Optimization of Cell-Aware ATPG Results by Manipulating Library Cells' Defect Detection Matrices

Zhan Gao[1,2,3] Min-Chun Hu[1,2,4] Joe Swenton[2]
Santosh Malagi[2] Jos Huisken[3] Kees Goossens[3] Erik Jan Marinissen[1,3]

[1] IMEC
Kapeldreef 75
3001 Leuven
Belgium
zhan.gao.ext@imec.be
min-chun.hu.ext@imec.be
erik.jan.marinissen@imec.be

[2] Cadence Design Systems
1701 North Street
Endicott, NY 13760
United States of America
zgao@cadence.com
malagi@cadence.com
swenton@cadence.com

[3] TU Eindhoven
Den Dolech 2
5612 AZ Eindhoven
the Netherlands
z.gao@tue.nl
j.a.huisken@tue.nl
k.g.w.goossens@tue.nl

[4] National Tsing-Hua University
101, Section 2, Kuang-Fu Road
Hsinchu, 30013
Taiwan
minchunhu@gapp.nthu.edu.tw

Abstract – Cell-aware test (CAT) explicitly targets defects inside library cells and therefore significantly reduces the number of test escapes compared to conventional automatic test pattern generation (ATPG) approaches that cover cell-internal defects only serendipitously. CAT consists of two steps, viz. (1) *library characterization* and (2) *cell-aware ATPG*. Defect detection matrices (DDMs) are used as the interface between both CAT steps; they record which cell-internal defects are detected by which cell-level test patterns. This paper proposes two algorithms that manipulate DDMs to optimize cell-aware ATPG results with respect to fault coverage, test pattern count, and compute time. Algorithm 1 identifies don't-care bits in cell patterns, such that the ATPG tool can exploit these during cell-to-chip expansion to increase fault coverage and reduce test-pattern count. Algorithm 2 selects, at cell level, a subset of *preferential* patterns that jointly provides maximal fault coverage at a minimized stimulus care-bit sum. To keep the ATPG compute time under control, we run cell-aware ATPG with the preferential patterns first, and a second ATPG run with the remaining patterns only if necessary. Selecting the preferential patterns maps onto a well-known $\mathcal{NP}$-hard problem, for which we derive an innovative heuristic that outperforms solutions in the literature. Experimental results on twelve circuits show average reductions of 43% of non-covered faults and 10% in chip-pattern count.

1 Introduction

To test digital logic integrated circuits (ICs) that are commonly synthesized with pre-defined libraries of standard cells, conventional ATPG tools target faults at the boundary of library cells only. Intra-cell defects are only covered fortuitously, and hence not surprisingly, these defects are found to be the root cause of a significant fraction of test escapes [1]. Cell-aware test (CAT) explicitly targets intra-cell defects and has shown to indeed significantly reduce the number of test escapes [2–6].

The CAT methodology consists of two steps. In Step 1, *library characterization*, for each library cell, all possible cell-internal defect locations are identified [7]. Subsequently, analog simulation is performed to identify which cell-internal defects are detected by which cell patterns. The simulation results are encoded in a defect detection matrix (DDM) which serves as input for Step 2, *cell-aware ATPG*. For each cell instance in a chip design, an intra-cell defect is covered if there is at least one cell pattern (1) which according to the DDM detects this particular defect, and (2) which the cell-aware ATPG tool is able to successfully expand from cell-to-chip level. Successful expansion of a cell pattern to the chip level through ATPG is not guaranteed and depends on the circuit environment of the cell in question. However, many defects are detected by multiple cell patterns, and these provide alternatives for the ATPG engine to get the defect in question covered by at least one cell pattern for which the cell-to-chip expansion is successful.

To the best of our knowledge, no prior work discloses details on how DDMs are used for cell-aware ATPG. Several papers [2, 6, 8] compare stuck-at and cell-aware pattern counts at cell level; however, none of them describes how the reported cell-aware pattern counts were identified from DDMs that contain exhaustive sets of fully-specified patterns, and whether those patterns are allowed to contain unspecified (*don't-care*) bits.

In this paper, we assume a given set of DDMs with patterns for which all stimulus bits and exactly one response bit are fully-specified, for example as output of Step 1 of the CAT tool flow. The paper proposes two algorithms that manipulate these DDMs, such that the resulting DDMs, when used in Step 2 optimize the cell-aware ATPG with respect to its key performance indicators (in order of decreasing priority): fault coverage, test pattern count, and compute time. Figure 1 shows the position of the two proposed DDM manipulation algorithms in the CAT flow. The two algorithms can be used in isolation, but actually deliver the best results when used in conjunction.

Algorithm 1 extends the DDMs to also include all partly-specified cell patterns, i.e., patterns for which one or more stimulus bits are *don't-care*. The new DDM patterns allow ATPG to significantly increase the fault coverage and reduce the final chip-pattern count. Algorithm 2 selects, for a given DDM, a subset of patterns containing minimized care-bit sum, that still detects the same defects as the ori-

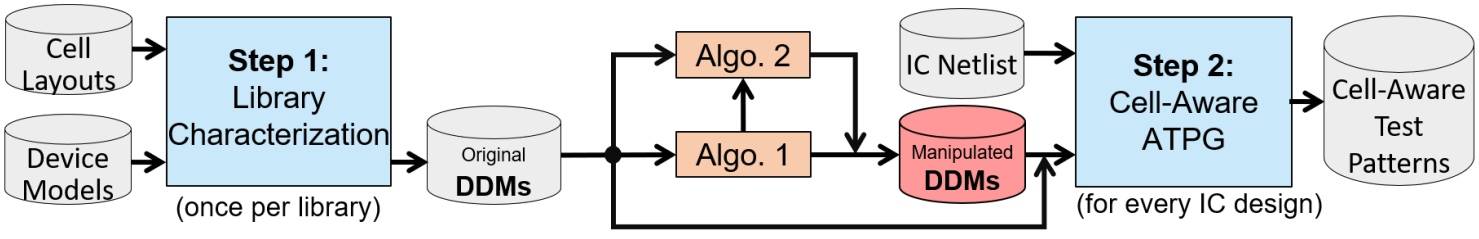


Figure 1: None, one, or both Algorithms 1 and 2 proposed in this paper can be used in the CAT tool flow to manipulate DDMs to optimize the subsequent ATPG results.

ginal DDM. Forcing ATPG to first try to expand these *preferential* patterns significantly reduces the ATPG compute times.

The remainder of this paper is organized as follows. In Section 2, we formally define DDMs and describe their usage in cell-aware ATPG. In Sections 3 and 4, we describe the motivation and detailed operation of Algorithms 1 and 2, respectively. Section 5 provides experimental ATPG results and Section 6 concludes this paper.

2 Defect Detection Matrix and Its Usage

A DDM is a two-dimensional binary matrix. An example DDM is shown in Figure 2; defects are represented by columns and cell patterns by rows. If and only if cell pattern p detects defect d, '1' is marked at the matrix entry on the cross point of d and p.

Patterns		Defects	cell: ADDHX1																									
	AB/SC	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24	d25	d26	d27
p1	00/L-													1	1				1				1		1	1		
p2	01/H-													1	1	1	1			1	1		1	1		1		
p3	10/H-											1						1		1	1				1	1		
p4	11/L-		1	1		1	1	1		1						1	1	1	1		1		1				1	
p5	00/-L		1	1						1																	1	
p6	01/-L		1	1	1		1						1	1				1			1							1
p7	10/-L		1	1		1		1		1									1		1				1	1		
p8	11/-H	1		1			1		1	1	1																	1

Figure 2: Example defect detection matrix.

We define DDMs for standard cells as follows. Let c be a standard cell with n inputs and m outputs, with $n, m \in \mathbb{N}^+$. Let D_c and P_c be the sets of defects respectively patterns for Cell c. The locations of the defects in D_c can be determined with algorithms described in [7]; the ohmic value of the defects depends on expectations regarding defect size. P_c can be any set of patterns for Cell c, provided that in every pattern a stimulus bit for at least one cell input and exactly one response bit for a cell output are specified. For cells with m outputs, m identical stimulus sets with responses on different outputs are considered as different patterns. A DDM for cell c is specified by function $DDM_c : D_c \times P_c \longrightarrow \{0,1\}$ with $DDM_c(d,p) = 1$ if and only if defect d is detected by pattern p.

The *pattern set* for defect d is defined by function $ps_c : D_c \longrightarrow \mathcal{P}(P_c)$ with $ps_c(d) = \{p \in P_c |\ DDM_c(d,p) = 1\}$. Similarly, the *detection set* of pattern p is defined by function $ds_c : P_c \longrightarrow \mathcal{P}(D_c)$ with $ds_c(p) = \{d \in D_c | DDM_c(d,p) = 1\}$.

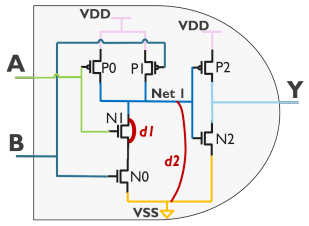
Cell-aware ATPG uses as inputs a cell-level netlist of a chip design and a set of DDMs corresponding to the cells occurring in that netlist (see Figure 1). A defect d in cell c is called *non-detectable* if no element of the exhaustive set of cell patterns detects d. Non-detectable defects are removed from D_c, as ATPG has no means to detect them. The remaining defects are detectable at cell level and also referred to as *faults*. ATPG will determine whether, given the chip design, cell-level test patterns for these faults can be expanded to chip level. Let I be the set of cell instances that occur in the chip's netlist and let function $t : I \longrightarrow C$ define the cell *type* of a given cell instance. The set of faults F is defined by $F = \{(i,d) \in I \times D_{t(i)} | ps_{t(i)}(d) \neq \emptyset\}$.

The goal of the cell-aware ATPG engine is for every fault $f = (i,d)$, i.e., detectable defect d in cell instance i, to expand at least one cell pattern p that detects f (i.e., $DDM_{t(i)}(d,p) = 1$) successfully from cell to chip level. Expansion involves propagating the responses at the outputs of library cell i to chip outputs, and justifying the specified stimuli at the inputs of library cell i to chip inputs. Expansion might propagate and/or justify via other cell instances and their interconnecting nets. Successful expansion of the stimulus and response bits for one fault in one particular cell instance typically covers other faults as well; fault simulation can identify these cases and add the detections to the set of overall detections. If expansion is not successful, typically there are alternative cell patterns for the fault in question, and their expansion can be tried as well. Only if expansion is unsuccessful for *all* cell patterns that detect fault f at cell-level, we declare fault f *undetectable*.

Conventional test methods consider per fault a single expansion task location only. For example, for stuck-at ATPG, the propagation and justification task are both defined at a single cell-terminal. For cell-aware ATPG, a single propagation task is defined at one cell output and, for k specified stimulus bits out of n inputs (with $k \leq n$), k additional justification tasks are defined at the k cell inputs. The other $(n-k)$ bits, a.k.a *don't-care* bits are not induced in justification tasks and therefore cause a higher probability of successful expansion and hence higher fault coverage and fewer specified bits in expanded chip patterns that allow further test pattern compaction.

3 Don't-Care Bit Identification Algorithm

For some cell-internal faults, detection requires all inputs of the cell to be specified. However, many faults can be detected while one or more cell inputs are don't-care. Figure 3 shows an example of both cases in the AND2X1 cell from the Cadence GPDK045 library [9]. Locations of two cell-internal short defects are shown in Figure 3(a). Defect $d1$ is a short between source and drain of NMOS transistor N1 controlled by input A and $d2$ a short between Net1 and VSS. To detect $d1$, both inputs should be specified. For $d2$ detection, when either input is specified as '0', the other input is don't-care. Figure 3(b) shows all fully- and partly-specified patterns for $d1$ and $d2$. Defect $d1$ is detected by only fully-specified pattern $p2$; defect $d2$ can be detected by all fully- and partly-specified patterns except pattern $p4$.



(a) $d1$ and $d2$ in cell schematic.

Patterns	AB/Y	Defects d1	d2
p1	00/L		1
p2	01/L	1	1
p3	10/L		1
p4	11/H		
p5	0X/L		1
p6	X0/L		1

(b) DDM for $d1$ and $d2$.

Figure 3: Examples of faults requiring either fully- or partly-specified cell patterns for detection in an AND2X1 cell.

We first introduce some notation conventions. Let *FP* be a set of fully-specified patterns and *PP* an extended pattern set including *FP* and all partly-specified patterns derived from *FP*. Function $r : PP \times [1,m] \longrightarrow \{H,L\}$ with $r(p,j) = s$ gives the response value s of pattern p on output bit j.

We develop Algorithm 1, which extends a given DDM containing fully-specified patterns with all partly-specified cell patterns that are implied by the original DDM. A pattern consists of stimulus bits for all n inputs and a response bit on exactly one of m outputs. To determine if two given patterns p_1 and p_2 can generate a pattern with one additional don't-care stimulus bit, p_1 and p_2 should be identical in all bits, except for one stimulus bit j, where the two patterns should have opposite logic values (thus, $p_1(j) =$'0' and $p_2(j) =$'1' or vice versa). Algorithm 1 consists of three main steps, which are

described below. Steps A and B are based on the well-known Quine-McCluskey algorithm [10].

Algorithm 1 [DON'T-CARE BIT IDENTIFICATION ALGORITHM]

```
01: input: F, FP, DDM(F, FP);
02: output: PP, DDM(F, PP);
03: dictionary = ∅; PPdictionary = ∅; // Initialization
04: // Step A: Divide fully-specified patterns in lists, based on response bits
05: for all j ∈ [1, m] do {
06:   dictionary[o_j,'L'] = ∅; dictionary[o_j,'H'] = ∅; // o_j is the output pin name
07:   for all p ∈ FP do {
08:     if r(p, i) = 'L' then { dictionary[o_j,'L'] ∪ p };
09:     else { dictionary[o_j,'H'] ∪ p};
10: } }
11: // Step B: Identify all partly-specified patterns for each list of dictionary
12: for all i ∈ [1, m] do {
13:   Quine-McCluskey Algorithm identifying PP for dictionary[o_i,'L']
14:   Quine-McCluskey Algorithm identifying PP for dictionary[o_i,'H']
15:   // Record the corresponding fully-specified pattern set for pp
16:   for all pp ∈ PP do { PPdictionary[pp] = FP_pp; }
17: // Step C: Identify the detection set for each partly-specified pattern pp
18: for all pp ∈ PP do {
19:   for all f ∈ F do {
20:     DDM(f, pp) = DDM(f, FP_pp[1]) ∧ ... ∧ DDM(f, FP_pp[|FP_pp|])); } }
```

Step A: We partition the fully-specified pattern set *FP* into at most $2m$ lists of which each has an identical response value on one of the m outputs. Variable *dictionary* stores these lists of patterns (Line 4–10). An example is shown as column "0 'X' Bits" in Figure 4.

Step B: For each pattern list, we identify all partly-specified patterns of *PP* that correspond to the fully-specified patterns in the list. Both *pp* ∈*PP* and its corresponding fully-specified patterns FP_{pp} are recorded for the next step (Line 11–16).

	Cell AOI21: Response 'L' on Output 'Y'		
	0 'X' Bits	1 'X' Bit	2 'X' Bits
1 '1' Bit	1 001/L ✓	1 3 0X1/L ✓ 1 5 X01/L ✓	1 3 5 7 XX1/L
2 '1' Bits	3 011/L ✓ 5 101/L ✓ 6 110/L ✓	3 7 X11/L ✓ 5 7 1X1/L ✓ 6 7 11X/L	
3 '1' Bits	7 111/L ✓		

Figure 4: Example matrix used in Step B in Algorithm 1 (based on [10]).

Each list identified in Step A is displayed in a matrix [10]; the columns in this matrix display the cell patterns with increasing 'X' count, while the rows correspond to increasing '1' count – see Figure 4. The fully-specified patterns end up in Column 0 'X' Bits. The green numbers in front of patterns are the identifiers of the patterns. We can add a pattern into Column $(i+1)$ 'X' Bits on input j $(1 \leq j \leq n)$ if and only if in Column i 'X' Bits we already have two patterns identical to the new partly-specified pattern, apart from input j where one pattern has a '0' and the other pattern has a '1'. Iteratively, we search the patterns in Column i 'X' Bits and identify patterns for Column $i+1$ 'X' Bits, starting of course with the given fully-specified patterns in Column 0 'X' Bits. The algorithm will certainly end in Column n 'X' Bits, but in typical cases sooner. As per column, we are searching for patterns which differ in only one stimulus bit, we can reduce compute time by considering only pattern pairs in adjacent rows in the same column. In Figure 4, one pattern with two don't-care bits is displayed in Column 2 'X' Bits: 'XX1/L'. As indicated in green, this pattern originates from four fully-specified patterns {1, 3, 5, 7} = {001/L, 011/L, 101/L, 111/L}. An important difference between the usage of the Quine-McCluskey in logic optimization [10] and in our application is that in the former, only patterns without check-mark are kept (two in the example of Figure 4), while in the latter we use *all* patterns (eleven in Figure 4).

Step C: In this step, the detection sets for the new partly-specified patterns are calculated. A fault f is included in the detection set of partly-specified pattern *pp* if and only if f is element of the detection sets of *all* original fully-specified patterns that were used to derive *pp* (Line 17–20).

We identify for 324 combinational cells of Cadence's GPDK045 45nm CMOS library [9] 4,766 fully-specified cell patterns after 8 hours of analog simulations. Algorithm 1 derives 13,266 additional partly-specified cell patterns (= 2.8× increase) in which 20,466 (= 30%) bits are identified as don't-care. Compute time for Algorithm 1 is just ∼1 minute. However, this extended cell pattern set significantly increases ATPG compute time, which we reduce in the next section of this paper by Algorithm 2.

4 MINCOVER Algorithms

A library cell's DDM can contain many cell patterns, especially if it is extended to include all applicable partly-specified patterns. Typically, there is quite some overlap in the fault detection sets of any pair of cell patterns. Successful pattern expansion from cell-to-chip by the ATPG engine depends on the circuitry surrounding the cell in question, and is not guaranteed. Consequently, the order in which ATPG tries to expand the various cell patterns in a DDM has a big impact on the ATPG compute time. Ideally, we want to expand a small subset of the DDM patterns for which the union of their detection sets provides maximum fault coverage at cell level. However, a traditional ATPG engine has no insight in overlap of the various DDM patterns. Also, by expanding cell patterns with fewer specified bits, we increase the chance for success, but possibly at the expense of a smaller fault detection set.

In this section, we first describe Algorithm 2, which selects a minimized subset of DDM patterns with maximal cell-level fault coverage, i.e., identical to the combined fault coverage of all cell patterns in the DDM. In this paper, we apply Algorithm 2 on DDMs with exhaustive cell-patterns to get full fault coverage at cell level. We refer to the resulting patterns as the *preferential* patterns for that cell. We show that identifying a minimal set of preferential patterns for a cell is an $\mathcal{NP}$-hard problem, we present a framework that enables us to efficiently describe all prior algorithms addressing this problem, and subsequently present an innovative and effective heuristic algorithm that outperforms the prior art.

We use the subset of preferential cell patterns for two purposes. At library level (even before a specific chip design on the basis of this library is available), the size of the preferential pattern subset is a useful metric for how many patterns a particular library cell will need as a minimum. However, the main usage of the preferential pattern subset is during ATPG on a specific design, where we run ATPG in two stages. In Stage 1, we only try to expand the preferential patterns of the cell instances in the design. If the entire set of preferential cell patterns expands successfully, we have obtained maximal fault coverage with the smallest-possible number of cell patterns at minimal ATPG compute time. However, typically a small fraction of the preferential patterns fail to expand to chip level, and therefore the chip-level fault coverage is not maximal. Thus, in Stage 2, we run ATPG again, now providing the DDM information of the remaining (non-preferential) patterns for all cell instances. This two-stage

ATPG keeps the ATPG compute time under control.

We have some variable definitions for the preferential pattern selection problem. P is the exhaustive fully-specified pattern set, thus $|P| = m \times 2^n$. $P' \subseteq P$ and represents the patterns that were specified to us in the input DDM. D is the set of all defects, for which the locations were identified by [7]. $F \subseteq D$ is the set of all detectable defects, which we also call faults. $F' \subseteq F$ is the union of the detection sets for all patterns in P'.

Problem 1: Given fault set F and pattern sets P and P' with $P' \subseteq P$. Each $p \in P$ is associated with a detection set $ds(p) \subseteq F$ and $\bigcup_{i \in P'} \big(ds(i)\big) = F' \subseteq F$. Find a minimal set $MC \subseteq P'$ such that $\bigcup_{i \in MC} \big(ds(i)\big) = F'$. □

Problem 1 is equivalent to the Set Covering Problem, which was proven to be $\mathcal{NP}$-hard [11]. Therefore, we developed a heuristic algorithm framework called MINCOVER to optimize MC. The algorithm framework is outlined in Algorithm 2. In every iteration, we employ a routine to move patterns from P' to either MC ('select') or *REST* ('deselect') (Line 05) and update P' and F' accordingly (Lines 06–07), until F' has been exhausted (Line 08).

Algorithm 2 [MINCOVER Framework]

```
01: input: F', P, P', DDM(F, P);
02: output: MC, REST;
03: MC = ∅; REST = ∅; // Initialization
04: repeat {
05:   Select/Deselect non-empty set of patterns Pat ⊆ P';
06:   if Select then {MC = MC ∪ Pat; P' = P'\Pat; F' = F' \ ⋃_{i∈Pat} ds(i);}
07:   else { REST = REST ∪ Pat; P' = P'\Pat; }
08: } until F' = ∅
```

The MINCOVER algorithm framework has numerous algorithm instances, which all differ with respect to the sequence of select/deselect routines which are employed in Line 05. We refer to the various routines we have developed with a single letter. A specific algorithm instance of MINCOVER is expressed by a regular expression describing a string of these letters, which represents the sequence of routines employed subsequently in Line 05. The various routines which we use are described below.

Definition 4.1 Pattern p is called an *essential* pattern if and only if $\exists_{f \in F'} \big(ps(f) = \{p\}\big)$. □

To cover all faults, all essential patterns must be selected. Typically, essential patterns also detect other faults, for which they are not essential. As essential patterns need to be selected anyway, we better do this early in the algorithm, thereby avoid having to include other cell patterns covering the faults that are detected by essential patterns. The routine of selecting all essential patterns for a given DDM is denoted by letter 'E', has compute complexity $\mathcal{O}(|F'|)$, and is described as follows:

E: *Pat*= ∅; **for all** $f \in F'$ **do** { **if** $|ps(f)| = 1$ **then** *Pat*=*Pat*∪ $ps(f)$}.

Definition 4.2 Pattern p is called a *subset* pattern if and only if $\exists_{i \in P'} \big(ds(p) \subseteq ds(i)\big)$. □

Subset patterns are not necessary to be expanded by the ATPG engine, provided their superset patterns can be expanded successfully. Hence, the subset patterns can be deselected and moved from P' to *REST*. Routine S, in which for a given DDM all subset patterns are collected in *Pat* consists of two nested loops and hence has compute complexity $\mathcal{O}(|P'|^2)$:

S: *Pat* = ∅; **for all** $p_1, p_2 \in P'$ **do** {
if $ds(p_1) \subseteq ds(p_2)$ **then** *Pat* = *Pat* ∪{p_1}}.

For many DDMs, routines E and S alone are insufficient to assign all DDM patterns to either MC or *REST*. In that case, we resort to a *greedy* selection of a single cell pattern which we add to the selected pattern set MC. We have formulated two greedy routines differentiated only by their cost functions $h(p)$. In both greedy routines, with compute complexity $\mathcal{O}(|P'|)$, a pattern p_g is selected for which holds $h(p_g) = MAX_{p \in P'}\big(h(p)\big)$.

tmp = 0;
for all $p \in P'$ **do** {**if** $h(p) \geq$*tmp* **then** *tmp*=$h(p)$; $p_g = p$};
Pat = {p_g}

In greedy routine G, $h(p) = \sum_{f \in ds(p)} \big(DDM(f,p)\big) = |ds(p)|$. Our second greedy routine, denoted by letter W, assigns a *weight* $h(p) = \sum_{f \in ds(p)} \big(\frac{DDM(f,p)}{|ps(f)|}\big)$ to each pattern $p \in P'$. Weight $h(p)$ increases if p has a large detection set $ds(p)$, but also increases for detected faults $f \in ds(p)$ that have a small pattern set $ps(f)$. Faults with a small pattern set have only few alternatives for detection, and hence trying expansion of such a pattern early on might be important.

In prior work addressing the Set Covering Problem, a classic heuristic solution is the Greedy Algorithm [12], denoted in our framework as G+: routine G is iteratively performed until F' is empty. Chvátal [13] showed that solutions obtained by G+ can be at most a factor $ln|F'|$ larger than an optimum solution. Selecting all essential patterns before applying the Greedy Algorithm (EG+) can reduce this factor from at most $ln|F'|$ to $log|F'|$, which was proven later by Qayyum [14]. Agathos & Papapetrou [15] used routines S, E, and G to compose two optimization algorithms, viz. SEG+ and (SE)+G+, in a communication application.

We make the following observations regarding algorithm compositions. Essential patterns should be selected first: E. Selecting essential patterns can introduce new subset patterns, and vice versa, deselecting subset patterns (S) can introduce new essential patterns on the still unclassified remaining test patterns in the DDM. Applying sequences of only E and S routines can lead to a situation where no further essential or subset patterns can be identified, while still not all patterns in the DDM are classified as selected or deselected. In such a case, we might need to apply a greedy routine (G or W) to make progress. Execution of G or W cannot introduce new essential patterns, but it can generate new subset patterns. Performing S only

Algorithm Composition		Preferential Patterns			Bits in Pref. Patterns	
		Essential	Greedy	Total	Care	Don't-Care
G+	[12]	0	1428	1428	5092	0
SEG+	[15]	1038	375	1415	5028	67
EG+	[14]	1007	408	1413	4980	9
(SE)+G+	[15]	1200	210	1410	5013	9
(ES)+G+		1200	210	1410	4956	66
((SE)+G)+		1340	68	1408	5002	9
(ES)+(G(SE)+)+		1340	68	1408	4946	65
EW+		1007	393	1400	4761	207
(ES)+W+		1147	253	1400	4959	66
(SE)+W+		1139	261	1400	4902	9
(ES)+(W(SE)+)+		1139	261	1399	4810	140
ED+	($x = 33$)	1007	395	1402	4747	233
(ES)+D+	($x = 33$)	1139	262	1401	4819	155
(SE)+D+	($x = 1$)	1147	253	1400	4956	12
(ES)+(D(SE)+)+	($x = 33$)	1139	262	1401	4819	157

Table 1: Results for 15 MINCOVER algorithm compositions for 324 combinational cells of the GPDK045 library [9].

before G or W does not make sense, because greedy selection will never select subset patterns and S has higher compute complexity.

We have defined a number of MINCOVER algorithms: seven using greedy routine G, and four using greedy routine W. Table 1 lists the aggregate results for these 11 MINCOVER algorithms on 324 combinational cells of Cadence's GPDK045 library [9]. The algorithms with our novel greedy routine W consistently outperform the algorithms based on greedy routine G, which is described in the literature. Based on the observations above, we expected algorithm composition (ES)+(W(SE)+)+ to perform the best, and the experimental results in Table 1 indeed confirm that.

Actually, what we are really after is not minimization of the number of preferential patterns, but minimization of the sum of care bits in the preferential patterns. Fewer care bits implies (1) lower probability for conflicts and hence more successful expansions, which should increase fault coverage and reduce test pattern count, (2) fewer specified bits in the chip-patterns, which should allow more compaction and reduce test pattern count, and (3) less work to expand them, which implies reduced ATPG compute times.

Problem 2: Given fault set F and pattern sets P and P' with $P' \subseteq P$. Each $p \in P$ is associated with a detection set $ds(p) \subseteq F$ and $\bigcup_{i \in P'} \big(ds(i)\big) = F'$. All $p \in P$ have n stimulus bits of which $cb(p)$ are care bits. Find a set $MB \subseteq P'$ with minimal $\sum_{i \in MB} \big(cb(i)\big)$ such that $\bigcup_{i \in MB} \big(ds(i)\big) = F'$. □

Problem 2 is $\mathcal{NP}$-hard; the proof is as follows.

Proof: Assume Problem 2 is *not* an $\mathcal{NP}$-hard problem. If for a certain problem instance $\forall_{p \in P'} \big(cb(p) = n\big)$, then $\sum_{i \in MB} \big(cb(i)\big) = n \times |MB|$. In that case, Problem 2 becomes Problem 1, viz. find a minimum set MB such that $\bigcup_{i \in MB} \big(ds(i)\big) = F'$. Problem 1 was already shown to be $\mathcal{NP}$-hard. This implies a contradiction with our assumption and hence Problem 1 *is* $\mathcal{NP}$-hard. □

The algorithm we propose for Problem 2 is based on what we have learned from the algorithms for Problem 1. Essential patterns should be selected first if we want to cover all faults: routine E. Partly-specified patterns are typically subset patterns of their fully-specified counterparts. Therefore, we will not use routine S, as that will deselect many partly-specified patterns and that does not help to reduce the number of care bits. Finally, we define a new greedy selection routine, which we call D. It is a modified version of routine W, as we have seen that W consistently outperformed G. To let the cost factor that governs greedy selection also include the number of don't-care bits in a particular pattern, we use $h(p) = \sum_{f \in ds(p)} \big(\frac{DDM(f,p)}{|ps(f)|}\big) \times (n - cb(p) + x)$. Here, $x \in \mathbb{N}^+$ is a means to balance the relative importance of both cost-factor components $\sum_{f \in ds(p)} \big(\frac{DDM(f,p)}{|ps(f)|}\big)$ and $n - cb(p)$.

We experimented on the 324 combinational cells of the GPDK045 library [9] with all MINCOVER algorithms from Table 1, where we replaced greedy routines G or W by routine D, while varying x. Based on the observations above, we expected algorithm ED+ to perform best, and that was indeed the case. In Figure 5, we show the pattern counts and care-bit sums for $x \in [1..500]$. The minimum care-bit sum is obtained for $x = 33$. The exact pattern and care-bit counts are given in the last row of Table 1. The care-bit sum is lower than with all other MINCOVER algorithms based on greedy routines G and W, as now this has become an explicit selection criterium in D, while the total preferential pattern count has only increased marginally from 1399 to 1402. Compute times for each algorithms on all 324 library cells was ∼0.1 second and thus negligible.

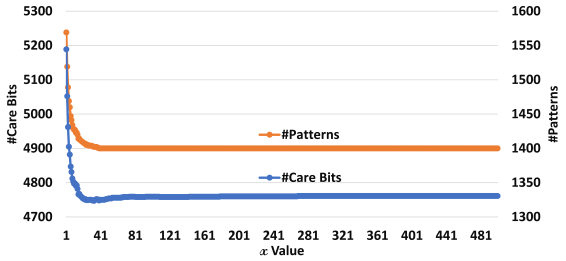


Figure 5: Results of algorithm ED+ for varing values of x on 324 combinational cells of the GPDK045 library [9].

5 Experimental Results

We performed cell-aware ATPG with Cadence's ATPG tool Modus (version 19.1) on twelve circuits [16–19] that were first mapped onto Cadence's GPDK045 45nm CMOS library [9]. For each circuit, we compare the cell-aware ATPG results for optimized DDMs versus non-optimized DDMs. The *non-optimized DDMs* contain *exhaustive* sets of fully-specified cell patterns (i.e., with $m \times 2^n$ patterns for a cell with n inputs and m outputs). The *optimized DDMs* were obtained by two steps. Step a, Algorithm 1 was used on the non-optimized DDMs to create *extended DDMs*, which in addition to the exhaustive set of fully-specified patterns also contain all possible partly-specified patterns; Step b, we generated two DDMs out of each extended DDM: one with only preferential patterns, identified by MINCOVER algorithm ED+, and one with the remaining cell patterns. These two sets of DDMs were used in two-stage ATPG.

Results are presented in Table 2. This table has four main columns: (1) design data, (2) $\sum$ cell patterns, (3) $\sum$ care bits, and (4) Δ ATPG results (comparing ATPG with optimized to non-optimized DDMs).

More cell instances with more inputs per cell improve the effectiveness of our optimization algorithms. The GPDK045 library that we used in the experiments reported in this paper is rather small and its cells have at most six inputs; this means that the benefits obtained with circuits based on GPDK045 are rather pessimistic with what would be possible for larger, industrial standard-cell libraries. The column 'Design Data' consists of three sub-columns: the circuit name, the number of combinational cell instances in the circuit $|I|$, and the weighted average number of inputs per cell instance $\sum_{k \in [1..6]} \big(k \times |\{i \in I | n_i = k\}|\big)/|I|$.

Subsequently, columns '$\sum$ cell patterns' and '$\sum$ Care Bits' present the results of our DDM optimization algorithm, each in five identical sub-columns. The first and second sub-columns give the summed number of cell patterns resp. care bits in the non-optimized and the extended DDMs (as obtained by Algorithm 1) for all instances $i \in I$, while the third sub-column gives the increase factor from non-optimized to extended DDMs. Sub-columns 4 list how many of the patterns in the extended DDMs are selected by MINCOVER algorithm ED+ and the last sub-columns express that number as a fraction of the pattern count in the extended DDMs; the smaller that fraction, the more effective our two-stage ATPG approach might be.

Finally, in the last column, we compare cell-aware ATPG based on optimized DDMs to non-optimized DDMs. The three sub-columns address the three key performance indicators of ATPG: test quality, test execution time, and test generation time. Test quality is expressed as the fraction the non-covered faults, i.e., 100% - fault

Design Data			$\sum_{i \in I}$ cell patterns					$\sum_{i \in I}$ Care Bits					Δ ATPG Results: Optimized vs. Non-Optimized		
Name	$\|I\|$	WAvg. Inp.	NonOpt	Extended	Ext/NonOpt	Pref.	Fraction	NonOpt	Extended	Ext/NonOpt	Pref.	Fraction	Non-Cov. Faults	Chip Patterns	ATPG Time (s)
b15 [17]	2933	2.95	31800	92166	2.90×	12505	13.6%	122140	309736	2.54×	38468	12.4%	-66.81%	-15.55%	+10.6 = +33.28%
b20 [17]	3212	2.65	30200	72682	2.41×	14985	20.6%	103676	229320	2.21×	42488	18.5%	-7.00%	-8.26%	+9.6 = +60.21%
aes [18]	3320	2.82	35284	111762	3.17×	13606	12.2%	143812	415573	2.89×	40869	9.8%	-53.01%	-0.93%	+29.8 = +409.47%
b21 [17]	3482	2.63	32068	75666	2.36×	16520	21.8%	107832	234869	2.18×	45909	19.5%	-3.93%	-13.07%	+10.2 = +47.62%
dtmf[16]	3726	2.72	38542	121242	3.15×	15150	12.5%	151670	430595	2.84×	45190	10.5%	-41.49%	-21.80%	+24.6 = +81.32%
b22 [17]	5036	2.64	46752	110196	2.36×	23668	21.5%	158148	341442	2.16×	66579	19.5%	-5.36%	-12.95%	+19.1 = +63.40%
M0 [19]	5289	2.94	61568	207910	3.38×	20475	9.8%	253516	738554	2.91×	65407	8.9%	-49.18%	-20.45%	+36.2 = +79.64%
b17 [17]	8981	2.94	97784	288230	2.95×	38043	13.2%	378564	976797	2.58×	117426	12.0%	-64.32%	-8.30%	+49.1 = +50.52%
fpu [18]	19136	2.97	220544	616994	2.80×	84950	13.8%	844492	2092037	2.48×	265413	12.7%	-60.98%	-11.89%	+128.9 = +270.86%
b18 [17]	20545	2.94	227466	659616	2.90×	88879	13.5%	877958	2238668	2.55×	274329	12.3%	-64.32%	-5.49%	+133.9 = +59.04%
M3 [19]	32626	2.95	388412	1293250	3.33×	129329	10.0%	1602836	4615396	2.88×	414499	9.0%	-42.42%	+6.03%	+135.3 = +10.96%
b19 [17]	40298	2.93	441896	1283378	2.90×	173509	13.5%	1702888	4349339	2.55×	533648	12.3%	-62.61%	-12.56%	+314.2 = +76.95%
Avg.	12382	2.84	137693.0	411091.0	2.88×	52634.9	14.7%	537294.3	1414360.5	2.56×	162518.8	13.1%	-43.45%	-10.43%	+75.1 = +103.61%

Table 2: The results of Algorithms 1 and 2 on the cell pattern and care-bit counts, and the effect on key ATPG performance indicators for twelve circuits.

coverage. We report how much our optimization reduces the fraction of non-covered faults. For example: for benchmark circuit $b15$, the non-optimized cell-aware fault coverage was 97.65%, while the optimized fault coverage increased to 99.22%; this constitutes a reduction of the fraction of non-covered faults with $1-(1-99.22\%)/(1-97.65\%) = -66.81\%$. In column '*Chip Patterns*' we give the delta-percentage of the number of test patterns generated with optimized DDMs in comparison to what non-optimized ATPG can achieve. Both these columns show that our approach has only benefits for test quality (on average, 43.45% less non-covered faults) and test execution time (10.43% less test patterns). However, these benefits are achieved through a significantly extended DDM (on average for the twelve circuits 2.84× more cell patterns), ATPG execution times are bound to increase (on average +103.61%). The two-stage ATPG approach makes that the increase in ATPG compute time remains under control. Running ATPG on the extended DDMs in a single run increased the compute time on average with an additional 101.3%.

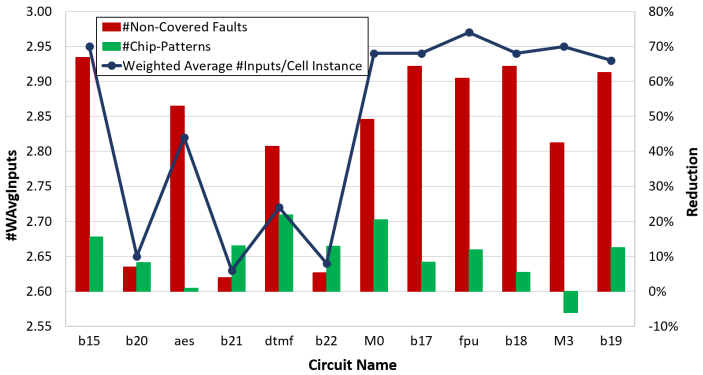


Figure 6: Optimization of ATPG results with minBit pattern sets.

Figure 6 shows the weighted average number of inputs of the cell instances in a circuit is a good indicator for the reductions in non-covered faults and test patterns that can be achieved by the combination of Algorithms 1 and 2 as described in this paper. Only circuit $M3$ is an anomaly, perhaps due to its specific circuit structure.

6 Conclusion

DDMs represent the result of characterizing which cell-internal defects are detected by which cell patterns. This paper presents two algorithms that manipulate DDMs to improve the results of subsequent cell-aware ATPG runs. Algorithm 1 extends a given DDM to include all patterns with don't-care bits, with as objectives to increase fault coverage and reduce test-pattern count. This typically leads to a large increase in cell patterns. Algorithm 2, an innovative heuristic solution for a well-known $\mathcal{NP}$-hard problem, identifies a subset of cell patterns with minimized care-bit sum, yet maximal fault coverage. These preferential patterns are tried for cell-to-chip expansion first by the cell-aware ATPG tool, while, if necessary, the remaining patterns are handled in a second ATPG run.

An attractive benefit of both algorithms is that they are executed as part of library characterization: once per library, instead of once per IC design. The two algorithms can be used stand-alone, but provide best results when used in conjunction. In this paper, we report results on their combined usage. For a set of twelve benchmark circuits, on average the non-covered faults were reduced with 43% and the test pattern count with 10%. The two-stage ATPG approach helped to keep the compute time under control. Despite that, the compute time increased, but as compute time is an non-recurring engineering cost only, we consider it acceptable in view of the improvements in the other two key ATPG performance indicators.

Acknowledgments

We thank the following persons for their support. At Cadence Design Systems: Vivek Chickermane, Franck Gerome, Anton Klotz, and Mike Vachon. At IMEC: Kristof Croes, Peter Debacker, Ingrid De Wolf, Alessio Spessot, Ibrahim Tatar, and Diederik Verkest. At National Tsing-Hua University: Cheng-Wen Wu. Zhan Gao is financially supported by the China Scholarship Council (CSC) and Cadence Design Systems.

References

[1] S. Eichenberger et al. Towards a World Without Test Escapes: The Use of Volume Diagnosis to Improve Test Quality. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10, October 2008. doi:10.1109/TEST.2008.4700604.

[2] F. Hapke et al. Defect-Oriented Cell-Aware ATPG and Fault Simulation for Industrial Cell Libraries and Designs. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10, November 2009. doi:10.1109/TEST.2009.5355741.

[3] F. Hapke et al. Cell-Aware Test. *IEEE Transactions on Computer-Aided Design*, 33(9):1396–1409, September 2014. doi:10.1109/TCAD.2014.2323216.

[4] A.D. Singh. Cell Aware and Stuck-Open Tests. *Proc. IEEE European Test Symposium (ETS)*, pages 1–6, May 2016. doi:10.1109/ETS.2016.7519316.

[5] W. Howell et al. DPPM Reduction Methods and New Defect Oriented Test Methods Applied to Advanced FinFET Technologies. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10, October 2018. doi:10.1109/TEST.2018.8624906.

[6] S.P. Dixit, D.D. Vora, and K. Peng. Challenges in Cell-Aware Test. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6, May 2018. doi:10.1109/ETS.2018.8400700.

[7] Z. Gao et al. Defect-Location Identification for Cell-Aware Test. In *Proc. IEEE Latin-American Test Symposium (LATS)*, pages 1–6, March 2019. doi:10.1109/LATW.2019.8704561.

[8] F. Hapke et al. Defect-Oriented Cell-Internal Testing. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10, November 2010. doi:10.1109/TEST.2010.5699229.

[9] Cadence Design Systems. Reference Manual Generic 45nm Salicide 1.0V/1.8V 1P 11M Process Design Kit and Rule Decks (PRD) Revision 4.0, 2014.

[10] E.J. McCluskey. Minimization of Boolean Functions. *The Bell System Technical Journal*, 35(6):1417–1444, Nov 1956. doi:10.1002/j.1538-7305.1956.tb03835.x.

[11] R.M. Karp. Reducibility among combinatorial problems. *Proc. a Symposium on the Complexity of Computer Computations*, page 85–103, March 1972. doi:10.2307/2271828.

[12] D.S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, 9(3):256–278, December 1974. doi:10.1016/S0022-0000(74)80044-9.

[13] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[14] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Hawaii International Conference on System Sciences*, pages 3866–3875, Jan 2002. doi:10.1109/HICSS.2002.994521.

[15] S. Agathos and E. Papapetrou. On the Set Cover Problem for Broadcasting in Wireless Ad Hoc Networks. *IEEE Communications Letters*, 17(11):2192–2195, November 2013. doi:10.1109/LCOMM.2013.091913.131765.

[16] Santosh Malagi. Cadence Cell-Aware Test Rapid Adoption Kit. `http://support.cadence.com`, 2019.

[17] Scott Davidson. ITC99 Benchmark Home Page. `https://www.cerc.utexas.edu/itc99-benchmarks/bench.html`, 1999.

[18] OpenCores.org. `https://www.opencores.org`, 1999.

[19] Arm Cortex-M Series Processors. `https://developer.arm.com/ip-products/processors/cortex-m`.