Optimal Don't Care Filling for Minimizing Peak Toggles during At-speed Stuck-at Testing

A. Satya Trinadh, Department of Computer Science and Engineering, IIT Hyderabad
Seetal Potluri, Department of Electrical Engineering, IIT Madras
Sobhan Babu Ch., Department of Computer Science and Engineering, IIT Hyderabad
V. Kamakoti, Department of Computer Science and Engineering, IIT Madras
Shiv Govind Singh, Department of Electrical Engineering, IIT Hyderabad

Due to the increase in manufacturing/environmental uncertainties in the nanometer regime, testing digital chips under different operating conditions becomes mandatory. Traditionally, stuck-at tests were applied at slow speed to detect structural defects and transition fault tests were applied at-speed to detect delay defects. Recently, it was shown that certain cell-internal defects can only be detected using *at-speed stuck-at testing*. Stuck-at test patterns are power hungry, thereby causing excessive voltage droop on the power grid, delays the test response and finally leading to false delay failures on the tester. This motivates the need for peak power minimization during at-speed stuck-at testing. In this paper, we use input toggle minimization as a means to minimize circuit's power dissipation during at-speed stuck-at testing under the CSP-scan DFT scheme. For circuits whose test sets are dominated by don't cares, this paper maps the problem of optimal X-filling for peak input toggle minimization to a variant of interval coloring problem and proposes a dynamic programming (DP) algorithm (DP-fill) for the same along with a theoretical proof for its optimality. For circuits whose test sets are not dominated by don't cares, we propose a max scatter Hamiltonian path algorithm, which ensures that the ordering is done such that the don't cares are evenly distributed in the final ordering of test cubes, thereby leading to better input toggle savings than DP-fill. The proposed algorithms, when experimented on ITC99 benchmarks, produced peak power savings of up to 48% over the best known algorithms in literature. We have also pruned the solutions thus obtained, using Greedy and Simulated Annealing strategies with iterative 1-bit neighborhood, to validate our idea of optimal input toggle minimization as an effective technique for minimizing peak power dissipation during at-speed stuck-at testing.

Categories and Subject Descriptors: B.7.3 [**Integrated Circuits**]: Reliability and Testing

General Terms: Design, Algorithms, Reliability

Additional Key Words and Phrases: At-speed Stuck-at testing, Peak Test Power, Test cube ordering, Don't care filling, Dynamic Programming, Max Scatter Hamiltonian path algorithm, Greedy Pruning, Simulated Annealing

1. INTRODUCTION

Traditionally, stuck-at tests were applied at slow speed to detect structural defects and transition fault tests were applied at-speed to detect delay defects. However, in

Author's addresses:
A. Satya Trinadh and Sobhan Babu Ch., Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad; Shiv Govind Singh, Department of Electrical Engineering, Indian Institute of Technology Hyderabad; Seetal Potluri, Department of Electrical Engineering, Indian Institute of Technology Madras; V. Kamakoti, Department of Computer Science and Engineering, Indian Institute of Technology Madras;

DOI:http://dx.doi.org/10.1145/0000000.0000000

the nanometer complementary metal oxide semiconductor (CMOS) regime, process parameter variations cause unexpected variations in signaling delays. In the presence of path delays that are comparable to the clock interval, delayed signal transitions or timing hazards influence the detection of defects. Recently in [McCluskey and Tseng 2000; Vorisek et al 2004], it was shown that certain cell-internal defects can only be detected by *at-speed stuck-at testing*.

Launch-Off-Capture (LOC), Launch-Off-Shift (LOS) and Enhanced Scan (ES) are the well-known design-for-testability (DFT) schemes for the purpose of at-speed testing. Due to the excessive physical design overheads and limitations of ES, LOC and LOS are the two prevalently used schemes for this purpose. LOS scheme is known to achieve higher fault coverage while consuming lesser test time over LOC scheme, but dissipates higher peak power [Wu et al 2011] and requires generation of global at-speed scan enable signal. This excessive peak power in LOS scheme, leads to high *voltage droop* on the power grid, more than what the power grid is designed to handle. This excessive *voltage droop* specific to test mode, can lead to false delay failures, thereby leading to significant yield reduction [et al 2003; Girard et al. 2009; Pant et al 2010], that is unwarranted. In [Liu 2004], it was shown that transition fault testing can be performed with stuck fault patterns, with 46% lesser test time, if combinational state preservation (CSP) property can be satisfied in scan-shift mode. Since test compaction leads to increase in switching activity [Pomeranz 2015], at-speed stuck-at testing produces more switching activity, hence higher supply voltage droop [Potluri 2015], as compared to transition fault testing.

This paper focuses on reduction in peak circuit toggles during at-speed stuck-at testing, through minimization of peak input toggles. Recently, a new DFT scheme, called Combinational State Preservation scan (CSP-scan) [Potluri 2015] was proposed, which preserves the combinational logic state not only during scan-shift but also during capture and dissipates lower power consumption, with (1) no impact on test application time and fault coverage; (2) less than 2% overhead in each of area, functional timing and functional power; (3) the tester need not generate the at-speed scan-enable signal; and (4) does not need additional physical design effort, as compared to LOS scan testing, thus making it very practical to implement [Potluri 2015]. We assume that such a CSP-scan architecture is in place, and propose efficient algorithms for minimizing peak test power in its presence. During at-speed testing, the test patterns are applied to the combinational logic and the responses are captured. Automatic test pattern generation (ATPG) tools provide test cubes, in which some of the bits are specified and some are don't cares. The test patterns are obtained by filling the don't cares in these test cubes. The test cubes for some circuits are typically dominated by don't care bits, as shown in column 4 of Table I, clearly making don't care filling an effective technique for minimizing peak test power, for such circuits. On the contrary, the test cubes for some circuits have few don't care bits, as shown in column 4 of Table I, hence don't care filling does not play the primary role in minimizing peak test power, for such circuits. Motivated by this, this paper proposes an analytical solution for solving the problems of (1) optimal don't care filling for a given test cube ordering, and (2) test cube ordering, for minimizing peak input toggles during test application of at-speed stuck-at testing, under CSP-scan scheme

The main contributions of this paper are as follows:

— Given a test cube ordering, mapping the problem of optimal don't care filling for minimizing peak input toggles during testing to a variant of *interval coloring problem*;
— Proposing a polynomial time algorithm for the variant of *interval coloring problem* (DP-fill), its proof of correctness and optimality of the proposed algorithm;

Table I. Average % of don't care bits in test cubes. PIs and FFs stand for primary inputs and flip-flops respectively

Benchmark	$\#(PIs + FFs)$	# Gates	% of Don't cares
b01	5	57	7.1
b02	4	31	5
b03	29	103	70.4
b04	77	615	64.4
b05	35	608	36.8
b06	5	60	12.5
b07	50	431	58.6
b08	30	196	60.4
b10	28	217	58.7
b11	38	574	64.1
b12	126	1.6K	76.9
b13	53	596	65.4
b14	275	5.4K	77.9
b15	485	8.7K	87.8
b17	1452	27.99K	89.9
b18	3357	75.8K	86.9
b19	6666	146.5K	89.8
b20	522	9.4K	75.3
b21	522	9.4K	73.2
b22	767	13.4K	74.1

— We propose a Interleaving-based test cube ordering algorithm for test sets dominated by dont cares, that when used in conjunction with DP-fill, produces significant savings in input toggles;
— We propose a Max Scatter Hamiltonian path algorithm based test cube ordering scheme for test sets not dominated by don't cares, that when used in conjunction with DP-fill, produces significant savings in input toggles; and
— Local Search using Iterative 1-bit neighborhood to verify our assumption that optimizing peak input toggles leads to peak power reduction during at-speed stuck-at testing.

The next section motivates the need for at-speed stuck-at testing. Section 3 describes the related work for minimizing power during at-speed testing. Section 4 describes the combinational state preservation (CSP) property and the underlying CSP-scan [Potluri 2015] scheme, that satisfies this property. Sections 5 and 6 motivate the need for don't care filling and the formal definition of the problem of don't care filling for peak input toggle minimization respectively. Following this, our mapping of this problem to a variant of interval coloring problem, which we call as *bottleneck coloring problem*, is explained in section 7. Following this, the proposed algorithm for optimal don't care filling (DP-fill), along with its proof of correctness and the results obtained are shown in section 8. Section 9 contains the details of the proposed max scatter Hamiltonian path algorithm, used to optimize test cube ordering, for reducing peak test power. The improvement in solution quality obtained by pruning the solution thus obtained, with local search technique using iterative 1-bit neighborhood, is shown in section 10. Section 11 concludes the paper.

2. MOTIVATION FOR AT-SPEED STUCK-AT TESTING

The real defect is a short or an open between two nodes inside a gate. Such a defect can change the truth table of a gate, which may not exactly resemble stuck-at 0 or stuck-at 1 behavior. Apart from changing the truth table of a gate, the manifested defect can also change the delay of the gate. Such defects will not be caught by slow speed stuck-at tests and were known to be caught by at-speed stuck-at tests [McCluskey and Tseng

2000]. In fact, it was shown practically using the data from foundry, that at-speed stuck-at testing can greatly reduce test escapes [McCluskey and Tseng 2000; Vorisek et al 2004]. This motivates the need for at-speed stuck-at testing and is especially true for today's chips which are fabricated in deep-submicron technologies, that contain many small delay defects [Ahmed et al. 2006; Goel et al 2010; Yilmaz et al 2010; Bao et al 2013].

As explained earlier, peak power dissipation during at-speed stuck-at testing is higher than that during transition fault testing [Liu 2004], thereby causing excessive supply voltage droop and finally resulting in increase of gate delays inside the chip. Thus, after launching the test pattern into the combinational logic, the response time of the combinational logic may be delayed, and since we are capturing at-speed, we observe faulty response and discard a good chip, although it works well in the functional mode of operation (during when, the excessive delay on gates won't occur). Thus, the advantage of minimizing peak power dissipation during at-speed stuck-at testing is to avoid a good chip being categorized as defective, which is the problem of *false negatives*, that impact the chip's yield and hence a financial loss to the chip manufacturer. This motivates the need to minimize peak power dissipation during at-speed stuck-at testing. The next two sections shall describe the related work in the area of low power at-speed testing and a short description of the combinational state preservation scan (CSP-scan) architecture for low power at-speed stuck-at testing.

3. RELATED WORK

Several techniques were proposed in the past for minimizing peak test power [Girard et al. 2009]. These techniques can be broadly categorized into circuit level [Gerstendorfer and Wunderlich 1999; Parimi and Sun 2004; Bhunia et al 2005; Devanathan et al. 2007], gate level [Girard et al 1999; et al 2000; Almukhaizim et al 2008; et al 2008] and system level [Girard et al 1998; Dabholkar et al 1998; Sankaralingam and Touba 2002; et al 2007; Yao et al 2011] techniques. Circuit level techniques include supply gating [Bhunia et al 2005], scan flip-flop redesign [Gerstendorfer and Wunderlich 1999; Parimi and Sun 2004] and supply voltage scaling [Devanathan et al. 2007; Potluri et al. 2013]. Gate level techniques include clock gating [et al 2000; Sankaralingam and Touba 2002], scan cell output gating [et al 2008], and low power scan chain synthesis [Gerstendorfer and Wunderlich 1999; Girard et al 1999; et al 2000; Parimi and Sun 2004; Potluri et al. 2013]. System level techniques include low power test pattern generation [et al 2007], power aware test scheduling [Yao et al 2011], test pattern ordering [Girard et al 1998; Dabholkar et al 1998; Trinadh et al 2013] and don't care filling [Devanathan et al 2007; Wu et al 2011; Trinadh et al 2014]. All of these test pattern ordering and don't care filling techniques for Launch-Off-Shift (LOS) scheme [Devanathan et al 2007; Wu et al 2011; Trinadh et al 2014] are heuristics without a performance guarantee. A new scan flipflop was proposed by [Potluri et al 2015] that preserves combinational state of the circuit under test. The above paper showed empirically that the proposed scan-flop in conjunction with a naive dont't care (X)-aware test pattern ordering scheme resulted in significant reduction in Launch-To-capture switching activity. Keeping this in mind, this paper proposes a theoretical framework to arrive at an optimal X-filling and correspondingly efficient test cube ordering for minimizing peak test power. The following sections motivate this theoretical framework, the underlying design for testability (DFT) scheme necessary to apply the proposed technique, its proof of optimality and the results thereby obtained by applying the same on benchmarks. To the best of our knowledge, *this is the first ever reported don't care filling algorithm that is optimal in input toggle minimization*, in this connection.

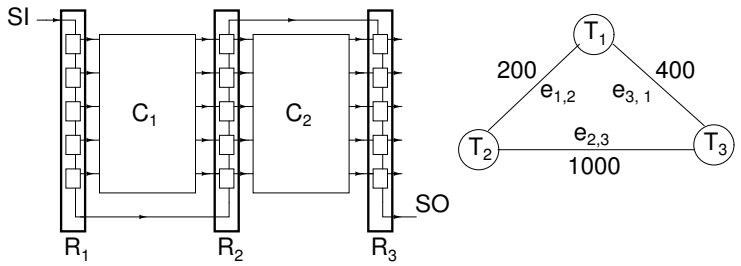

Fig. 1. Combinational State Preservation proposed in [Potluri 2015]

While it is true that there are already several papers on X-filling, none of them look at the problem from a theoretical perspective and hence not very effective. An interesting aspect of X-filling is that it neither incurs area nor timing overheads. However, high power dissipation during testing of today's complex chips, still continues to remain a major concern, because existing X-filling algorithms do not systematically target test power reduction. It is noteworthy that due to the several scan partitions available, interleaving these scan partitions, to control power consumption during testing is prevalently used in the industry. However, it is also well-known that this kind of test scheduling for controlling test power usually comes with a significant test time overhead. Thus, X-filling techniques can complement these existing scan chain interleaving techniques to reduce test power, without majorly increasing the test time. Hence, we focus on an X-filling algorithm that systematically reduces toggles during testing. To begin the discussion, we next provide a short description of combinational state preservation (CSP) and the DFT technique that preserves the CSP property [Potluri 2015].

4. COMBINATIONAL STATE PRESERVATION

All the existing scan architectures proposed in the past, violate combinational state preservation (CSP) property during the capture phase [Potluri 2015]. To address this issue, the CSP-scan architecture is proposed in [Potluri 2015], that preserves combinational logic states during scan-shift as well as capture phases of at-speed testing. Under CSP-scan, test pattern ordering can be very effective in reducing peak launch-to-capture power during scan based testing of sequential circuits. To understand this in little more detail, let S be a sequential circuit and $\tau = \{t_1, t_2 \dots\}$ be the test pattern set designed for it by the ATPG tool. Let the combinational parts of S be labeled as $\{C_1,\ C_2 \dots\}$. Let $s^k_{i,j}$ be the state of the i^{th} wire in combinational part C_k after launching test pattern T_j. If the state in which the combinational parts settle down, after launching test pattern T_j, is preserved until launching of the next test pattern T_{j+1}, the switching activity in C_k immediately after launching T_{j+1} is given by $\sum_{i \in set\ of\ wires} s^k_{i,j} \oplus s^k_{i,j+1}$. It is interesting to note that the switching activity in S is dependent on the order of application of test patterns. Figure 1 shows how the combinational logic states are so preserved that the sequential circuit can practically be treated like a combinational circuit, and we can perform test pattern ordering for minimizing peak switching activity. To illustrate the above, consider the circuit shown in Figure 1. There are 2 combinational parts in this circuit, namely C_1 and C_2, separated by 3 register stages R_1, R_2 and R_3. All the flip-flops in the register stages are connected together to form a scan chain, which is subsequently used to load the test pattern. The test patterns are generated post-synthesis for combinational circuits C_1

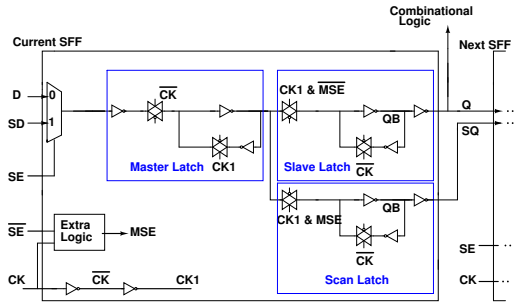

Fig. 2. Scan flip-flop that implements the CSP-scan scheme [Potluri 2015]

and C_2 separately. Let $\{t_1, t_2 \ldots t_r\}$ be the set of tests for C_1 and $\{p_1, p_2 \ldots p_s\}$ be the set of tests for C_2.

The testing of the chip comprises repetition of the three steps, namely, *scan-shift*, *launch* and *capture*. First, $\langle p_1, t_1 \rangle$ is shifted in that order into register stages R_1 and R_2 respectively (*scan-shift step*) through the scan input pin SI as shown in Figure 1; t_1 is applied to C_1 and p_1 is applied to C_2 (*launch step*); and, the result r_1 due to application of t_1 on C_1, and the result r_2 due to application of p_1 on C_2, is captured in register stages R_2 and R_3 respectively (*capture step*). In the next iteration, $\langle p_2, t_2 \rangle$ is shifted into the circuit which causes the responses $\langle r_1, r_2 \rangle$ to be shifted out of the circuit through the scan output pin SO as shown in Figure 1. Note that the last shift operation in the shift step of the scan-in operation is used as the launch step too.

Let us assume that there are 3 test patterns in the test set, namely T_1 (includes t_1 and p_1), T_2 (includes t_2 and p_2) and T_3 (includes t_3 and p_3). The same figure shows the *test pattern graph* for this circuit. This is a complete graph, with each vertex representing a test pattern, and edge-weights representing the switching activity in S, based on the order of application of test pattern pairs. Since the goal is to reduce dynamic power during launch-capture window without affecting the at-speed stuck-at fault coverage, we have used switching activity (which is a measure of dynamic power) as a cost function to represent the edge-weights in the test pattern graph. As shown in this figure, applying T_1 after T_2 or vice-versa causes 200 toggles (represented as edge-weight $e_{1,2}$) and so on. We assume that the edge-weights $e_{1,2}$, $e_{2,3}$ and $e_{3,1}$ are equal to 200, 1000 and 400 respectively. In this case, if the order of application of test patterns is $T_1 \rightarrow T_2 \rightarrow T_3$, then peak switching activity occurs when test pattern T_3 is applied after test pattern T_2. This corresponds to 1000 toggles in the entire circuit, as represented by edge-weight $e_{2,3}$. On the other hand, if the order of application of test patterns is $T_3 \rightarrow T_1 \rightarrow T_2$, then peak switching activity occurs when test pattern T_1 is applied after test pattern T_3, which corresponds to 400 toggles, as represented by the edge $e_{3,1}$ in Figure 1. Thus, test pattern ordering [Girard et al 1998] has significant impact on the peak switching activity in the circuit, under CSP-scan.

The scan flip-flop that implement the CSP-scan scheme is shown in Figure 2. The timing diagram corresponding to the CSP-scan scheme is shown in Figure 3. The CSP-scan can be summarized [Potluri 2015] as follows:

(1) A Muller C-element is used to generate SE_{latch} signal, as shown in Figure 4, which takes SE and CK signals as input. A $1 \rightarrow 0 \rightarrow 1$ transition is produced on SE signal just before the launch point, to facilitate the $1 \rightarrow 0$ transition on SE_{latch} (Figure 3 - pt. **b**). This ensures the functional slave latch turns ON (and the alternate slave latch turns OFF), thus launching the test pattern into the combinational logic.
(2) Some extra logic is used within the scan flip-flop, as shown in Figure 4, which produces at-speed $1 \rightarrow 0$ transition on SE_{mux}, marked **f** in Figure 3.

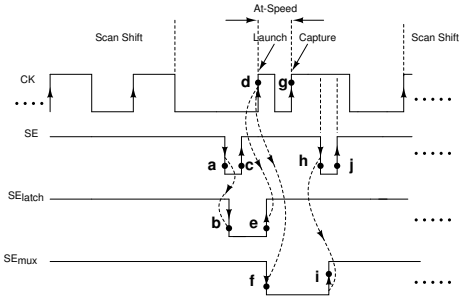

Fig. 3. Timing diagram for CSP-scan scheme [Potluri 2015]

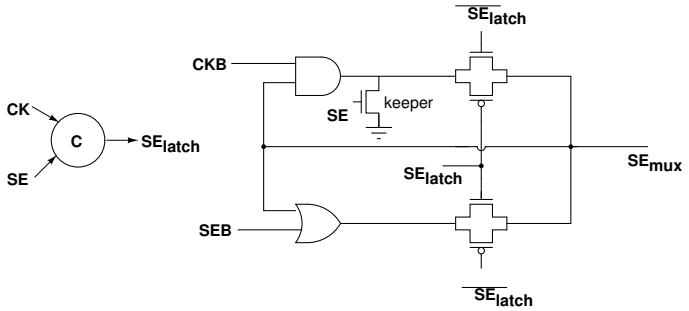

Fig. 4. Extra Logic [Potluri 2015]

(3) After launch point **d**, when $\{SE, CK\}$ reaches a 11 state, there is a $0 \rightarrow 1$ transition on SE_{latch}, marked as **e** in Figure 3. This transition on SE_{latch} signal, turns OFF the functional slave latch and turns ON the alternate slave latch before the capture point marked **g** shown in the same figure. As a result, at the positive edge of the capture clock, the response data that is waiting at the output of the master latch of each scan flip-flop gets latched into the corresponding alternate slave latches, **leaving the functional slave latches undisturbed, thus satisfying the CSP during the capture step**.
(4) An extra $1 \rightarrow 0 \rightarrow 1$ pulse on SE signal is produced to ensure that SE_{mux} signal changes back from $0 \rightarrow 1$ after the capture step. To ensure that SE_{latch} signal is not disturbed due to this modification of SE signal, the positive pulse of the capture clock is slightly stretched till the end of SE pulse, marked **j** in Figure 3 .

Similar to the techniques proposed in [Gerstendorfer and Wunderlich 1999; Parimi and Sun 2004], CSP-scan also avoids shift power by ensuring that the changing flip-flop values do not propagate to this circuit during shift mode. However, the most important difference is that, these techniques ensure combinational state preservation (CSP) only during scan-shift, whereas CSP-scan ensures CSP both during scan-shift and capture steps. Because CSP-scan ensures CSP during the capture step, the sequential circuit can be treated as combinational circuit during scan based testing, thereby making test pattern ordering a very effective technique for reducing peak capture power.

The CSP scan is also best suited for at-speed delay testing. Assume that a $0 \leftarrow 1$ transtion delay need to be tested on an logic element l of the circuit under test. Let t_1 and t_2 be the test vectors that shall set l to 0 and 1 respectively. First t_1 is shifted, then

launched and results re caprtured at-speed. At this point l is set to 0. Now, t_2 is shifted in. Due to the CSP property, l remains at 0 till t_2 is launched and results captured at-speed. Once t_2 is launched there is a $0 \leftarrow 1$ transition on l that is captured at-speed, thus, carrying out the desired at-speed delay testing. Since two test vectors t_1 and t_2 are involved these technqiues are called two-pattern delay tests.

The enhanced scan [Dervisoglu and Stong 1991] and hold [Bhunia et al. 2004; 2005], techniques are also used to implement two-pattern delay tests. The main difference between the propose scheme and the enhanced-scan and hold schemes is that in the latter the combinational logic state of the circuit under test is undisturbed during the scan-shift of the second pattern, however the combinational logic state is disturbed during capture, while the proposed CSP-scan preserves the state of combinational logic not only during scan-shift but also the capture step of *every pattern*. the advantage of this can be best explained through an example. Assume in Figure 1 that the scan registers R_1, R_2 and R_3 are designed according to enhanced scan or hold techniques as proposed in [Bhunia et al. 2004; 2005; Dervisoglu and Stong 1991]. Consider logic elements l_{21} and l_{22} in the combinational circuit C_2 on whom a $0 \leftarrow 1$ delay test need to be performed. Assume test vectors t_{21}, t_{22} and t_{23} will set l_{21} to 0, 1, 1 and l_{22} to 0, 0, 1 respectively. Thus, (t_{21}, t_{22}) and (t_{22}, t_{23}) are two-pattern delay tests for testing the $0 \leftarrow 1$ transition delay on l_{21} and l_{22} respectively. To carry out these delay tests, first the (t_{21}, t_{22}) pair is loaded into R_2. In practice there will also be tests loaded into R_1 (say (t_{11}, t_{12}) for testing C_1 while the faults in C_2 are tested. At the end of this test, R_2 will be loaded with the response of C_1 to t_{12} and hence line l_{22} of C_2 need not necessarily carry the value 0. Therefore, to test a $0 \leftarrow 1$ transition on l_{22} again t_{21} need to be shifted into R_2 followed by t_{22}. This is not needed in the case of the proposed CSP-scan as it preserves the combinational state during the capture step. In this case, (t_{21}, t_{22}, t_{23}) can be shifted one after another in the same sequence to test the $0 \leftarrow 1$ delay in L_{21} and l_{22}, thereby saving an extra shift of t_{22}. Based on the above explanation, it is also straightforward to note that the test vector reordering explained earlier in this section (Figure 1) is *not applicable* in the case of enhanced-scan and hold techniques proposed in [Bhunia et al. 2004; 2005; Dervisoglu and Stong 1991].

To sumup, the proposed CSP-scan can do the same job of enhanced scan/hold in applying two-pattern delay tests. However, because of preserving combinational logic state during capture, CSP-scan is also capable of reducing peak launch-capture power during at-speed stuck-at testing, through test pattern ordering, which is not possible with enhanced scan/hold. That is the important additional capability of CSP-scan, over and above enhanced scan/hold. Additionally, CSP-scan has significantly lesser area, timing and power overheads as compared to enhanced scan and hold, as explained in [Potluri 2015].

This paper focuses on ordering the test cubes and selectively filling the don't care (X) bits in the test cubes to minimize peak power during at-speed stuck-at testing, under CSP-scan scheme. Next, we motivate the need for don't care filling for power reduction during at-speed stuck-at testing, under CSP-scan.

5. MOTIVATION FOR DON'T CARE FILLING FOR POWER REDUCTION DURING AT-SPEED STUCK-AT TESTING UNDER CSP-SCAN SCHEME

In scan based test, the input test pattern is serially shifted in, while serially shifting out the response for previous test pattern. In CSP-scan [Potluri 2015], since the combinational logic is undisturbed during scan-shift and capture phases, *as far as application of test patterns is concerned, the sequential circuit behaves like a combinational circuit. Thus, test pattern ordering technique that was proposed earlier for reducing test power in combinational circuits [Girard et al 1998; Dabholkar et al 1998] becomes equally effective for sequential circuits.* Having understood this, the next step

Test Vector Sequence					X-Stat										Optimum-Fill				
					Phase 1					Phase 2									
V1	V2	V3	V4	V5	V1	V2	V3	V4	V5	V1	V2	V3	V4	V5	V1	V2	V3	V4	V5
0	X	X	1	X	0	0	X	1	1	0	0	0	1	1	0	1	1	1	1
1	X	X	X	0	1	1	1	X	0	1	1	1	0	0	1	1	1	1	0
0	X	X	1	X	0	0	X	1	1	0	0	1	1	1	0	0	1	1	1
1	X	X	X	0	1	1	1	X	0	1	1	1	0	0	1	1	1	1	0
1	X	1	X	0	1	1	1	X	0	1	1	1	1	0	1	1	1	0	0
X	0	X	X	1	0	0	0	X	1	0	0	0	0	1	0	0	1	1	1
1	0	X	X	1	1	0	0	X	1	1	0	0	0	1	1	0	0	1	1
										1	1	3	3		2	2	2	2	

Fig. 5. X-Stat [Trinadh et al 2014] vs Optimum-Fill

is to compute a test cube ordering that achieves the same for sequential circuits under CSP-scan. Once the test cube ordering is computed, the next step is to minimize the peak toggles at the inputs (primary inputs and the scan cell outputs) through filling of the don't care bits in test cubes with binary values. The expectation is that reducing the input toggles leads to reduction in power dissipation inside the circuit, as shown previously in [Girard et al 1998].

The most recent and effective don't care filling algorithm for peak power minimization during at-speed stuck-at testing under CSP-scan scheme is X-Stat [Trinadh et al 2014]. The X-Stat algorithm follows a two phase approach. In the first phase, it uses adjacent don't care fill technique to convert don't care stretches 0XX...X1 and 1XX...X0 into smaller don't care bit stretches 0X1 and 1X0 respectively, as shown in *Phase 1* column of Figure 5. In the second phase, it replaces don't care bits by either 0 or 1 in order to minimize peak toggles as shown in *Phase 2* column of Figure 5. This figure shows that the global minimum peak toggles is 2 (shown under the *Optimum-Fill* column), while the minimum peak toggles achieved by XStat technique is 3, making it sub-optimal. Because of the greedy approach used in *Phase 1* of XStat technique, it does not achieve the global optimum. Motivated by this, we choose a *Dynamic Programming* paradigm which takes global picture into consideration and optimally fills the don't care bits with binary values to achieve the best reduction in peak toggles.

6. PEAK INPUT TOGGLE MINIMIZATION PROBLEM

The power dissipation is directly proportional to the number of toggles in the circuit. A best estimate of power can be obtained by considering the toggle axctivity in the entire circuit. In the present case of X-filling, every test vector with k don't cares imply 2^k fully-specified test vectors. Assume n test vectors, each with k_i don't cares, $1 \leq i \leq n$, we land up with $\sum_{i=1}^{n} 2^{k_i}$ test vectors. Constructing a test pattern graph as in Figure 1 with the edge-weights as the total toggle in the circuit is a prohibitively time and memory intensive computational task. The results reported in [Girard et al 1998] show that the input switching activity between two test patterns can be used as a reasonable estimate of the switching activity of the entire circuit in general though there are circuits including code converters and decoders that may not adhere to this property. This has motivated the technique proposed in the paper to fill the don't cares with an objective to minimize the input switching activity. A subsequent test pattern ordering of the n fully specified test vectors may be performed assuming the entire circuit switching activity. This will involve only ${}^nC_2$ computations to construct a test pattern graph as shown in Figure 1.

Objective: Given a sequence of test cubes $T_1, T_2, ...T_n$, each of length m, replace each don't care in test cubes by either 0 or 1, such that $max\{Hd(T_1,\ T_2),\ Hd(T_2,\ T_3) \ldots Hd(T_{n-1},\ T_n)\}$ is $minimized$, where $Hd(T_i, T_{i+1})$ is the Hamming distance between test cubes T_i and T_{i+1}, after replacing don't cares by either 0 or 1.

This problem can be formulated as a variant of interval coloring problem, which we call *Bottleneck Coloring Problem*. Next, we define and explain the *Bottleneck Coloring Problem* and show that peak power minimization is an instance of this problem. Since our objective is to minimize the peak toggles, we call this problem as *Bottleneck Coloring Problem*.

7. BOTTLENECK COLORING PROBLEM (BCP)

7.1. Problem Explanation in Terms of Hotel Room Booking

Suppose a hotel received several guest requests for accommodation, each of which has a *start-date* and an *end-date* of a time period, and asks the hotel to provide accommodation for exactly one day which falls in the given period. The aim of the hotel is to assign rooms to all guest requests such that the number of guests staying in the hotel on any given day is minimized, which is a variant of the *interval coloring problem* [West 2000].

7.2. Mathematical Definition of the Problem

(1) Let S = (s_1, e_1), $(s_2, e_2)\ \ldots\ (s_k, e_k)$ be a sequence of intervals such that s_i and e_i are integers corresponding to starting and ending times of interval i respectively, $\forall 1 \leq i \leq k$;
(2) Let $max\_color = max(e_1,\ e_2,\ e_3, \ldots e_k)$;
(3) Let $\{\ c_1,\ c_2,\ c_3 \ldots c_{max\_color}\ \}$ be a set of colors;
(4) For each interval (s_i, e_i), assign a color c_j such that $s_i \leq j \leq e_i$;
(5) Let $h_1,\ h_2,\ h_3 \ldots h_{max\_color}$ be a sequence of integers such that h_j be the number of intervals which are assigned color c_j; and
(6) Our objective is to assign colors to intervals such that $max(h_1,\ h_2 \ldots h_{max\_color})$ is minimized.

Here, each interval corresponds to an accommodation request (as explained in subsection 7.1). Each color corresponds to a day. Assigning color c_j to the interval (s_i, e_i) is same as allocation of hotel room on j^{th} day to this request. Note that h_j denotes the number of guests who are assigned room on j^{th} day. Having defined BCP, next we shall explain how the peak input toggle minimization problem maps to BCP.

7.3. Mapping of Peak Input Toggle Minimization Problem to BCP

(1) Let $T_1, T_2, ...T_n$ be a sequence of test cubes each of length m;
(2) Construct a $m \times n$ matrix A such that i^{th} column of A is equal to the test cube T_i;
(3) **for** $i = 1 \rightarrow m$ **do**

```
/* Preprocessing of 0XX..X0,1XX..X1 stretches                    */
```

If $\{\ i^{th}$ row contains a sub-sequence 0XX...X0$\}$ then replace every don't care in this sub-sequence by zero, since there exists an optimal solution, in which all of these don't cares are replaced by zeroes, irrespective of how other don't cares are replaced.

If $\{\ i^{th}$ row contain a sub-sequence 1XX...X1$\}$ then replace every don't care in this sub-sequence by one, since there exists an optimal solution, in which all of these don't cares are replaced by ones, irrespective of how other don't cares are replaced.

(4) Let S=ϕ

(5) **for** $i = 1 \rightarrow m$ **do**

`/* Creating intervals for 0XX..X1,1XX..X0 stretches */`

If there exist $k < l$ such that $A_{i,k}$ = **0**, $A_{i,l}$ = **1** and $A_{i,k+1}...A_{i,l-1}$ are don't cares, then append an interval $(k, l-1)$ to sequence of intervals S.
Comment 1 : Note that there exists an optimal solution to *Peak Toggle Minimization Problem* such that $A_{i,k}$ = **0**, $A_{i,k+1}$ = **0**, ... , $A_{i,j}$ = **0**, $A_{i,j+1}$ = **1**, $A_{i,j+2}$ = **1**, ... , $A_{i,l}$ = **1**, where $k \leq j < l$, irrespective of how other don't cares are replaced. There is only one toggle between j^{th} and $j + 1^{th}$ test vectors in this sub-sequence. The color assigned to this newly added interval in the solution of *BCP* captures the location of this toggle in this sub-sequence.

If there exist $k < l$ such that $A_{i,k}$ = **1**, $A_{i,l}$ = **0** and $A_{i,k+1}...A_{i,l-1}$ are don't cares then append an interval $(k, l-1)$ to sequence of intervals S.
Comment 2 : Note that there exists an optimal solution to *Peak Toggle Minimization Problem* such that $A_{i,k}$ = **1**, $A_{i,k+1}$ = **1**, ... , $A_{i,j}$ = **1**, $A_{i,j+1}$ = **0**, $A_{i,j+2}$ = **0**, ... , $A_{i,l}$ = **0**, where $k \leq j < l$, irrespective of how other don't cares are replaced. There is only one toggle between j^{th} and $j + 1^{th}$ test vectors in this sub-sequence. The color assigned to this newly added interval in the solution of *BCP* captures the location of this toggle in this sub-sequence.

Each row of the matrix represents an input pin to the circuit (corresponds to a guest in BCP) and each column represents a test cube (corresponds to a *day* in the BCP formulation as per section 7.1). A toggle in i^{th} row, from j^{th} position to $(j + 1)^{th}$ position corresponds to a hotel room allocation for i^{th} customer on j^{th} day. The BCP ensures that the number of allocations on any given day is minimized, which in the current context, translates to minimization of number of the peak input toggles on any given test cycle (launch-capture duration). Having explained how the problem under consideration maps to BCP, next we proceed to explain how to construct solution to the given problem.

7.4. Constructing Optimal solution for *Peak Input Toggle Minimization Problem* from Optimal solution for *Bottleneck Coloring Problem*

Having known how to construct optimal solution to BCP, the following steps are followed in constructing the optimal solution for the given problem:

(1) Suppose color c_j is assigned to interval (s_i, e_i) in the given optimal solution for *Bottleneck Coloring Problem*;
(2) Look at the row in matrix A, corresponding to interval (s_i, e_i), and make all bits from column s_i to column j same as bit value at column s_i and make all bits from column $j + 1$ to column $e_i + 1$ same as bit value at column $e_i + 1$

Having understood the solution for the given problem, through solution to BCP, next we shall see how to solve BCP optimally.

8. ALGORITHM

The Algorithm used to compute BCP is composed of two phases: (1) lower bound computation; and (2) achieving the lower bound, and thereby the optimal solution. First, we shall describe the lower bound computation phase.

ALGORITHM 1: Computing Lower-Bound

Input: S= (s_1, e_1),$(s_2, e_2) \ldots (s_k, e_k)$ be a sequence of intervals
Output: *Lower-Bound Value.*

1 Let $T_{i,j}$, where $i \leq j$, denote the number of intervals whose starting time is $\geq i$ and ending time is $\leq j$;

2 If $i > j$ then let $T_{i,j}$ = 0, else $T_{i,j}$ can be expressed recursively as follows : $T_{i,j} = T_{i,j-1}$ + $T_{i+1,j}$ - $T_{i+1,j-1}$ + Number of intervals whose staring time is equal to i and ending time is equal to j.
/* Note that $T_{i+1,j-1}$ is subtracted since the set of intervals whose starting time is at least $i+1$ and ending time is at most $j-1$, are counted in both $T_{i,j-1}$, $T_{i+1,j}$. */

3 Let $max\_color = max(e_1, e_2 \ldots e_k)$

4 $Lowerbound\ LB = max\{\lceil \frac{T_{i,j}}{j-i+1} \rceil | 1 \leq i \leq j \leq max\_color\}$
/* If we take any interval whose starting time is at least i and ending time at most j, then we should assign a color c_k to this interval, such that $i <= k <= j$. This means there exists a color c_k, such that at least $\lceil \frac{T_{i,j}}{j-i+1} \rceil$ intervals are assigned color c_k, where $i <= k <= j$ */

Result: $return\ LB$

ALGORITHM 2: Assigning color to intervals

Input: S= (s_1, e_1),$(s_2, e_2) \ldots (s_k, e_k)$ be a sequence of intervals, LB - lower-bound
Output: Intervals with assigned colors

1 Sort the intervals in S based on starting time.

2 Let *H be a min heap*. Each node of this heap can store information of an interval (starting time and ending time). Nodes of this heap are ordered by ending times of intervals i.e ending time of interval stored in a node is less than or equal to ending times of intervals stored in that node's children.

3 **for** $i = 1 \rightarrow n$ **do**
4 Insert into heap H all intervals whose starting time is equal to i.
/* if we take any interval in H starting time is at most i. */
5 Remove top l elements from heap and assign color c_i, where $l = min(current\ heap\ size, LB)$;
/* The reason for picking top elements and assigning colors c_i is we want to assign colors to intervals which are ending soon. We prove in section *Proof of correctness* that ending times of all these removed intervals are at least i. */

8.1. Dynamic Programming Approach to compute Lower-Bound (LB) for the *Bottleneck Coloring Problem*

Algorithm 1 gives the lower bound on the number of intervals which are assigned the same color. This algorithm can be implemented such that the running time is $O(k^2)$, where k is the number of intervals. Having described the lower bound computation phase, we next proceed to describe the next phase of achieving the lower bound, using a greedy approach.

8.2. Greedy Approach to *Bottleneck Coloring Problem*

Algorithm 2 assigns a color c_j for each interval $(s_i$,$e_i)$, where $s_i \leq j \leq e_i$, and the maximum number of intervals which are assigned the same color, is at most the lower

bound value computed in Algorithm 1. Since the lower bound computation involves dynamic programming (DP), we call this X-filling algorithm as DP-Fill. Running time of this algorithm is O ($k\ log\ k$), where k is the number of intervals.

8.3. Proof of correctness

In the following paragraph, we will prove that, at the end of i^{th} iteration of Algorithm 2, ending times of all intervals contained in *min heap* are greater than i. This means that each interval (s_i,e_i) is assigned a color c_j such that $s_i \leq j \leq e_i$.

Suppose, at the end of some iteration i, *min heap* contains an interval, whose ending time is greater than i. Let i be such that it's value is minimum. Let $j < i$ such that number of intervals which are assigned color in j^{th} iteration, is less than the *lower bound*. Let j be such that it's value is maximum. If there is no such j, then let $j = 0$. Let $j < k < i$ such that in the k^{th} iteration, the above algorithm assigned color to an interval whose ending time is more than i. Let k be such that it's value is maximum. If there is no such k, then let $k = j$. Ending times and starting times of all intervals, which are assigned color from $k + 1^{th}$ iteration to i^{th} iteration, are less than or equal to i and greater than k respectively. Note that the number of intervals which are assigned colors from $k + 1^{th}$ iteration to i^{th} is equal to $lowerbound * (i - k)$ and *min heap* contains an interval whose ending time is equal to i and starting time is greater than k. This implies that the number of intervals, whose starting time is greater than k and ending time is less than or equal to i, is more than $lowerbound * (i - k)$, which is a contradiction.

Hence, we have proposed an *optimal* algorithm using dynamic programming, for minimizing peak input toggles during at-speed stuck-at testing, for a given test cube ordering, under CSP-scan scheme. Thus, we call this algorithm as dynamic programming based don't care filling (DP-fill) algorithm. As already discussed, CSP-scan satisfies CSP property, thereby making the ordering of the test cubes influence the peak input toggles during at-speed stuck-at testing. The next section solves the problem of ordering the test cubes, with large don't care stretches, for minimizing the peak input toggles, using the technique of *interleaving*.

The entire mapping of input toggle minimization, consists of two phases: optimal test cube ordering, followed by X-filling. It should be noted that the proposed X-filling algorithm gives the optimal X-filling, after the test cube ordering phase is completed. Since TSP can be reduced to an instance of the test pattern ordering problem [Girard et al 1998; Dabholkar et al 1998], hence the test pattern ordering problem is NP-hard. Hence, it is not possible to prove the optimality of the test pattern ordering, and hence the optimality of the entire mapping. Thus, our attempt in this paper was to target optimality on an important sub-space of the mapping i.e, X-filling after the test cube ordering is completed. So, we propose an optimal algorithm based on interval-coloring for X-filling for a given test cube ordering, to minimize peak input toggles.

8.4. Test Vector Ordering Algorithm for Large Don't Care Stretches

For a given vector ordering, Algorithm 2 gives the optimum value of peak input toggles. Note that if the length of don't care stretches in the rows of matrix A (which is defined in section 7) is high, then the optimum value of peak input toggles is small. To achieve such a large don't cares stretches in the rows of matrix A, we propose Algorithm 3 for test vector ordering. We call this ordering as *interleaved* test vector ordering (I-Ordering). The number of times the *while loop* in Algorithm 3 gets executed is shown in Figures 6 and 7. These experimental observations show that the number of iterations grow as $O(log(n))$, where n is the number of test vectors. Figure 8 analyzes the don't care stretch statistics in the test cubes of `b19` circuit, for different test cube orderings. One can observe that I-Ordering increases the sizes of don't care stretches,

ALGORITHM 3: Computing Test vector Ordering for Large Don't Care Stretches

Input: T = T_1, T_2,.. T_n be the set of input test vectors.
Output: S = Ordering of input test vectors.

Let $T^{'}$ = T_1^1, $T_2^{'}$,.. $T_n^{'}$ be the ascending order of input test vectors, sorted based on the number of don't cares.
Let $current\_optimal\_value = \infty$
Let $k = 0$
Let $exit\_flag = false$
while $exit\_flag = false$ **do**
    Let $k = k + 1$ `/* Interleaving size */`
    Let $S = \emptyset$
    for $i = 1 \rightarrow \lfloor \frac{n}{k+1} \rfloor$ **do**
        `/* pick` i^{th} `vector from` $T^{'}$ `and append to S */`
        $S = S, T_i^{'}$
        `/* pick` $n-(i-1)*k$ `th vector to` $n-(i-1)*k-k+1$ `th vector from` $T^{'}$ `and append to S */`
        $S = S, T^{'}_{n-(i-1)*k}, T^{'}_{n-(i-1)*k-1}, ..T^{'}_{n-(i-1)*k-k+1}$
    S = $S \cup (T^{'} - S)$.
    Let $temp\_optimal\_value$ be the optimal bottleneck value computed on sequence S using Algorithm 2
    if $temp\_optimal\_value < current\_optimal\_value$ **then**
        $current\_optimal\_value = temp\_optimal\_value$;
    else
        $exit\_flag = true$;

Result: $return\ S$

which are finally exploited by the proposed Algorithm 2, to achieve the best possible reduction in peak input toggles. The results obtained by using this technique, is shown for different possible don't care fillings, in Table II. It can be seen that for test sets with large number of don't cares, the combination of I-Ordering and DP-fill, which is proposed in this paper, has best reduction in peak input toggles. Next, we will see how to reduce peak input toggles, when the test sets have small don't care stretches. The next section maps the problem of ordering the test cubes, for minimizing the peak input toggles, when the test sets have small don't care stretches, to an instance of Max Scatter Traveling Salesman Problem.

9. TEST VECTOR ORDERING ALGORITHM FOR SMALL DON'T CARE STRETCHES

In this section, we map the problem of ordering test cubes for minimizing peak input toggles during at-speed stuck-at testing, to an instance of the Maximum Scatter Traveling Salesman Path (or Max Scatter Hamiltonian Path) Problem (MSTSPP). The definition of MSTSPP is as follows:

Maximum Scatter Traveling Salesman Path Problem (MSTSPP):
Given an edge-weighted undirected graph G, the *Maximum Scatter Traveling Salesman Path Problem (MSTSPP)* is to find an *Hamiltonian path* in G such that the *smallest edge cost in this path is maximized*. The MSTSPP is NP-Hard [Arkin et al 1997].

Consider an edge-weighted undirected complete graph $G = (V, E)$, with test cubes representing the vertices, and the edge-cost function shown in Table III, used to represent the edge weight, $\forall 1 \leq i < j \leq n$, where n is the test set size. The intuition is that, if we solve the Max Scatter Hamiltonian path problem in this graph, we ensure

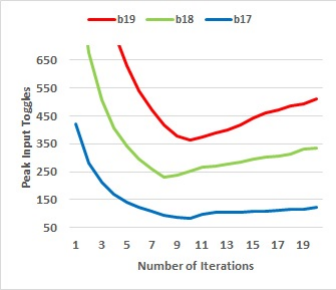

Fig. 6. Number of Iterations vs Peak Input Toggles

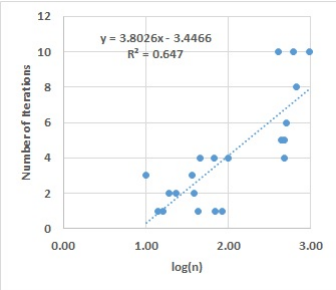

Fig. 7. Optimum Number of Iterations vs log(n)

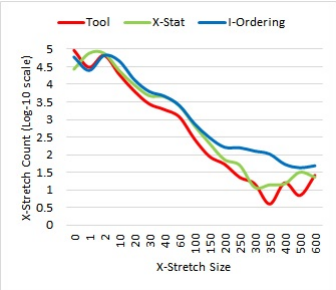

Fig. 8. Don't care stretch statistics for b19 circuit (Tool vs X-Stat[Trinadh et al 2014] vs I-Ordering

that, the test cube pair with the lowest number don't cares in the ordering, is rich with don't cares (min-max optimization), which in turn maximizes the reduction in peak test power. It is well-known that the problem of test vector ordering, for minimizing test power in combinational circuits, maps to traveling salesman problem (TSP) [Girard et al 1998]. The following points clarify the difference between this mapping and the proposed mapping:

(1) An important difference to note here is that we are not dealing with combinational circuits, but with sequential circuits. Because of CSP-scan, ordering becomes effective to reduce test power in sequential circuits also through test vector ordering;
(2) Our claim is that when test cubes are rich with dont care bits, interleaved ordering, when combined with DP-fill, gives the best savings in launch-capture power. On the other hand, when test cubes are not rich with dont care bits, max scatter TSP based ordering, combined with DP-fill, gives the best savings in launch-capture power. Again, it should be noted that max scatter TSP is not the same as TSP, because in TSP, we try to reduce the overall tour length, whereas in max scatter TSP, we try to maximize the smallest edge-weight along the tour;

Table II. Peak Input Toggles : I-Ordering with different don't care fillings

Circuit	MT-fill	R-fill	0-fill	1-fill	B-fill	DP-fill
b01	3	4	4	3	3	3
b02	3	3	3	3	3	3
b03	12	19	15	15	8	6
b04	41	45	43	39	23	15
b05	20	22	21	23	15	14
b06	4	4	4	4	4	4
b07	24	31	38	23	15	11
b08	16	18	16	14	8	6
b09	14	18	16	16	11	11
b10	10	18	14	13	9	7
b11	15	25	22	18	10	9
b12	59	72	99	65	30	15
b13	28	31	28	23	15	10
b14	168	158	208	148	77	40
b15	296	267	314	193	141	33
b17	882	770	953	676	419	85
b18	2030	1741	2200	1550	980	232
b19	3862	3436	4340	3167	1871	364
b20	301	285	352	284	143	65
b21	280	286	333	237	129	67
b22	451	409	475	425	210	91

Table III. Edge cost function used for MSTSP

V_i	V_j	$Cost$
0	0	+2
1	1	+2
0	X	+1
1	X	+1
X	0	+1
X	1	+1
0	1	-1
1	0	-1
X	X	+3

(3) There is another major difference. The standard mapping of test vector ordering to TSP, to minimize test power in combinational circuits, is when the test cubes are completely specified. In our case, we are using max scatter TSP, to order the test cubes which are not completely specified i.e., some of the bits are dont cares; and finally
(4) The goal of the standard mapping of test vector ordering to TSP, is to reduce toggles. On the contrary, we are using max scatter TSP to order test cubes, in order to maximize the dont care pairs in the adjacent test cubes along the ordering, so that the subsequent dont care filling step will effectively reduce toggles.

We used the algorithm for Max Scatter Traveling Salesman Problem (MSTSP) given in [Larusic et al. 2012] to find the Max Scatter Traveling Salesman Path in G. Since our requirement is Max Scatter Hamiltonian path, but MSTSP algorithm gives Max Scatter Hamiltonian cycles, we modified G to $G^{'}$ as follows and given $G^{'}$ as input to MSTSP algorithm. Construct a graph $G^{'}$ as follows:

— Add an vertex v_{k+1} to G.
— Place edge between v_{k+1}, v_i in G with a cost zero, where $1 \leq i \leq k$.

Note that there is an one to one correspondence between Hamiltonian path in G and Hamiltonian cycle in $G^{'}$, since vertex v_{k+1} is connected to every vertex in G. Since the edge cost of any edge between v_{k+1} to vertex in G is zero, Optimal Max Scatter Hamiltonian Cycle cost in $G^{'}$ is same as Optimal Max Scatter Hamiltonian Path cost in G. In [Larusic et al. 2012], efficient heuristic algorithm was developed to solve the MSTSP problem by converting it to an instance of the Bottleneck TSP (BTSP) [Larusic et al. 2012]. As part of this algorithm, construct a graph $G^{''}$ as follows:

— Find the maximum edge-weight of all edges in graph $G^{'}$, and let it be e_{max}; and
— Replace each edge, $e_{i,j}$ in $G^{'}$ by $e_{max} - e_{i,j}$ to obtain $G^{''}$.
— Find the Hamiltonian path in $G^{''}$ such that minimum edge cost is maximized. Algorithm 4 gives a brief sketch.

The major idea of this algorithm is *controlled shake operation*.

9.0.1. Controlled shake operation. Let $G^{''}$ be a graph and δ be a positive number. Controlled shake operation on graph $G^{''}$ with value δ creates a graph G^s as follows

— Vertex set of G^s is the same as vertex set of $G^{''}$
— Edge set of G^s is the same as edge set of $G^{''}$
— cost of an edge e in G^s is zero if the cost of the corresponding edge in $G^{''}$ is less than or equal to δ
— cost of an edge e in G^s is any positive random number if the cost of the corresponding edge in $G^{''}$ is greater than δ

9.0.2. Algorithm. $Algorithm\ 4$ gives a brief sketch of the MSTSP Algorithm. For detailed description refer [Larusic et al. 2012]. The following are major ideas in this algorithm

— Let G^s be a graph obtained from a graph $G^{''}$ by controlled shake operation with value δ. Note that if G^s contains Hamiltonian tour with cost zero then $G^{''}$ contains a BTSP tour with cost at most δ. In this algorithm we take δ equal to one of the edge cost in the given graph $G^{''}$.
— Suppose BTSP tour cost in a graph $G^{''}$ is $\leq \delta$. Then if we apply controlled shake operation on $G^{''}$ several times with the same δ then one of the graphs generated by these operations will have Hamiltonian tour with cost zero with high probability.
— This algorithm uses binary search to find a Z_i such that given graph $G^{''}$ contains a BTSP tour with cost at most Z_i.
— In the algorithm when ever we are setting upper bound u equal to mid then we are certain that BTSP tour cost in $G^{''}$ is at most Z_u.
— If we are setting lower bound l to $mid + 1$ does not mean that BTSP tour cost in $G^{''}$ is at least Z_l. It can be less that Z_l with some small probability. This is because we are using heuristic to test whether the given graph contains a Hamiltonian cycle or not
— This algorithm terminates when lower bound l equal to upper bound u, and given graph $G^{''}$ contains a BTSP tour with cost at most Z_u, which means that graph $G^{'}$ contains an MSTSP tour with cost at least $e_{max} - Z_u$. This effectively means that graph G has a Max Scatter Hamiltonian path with cost at least $e_{max} - Z_u$.

The results obtained by using the proposed MSTSP based test cube ordering technique, is shown for different possible don't care fillings, in Table IV. In this table, MT-fill stands for Minimum Transition fill, where one would fill all the don't cares inside

ALGORITHM 4: MSTSP Algorithm

Input: Graph $G^{'}$
Output: Bottleneck Edge
Replace each edge, $e_{i,j}$ in $G^{'}$ by $e_{max} - e_{i,j}$ to obtain $G^{''}$;

Compute lowerbound lb and upper bound ub using Bottleneck Biconnected Spanning Subgraph Problem (BBSSP) algorithm and Nearest Neighbor Heuristic(NNH) respectively in given graph $G^{''}$;

Let $Z_1 < Z_2 < \cdots < Z_k$ be an ascending arrangement of the distinct edge costs in graph $G^{''}$ such that $Z_1 \geq lb$ and $Z_K \leq ub$;

Let $l = 1$, $u = k$;
while $l < u$ **do**
    mid = $\lfloor (l + u)/2 \rfloor$;
    count = some positive integer say N ;
    flag = 1 ;
    $\delta = Z_{mid}$;
    while *count* > 0 *and flag==1* **do**
        Apply controlled shake on graph $G^{''}$ with value δ to get graph G^s ;
        Find a lowest cost TSP tour in G^s using *Lin-Kernighan TSP heuristic* ;
        Let T be this tour ;
        if *the length of* T *is zero* **then**
            flag=0;
        count = count-1;
    if *flag == 0* **then**
        u=mid;
    else
        l=mid+1;
MSTSP cost is equal to $e_{max} - Z_u$.

a don't care stretch of a test cube, with the filled bit left-adjacent to the stretch. Similarly, *0-fill* and *1-fill* fill all the don't cares in the test cubes with '0' and '1' respectively. And, *R-fill* fills the don't cares in the test cubes with random values, whereas *B-fill* stands for balanced X-filling proposed in [Trinadh et al 2014]. Finally, *DP-fill* refers to the dynamic programming based X-filling technique proposed in this paper. It can be seen that the combination of *MSTSP-Ordering* and *DP-fill* proposed in this paper, has best reduction in peak input toggles. Therefore, the test cube orderings produced by MSTSP lead to very effective reduction in peak test power. Tables V and VI show the comparison of peak input toggles for various don't care filling methods w.r.t to test cube orderings given by the $TetraMax^{TM}$ tool and the XStat method [Trinadh et al 2014] respectively. Each row in these tables corresponds to a benchmark circuit. The shaded cell in each row corresponds to best don't care filling method among all don't care filling methods for the given ordering. We can observe that the proposed DP-fill method consistently performed better than all the other don't care filling methods, under both the test cube orderings. *This is because, under a given ordering, DP-fill is an optimal algorithm for minimizing peak input toggles*. Additionally, it can be observed from Tables IV, V and VI that the combination of *MSTSP-ordering + DP-fill* is most effective in reducing peak toggles, especially for the circuits whose test sets have less number of don't cares.

Table IV. Peak Input Toggles : MSTSP-Ordering with different Don't care fillings

Circuit	MT-fill	R-fill	0-fill	1-fill	B-fill	DP-fill
b01	3	2	2	3	2	2
b02	1	2	2	1	1	1
b03	10	20	19	15	6	4
b04	34	46	42	39	22	13
b05	18	23	19	19	12	8
b06	2	2	2	2	2	2
b07	26	31	39	25	17	11
b08	14	21	18	13	6	6
b09	15	17	15	16	9	8
b10	10	19	12	10	7	6
b11	20	26	22	16	12	8
b12	47	83	90	67	34	16
b13	27	32	28	21	15	9
b14	104	166	208	150	74	39
b15	221	281	298	196	109	47
b17	762	799	908	683	343	167
b18	1515	1766	2200	1567	759	403
b19	2911	3462	4327	3167	1374	646
b20	288	291	323	277	118	76
b21	252	289	340	237	147	82
b22	327	421	470	423	217	120

Table V. Peak Input Toggles : Tool-Ordering with different don't care fillings

Circuit	MT-fill	R-fill	0-fill	1-fill	B-fill	DP-fill
b01	4	4	4	4	4	4
b02	4	4	4	4	4	4
b03	15	21	17	16	14	14
b04	41	50	47	45	39	39
b05	20	23	19	20	17	17
b06	4	4	5	4	4	4
b07	31	30	34	27	23	23
b08	20	20	20	18	14	12
b09	18	20	22	18	18	18
b10	12	19	17	15	10	10
b11	22	27	29	21	20	20
b12	63	76	62	89	59	58
b13	31	34	38	30	30	29
b14	181	180	194	159	157	156
b15	305	334	344	298	292	282
b17	916	923	943	880	871	841
b18	2134	2167	2251	2114	2066	2009
b19	3926	4099	4201	3955	3819	3753
b20	309	314	315	305	302	299
b21	317	307	315	305	276	260
b22	489	494	507	471	472	466

Next, we will compare *MSTSP-ordering* + *DP-fill* and *I-ordering* + *DP-fill* with other existing techniques in the literature. Table VII shows the *peak input toggles* comparison between the proposed techniques and best known existing techniques. Column 1 shows the benchmark name and column 2 shows minimum peak input toggles obtained among all aforementioned don't care filling methods, under test cube ordering given by the $TetraMax^{TM}$ tool. In the technique proposed in [Trinadh et al 2013], only the impact of test cube ordering is considered, while in [Wu et al 2011] only the impact

Table VI. Peak Input Toggles : XStat-Ordering [Trinadh et al 2014] with different don't care fillings

Circuit	MT-fill	R-fill	0-fill	1-fill	B-fill	DP-fill
b01	3	4	4	3	3	3
b02	4	4	4	4	4	4
b03	15	19	18	15	8	7
b04	45	52	47	43	25	24
b05	21	24	21	23	15	14
b06	5	4	5	5	5	4
b07	27	33	38	25	15	14
b08	16	20	18	15	8	7
b09	20	19	17	16	14	14
b10	14	20	16	14	10	7
b11	18	26	22	20	10	9
b12	60	76	99	68	31	31
b13	37	32	28	23	17	17
b14	181	164	208	152	79	79
b15	308	277	314	198	144	144
b17	912	774	953	680	421	421
b18	2130	1752	2200	1569	1011	1008
b19	3926	3457	4340	3168	1877	1877
b20	314	291	352	297	152	152
b21	288	290	346	237	130	130
b22	483	419	475	440	237	234

of don't care filling is considered. Columns 3, 4 and 5 show the minimum peak input toggles obtained using the techniques proposed in [Trinadh et al 2013], [Wu et al 2011] and [Trinadh et al 2014] respectively. Column 6 shows the minimum peak input toggles obtained after applying the proposed DP-fill method under the proposed I-Ordering based test cube ordering scheme. Column 7 shows the percentage improvement of proposed *I-ordering+DP-fill* technique over the best of existing don't care filling methods under the test cube ordering given by the commercial ATPG tool. Columns 8-10 of this table show the percentage improvement of proposed *I-ordering* + *DP-fill* technique over these best known low power testing techniques. Column 11 shows the minimum peak input toggles obtained after applying the proposed DP-fill method under the proposed MSTSP-Ordering based test cube ordering scheme. Column 12 shows the percentage improvement of proposed *MSTSP-ordering+DP-fill* technique over the best of existing don't care filling methods under the test cube ordering given by the commercial ATPG tool. Columns 13-15 of this table show the percentage improvement of proposed *MSTSP-ordering* + *DP-fill* technique over the mentioned best known low power testing techniques in the literature. It is evident that the proposed techniques outperforms all the existing techniques for most of the benchmark circuits. Among *I-ordering* + *DP-fill* and *MSTSP-ordering* + *DP-fill*, the former performs the best benchmarks whose test sets are dominated by don't cares ($> \approx 75\%$ don't cares), and the latter performs the best otherwise. So, *I-ordering* + *DP-fill* is good for circuits whose test sets are dominated for don't cares and *MSTSP-ordering* + *DP-fill* is good circuits whose test sets are not dominated by don't cares.

Table VIII shows the comparisons of actual peak power dissipation during test, between the proposed techniques and the existing techniques. It can be observed that similar to reduction in peak toggles, the proposed techniques performs better than all the existing techniques in reducing peak power for most of the benchmarks and percentage improvement increases with increase in circuit size. This can be attributed to the well known fact that there is a good correlation between input toggles and circuit toggles, as explained in [Girard et al 1998].

Table VII. Peak Input Toggles : Comparison of DP-fill Method (I-Ordering + DP-fill and MSTSP-Ordering + DP-fill) Over Existing Ordering + Filling Methods

					I-Ordering + DP-fill over					MSTSP-Ordering + DP-fill over				
						% Improvement of I-Ordering + DP-fill					% Improvement of MSTSP-Ordering + DP-fill			
Circuit	Tool	ISA [Trinadh et al 2013]	Adj-fill [Wu et al 2011]	XStat [Trinadh et al 2014]	Proposed	Tool	ISA [Trinadh et al 2013]	Adj-fill [Wu et al 2011]	XStat [Trinadh et al 2014]	Proposed	Tool	ISA [Trinadh et al 2013]	Adj-fill [Wu et al 2011]	XStat [Trinadh et al 2014]
b01	4	2	4	3	3	25	-50	25	0	2	50	0	50	33.3
b02	4	1	3	4	3	25	-200	0	25	1	75	0	66.7	75
b03	14	8	6	8	6	57.1	25	0	25	4	71.4	50	33.3	50
b04	39	31	29	25	15	61.5	51.6	48.3	40	13	66.7	58.06	55.2	48
b05	17	12	19	15	14	17.6	-16.7	26.3	6.7	8	52.9	33.3	57.9	46.7
b06	4	2	4	4	4	0	-100	0	0	2	50	0	50	50
b07	23	18	17	15	11	52.2	38.9	35.3	26.7	11	52.2	38.9	35.3	26.7
b08	14	10	9	8	6	57.1	40	33.3	25	6	57.1	40	33.3	25
b09	18	11	17	14	11	38.9	0	35.3	21.4	8	55.6	27.3	52.9	42.9
b10	10	9	9	10	7	30	22.2	22.2	30	6	40	33.3	33.3	40
b11	20	12	18	10	9	55	25	50	10	8	60	33.3	55.6	20
b12	59	46	77	31	15	74.6	67.4	80.5	51.6	16	72.9	65.2	79.22	48.4
b13	30	20	26	17	10	66.7	50	61.5	41.2	9	70	55	65.4	47.1
b14	157	89	69	79	40	74.5	55.1	42	49.4	39	75.2	56.2	43.5	50.6
b15	292	172	149	144	33	88.7	80.8	77.9	77.1	47	83.9	72.7	68.5	67.4
b17	871	573	438	421	85	90.2	85.2	80.6	79.8	167	80.8	70.9	61.9	60.3
b18	2066	1384	1065	1011	232	88.8	83.2	78.2	77.1	403	80.5	70.9	62.2	60.1
b19	3819	2609	2100	1877	364	90.5	86	82.7	80.6	646	83.1	75.2	69.2	65.6
b20	302	214	198	152	65	78.5	69.6	67.2	57.2	76	74.8	64.5	61.6	50
b21	276	181	182	130	67	75.7	63	63.2	48.5	82	70.3	54.7	55.0	36.9
b22	471	324	232	237	91	80.7	71.9	60.8	61.6	120	74.5	63.0	48.3	49.4

Table VIII. Peak Circuit Power (in μW) : Comparison of DP-fill-Method (I-Ordering + DP-fill and MSTSP-ordering + DP-fill) Over Existing Ordering + Filling Methods

					I-Ordering + DP-fill					MSTSP-Ordering + DP-fill				
						% Improvement of I-Ordering + DP-fill over					% Improvement of MSTSP-Ordering + DP-fill over			
Circuit	Tool	ISA [Trinadh et al 2013]	Adj-fill [Wu et al 2011]	XStat [Trinadh et al 2014]	Proposed	Tool	ISA [Trinadh et al 2013]	Adj-fill [Wu et al 2011]	XStat [Trinadh et al 2014]	Proposed	Tool	ISA [Trinadh et al 2013]	Adj-fill [Wu et al 2011]	XStat [Trinadh et al 2014]
b01	3.8	2.3	3.3	3.1	3.1	18.8	-33.1	6.1	0	2.3	38.4	-0.9	28.8	24.2
b02	2.4	1.5	2.8	2.6	2.6	-6.2	-68.3	7.3	0	1.4	42.8	9.4	50.1	46.2
b03	5.6	4	4.6	3.9	4.2	25	-5.5	9.2	-5.6	3.0	45.9	23.9	34.5	23.8
b04	17.2	17.1	15.8	16.9	14.8	14	13.9	6.6	12.7	13.17	23.3	23.2	16.6	22.1
b05	15.6	13.6	16.4	14.6	14.9	4.4	-9.8	9	-2	12.7	19.0	6.9	22.9	13.5
b06	4.4	2.6	4.4	4.3	4.4	0.9	-67.2	-0.1	-1.7	2.2	49.4	14.5	48.8	48.0
b07	15.7	14.8	13.1	14.6	13.3	15.7	10.6	-1.5	8.9	13.01	17.3	12.3	0.4	10.6
b08	7.8	6.8	8.1	7.7	6.3	18.5	6.8	21.5	18.1	5.51	29.1	18.9	31.7	28.7
b09	9.8	8.4	10.7	8.9	7.4	24.7	12.1	30.8	17.2	8.0	18.1	4.4	24.8	9.9
b10	9.3	8.8	9	8.7	8.2	11.6	6.5	9.2	6.3	7.1	23.3	18.9	21.3	18.7
b11	16.4	15.4	15.2	14.6	13.9	15.2	9.6	8.9	4.8	12.8	21.6	16.4	15.8	12.0
b12	56.5	49.4	58.4	39.3	36.4	35.5	26.3	37.6	7.2	35.9	36.6	27.4	38.6	8.7
b13	18	13.7	15.1	14.7	10.9	39.4	20.1	27.6	25.3	10.3	43.1	25.0	32.08	30.0
b14	99.3	101.7	99	86.5	85.4	14	16.1	13.8	1.3	84.9	14.5	16.5	14.2	1.8
b15	197.1	171	155.3	140.4	122	38.1	28.7	21.4	13.1	124.2	37.0	27.4	20.0	11.5
b17	1085.5	847.1	665.5	641.7	431.6	60.2	49.1	35.1	32.7	519.6	52.1	38.7	21.9	19.0
b18	3350.7	2405.3	2012.2	1761	1192	64.4	50.4	40.8	32.3	1406.5	58.0	41.5	30.1	20.1
b19	7621.6	6708.3	5885	4135	2699.4	64.6	59.8	54.1	34.7	3351.0	56.0	50.1	43.1	19.0
b20	252.8	243	214.8	202.6	195.3	22.7	19.6	9.1	3.6	193.6	23.4	20.3	9.9	4.4
b21	248.4	226.1	223.8	183.2	166.4	33	26.4	25.6	9.2	170.1	31.5	24.8	24.0	7.1
b22	395.6	372.8	328.9	304.8	277.1	30	25.7	15.8	9.1	285.5	27.8	23.4	13.2	6.3

Additionally, we can observe that the magnitude of improvement in Tables VII and VIII is not same. The difference is due to the fact that the relation between input toggles and circuit toggles is not perfectly linear and while computing actual power dissipation of the circuit, we need to take interconnect capacitances into account. However, our proposed techniques outperforms all the existing techniques considerably, in both peak input toggles as well as actual peak circuit power.

Thus, in reducing peak input toggles during testing, while MSTSP-ordering is effective when test sets have small don't care stretches, while I-ordering is effective when test sets have large don't care stretches.

In [Girard et al 1998], it was shown that there is a strong correlation between input toggles and internal toggles inside the circuit. Under this assumption, we have proposed an optimal algorithm that will minimize the input toggles to the circuit during the testing phase. In the next section, we relax this assumption and try to search for solutions near the solution so-far obtained, using the local search technique.

10. LOCAL SEARCH USING ITERATIVE 1-BIT NEIGHBORHOOD

Although for a given test cube ordering, DP-fill gives the optimal X-filling for minimizing peak input toggles, and there is a good correlation between input toggles and circuit toggles [Girard et al 1998], there is no guarantee that this solution also optimally minimizes peak circuit toggles. However, from the results observed in previous sections, we observed that there is a significant reduction in peak circuit toggles also by the proposed DP-fill method. In this section, we will explore and analyze the quality of solutions obtained by DP-fill, for peak circuit toggle reduction, by pruning using the local search technique.

We denote $\mathcal{S}_{DP-fill}$ as the solution obtained using DP-fill suggested in this paper. In every iteration, $\mathcal{S}_{cur}$ stands for the best-so-far solution in the current iteration of the local search technique. The 1-bit neighborhood of a test vector T contains all those vectors that can be obtained by flipping exactly 1-bit in T. The local search technique searches for better solutions in the 1-bit neighborhood of the vectors in the test vector pair, that contribute to the peak power dissipation in $\mathcal{S}_{cur}$. This local search technique was used to prune the solutions generated by DP-fill is outlined in Figure 9.

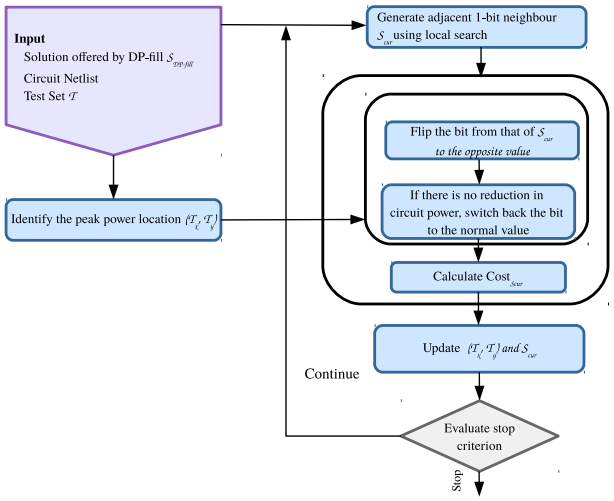

Fig. 9. Flow chart description of the local search technique with 1-bit neighborhood

Although we have adhered to 1-bit neighborhood in this paper, in principle, the local search technique shown in Figure 9, can be extended to a more general case of n-bit neighborhood, for a given n, in a straightforward manner. However, it should be noted that searching all the possible n-bit neighborhoods ($1 \leq n \leq T$, where T is test vector size) is intractable, because the size of the search space is $\sum_{n=1}^{T} n! = 2^n$. The results obtained by applying the described local search technique for greedy as well simulated annealing (SA) strategies is shown in Table IX. It can be seen that the savings is marginal, thereby validating our idea of optimizing input toggles as an effective technique for minimizing peak power dissipation during testing.

Table IX. Additional Peak Power Savings obtained by the Local Search Technique with 1-bit Neighborhood

		Greedy Pruning			Simulated Annealing (SA) Pruning		
Circuit	DP-fill (in μW)	DP-fill + Greedy Pruning (in μW)	%Improvement	Additional Simulation time	DP-fill + SA Pruning (in μW)	%Improvement	Additional Simulation time
b01	3.07	3.07	0	0s	3.07	0	0.001s
b02	2.6	2.19	15.67	0m0.03s	2.19	15.59	0m0.481s
b03	4.17	4.12	1.18	0m0.64s	4.12	1.3	0m4.728s
b04	14.76	13.23	10.36	0m6.83s	13.23	10.39	1m49.907s
b05	14.92	14.91	0.09	0m0.91s	14.86	0.43	0m13.423s
b06	4.35	4.28	1.68	0m0.06s	4.28	1.63	0m0.208s
b07	13.26	12.24	7.71	0m3.10s	12.24	7.68	0m26.645s
b08	6.89	6.79	1.47	0m0.48s	6.79	1.54	0m8.717s
b09	7.4	6.94	6.15	0m0.17s	6.94	6.1	0m0.790s
b10	8.19	8.03	1.92	0m0.41s	8.03	1.93	0m13.768s
b11	13.88	13.88	0	0m1.023s	13.88	0	0m26.877s
b12	36.42	36.12	0.82	0m30.62s	36.12	0.83	4m23.086s
b13	10.94	10.79	1.39	0m1.68s	10.79	1.36	0m21.156s
b14	85.37	82.78	3.03	2m47.09s	81.48	4.56	8m13.430s
b15	122.01	113.73	6.78	66m14.21s	117.18	3.96	39m20.670s
b17	431.6	422.17	2.19	45h37m37s	424.57	1.63	28h10m45s
b18	1192.03	1179.93	1.02	46h7m22s	1184.7	0.61	28h40m35s
b19	2699.35	2696.11	0.12	47h35m47s	2696.11	0.12	30h14m20s
b20	195.34	190.22	2.62	57m0.19s	189.76	2.86	56m56.429s
b21	166.38	161.54	2.91	3m35.40s	161.54	2.91	20m44.411s
b22	277.07	265.98	4.0	14h15m51s	267.39	3.5	9h12m25s
Average	-	-	3.39			3.28	-

Finally, the results obtained by applying the described local search technique for greedy as well simulated annealing (SA) strategies is shown in Table IX. It can be seen that the savings is marginal, thereby validating our idea of optimal minimization of input toggles as an effective technique for minimizing peak power dissipation during testing.

11. CONCLUSIONS

We address the problem of peak capture power reduction during at-speed stuck-at testing, that leads to false delay failures. Under Combinational State Preservation (CSP-scan) based design for testability (DFT) technique, we show that test cube ordering produced by the proposed Max Scatter Hamiltonian path algorithm gives best reduction in peak test power when the test cubes are not dominated by don't cares, and the proposed I-ordering algorithm gives the best reduction in peak test power otherwise. When test cubes have significant don't care bits and there is a good correlation of input toggles to circuit toggles, don't care filling is very effective for reducing peak capture power. We have mapped the problem of optimal don't care filling for a given test cube ordering, to a variant of *interval coloring problem*, so as to minimize peak input toggles of the circuit. The algorithm uses *Dynamic Programming* to obtain a lower bound, and

uses pigeon-hole principle to fill the don't cares (DP-fill), such that the lower bound is achieved. The algorithm is proven to be correct and optimal.

The proposed techniques were applied to all the benchmarks from ITC suite, and found to produce significant reductions in peak capture power dissipated inside the circuit during at-speed stuck-at testing. To the best of our knowledge, *DP-fill is the first ever reported don't filling algorithm that is optimal*. We have relaxed the assumption that circuit toggles is closely correlated to input toggles, and performed local search on the 1-bit neighborhood of the solution produced by the proposed technique. Both greedy and simulated annealing strategies are adopted to perform the search, and found to have very minimal extra savings. From this, we can conclude that the proposed technique not only optimally minimizes peak input toggles, but also produces significant savings in peak power dissipation during at-speed stuck-at testing.

REFERENCES

N. Ahmed, M. Tehranipoor, and V. Jayaram. 2006. Timing-based delay test for screening small delay defects. In *Design Automation Conference*. ACM/IEEE, 320–325.

S. Almukhaizim et al. 2008. Peak Power Reduction Through Dynamic Partitioning of Scan Chains. In *International Test Conference*. IEEE, 1–10.

Esther M. Arkin et al. 1997. On the Maximum Scatter TSP. In *Symposium on Discrete Algorithms*. ACM-SIAM, 211–220.

Fang Bao et al. 2013. Efficient Pattern Generation for Small-Delay Defects Using Selection of Critical Faults. *Journal of Electronic Testing* 29, 1 (2013), 35–48.

Swarup Bhunia, Hamid Mahmoodi, Arijit Raychowdhury, , and Kaushik Roy. 2004. First Level Hold: A Novel Low-overhead Delay Fault Testing Technique. In *Defect and Fault Tolerance*. IEEE, 314–315.

Swarup Bhunia, Hamid Mahmoodi, Arijit Raychowdhury, , and Kaushik Roy. 2005. A Novel Low-overhead Delay Testing Technique for Arbitrary Two-Pattern Test Application. In *Design Automation and Test in Europe*. IEEE, Munich, Germany, 1136–1141.

S. Bhunia et al. 2005. Low-power scan design using first-level supply gating. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, 3 (2005), 384–395.

V. Dabholkar et al. 1998. Techniques for minimizing power dissipation in scan and combinational circuits during test application. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17, 12 (1998), 1325–1333.

B. Dervisoglu and G. Stong. 1991. Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement. In *International Test Conference*. IEEE, Nashville, TN, USA, 365–374.

V. R. Devanathan, C. P. Ravikumar, and V. Kamakoti. 2007. A Stochastic Pattern Generation and Optimization framework for Variation-Tolerant, Power-Safe Scan Test. In *International Test Conference*. IEEE, Santa Clara, CA, USA, 1–10.

V.R. Devanathan et al. 2007. A stochastic pattern generation and optimization framework for variation-tolerant, power-safe scan test. In *International Test Conference*. IEEE, 1–10.

Jayashree Saxena et al. 2003. A Case Study of IR-Drop in Structured At-Speed Testing. In *International Test Conference*. 1098–1104.

Kuen-Jong Lee et al. 2000. Peak-power reduction for multiple-scan circuits during test application. In *Asian Test Symposium*. IEEE, 453–458.

V. R. Devanathan et al. 2007. Glitch-Aware Pattern Generation and Optimization Framework for Power-Safe Scan Test. In *VLSI Test Symposium*. IEEE, Berkeley, CA, USA, 167–172.

Xijiang Lin et al. 2008. Test Power Reduction by Blocking Scan Cell Outputs. In *Asian Test Symposium*. IEEE, 329–336.

S. Gerstendorfer and H. J. Wunderlich. 1999. Minimized Power Consumption for Scan-based BIST. In *International Test Conference*. IEEE, Atlantic City, NJ, USA, 77–84.

P. Girard, N. Nicolici, and X. Wen. 2009. *Power-Aware Testing and Test of Low Power Design*. Springer.

P. Girard et al. 1998. Reducing power consumption during test application by test vector ordering. In *International Symposium on Circuits and Systems*. IEEE, 296–299.

P. Girard et al. 1999. Circuit partitioning for low power BIST design with minimized peak power consumption. In *Asian Test Symposium*. IEEE.

S.K. Goel et al. 2010. Circuit Topology-Based Test Pattern Generation for Small-Delay Defects. In *Asian Test Symposium*. IEEE, 307–312.

John Larusic, Abraham P. Punnen, and Eric Aubanel. 2012. Experimental analysis of heuristics for the Bottleneck Traveling Salesman Problem. *Journal of Heuristics* 18, 3 (June 2012), 473–503.

X. Liu. 2004. *ATPG and DFT Algorithms for Delay Fault Testing*. Ph.D. Dissertation. Virginia Polytechnic Institute and State University, Virginia, USA.

E.J. McCluskey and Chao-Wen Tseng. 2000. Stuck-fault tests vs. actual defects. In *International Test Conference*. IEEE, 336–342.

P. Pant et al. 2010. Lessons from at-speed scan deployment on an Intel Itanium microprocessor. In *International Test Conference*. IEEE, 1–8.

N. Parimi and Xiaoling Sun. 2004. Toggle-masking for test-per-scan VLSI circuits. In *International Symposium on Defect and Fault Tolerance in VLSI Systems*. IEEE, 332–338.

I. Pomeranz. 2015. Static Test Compaction for Low-Power Test Sets by Increasing the Switching Activity. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 9 (2015), 1936–1940.

S. Potluri. 2015. *Power:Its Manifestations during Digital Systems Testing*. Ph.D. Dissertation. Indian Institute of Technology Madras, Chennai, India.

S. Potluri, S.T. Adireddy, C. Rajamanikkam, and S. Balachandran. 2013. LPScan: An algorithm for supply scaling and switching activity minimization during test. In *International Conference on Computer Design*. IEEE, 463–466.

S. Potluri et al. 2015. DFT Assisted Techniques for Peak Launch-to-Capture Power Reduction during Launch-On-Shift At-Speed Testing. *ACM Transaction on Design Automation of Electronic Systems (TODAES)* 21, 1 (2015).

K. Sankaralingam and N.A. Touba. 2002. Controlling peak power during scan testing. In *VLSI Test Symposium*. IEEE, 153–159.

A. Satya Trinadh et al. 2013. An Efficient Heuristic for Peak Capture Power Minimization During Scan-Based Test. *Journal of Low Power Electronics* 9, 2 (2013), 264–274.

A. Satya Trinadh et al. 2014. XStat: Statistical X-Filling Algorithm for Peak Capture Power Reduction in Scan Tests. *Journal of Low Power Electronics* 10, 1 (2014), 107–115.

Vlado Vorisek et al. 2004. At-Speed Testing of SOC ICs. In *Design, Automation and Test in Europe*. IEEE, 30120.

Douglas B. West. 2000. *Introduction to Graph Theory*. Prentice Hall.

F. Wu et al. 2011. Power reduction through X-filling of transition fault test vectors for LOS testing. In *International Conference on Design Technology of Integrated Systems in Nanoscale Era*. IEEE, 1–6.

C. Yao et al. 2011. Power and Thermal Constrained Test Scheduling Under Deep Submicron Technologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 2 (2011), 317–322.

Mahmut Yilmaz et al. 2010. Test-Pattern Selection for Screening Small-Delay Defects in Very-Deep Submicrometer Integrated Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 5 (2010), 760–773.

Received ; revised ; accepted