Verlustleistungsoptimierende Testplanung zur Steigerung von Zuverlässigkeit und Ausbeute

Michael E. Imhof, Christian G. Zöllin,
Hans-Joachim Wunderlich
Institut für Technische Informatik
Universität Stuttgart
Pfaffenwaldring 47; D-70569 Stuttgart, Germany
email: {imhof, zoellin, wu}@iti.uni-stuttgart.de

Nicolas Mäding, Jens Leenstra

IBM Deutschland Entwicklung

Schönaicherstr. 220, D-71032 Böblingen, Germany
email: {nmaeding, leenstra}@de.ibm.com

***Zusammenfassung*— Die stark erhöhte durchschnittliche und maximale Verlustleistung während des Tests integrierter Schaltungen kann zu einer Beeinträchtigung der Ausbeute bei der Produktion sowie der Zuverlässigkeit im späteren Betrieb führen. Wir stellen eine Testplanung für Schaltungen mit parallelen Prüfpfaden vor, welche die Verlustleistung während des Tests reduziert. Die Testplanung wird auf ein Überdeckungsproblem abgebildet, das mit einem heuristischen Lösungsverfahren effizient auch für große Schaltungen gelöst werden kann. Die Effizienz des vorgestellten Verfahrens wird sowohl für die bekannten Benchmarkschaltungen als auch für große industrielle Schaltungen demonstriert.**

1 Einführung

Die fortschreitende Miniaturisierung der Schaltungsstrukturen, die steigende Betriebsfrequenz und die wachsende Komplexität und Fläche führen zu einer ständigen Zunahme sowohl der dynamischen als auch der statischen Verlustleistung hochintegrierter Schaltungen. Zusätzlich ist die Schaltaktivität während des strukturorientierten Tests um nahezu eine Größenordnung erhöht [1], so daß die dynamische Verlustleistung beträchtlich zunimmt und sowohl die Ausbeute als auch die Zuverlässigkeit beeinträchtigt. Die hierfür wesentlichen Mechanismen sind beispielsweise in [2] beschrieben.

Ohne geeignete Gegenmaßnahmen kann die maximale Verlustleistung, die während eines Zeitpunkts im Verlauf des Tests auftritt, deutlich über derjenigen liegen, die für den Systembetrieb spezifiziert ist. Spannungseinbrüche, induktive Effekte oder Rauschen auf den Signalleitungen führen dann dazu, daß auch defektfreie Schaltungen fehlerhafte Ausgaben liefern, aussortiert werden und somit die Ausbeute mindern.

Die durchschnittliche Verlustleistung ist die während eines Tests oder eines Testabschnitts umgesetzte Energie dividiert durch die Dauer und kann ohne geeignete Gegenmaßnahmen beträchtlich über derjenigen im Systembetrieb liegen. Die erhöhte Stromdichte und auch die daraus folgende erhöhte Temperatur setzen die Schaltung einem Stress aus, der unkontrolliert die Lebenserwartung verringert [3].

Die auf Grund der Skalierung schnell wachsenden Leckströme beeinträchtigen den System- und Testbetrieb in gleicher Weise, wobei die Reduktion der statischen Verlustleistung nicht Gegenstand dieses Beitrags ist. Dagegen ist die überproportional wachsende dynamische Verlustleistung testspezifisch und muß in der industriellen Fertigung entsprechend berücksichtigt werden. Zu den üblichen Gegenmaßnahmen gehören die Reduktion der Testgeschwindigkeit, Partitionierung der Schaltung und besonderer Aufwand bei der Kühlung [2]. Diese Techniken sind nicht nur mit hohen Kosten verbunden, sondern können auch die Testqualität beeinträchtigen.

Es haben sich daher in der Vergangenheit zahlreiche Publikationen mit der Verminderung der Schaltaktivität während des prüfpfadbasierten Selbsttests beschäftigt. Das Schieben von Testmustern und -antworten erfordert den größten Teil des Energieaufwands nicht nur im Prüfpfad selbst sondern auch durch die in der kombinatorischen Logik hervorgerufene Schaltaktivität. Diese kann verringert werden, indem der funktionale Ausgang während des Schiebevorgangs konstant gehalten wird [4], [5]. Einige Ansätze unterdrücken im Selbsttest solche Muster, die nicht zur Testqualität beitragen und verringern so den Energieverbrauch [4], [6], [7]. Andere Methoden modifizieren

beispielsweise durch Umordnen oder Einfügen von Gattern den Prüfpfad, um den Test zu beschleunigen oder die Schaltaktivität zu reduzieren [8], [9], [10]. Die Parallelität des Tests und damit die Verlustleistung können reduziert werden, indem die Schaltung partitioniert und ein entsprechender Testplan berechnet wird [11]. Weiterhin wurden verlustleistungsgerechte Mustergeneratoren [12], [13] oder die Verwendung von globalem Clock Gating [14], [15] vorgeschlagen.

Am Beispiel des Cell Prozessors wurde bereits die Möglichkeit der individuellen Abschaltung einzelner Prüfpfade einer STUMPS Konfiguration (Self-Test Using MISR and Parallel Shift Register Sequence Generator, Abb. 1) erfolgreich untersucht [16]. Dabei wurde gezeigt, dass es möglich ist, einzelne bzw. mehrere Prüfpfade vorübergehend zu deaktivieren ohne die Fehlerabdeckung zu beeinträchtigen, wodurch die Verlustleistung signifikant reduziert und deren Verlauf an verschiedene Hüllkurven angepasst werden kann. Das dort vorgestellte Verfahren war jedoch wegen der zur Verfügung stehenden kommerziellen Entwurfsautomatisierungssoftware auf einige einfache Heuristiken beschränkt, welche das ganze Potential des Ansatzes nur unvollkommen ausnutzen konnten. In dem vorliegenden Beitrag zeigen wir, dass durch Anpassung der Entwurfswerkzeuge und Berücksichtigung zusätzlicher Freiheitsgrade während der Optimierung die Verlustleistung deutlich weiter reduziert werden kann.

Der Rest des Beitrags ist wie folgt aufgebaut: In Abschnitt 2 geben wir einen kurzen Überblick über die zugrunde liegende Selbsttestarchitektur. Darauf folgt eine genaue Definition des Problems sowie dessen Abbildung auf ein allgemeines Überdeckungsproblem. Es wird ein Verfahren vorgestellt, mit dem eine kostengünstige Überdeckung in vertretbarer Zeit approximiert werden kann. In Abschnitt 4 zeigen wir die Effizienz der vorgestellten Methode an Hand der verbreiteten Benchmarkschaltungen sowieso für verschiedene industrielle Schaltungen.

2 Selbsttest mit Parallelen Prüfpfaden

Für den Selbsttest mit mehreren Prüfpfaden wurde in [17] die STUMPS-Konfiguration (Abb. 1) vorgestellt, welche inzwischen die am weitesten verbreitete Struktur für den Logik-Selbsttest ist [18], [19]. Hierbei werden mehrere parallele Prüfpfade von einem Generator für Pseudo-Zufallsmuster (Pseudo-Random Pattern Generator, PRPG) mit Testmustern versorgt. Der PRPG besteht aus einem linear rückgekoppelten Schieberegister (Linear-Feedback Shift Register, LFSR), einem XOR-Netzwerk zur Phasenverschiebung und einer Logik zur Gewichtung der Häufigkeit von Einsen und Nullen im Testmuster [20]. Die Schaltungsantworten werden durch ein Signaturregister mit mehreren Eingängen (Multiple-Input Signature Register, MISR) ausgewertet, wobei eine Maskierungslogik die Möglichkeit bietet, einzelne defekte Prüfpfade oder solche mit unbekannten Zuständen zu diagnostischen Zwecken auszublenden.

Erweiterungen dieser Selbsttestruktur gestatten es, den Schiebevorgang für einzelne Prüfpfade zu deaktivieren. Zum Beispiel erlaubt der Cell Prozessor, das Taktsignal für individuelle Prüfpfade gänzlich zu unterdrücken [21] und dadurch nicht nur die durch den Schiebevorgang verursachte Schaltaktivität zu blockieren, sondern auch die Verlustleistung im Taktbaum zu verringern. Der Cell Prozessor benutzt mehrere Testregister (Abb. 1), über die ein Tester Zugriff auf die Werte für den LFSR-Startwert ($seed$), die Gewichtung ($weight$), die Maskierung ($mask$), die Signatur ($signature$) sowie die Prüfpfadaktivierung ($scanena$) hat. Eine ausführliche Beschreibung der Selbsttestausstattung des Cell Prozessors wurde in [16] vorgestellt, und ähnliche Eigenschaften sind auch in anderen großen industriellen Schaltungen implementiert.

3 Berechnung eines optimierten Testplans

Ziel der Berechnung eines optimierten Testplans ist das Erkennen einer gegebenen Fehlermenge mit minimierter Verlustleistung. Zu diesem Zweck wird für jeden Startwert des LFSRs eine möglichst große Menge von Prüfketten bestimmt, die abgeschaltet werden können, ohne die Fehlererfassung zu beeinträchtigen.

Ein Testblock b ist ein Tupel (s, sc) bestehend aus einem Startwert des Testmustgenerators (und der zugehörigen Testmenge fester Größe), sowie einer Menge von aktivierten Prüfpfaden $sc \subset SC$ im folgenden Konfiguration genannt. Durch Fehlersimulation unter Einbeziehung der Prüfpfadkonfiguration kann zu jedem Testblock die Fehlermenge F_b ermittelt werden, die durch diesen Block erkannt wird. Gesucht ist eine Menge von Blöcken B, welche die Ge-

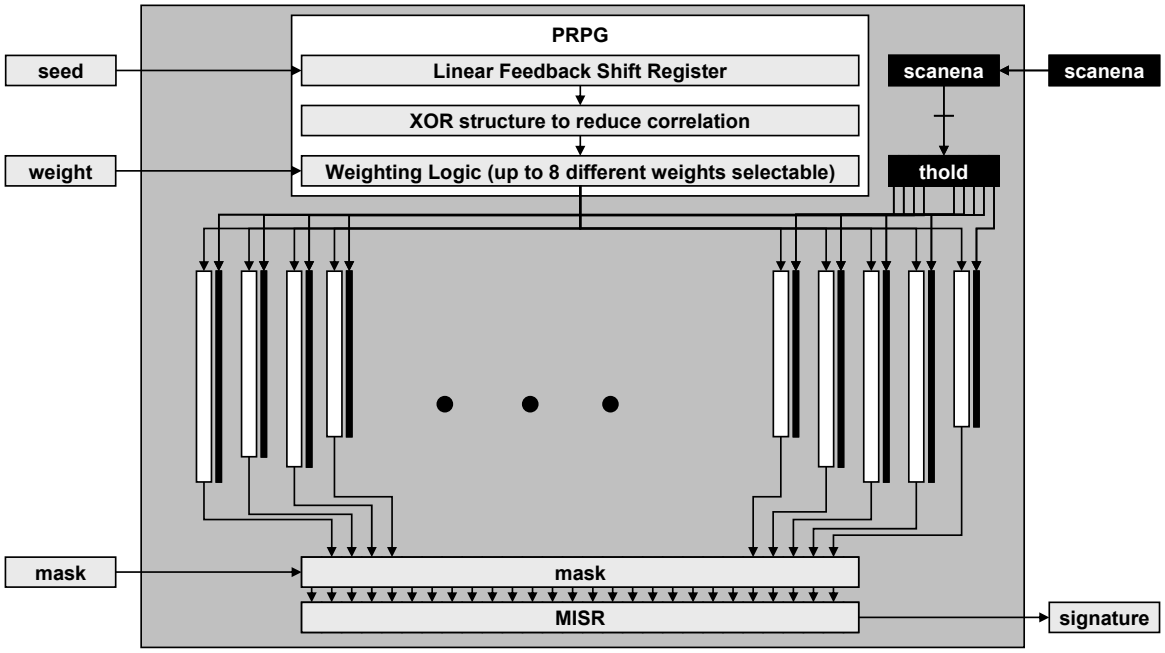

Abb. 1. STUMPS Selbsttest-Konfiguration implementiert im Cell-Prozessor

samtfehlermenge F mit minimalem Energieaufwand detektieren kann.

Ein Fehler f kann durch unterschiedliche Konfigurationen entdeckt werden. Man beachte, dass zu einem Startwert mehrere Blöcke existieren können, die sich jeweils in der Konfiguration unterscheiden und deshalb unterschiedliche Fehlermengen überdecken. Ein Block (s, sc) wird als minimal bezüglich des Fehlers f bezeichnet, wenn f nicht mehr durch den Block erkannt werden würde, falls ein beliebiger Prüfpfad aus sc entfernt wird.

Die Kosten einer Überdeckung sind eine Schätzung ihres Energieverbrauchs, indem die Zahl der Ketten, die pro Startwert aktiviert werden sollen, aufsummiert wird. Sei S_B die Menge der Startwerte aus B, dann kann zu jedem Startwert $s \in S_B$ die zugehörige Menge von Blöcken B_s angegeben werden. Die Kosten einer Menge von Blöcken kann nun abgeschätzt werden zu

$$cost(B) = \sum_{s \in S_B} \left| \bigcup_{(s,sc) \in B_s} sc \right|.$$

Falls die Prüfpfade nicht, wie allgemein üblich, von gleicher Länge sind, kann hier auch mit der Zahl der Flipflops gewichtet werden.

Für realistische Schaltungsgrößen ist die Bestimmung eines globalen Optimums nicht sinnvoll möglich, da die Zahl aller möglichen Startwerte des Testmustergenerators sehr groß ($> 2^{32}$) ist und die benötigten Fehlersimulationen sehr rechenintensiv sind. Im Folgenden zeigen wir eine Heuristik, welche das Überdeckungsproblem löst, indem für eine gegebene Menge S von Startwerten Blöcke berechnet werden, welche die zugehörige Fehlermenge annähernd optimal überdecken.

Beispiel: Zum Test der Schaltung in Abbildung 2 gehört die Menge der Prüfpfade $SC = \{sc_1, sc_2, sc_3\}$, die Menge der Fehler $F = \{f_1, f_2, f_3, f_4, f_5\}$ und die Menge der Startwerte $S = \{s_1, s_2, s_3\}$.

3.1 Optimierungsalgorithmus

Jeder Block $b = (s, sc)$ bestimmt eine Menge von Fehlern F_b, die bei der Abarbeitung dieses Blocks erkannt werden. Falls in F_b Fehler enthalten sind, die von keinem anderen Block erkannt werden, gehört b auf jeden Fall zur optimalen Lösung und wird essentieller Block genannt. Wir erhalten $B_0 := \{b \mid b \text{ essentiell}\}$ als Zwischenlösung und müssen nur noch die Fehler aus $F_0 = F \setminus \bigcup_{b \in B_0} F_b$ überdecken.

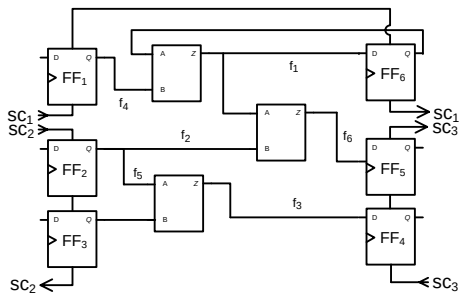

Abb. 2. Beispielschaltung

Auch bei den verbleibenden Fehlern wird die Komplexität durch einen „divide and conquer“ Ansatz reduziert. Zu diesem Zweck wird die Menge der Fehler in drei verschiedene Klassen unterteilt:

1) Harte Fehler sind solche, die nur durch einen Startwert aus S erkannt werden können. Die entsprechenden Startwerte nennen wir essentiell.
2) Schwer erkennbare Fehler können nur durch eine Zahl von Startwerten erkannt werden, die unter einer vom Anwender vorgegebenen Konstante lim liegt.
3) Die übrigen Fehler sind leicht erkennbar.

Diese drei Klassen werden mit unterschiedlichen Verfahren und Heuristiken behandelt.

3.1.1 Harte Fehler: Für jeden harten Fehler $f \in F_0$ mit zugehörigen Startwert s_f bestimmt die Funktion $c(f, s_f) = \{c \mid f \text{ wird von Block } (s_f, c) \text{ erkannt}\}$ die Scanketten und damit die Menge der minimalen Blöcke $B_f = \{(s_f, c) \mid c \in c(f, s_f)\}$.

Für harte Fehler wird zur Ermittlung der Blöcke eine Implementierung der Funktion $c(f, s)$ verwendet, die ausnutzt, dass zur Erkennung eines Fehlers bereits ein einzelnes Ausgangsflipflop ausreicht, in dem der Fehler zu einer Änderung der Schaltungsantwort führt. Damit wird für jedes erkennende Ausgangsflipflop eine Konfiguration ermittelt, die ausreichend ist, um den Fehler zu erkennen, und die nahe an einer minimalen Konfiguration liegt.

Für die Funktion $c(f, s)$ werden also aus der Fehlersimulation des Startwerts s_f diejenigen Flipflops ermittelt, welche f beobachten können. Zu jedem Flipflop werden anschliessend die Flipflops des zugehörigen Eingangskegels hinzugenommen und auf die Menge der benötigten Prüfpfade geschlossen.

Für jeden Block $b \in \bigcup_{f \text{ hart}} B_f$ wird wieder die Menge F_b aller Fehler bestimmt, die in b erkannt werden. Da die Zahl dieser Blöcke gering ist, kann mit einem Branch-and-Bound Verfahren eine Teilmenge $B_1 \subset \bigcup_{f \text{ hart}} B_f$ gefunden werden [22], so daß $\bigcup_{b \in B_1} F_b$ alle harten Fehler aus F_0 überdeckt und $cost(B_0 \cup B_1)$ minimal ist. Die verbleibende Fehlermenge ist $F_1 = F_0 \setminus \bigcup_{b \in B_1} F_b$.

Beispiel: In der Beispielschaltung aus Abb. 2 werde f_4 nur durch s_1 erkannt und sei ein harter Fehler (wir nehmen $B_0 = \emptyset$ an). Bei der Fehlersimulation werde ermittelt, dass f_4 durch die Flipflops FF_5 sowie FF_6 beobachtet wird, wenn $s_1$unter Aktivierung aller Prüfpfade angewendet wird. Daraus folgt gemäß der Approximation $c(f_4, s_1)$, dass f_4 durch Aktivierung einer der Flipflop Kombinationen $\{FF_1, FF_2, FF_5, FF_6\}$ oder $\{FF_1, FF_6\}$ bzw. der zugehörigen Prüfpfade $\{sc_1, sc_2, sc_3\}$ oder $\{sc_1\}$ erkannt wird. Die Menge der Blöcke für f_4 ergibt sich zu:

$$B_{f4} = \{(s_1, \{sc_1, sc_2, sc_3\}),\ (s_1, \{sc_1\})\}$$

Die optimierte Überdeckung ist $B_1 = \{(s_1, \{sc_1\})\}$ mit Kosten $cost(B_1) = 1$. Im Beispiel wird durch Fehlersimulation ermittelt, dass B_1 bereits auch den Fehler f_1 erkennt.

3.1.2 Schwer erkennbare Fehler: Die nächste Klasse enthält schwer erkennbare Fehler, welche durch mehrere Startwerte erkannt werden, deren Zahl aber unter einem gesetzten Limit lim liegt. Um den damit einhergehenden Komplexitätszuwachs zu kompensieren, wird pro Startwert nur genau eine erkennende Konfiguration berücksichtigt.

Dazu wird die Funktion $c(f, s_f)$, welche die zu aktivierenden Scanketten für den Fehler f bestimmt, für schwer erkennbare Fehler einfacher und unabhängig vom Startwert s_f bestimmt. Die dabei verwendete Funktion $c(f, s_f) = c_{wc}(f)$ liefert eine Obermenge der notwendigen Flipflops und verwendet hierzu die „Support Region“ [23] eines Fehlers f. Sie enthält sämtliche Flipflops des Ausgangskegels des Fehlerorts von f sowie wiederum alle Flipflops, die im Eingangskegel dieser Ausgangsflipflops liegen. Diese Approximation ist unabhängig von der generierten Testmenge und muss nicht durch Fehlersimulation für jeden möglichen

Startwert in S_f neu berechnet werden. Wiederum ist $B_f = \{(s_f, c_{wc}(f)) \mid s_f \text{ Startwert für } f\}$ die zugehörige Blockmenge, und eine Teilmenge $B_2 \subset \bigcup_{f \in F_{lim}} B_f$ wird erzeugt, so dass $\bigcup_{b \in B_2} F_b$ alle schwer erkennbaren Fehler aus F_1 überdeckt und $cost(B_0 \cup B_1 \cup B_2)$ minimal ist. Die verbleibende Fehlermenge $F_2 = F_1 \setminus \bigcup_{b \in B_2} F_b$ kann nur noch leicht erkennbare Fehler umfassen und wird daher sehr klein oder sogar leer sein.

Beispiel: Zu Abbildung 2 ergibt sich für $lim = 2$ die Menge $F_{lim} = \{f_2, f_3\}$. Für f_2 besteht die Support Region aus den Flipflops FF_1, FF_2, FF_5 sowie FF_6 woraus sich sofort ergibt, dass f_2 auf jeden Fall durch die Konfiguration $\{s_1, s_2\}$ erkannt wird. Für f_3 ergibt sich aus der Support Region FF_2, FF_3, FF_4 die zugehörige Konfiguration $\{s_2, s_3\}$. Dazu gehören die folgenden Mengen von Blöcken: $B_{f2} = \{(s_2, \{sc_1, sc_2\}), (s_3, \{sc_1, sc_2\})\}$, $B_{f3} = \{(s_1, \{sc_2, sc_3\}), (s_3, \{sc_2, sc_3\})\}$. Für Abb. 2 wird hierbei die optimale Teillmenge $B_2 = \{(s_3, \{sc_1, sc_2\}), (s_3, \{sc_2, sc_3\})\}$ mit Kosten $cost(B_2 \cup B_1 \cup B_0) = 4$ gefunden. Im Beispiel aus Abb. 2 wird bereits der letzte Fehler f_5 erkannt und der Algorithmus endet hier für diesen Fall. Die ermittelte Blockmenge beträgt $B = B_2 \cup B_1 \cup B_0 = \{(s_1, \{sc_1\}), (s_3, \{sc_1, sc_2\}), (s_3, \{sc_2, sc_3\})\}$ mit Kosten von $cost(B) = 4$ und detektiert alle Fehler.

3.1.3 Restliche Fehler: Um die restlichen Fehler zu überdecken, wird Schritt 2 nochmals mit $lim = \infty$ wiederholt, wodurch Blöcke für alle verbliebenen Fehler erzeugt werden. Da viele dieser Fehler durch eine sehr große Menge von Startwerten erkannt werden, ist der Suchraum bei der Optimierung sehr gross. Die Optimierung durch das Branch-and-Bound Verfahren kann abgebrochen werden, sobald die Zeit bis zum nächsten (besseren) Zwischenergebnis ein festgelegtes Zeitlimit überschreitet. Dies beeinträchtigt das Ergebnis nur unwesentlich, beschränkt aber die Worst Case Zeitkomplexität auf eine wählbare obere Schranke.

In den Verfahren für die Klassen der harten und schwer erkennbaren Fehler kann die Zahl der Blöcke im schlimmsten Fall quadratisch mit der Zahl der Prüfpfade wachsen. Die Komplexität für die Klasse der restlichen Fehler ist konstant. Die Experimente im folgenden Kapitel bestätigen, dass das Verfahren in der Praxis noch besser skaliert.

4 Ergebnisse

Das beschriebene Verfahren wurde in Java als Teil einer hauseigenen Entwurfsautomatisierungslösung implementiert und auf eine Reihe von Benchmark-Schaltungen angewendet. Alle Resultate wurden mit Hilfe eines Quad-Opteron mit 2,4 GHz und 32 GB Hauptspeicher generiert, wobei die hohe Parallelisierbarkeit des Problems ausgenutzt wurde, um rechenintensive Aufgaben wie z.B. die Fehlersimulation zu beschleunigen.

4.1 Benchmarks und industrielle Schaltungen

Zur Evaluierung des vorgestellten Verfahrens wurden Schaltungsmodelle aus den folgenden Quellen verwendet:

- International Symposium on Circuits and Systems (ISCAS89)
- International Test Conference (ITC99)
- Industrielle Schaltungen von NXP
- Prozessorkern des Cell-Prozessors

Die Schaltungen aus ISCAS89 (s38417 sowie s38584) und ITC99 (b17, b18 und b19) sind die größten Schaltungen aus der jeweiligen Sammlung von sogenannten Benchmark-Schaltungen. Sie besitzen keine Selbsttestausstattung und wurden für diesen Beitrag um die benötigte Testarchitektur erweitert. Die Flipflops der Schaltungen wurden hierzu basierend auf der Schaltungstopologie in 32 parallelen Prüfpfaden angeordnet.

Die von NXP zur Verfügung gestellten Schaltungen (p286k, p330k, p388k, p418k und p951k) enthalten bereits einen prüfgerechten Entwurf mit parallelen Prüfpfaden und sind um ein vielfaches größer als die Schaltungen aus ISCAS89 und ITC99. Zudem repräsentieren sie die typischen Eigenschaften industrieller Schaltungen, nämlich kürzere Pfade sowie kleinere Ausgangskegel, bedingt durch die stärkere Optimierung auf hohe Taktfrequenzen bei geringer Fläche.

Als Beispiel für die Anwendung des vorgestellten Verfahrens auf eine Schaltung mit partiellen Prüfpfaden wurde die aktuelle Implementierung des Cell-Prozessors verwendet, welche aus etwa 250 Millionen Transistoren auf $235mm^2$ Chipfläche besteht und deren übergeordnete Selbsttestarchitektur aus 15

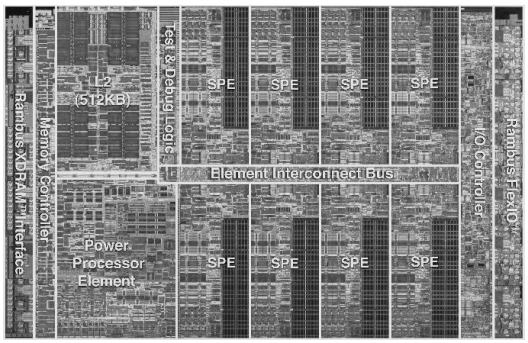

Abb. 3. Chip Foto des Cell-Prozessors

Selbsttestdomänen mit jeweils eigener STUMPS Instanz [24] besteht. Da 70% der Chipfläche von den 8 identischen Synergistic Processing Elements (SPE) eingenommen werden und diese eine recht heterogene Struktur besitzen, werden sie im folgenden als repräsentativer Teil des Cell Prozessors betrachtet. Eine detaillierte Beschreibung des Cell Prozessors und insbesondere seines prüfgerechten Entwurfs kann [16] entnommen werden. Die SPE besteht aus 20.9 Millionen Transistoren, davon 7 Millionen in den Logikteilen und 14 Millionen in den Speicherfeldern des lokalen Speichers.

Die für den Test der SPE relevanten Kenndaten sind:

- 1.8 Millionen Logikgatter, 7 Millionen Transistoren in der Logik, 14 Millionen Transistoren in Speicherfeldern
- 150.000 Flipflops
- 82.500 Flipflops sind angeordnet in 32 STUMPS Kanäle
- Speicher-Felder sind nicht Teil des Logikselbsttests und werden über einen gesonderten Selbsttest abgedeckt

Für alle industriellen Schaltungen wurden das Design und die (vorgegebenen) Prüfpfade nicht verändert. Bei der Berechnung der Verlustleistung könnten durch eine optimierte Prüfpfadzuordnung bessere Ergebnisse als hier dargestellt erzielt werden.

4.2 Ergebnisse

Der folgende Abschnitt zeigt Ergebnisse für verschiedene Kombinationen aus Startwerten und erzeugten Mustern pro Startwert. Dabei beträgt die Anzahl der zufällig gewählten Startwerte 200 und pro Startwert generiert der PRPG eine feste Anzahl von 512 oder 1024 Mustern. Daraus ergibt sich eine Gesamtanzahl der angelegten Testmuster von 102.400 und 204.800. Für Schritt 2 des Algorithmus wurde der Parameter $limit$ auf einen Erfahrungswert von 3 gesetzt, für den sich eine gute Verteilung zwischen Problemgröße und erreichter Überdeckung ergibt.

Die detaillierten Ergebnisse für die ausgewählten Schaltungen finden sich in Tabelle 1.

Der erste Spaltenblock zeigt neben dem Schaltungsnamen die Zahl der Haftfehler, sowie die Anzahl der durch die Startwerte erkannten Fehler.

Im zweiten Spaltenblock werden Vergleichswerte für das in [16] beschriebene Verfahren angegeben, wobei die Kosten des Endergebnisses ($cost()$) sowie die relative Gesamteinsparung ($\%$) angegeben sind. Für das Verfahren aus [16] sowie den im Beitrag vorgestellten Algorithmus werden jeweils die gleichen Mengen von Startwerten verwendet.

Der letzte Spaltenblock zeigt die mit dem vorgestellten Algorithmus erreichten Ergebnisse. Er ist entsprechend dem vorgestellten Optimierungsalgorithmus in mehrere Teilblöcke unterteilt. Für die harten Fehler ist deren absolute Anzahl sowie die Kosten der hierfür generierten Blöcke $cost(B_1 \cup B_0)$ angegeben. Die folgende Spalte $FSIM$ gibt Auskunft über die Anzahl der durch die Blöcke entdeckten Fehler nach der Fehlersimulation. Für die schwer erkennbaren Fehler wird das Ergebnis der Kostenfunktion $cost(B_2 \cup B_1 \cup B_0)$ nach der Optimierung angegeben, wiederum gefolgt von der durch Fehlersimulation des Zwischenergebnisses errechneten Abdeckung. Schliesslich werden die für das Endergebnis benötigten Kosten sowie die erreichte Ersparnis ($\%Red.$) angegeben.

Es zeigt sich, dass der Algorithmus über die gesamte Bandbreite verschiedener Schaltungstypen bessere Ergebnisse erzielt, als der in [16] vorgestellte. Besonders bei den Schaltungen b18, b19 und p286k ergibt sich eine drastische Verbesserung. Der Pseudozufallstest verlangt in der Regel mit steigender Schaltungsgröße auch eine Zunahme der Testlänge. Bleibt die Testlänge konstant, wächst die Zahl der harten Fehler und beeinträchtigt die Effizienz des Verfahrens. Denn in der Folge werden für diese Schaltungen die Kosten stark von den harten Fehlern dominiert

200,512	# Prüf-pfade	# Fehler		[16]		Harte Fehler		FSIM	Schwere + Harte F.	FSIM	Alle Fehler	
200,1024		gesamt	erkannt	cost()	% Red.	# Fehler	cost()	# Fehler	cost()	# Fehler	cost()	% Red.
s38417	32	32320	31144	2723	57.44	555	1171	30330	1760	31023	2081	67.48
			31634	2809	56.10	467	1207	30632	1917	31347	2496	60.99
s38584	32	38358	36370	1852	71.05	45	208	21458	568	29570	1384	78.36
			36395	1489	76.73	15	94	16502	345	24896	1276	80.05
b17	32	81330	68598	4097	35.97	3263	2750	68445	3059	68570	3143	50.89
			70256	4379	31.57	2148	2678	70056	3116	70229	3210	49.84
b18	32	277976	234852	5431	15.14	9754	3957	234464	4348	234780	4418	30.96
			239652	5488	14.25	6080	3915	239321	4343	239601	4406	31.15
b19	32	560696	468017	5660	11.55	20460	4693	467639	4933	467956	4955	22.57
			479119	5807	9.26	13135	4648	478591	5039	479044	5064	20.87
p286k	55	648044	605342	10698	2.75	7874	9970	598746	10071	598792	10093	8.25
			609701	10696	2.76	6171	9814	603119	9906	603162	9918	9.84
p330k	64	547808	488889	9772	23.65	4066	3790	471817	8384	479429	8932	30.21
			491305	9384	26.68	3040	6968	480774	7962	481874	8550	33.19
p388k	50	856678	836398	6233	37.66	3656	3857	794256	4688	825105	5450	45.49
			838914	5947	40.52	3176	3532	799977	4408	817713	5100	49.00
p418k	64	688808	632941	9400	26.55	9477	6982	597112	7892	614594	8470	33.82
			638702	9050	29.29	9005	6364	600086	7514	630422	8088	36.81
p951k	82	1590490	1538987	10924	33.39	9621	8553	1443351	9574	1462481	10335	36.97
			1544946	10599	35.37	8964	8246	1468258	9115	1468258	10104	38.38
SPE	32	1065190	903645	3388	47.06	2207	1323	897242	1917	900753	2569	59.85
			904317	3180	50.32	2081	1251	898322	1721	901870	2340	63.43

Tabelle 1

200 STARTWERTE, 512 BZW. 1024 MUSTER

und die zusätzlichen Freiheitsgrade in den beiden späteren Schritten können nicht ausgenutzt werden. Zum Beispiel werden bei p951k im ersten Schritt bei $200 * 512$ Mustern bereits 83% der Kosten zur Erkennung der harten Fehler benötigt, bei p286k gar 99%. Die erzielte Minderung der Verlustleistung ist also von den Parametern „Anzahl der Startwerte“ und „Muster pro Startwert“ abhängig.

Darum wird in Abbildung 4 für die Benchmarkschaltungen aus ISCAS und ITC die Anzahl der Startwerte variiert, wobei die Gesamtzahl der Muster konstant bei $102,400$ gehalten wird. Es zeigt sich, dass durch eine feinere Granularität die Reduzierung der Verlustleistung erheblich verbessert werden kann. Dies ist darauf zurückzuführen, dass für eine höhere Zahl von Startwerten die Zahl von Mustern und damit die Zahl von Fehlern, die durch jeden Startwert erkannt werden, entsprechend abnimmt.

Im Vergleich zu [16] zeigen sich auch Verbesserungen der Qualität der gefundenen Lösung. Hierzu wird die Zahl der Flip-Flops, die für die jeweils gefundene Blockmenge pro Startwert aktiviert wird (Abb. 5), betrachtet. Es zeigt sich, dass durch die in [16] vorrausgesetzte Reihenfolge für die zuerst bearbeiteten Startwerte keine oder nur sehr wenige Prüfpfade deaktiviert werden können. Dagegen ist

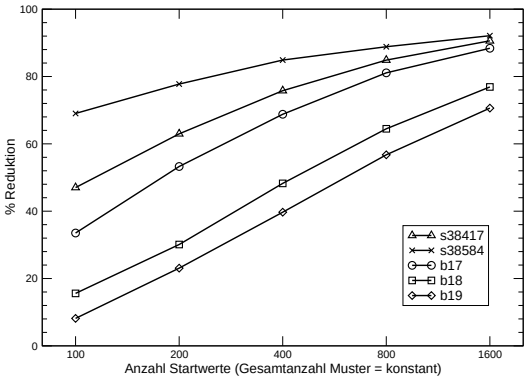

Abb. 4. Abhängigkeit von der Anzahl der Startwerte für ISCAS und ITC

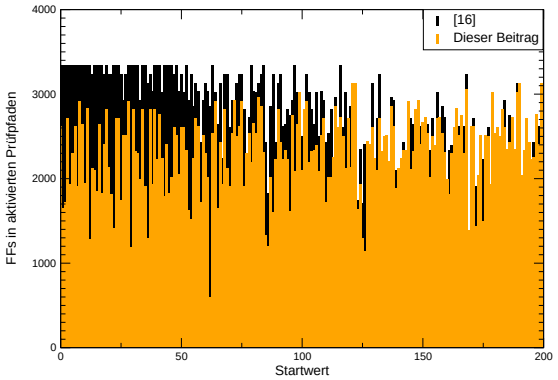

Abb. 5. Lösung für b18, 200 Startwerte, 512 Muster/Startwert

die durch den vorgestellten Algorithmus berechnete Lösung nicht von diesem Überhang betroffen. In [16] wurde vorgeschlagen die Startwerte (mit deren zugeordneter Konfiguration) entsprechend umzuordnen, um eine gegebene Hüllkurve der Verlustleistung zu erfüllen, was auch für die durch den vorgestellten Algorithmus berechnete Lösung getan werden kann.

5 Schlussbemerkungen

In vielen aktuellen Chip-Entwürfen können Prüfpfade während des Tests individuell deaktiviert werden. Im vorliegenden Beitrag wurde ein Algorithmus vorgestellt, mit dem ein optimierter Testplan generiert wird, der während des Tests möglichst viele Prüfpfade vorübergehend deaktiviert um die durchschnittliche Verlustleistung zu reduzieren. Die ermittelten Ergebnisse zeigen eine wesentliche Verbesserung gegenüber dem bestehenden Ansatz, sowohl unter quantitativen als auch qualitativen Aspekten und demonstrieren die Skalierbarkeit des Verfahrens.

Danksagung

Die vorliegende Arbeit wurde unterstützt durch das IBM CAS Projekt „Improved Testing of VLSI Chips with Power Constraints“, sowie durch die Deutsche Forschungsgemeinschaft unter dem DFG Projekt „Realtest“ Wu245/5-1. Die Schaltungen von NXP wurden im Rahmen des Projekts „Realtest“ zur Verfügung gestellt.

Literatur

[1] Y. Zorian, “A distributed BIST control scheme for complex VLSI devices,” in *Proceedings of the 11th IEEE VLSI Test Symposium (VTS)*, 1993, pp. 4–9.

[2] P. Girard, “Survey of low-power testing of VLSI circuits,” *Design & Test of Computers, IEEE*, vol. 19, no. 3, pp. 80–90, 2002.

[3] N. H. E. Weste, K. Eshraghian, and M. J. S. Smith, *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, 2000.

[4] S. Gerstendoerfer and H.-J. Wunderlich, “Minimized power consumption for scan-based BIST,” in *Proceedings IEEE International Test Conference 1999, Atlantic City, NJ, USA, 27-30 September 1999*, 1999, pp. 77–84.

[5] S. Ghosh, E. MacDonald, S. Basu, and N. A. Touba, “Low-power weighted pseudo-random BIST using special scan cells,” in *Proceedings of the 14th ACM Great Lakes Symposium on VLSI 2004, Boston, MA, USA, April 26-28, 2004*, 2004, pp. 86–91.

[6] P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch, “A test vector inhibiting technique for low energy BIST design,” in *17th IEEE VLSI Test Symposium (VTS ’99), 25-30 April 1999, San Diego, CA, USA*, 1999, pp. 407–412.

[7] S. Manich, A. Gabarro, M. Lopez, and J. Figueras, “Low power BIST by filtering non-detecting vectors,” *Journal of Electronic Testing Theory and Applications (JETTA)*, vol. 16, no. 3, pp. 193–202, 2000.

[8] L. Whetsel, “Adapting scan architectures for low power operation,” in *Proceedings IEEE International Test Conference 2000, Atlantic City, NJ, USA*, 2000, pp. 863–872.

[9] V. Dabholkar, S. Chakravarty, I. Pomeranz, and S. Reddy, “Techniques for minimizing power dissipation in scan and combinational circuits during test application,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1325–1333, 1998.

[10] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, and A. Virazel, “Design of routing-constrained low power scan chains,” in *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France*, 2004, pp. 62–67.

[11] P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch, “Low power BIST design by hypergraph partitioning: Methodology and architectures,” in *Proceedings IEEE International Test Conference 2000, Atlantic City, NJ, USA, October 2000*, 2000, pp. 652–661.

[12] S. Wang and S. K. Gupta, “LT-RTPG: a new test-per-scan BIST TPG for low heat dissipation,” in *Proceedings IEEE International Test Conference 1999, Atlantic City, NJ, USA, 27-30 September 1999*, 1999, pp. 85–94.

[13] F. Corno, M. Rebaudengo, M. S. Reorda, G. Squillero, and M. Violante, “Low power BIST via non-linear hybrid cellular automata,” in *18th IEEE VLSI Test Symposium (VTS 2000), 30 April - 4 May 2000, Montreal, Canada*, 2000, pp. 29–34.

[14] R. Sankaralingam, N. A. Touba, and B. Pouya, “Reducing power dissipation during test using scan chain disable,” in *19th IEEE VLSI Test Symposium (VTS 2001), 29 April - 3 May 2001, Marina Del Rey, CA, USA*, 2001, pp. 319–325.

[15] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, and H.-J. Wunderlich, “A modified clock scheme for a low power bist test pattern generator,” in *19th IEEE VLSI Test Symposium (VTS 2001), Test and Diagnosis in a Nanometric World, 29 April - 3 May 2001, Marina Del Rey, CA, USA*, 2001, pp. 306–311.

[16] C. Zoellin, H.-J. Wunderlich, N. Maeding, and J. Leenstra, “Bist power reduction using scan-chain disable in the cell processor,” in *IEEE International Test Conference (ITC 2006), Santa Clara, CA, USA, October 24 - 26, 2006*, 2006.

[17] P. H. Bardell and W. H. McAnney, “Self-testing of multichip logic modules,” in *Proceedings International Test Conference 1982, Philadelphia, PA, USA, November 1982*, 1982, pp. 200–204.

[18] B. L. Keller and T. J. Snethen, “Built-in self-test support in the IBM engineering design system,” *IBM Journal of Research and Development*, vol. 34, no. 2/3, pp. 406–415, 1990.

[19] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. S. M. Hassan, and J. Rajski, “Logic BIST for large industrial designs: real issues and case studies,” in *Proceedings IEEE International Test Conference 1999, Atlantic City, NJ, USA, 27-30 September 1999*, 1999, pp. 358–367.

[20] H.-J. Wunderlich, “Multiple distributions for biased random test patterns,” in *Proceedings International Test Conference 1988, Washington, D.C., USA, September 1988*, 1988, pp. 236–244.

[21] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, “The design and implementation of a first-generation Cell processor,” in *International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers, 6-10 Feb. 2005, San Francisco CA*, 2005, pp. 184–185.

[22] O. Coudert, “On solving covering problems,” in *Proceedings of the 33st Conference on Design Automation, Las Vegas, Nevada, USA, June 3-7, 1996*, 1996, pp. 197–202.

[23] I. Hamzaoglu and J. H. Patel, “New techniques for deterministic test pattern generation,” in *16th IEEE VLSI Test Symposium (VTS ’98), 28 April - 1 May 1998, Princeton, NJ, USA*, 1998, pp. 446–452.

[24] M. Riley, L. Bushard, N. Chelstrom, N. Kiryu, and S. Ferguson, “Testability features of the first-generation Cell processor,” in *Proceedings of the IEEE International Test Conference (ITC), 8-10 Nov. 2005, Austin TX*, 2005, p. 6.1.