Analyse von Corner Cases und funktionaler Abdeckung auf Basis von Entscheidungsdiagrammen

Jan Langer and Ulrich Heinkel
Chemnitz University of Technology
Circuit and System Design Group
09126 Chemnitz, Germany
Email: {laja,heinkel}@infotech.tu-chemnitz.de

Vasco Jerinić and Dietmar Müller
Core Mountains GmbH
Leonhardtstraße 30
09112 Chemnitz, Germany
Email: {vje,dm}@coremountains.com

***Zusammenfassung*— Ein stetig wachsender Anteil des Aufwands zum Entwurf digitaler Schaltungen entfällt auf die funktionale Verifikation. Der Verifikationsraum als Menge aller möglichen Kombinationen von Attributen einer Komponente, d. h. der Parameter und Eingangsdaten, ist oftmals sehr groß, wodurch die Verifikation aller Kombinationen unpraktikabel wird. Deshalb verwenden moderne Methoden der funktionalen Verifikation die zufallsgesteuerte Erzeugung von Stimuli in Verbindung mit manuell definierten Spezialfällen, sog. Corner Cases, um eine möglichst hohe funktionale Abdeckung in der angestrebten Verteilung zu erzielen. Als großer Nachteil diese Ansätze führen steigende Abdeckungsanforderungen zu exponentiell ansteigenden Laufzeiten. Um diesen Nachteil auszugleichen, wurden Generatoren propagiert, die nur solche Kombinationen erzeugen, die nicht bereits abgedeckt worden sind. Leider können die dabei verwendeten Verfahren das Problem nicht zufriedenstellend lösen, da auch sie im Allgemeinen zufällige Kombinationen erzeugen, um in einem zweiten Schritt zu prüfen, ob diese bereits abgedeckt sind. Im vorliegenden Beitrag werden Entscheidungsdiagramme zur Repräsentation aller zulässigen Kombinationen innerhalb des Verifikationsraums verwendet. Mit Hilfe dieses analytischen Modells kann jede beliebige Anzahl von Kombinationen in linearer Zeit erzeugt werden. Wird die vorgestellte Methode auf die Zufallserzeugung zur funktionalen Verifikation angewendet, kann diese um Größenordnungen beschleunigt werden.**

I. Einleitung

Die funktionale Verifikation nimmt einen stetig wachsenden Anteil des Entwurfsaufwands ein. Der Verifikationsraum als Menge aller möglichen Kombinationen von Parametern und Eingangssignalen ist zu groß, als dass die Verifikation aller dieser Kombinationen möglich ist [1]. Verifikationsumgebungen, wie *Specman Elite*^{TM} von *Cadence* oder *VERA*^{TM} von *Synopsys*, verwenden zufällig erzeugte Stimuli in Kombination mit manuell definierten Spezialfällen, sog. *Corner Cases*, um akzeptable Abdeckungen mit der gewünschten Verteilung zu erreichen. Dabei kommen i. allg. spezielle Hardwareverifikationssprachen (Hardware Verification Language–HVL) zum Einsatz, um die Stimuli für das Modell in einer Hardwarebeschreibungssprache (Hardware Description Language–HDL) zu generieren und die Reaktion des Modells auszuwerten. Neben dem Soll-Ist-Vergleich des Modellverhaltens werden Daten gesammelt, die zu einem späteren Zeitpunkt zur Analyse der funktionalen Abdeckung herangezogen werden können. Im Gegensatz zur sog. *Code Coverage*, die sicherstellen soll, dass jede Zeile eines Modells zumindest einmal ausgeführt wurde, wird für die funktionale Abdeckung die Funktionalität eines Entwurfs genauer betrachtet. Aus diesem Grund sind Verifikationsstrategien, die den Verifikationsfortschritt anhand der funktionalen Abdeckung bewerten, Entwurfs- und sogar Implementierungsspezifisch [1], [2]. In diesem Beitrag stellen wir einen Ansatz vor, mit dessen Hilfe Corner Cases sehr viel genauer definiert und klassifiziert werden können, als mit den verbreiteten Verfahren. Dieser soll Verifikationsingenieure in die Lage versetzen, die Definition von Corner Cases zu automatisieren und bessere Abdeckungsgrade früher im Verifikationsprozeß zu erreichen.

Teile dieser Arbeit wurden unter dem Förderkennzeichen <01M 0348 A> Intellectual Property Qualifikation für effizientes Systemdesign (IPQ) vom Bundesministerium für Bildung und Forschung (BMBF) unter dem Schwerpunkt Entwurfsplattformen für komplexe angewandte Systeme und Schaltungen der Mikroelektronik (EkompaSS) gefördert.

II. Vorarbeiten

Bei der Nutzung funktionaler Abdeckung müssen eine Vielzahl von Verifikations- bzw. Abdeckungsmodellen (sog. *Coverage Tasks*) systematisch vom Nutzer erstellt werden, um dann zur Messung, Überwachung und Steuerung des Testvorgangs verwendet zu werden [1]. Jedes dieser Modelle deckt ein bestimmtes Gebiet des Verifikationsraumes ab [3], indem es ihn entsprechend definierter Grenzen einschränkt [4]. Um den gesamten Verifikationsraum abzudecken, muss jeder einzelne Punkt des Raumes durch eines dieser Abdeckungsmodelle behandelt werden.

Jede Kombination von Attributwerten, d. h. jedes einzelne Element des Kreuzproduktes der Attribute eines Modells, kann entweder gültig oder ungültig sein. Deshalb ist es notwendig, Nebenbedingungen (*Constraints*) einzuführen, die die gültigen Attributkombinationen identifizieren.

In [5] wird die Nutzung einer speziellen Art von Entscheidungsdiagrammen, sog. *Domänengraphen (DG)* vorgeschlagen, um den Verifikationsraum in Unterräume (Domänen) aufzuteilen. Dies geschieht auf Grundlage formaler Beschreibungen der Attribute und ihrer gegenseitigen Abhängigkeiten. Bei der Konstruktion des DG werden alle ungültigen bzw. verbotenen Attributkombinationen entfernt und alle Kombinationen, die im Sinne der Verifikation gleich sind, d. h. zu identischem Verhalten führen, werden zusammengefaßt [6]. Als Beispiel für identisches Verhalten kann eine veränderte Position des Paritätsbits in einem seriellem Bitstrom dienen, dessen Paritätsbitgenerierung ausgeschaltet ist. Der entstehende Domänengraph beinhaltet alle unterschiedlichen, gültigen Attributkombinationen und wird im Folgenden als *wirksamer Verifikationsraum* bezeichnet.

Ausgehend vom DG werden automatisch nichtüberlappende Abdeckungsmodelle generiert. Diese vollständige Menge von Abdeckungsmodellen füllt den Verifikationsraum komplett aus und verhindert somit das Entstehen zusammenhängender Löcher. Um eine schnelle Traversierung des wirksamen Verifikationsraums für die automatische Stimuligenerierung zu ermöglichen, ist eine effiziente Repräsentation erforderlich. Aus diesem Grund wird der vorgeschlagene DG nicht direkt implementiert, sondern auf binäre Entscheidungsdiagramme abgebildet. In dem Softwarepaket PARAGRAPH [7] wurde die DG-Methodik bereits implementiert, um eine analytische Repräsentation des wirksamen Verifikationsraums zu erhalten. Dieses Softwarepaket dient als Grundlage der vorliegenden Arbeit.

In [8] werden eine Anzahl von Abdeckungskriterien vorgeschlagen, die die Oberfläche des Verifikationsraums betonen. Die Definition der Corner Cases ist der in dieser Arbeit gewählten Definition sehr ähnlich. Es gibt dort jedoch keine Möglichkeit sie zu klassifizieren oder ihre Kardinalität zu bestimmen, da der Algorithmus zur Generierung der entsprechenden Stimulidaten auf Constraintlogikprogrammierung (CLP) basiert, während im Gegensatz dazu die von uns vorgeschlagene Methode Entscheidungsdiagramme nutzt.

Eine Methode des pseudoerschöpfenden Testens wird in [9] untersucht. Sie kann, angewendet auf eine Softwarekomponente, eine höhere Abdeckung mit einer kleineren Anzahl Testcases erreichen. Die Anwendung dieser Methode auf Hardwarekomponenten stellt eine Alternative zu der in dieser Arbeit vorgestellten Corner Case Betonung dar.

III. VERIFIKATIONSRAUM

Ein Design wird während der Simulation durch Attribute parametrisiert, die verschiedene Konfigurationen und Zustände des Schaltkreises identifizieren. Jedes Design hat eine Menge von m Attributen $\{a_1, a_2 \ldots, a_m\}$. Jedes Attribut a_i kann einen der Werte aus seinem Wertebereich A_i annehmen. Dadurch spannen all diese Wertebereiche den Verifikationsraum $V = A_i \times A_2 \times \ldots \times A_m$ auf. Jedes Element $v = \langle a_1, a_2, \ldots, a_m \rangle$ aus V stellt genau eine bestimmte Kombination von Attributwerten dar, d. h. einen einzelnen Stimulivektor.

In den meisten realen Designs, sind nicht annähernd alle Attributkombinationen gültig. So kann es Abhängigkeiten zwischen den Attributen geben, die den Verifikationsraum V einschränken. Mit Hilfe dieser Abhängigkeiten kann eine Abbildungsfunktion $\texttt{isValid} : v \mapsto \{0,1\}$ definiert werden, die zu 1 evaluiert, wenn die Attributkombination v ein gültiger Stimulivektor ist, und anderenfalls zu 0. Mit Hilfe dieser Abbildung kann die Menge $G \subseteq V$ aller gültigen Kombinationen eingeführt werden:

$$G = \{\forall v \in V \mid \texttt{isValid}(v) = 1\}.$$

Es seien die Attributmenge $\{a,b\}$ gegeben, mit den entsprechenden Wertebereichen $A = \{1 \ldots 20\}$ und $B = \{1 \ldots 20\}$. Zwei zusätzliche Abhängigkeiten können als relationale Nebenbedingungen ausgedrückt werden:

$$a + b < 32$$

und

$$8b < a^2 + 16.$$

In diesem Fall ist die entsprechende Funktion `isValid` wie folgt definiert:

$$\texttt{isValid}(\langle a,b \rangle) = (a + b < 32) \wedge \left(8b < a^2 + 16\right).$$

Im konkreten Anwendungsfall wird die spezielle Definition von `isValid` von den Nebenbedingungen der Attribute in jeder Domäne hergeleitet. Diese müssen aus der Spezifikation des Designs extrahiert werden. In den von uns untersuchten realen Designs war dieser Vorgang ohne Probleme durchführbar. Dies wird möglich, da die Nebenbedingungen mit Hilfe aller logischen und arithmetischen Operatoren beschrieben werden, die in einer Reihe von HDL und HVL, wie z. B. *VHDL, Verilog* und *e* zur Verfügung stehen. Abb. 1 zeigt den Verifikationsraum V als ein zweidimensionales Gebiet und die entsprechenden Elemente des wirksamen Verifikationsraums G als graue Felder.

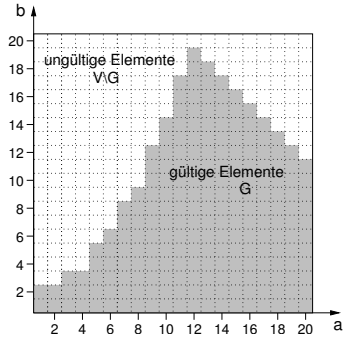


Abb. 1. Gesamter bzw. wirksamer Verifikationsraum

Jede der Mengen von Attributkombinationen innerhalb eines Domänengraphen wird als *mehrwertiges Entscheidungsdiagram (Multi-valued Decision Diagram – MDD)* [10] in Form einer charakteristischen Funktion wie folgt repräsentiert:

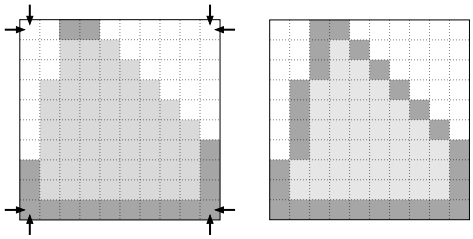
$$f(a_1, a_2, \ldots, a_m) = \begin{cases} 1: & \langle a_1, a_2, \ldots, a_m \rangle \in \text{Menge} \\ 0: & \textit{sonst} \end{cases}.$$

Die interne Darstellung der zugehörigen Datenstrukturen ist von *Binary Decision Diagrams* abgeleitet und erfuhr eine tiefgreifende Weiterentwicklung über *Multi-terminal Binary Decision Diagrams* und *Binary Moment Diagrams* zu *Kronecker Multiplicative Binary Moment Diagrams* [11], [12], [13], [14], [15]. Letztere erlauben die Darstellung der meisten Ausdrücke und ihrer Abhängigkeiten mit linearem Ressourcenbedarf. Dies erfüllt zwei üblicherweise gegensätzliche Ansprüche: schnelle Manipulation bei gleichzeitig geringem Speicherbedarf [15].

IV. Corner Case Definition

Der Verifikationsingenieur hat normalerweise die Aufgabe sogenannte Corner Cases zu definieren, die das Design in Zustände treiben, von denen man eine hohe Anfälligkeit für fehlerhaftes Verhalten erwartet. Die Erfahrung zeigt, dass Bugs, die durch eine Fehlinterpretation der Spezifikation oder durch Entwurfsfehler entstanden sind, oft im Zusammenhang mit Eingangdaten auftreten, deren Werte sich am Rande des zugehörigen Wertebereichs befinden. Deshalb ist die Möglichkeit erstrebenswert, die zufallsgesteuerte Generierung dahingehend zu beeinflussen, dass Corner Cases verstärkt generiert werden.

Es erscheint jedoch schwierig formal zu definieren, welche Werte bzw. Attributkombinationen Randwerte (Corner Cases) sind. In dieser Arbeit werden wir deshalb diejenigen Punkte als Corner Cases bezeichnen, die sich nahe an den Grenzen des wirksamen Verifikationsraumes G befinden [16]. Auf der linken Seite zeigt Abb. 2 an einem etweas kleineren Beispiel die Corner Case Definition laut einer kommerziell verfügbaren Verifikationsumgebung, die Corner Cases im Regelfall entsprechend den Randwerten der Wertebereiche der einzelnen Attribute definiert. Die Pfeile verweisen auf diese Regionen von Randwerten, d. h. die Werte 1 und 10 für beide Attribute. Die rechte Seite der Abbildung zeigt die Corner Case Definition entsprechend den wirklichen Grenzen des wirksamen Verifikationsraums G.



(a) Traditionelle Definition (b) DG-basierte Definition

Abb. 2. Abweichende Definitionen der Corner Cases

Im Allgemeinen kann man auch andere Kriterien zur Identifikation von Corner Cases heranziehen. Der Rest dieses Abschnitts beschreibt eine alternative Corner Case Definition, die besonders vorteilhaft für Verifikationsräume mit großen numerischen Wertebereichen ist. Logische Attribute, Attribute mit wenigen Werten oder Attribute ohne Ordnungsrelation (z. B. Enumerationen) können auch verwendet werden, fließen jedoch nicht in die Definition der Corner Cases ein.

Anfangs wird eine unklassifizierte Definition gegeben, die den Verifikationsraum in nur zwei Mengen aufteilt, Oberflächen und innere Punkte. Im weiteren Verlauf wird eine Klassifikationsmethode inklusive zugehörigem Algorithmus vorgestellt, die jedem Punkt des wirksamen Verifikationsraums eine Ordnung zuweist, um die Corner Cases präziser selektieren zu können.

Die Menge $S \subseteq G$ der Randelemente, die während der Simulation betont werden sollen, beinhaltet alle Elemente $v = \langle a_1, a_2, \ldots, a_m \rangle$ aus G, die mindestens einen Nachbarn $v' = \langle a'_1, a'_2, \ldots, a'_m \rangle$ haben, der nicht in G enthalten ist. Der Ausdruck Nachbar ist formal mittels der folgenden Funktion definiert:

$$\texttt{isNeighbour}\left(v, v'\right) = \begin{cases} 1: & \left(\sum_{i=1}^{m} |a_i - a'_i|\right) < B \\ 0: & \textit{sonst} \end{cases} .$$

Diese nutzt die sogenannte Manhattan-Distanz um zu bestimmen, ob die Entfernung zwischen zwei Elementen kleiner ist, als eine bestimmte Oberflächenbreite B. Die Menge S ergibt sich nun aus

$$S = \left\{ v \in G \mid \exists v' \mid \left(\texttt{isNeighbour}\left(v, v'\right) \wedge v' \notin G\right) \right\}.$$

Die Elemente im Inneren von G können mit Hilfe des Ausdrucks $I = G \setminus S$ bestimmt werden. Die Mengen der Oberflächenelemente und inneren Elemente des Beispiels aus Abschnitt III sind in Abb. 3 dargestellt.

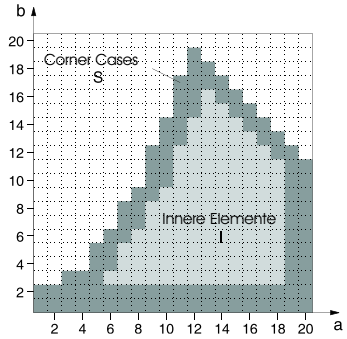


Abb. 3. Korrekte Definition der Oberflächenelemente

Die Definition der Corner Cases kann nun weiter verbessert werden, indem jeder Punkt der Oberfläche klassifiziert wird, d. h. jedem Punkt wird eine bestimmte Ordnung zugeordnet. Daraus folgt, dass der Raum S entsprechend nachfolgender Definition in nicht-überlappende Unterräume S_i aufgeteilt wird:

$$S_i = \left\{ v \in G \mid \left(i \equiv \left| v' \in V \backslash G \mid \texttt{isNeighbour}\left(v, v'\right) \right| \right) \right\}.$$

Jeder Punkt in S_i ist als Oberflächenpunkt der Ordnung i definiert und hat genau i ungültige Nachbarn. Dies schließt auch Nachbarn mit ein, die nicht Element von V sind, d. h. Elemente die außerhalb der Wertebereiche der Attribute liegen. Der Unterraum S_0 ist identisch zur Menge der inneren Punkte

I. Abb. 4 zeigt die Klassifikation der Oberflächenpunkte am Beispiel eines dreidimensionalen Körpers. Für orthogonale Räume, wie den gezeigten, stimmt diese Klassifikation implizit mit der eines beliebigen Zufallsgenerators überein, der einfach die Randwerte der einzelnen Attribute betont. Dabei werden Eckpunkte natürlich mit höherer Wahrscheinlichkeit selektiert als Kantenpunkte, und diese wiederum mit höherer Wahrscheinlichkeit als Punkte auf der Oberfläche des Körpers. Punkte im Inneren haben die geringste Selektionswahrscheinlichkeit. Der Ansatz alle Attribute getrennt zu betrachten ist jedoch auf Räumen mit nichtorthogonalen Grenzen nicht anwendbar. Stattdessen muss die Gesamtmenge der wirksamen Wertebereiche der Attribute in Betracht gezogen werden.

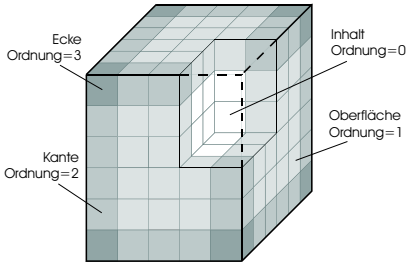


Abb. 4. Oberflächenklassifikation

Ein sehr einfacher und ineffizienter Algorithmus zur Klassifikation der Oberflächenpunkte kann direkt aus der Definition von `isNeighbour` abgeleitet werden. Für jeden Punkt aus *G* werden alle seine Nachbarn berechnet und die Anzahl der ungültigen darunter bestimmt. Das Resultat ist gleichzeitig die entsprechende Ordnung des Punkts. Wie bereits angedeutet, ist dieser Algorithmus sehr langsam, weshalb wir einen Entscheidungsdiagramm-basierten Algorithmus entwickelt haben, der *G* direkt manipuliert, um die Anzahl der Nachbarn für alle Punkte auf einmal zu berechnen. Abb. 5 zeigt die Schritte dieses Verfahrens für das vorgestellte einfache Beispiel. Für jeden möglichen Nachbarn eines Punkts aus *G* (durch die fett markierte Linie gekennzeichnet), wird der gesamte Raum in dessen Richtung verschoben. Danach werden die Werte aller verschobenen Räume zusammenaddiert, wobei der dadurch entstandene MDD die Anzahl der gültigen Nachbarn für jeden Punkt repräsentiert. Der Körper unten rechts in Abb. 5 zeigt das Resultat der vier Verschiebungen für das kleine Beispiel. Je dunkler die Farbe eines Elements dargestellt ist, desto mehr gültige Nachbarn hat dieser Punkt. Dieser Wert ist von der Gesamtanzahl der möglichen Nachbarn abzuziehen, um die Ordnung des Punktes zu erhalten. Weiterhin werden die Werte für Punkte außerhalb von *G* nicht betrachtet und auf 0 zurückgesetzt.

Die grundlegende Idee hinter der Bewertung der Corner Cases ist der Fakt, dass die Wahrscheinlichkeit fehlerhaften Verhaltens in (geometrischen) Ecken des Verifikationsraumes, an denen mehrere Attribute Randwerte einnehmen, höher ist, als auf den verbleibenden Oberflächengebieten. Der Zufallsgenerator kann nun, basierend auf dem Klassifikationsresultat, Corner Cases mit sehr hoher Ordnung, d. h. Punkte mit

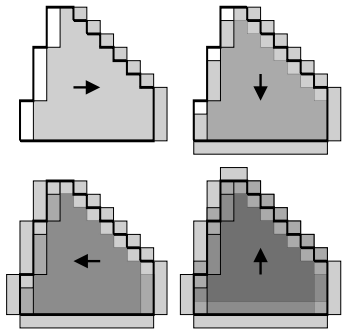


Abb. 5. Berechnungsschritte für den Corner Case Klassifikationsalgorithmus

einer hohen Anzahl von Nachbarn außerhalb des wirksamen Verifikationsraumes, zuerst auswählen. Damit werden die kritischsten Attributkombinationen selbst dann abgedeckt, wenn die Verifikation nur einen geringen Abdeckungsgrad erreicht. Dies kann das Vertrauen in das Ergebnis der simulativen Verifikation insbesondere dann verstärken, wenn der wirksame Verifikationsraum eines Designs so groß ist, dass die Überprüfung jedes einzelnen Punkts nicht praktikabel ist.

V. Abdeckungsgesteuerte Generierung

Moderne Verifikationsstrategien sammeln funktionale Abdeckungsdaten, um den Verifikationsfortschritt zu überwachen und den Verifikationsprozess dahingehend zu beeinflussen, dass die funktionale Abdeckung verbessert wird. Der Grad der Abdeckung berechnet sich aus dem Anteil derjenigen gültigen Attributkombinationen, die während der Simulation bereits generiert wurden. Die Menge dieser Kombinationen bezeichnen wir als $C \subseteq G$. Sie ist zu Beginn leer und füllt sich mit fortschreitender Verifikationsdauer mit Elementen aus *G*.

Es ist oft wünschenswert, Corner Cases zuerst zu verifizieren. Deshalb wurde die Menge der gültigen Attributkombinationen *G* in die bereits bekannten Untermengen *S* und *I* aufgeteilt, wobei *S* alle Kombinationen nahe der Oberfläche und *I* alle restlichen Elemente von *G* enthält. Da *S* die Corner Cases repräsentiert, die während der Simulation stärker betont werden sollen, kann der Verifikationsingenieur deren exakte Anzahl ermitteln und daraufhin eine anwendungsspezifische Verifikationsstrategie zum Einsatz bringen.

Es ist weiterhin möglich, *S* entsprechend den Ergebnissen des Corner Case Klassifikationsalgorithmus wiederum in mehrere Unterräume aufzuteilen. Somit wird es möglich, zufallsgesteuert zuerst Attributkombinationen unter den Elementen der höchsten Ordnung zu generieren, und erst danach mit Kombinationen absteigender Ordnungen fortzufahren.

Ein weiteres neues Verfahren ist die Generierungsmethode selbst. Sie garantiert, dass keine Attributkombination mehrfach erzeugt wird, während eine zufällige Reihenfolge mit Gleichverteilung eingehalten wird. Algorithmus 1 stellt einen einfachen Anwendungsfall dar, bei dem der Verifikationsingenieur alle Corner Cases generieren will, d. h. eine Oberflächenabdeckung von 100 % ist gefordert. Jede Operation des Algorithmus ist dabei eine effiziente MDD-Operation mit

konstanter Laufzeitkomplexität. Aus diesem Grund ist eine Generierungsgeschwindigkeit möglich, die proportional zur Anzahl der generierten Kombinationen ist.

Algorithmus 1 Generierungsalgorithmus für Attributkombinationen

$S' \leftarrow S$
$C_S \leftarrow \emptyset$
while $C_S \neq S$
    generate v from S'
    $C_S \leftarrow C_S \cup v$
    $S' \leftarrow S' \setminus v$
end

Konventionelle Verifikationsansätze erzeugen zufallsgesteuert Attributkombinationen, um dann mit Hilfe eines Constraint-Solvers die ungültigen wegzuwerfen, bis eine gültige Kombination gefunden wurde. Im Gegensatz dazu kann mit der vorgestellten Methode Zeit eingespart werden, da jede erzeugte Kombination wirksam zur Erhöhung der Abdeckung beiträgt.

Heutige Testgeneratoren sind mehr und mehr in der Lage effizient nur noch gültige Kombinationen zu generieren. Der Nachteil dieser Methoden bleibt jedoch, dass es nicht möglich ist, den genauen Abdeckungsgrad der Corner Cases, sowie den gesamten Abdeckungsgrad des wirksamen Verifikationsraums zu bestimmen. Mit Hilfe des MDD-basierten Ansatzes sind die exakten Abdeckungsgrade zu jedem Zeitpunkt während der Verifikation bekannt

$$\text{Corner Case Abdeckungsgrad} : \frac{|C_S|}{|S|}$$

$$\text{Gesamtabdeckungsgrad} : \frac{|C_S|+|C_I|}{|G|}.$$

VI. Ergebnisse

Die beschriebene Methodik wurde in das Softwarepaket ParaGraph [7] integriert. Aus einer Datei wird eine formale Beschreibung der Attribute gelesen, die ein spezielles Dateiformat nutzt [5] und mit dessen Hilfe ParaGraph den DG konstruieren kann. Ein hochgradig parametrisiertes serielles Kommunikationsinterface für Multimediaapplikationen, das in einem früheren Projekt mit einem industriellen Partner entwickelt wurde, dient als Demonstrationsbeispiel. Aus dem HDL Modell konnten die 15 in Tabelle I gezeigten Attributdefinitionen extrahiert werden. Abhängigkeiten zwischen

TABELLE I
Liste der Attribute

Attribut	Kardinalität	Attribut	Kardinalität
ALIGN	8	PARITY	3
DATALEN	4	STARTSTOP	2
FRAMEGEN	3	STOPBITS	2
FRAMELEN	10	VALID	2
FRAMEPOL	2	VALIDPOL	2
FRAMEPULSE	10	VALIDPOS	10
FRAMESYNC	2	WORDLEN	10
ORDER	2		

Attributen können, wie in Abschnitt III beschrieben, entweder als bedingte Constraints mit **REQUIRE** und **CONFLICT** ausgedrückt werden, oder in Form von relationalen Abhängigkeiten. Die Tatsache, dass das Attribut FRAMELEN das Schnittstellenverhalten nicht beeinflusst, wenn das Attribut FRAMEGEN den Wert none annimmt, kann beispielsweise mit Hilfe einer **CONFLICT** Anweisung deklariert werden:

```
CONFLICT framelen:
    framegen name: none;
```

Auf diese Art und Weise können insgesamt 9 Attributabhängigkeiten definiert werden:

```
CONFLICT framelen:
    framegen name: none;
CONFLICT framepulse:
    framegen name: none;
CONFLICT framepol:
    framegen name: none;
CONFLICT framesync:
    framegen name: none;
REQUIRE validpol:
    valid name:on;
REQUIRE validpos:
    valid name:on;
REQUIRE startstop:
    framegen name:none;
REQUIRE stopbits:
    framegen name:none;
REQUIRE wordlen sec:1:7:
    framegen name:none;
```

Zusätzlich wurden 7 relationale Constraints deklariert, die den wirksamen Verifikationsraum weiter einschränken:

```
RELATION validpos > align;
RELATION validpos <= wordlen;
RELATION framelen > framepulse;
RELATION datalen + align <= wordlen;
RELATION {(valid  == 'on') &&
          [(datalen + align) <= (wordlen - 1)]
         }
         || (valid  == 'off')
         ;
RELATION {(parity != 'off') &&
          [(datalen + align) <= (wordlen - 1)]
         }
         || (parity == 'off')
         ;
RELATION {(parity != 'off') &&
          (valid == 'on') &&
          [(datalen + align) <= (wordlen - 2)]
         }
         || (parity == 'off')
         || (valid == 'off')
         ;
```

Auf Grundlage dieser Definitionen konnte ParaGraph den Verifikationsraum in 6 Domänen aufteilen. Die Konstruktion des entsprechenden DG benötigt auf einer *Sun UltraSPARC®10* ungefähr 6 Sekunden. Die Größe des wirksamen Verifikationsraums G wird im Vergleich zum kompletten Verifikationsraum V um den Faktor 407 reduziert.

Im vorliegenden Beispieldesign wurden zuerst alle Corner Cases in zufälliger Reihenfolge verifiziert. Im Anschluss sind alle Nicht-Corner-Cases ebenso zufällig verifiziert worden. Dadurch wird die Wahrscheinlichkeit zu Beginn der Verifikation mehr Bugs zu finden signifikant erhöht.

Der entstandene Verifikationsfortschritt des betrachteten Designs ist in Abb. 6 dargestellt. Man kann deutlich erkennen, dass mit der vorgeschlagenen abdeckungsgesteuerten Verifikationsmethodik der Verifikationsfortschritt linear mit der Zeit wächst, sodass sie nach etwas weniger als drei Tagen beendet ist. Im Gegensatz dazu benötigt die zufallsgesteuerte

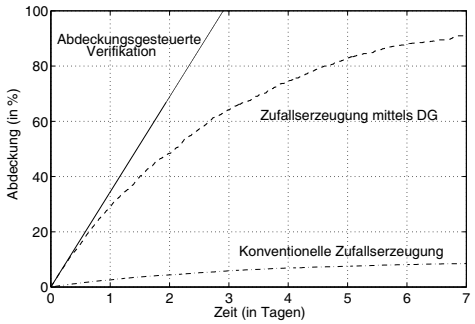


Abb. 6. Verifikationsfortschritt

Erzeugung von Punkten im wirksamen Verifikationsraum G in der untersten Kurve mehr als 6 Tage, um 90 % Abdeckung zu erreichen. Weiterhin ist mit konventionellen Methoden der Abdeckungsgrad der Corner Cases zu keinem Zeitpunkt der Verifikation bekannt, weshalb der Verifikationsingenieur sich nie sicher sein kann, dass er alle Corner Cases abgedeckt hat.

Um den genauen Abdeckungsgrad zu ermitteln, wird die Anzahl der gültigen Kombinationen, d. h. die Kardinalität von G benötigt. Dies ist für einfache bedingte Constraints (**`REQUIRE`**,**`CONFLICT`**) noch möglich, bereitet jedoch bei mehreren relationalen Constraints erhebliche Schwierigkeiten. Aus diesem Grund, erzeugen traditionelle Ansätze zufällig Punkte im kompletten Raum V, der jedoch den wirksamen Verifikationsraum G des Beispieldesigns um einen Faktor $10,87$ übersteigt. Dies führt zu einer Unterabschätzung des Abdeckungsgrades in der untersten der Kurven in Abb. 6, sodass die endgültige Abdeckung in diesem Fall noch nicht einmal 10 % beträgt.

Zusammenfassend lässt sich sagen, dass der wirksame Verifikationsraum G in Corner Cases und innere Elemente aufgeteilt werden konnte. Dabei hat sich herausgestellt, dass 90 % aller Attributkombinationen Corner Cases sind. Dies kommt daher, dass der Verifikationsraum durch die Constraints stark segmentiert wird und dadurch nur wenige größere zusammenhängende Gebiete mit gültigen Kombinationen vorhanden sind. Diese Information ist jedoch bereits sehr wertvoll für den Verifikationsingenieur, weil er damit in die Lage versetzt wird, einen exakt zugeschnittenen Verifikationsplan zu erstellen und die vorhandenen Ressourcen besser aufzuteilen.

Insgesamt gesehen ist das Beispieldesign relativ klein (50k Gatteräquivalente), und sein Verifikationsraum ist stark segmentiert. Daher ist der wesentlichste Vorteil der neuen Methodik, dass der Verifikationsingenieur genau über den Fortschritt und die zu erwartende verbleibende Verifikationszeit Bescheid weiß. Weiterhin konnte durch die Klassifikation der Corner Cases genauer gesteuert werden, welche Punkte wie stark betont werden sollen.

VII. Zusammenfassung und Ausblick

In dieser Arbeit wurde die DG Methodik auf die abdeckungsgesteuerte Verifikation angewendet. Durch die Nutzung von Entscheidungsdiagrammen kann eine beliebige Anzahl von Attributkombinationen zufällig und in linearer Zeit aus dem Verifikationsraum ausgewählt werden. Traditionelle Ansätze der gezielten (*directed*) Zufallserzeugung benötigen mit steigendem Abdeckungsgrad eine exponentielle Laufzeit. Im Gegensatz dazu kann mit der präsentierten Methodik ein erheblicher Laufzeitgewinn erzielt werden.

Zukünftige Arbeiten werden sich mit der Darstellung von Übergängen zwischen verschiedenen funktionalen Zuständen oder kompletten Stimulisequenzen beschäftigen. Weitere Untersuchungen sind nötig, um festzustellen, ob die vorgestellte Klassifikationsmetrik das Optimum bildet, oder ob bessere Ansätze gefunden werden können.

Literatur

[1] O. Lachish, E. Marcus, S. Ur, and A. Ziv, "Hole Analysis for Functional Coverage Data," *Design Automation Conference*, vol. DAC, pp. 807–812, June 2002.

[2] S. Ur and A. Ziv, "Off-The-Shelf Vs. Custom Made Covergae Models, Which Is The One for You?" in *Software Testing, Analysis, and Review (STAR98)*, May 1998.

[3] K. Kohno and N. Matsumoto, "A New Verification Methodology for Complex Pipeline Behavior," *Design Automation Conference*, vol. DAC, pp. 816–821, 2001.

[4] D.P. Appenzeller and A. Kuehlmann, "Formal Verification of a PowerPC Microprocessor," *IEEE International Conference on Computer Design*, vol. ICCD, no. '95, 1995.

[5] V. Jerinić and D. Müller, "Safe Integration of Parameterized IP," *INTEGRATION: The VLSI Journal*, vol. 37, no. 4, pp. 193–221, 2004.

[6] ——, "Assertion-Based Parameter Checking for IP," in *Proceedings of System-on-Chip and ASIC Design Conference DesignCon 2003*, International Engineering Consortium. IEC, jan 2003, santa Clara, CA, USA.

[7] ——, "Tool-Demo: ParaGraph - Parameter checking for IP," *Presentation of the edacentrum Design, Automation and Test in Europe Conference (DATE)*, vol. Munich, p. , March 2003.

[8] N. Kosmatov, B. Legeard, F. Peureux, and M. Utting, "Boundary coverage criteria for test generation from formal models," in *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 139–150.

[9] D. R. Kuhn and V. Okun, "Pseudo-exhaustive Testing For Software," in *30th NASA/IEEE Software Engineering Workshop*, April 2006.

[10] T. Y. K. Kam, "Multi-valued decision diagrams," Master's thesis, University of California, Departement of Electrical Engineering and Computer Sciences, Berkeley, CA 94720, USA, 1990.

[11] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and Their Applications," in *International Conference on Computer Aided Design (ICCAD '93), Santa Clara, California*, ACM/IEEE. IEEE Computer Society Press, Nov. 1993, pp. 188–191.

[12] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[13] R. E. Bryant and Y.-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," Carnegie Mellon University, Tech. Rep. CMU-CS-94-160, 1994.

[14] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid decision diagrams overcoming the limitations of MTBDDs and BMDs," in *International Conference on Computer Aided Design (ICCAD '95), Los Alamitos, CA, USA*. IEEE Computer Society Press, Nov. 1995, pp. 159–163.

[15] R. Drechsler, B. Becker, and S. Ruppertz, "K*BMDs: a new data structure for verification," in *IFI WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, Frankfurt, Germany*, 1995. [Online]. Available: citeseer.ist.psu.edu/article/drechsler96kbmds.html

[16] V. Jerinić, J. Langer, U. Heinkel, and D. Müller, "New Methods and Coverage Metrics for Functional Verification," in *Proceedings of Design, Automation and Test in Europe (DATE) 2006*. IEEE, March 2006, pp. 1025–1030.