

# Ein Verfahren zur Verifikation hochkomplexer Randbedingungen beim IC-Entwurf

Jan Freuer, Göran Jerke, Robert Bosch GmbH, Reutlingen  
André Schäfer, Kai Hahn, Rainer Brück, Universität Siegen, Institut für Mikrosystemtechnik  
Ammar Nassaj, Technische Universität Dresden, IFTE  
Wolfgang Nebel, Universität Oldenburg und OFFIS

## Kurzfassung

Die wachsenden Anforderungen an elektronische Baugruppen und die rasante technologische Entwicklung erzwingen die Absicherung der spezifizierten qualitativen und funktionalen Eigenschaften der Baugruppen mit Hilfe automatisierter Verifikationsverfahren. Die dabei verwendeten Verifikationswerkzeuge sind bisher auf die Überprüfung einer festen Menge vordefinierter und meist nicht erweiterbarer Entwurfsrandbedingungen beschränkt.

Der Artikel beschreibt eine neue Verifikationsmethodik auf der Basis einer vereinheitlichten Darstellung von Entwurfsrandbedingungen und werkzeugübergreifender Verifikationsaufgaben. Mit Hilfe des vorgestellten Constraint-Engineering-Systems wird erstmals eine flexible, erweiterbare und werkzeugübergreifende Definition von komplexen Entwurfsrandbedingungen und Verifikationsaufgaben höherer Ordnung ermöglicht. Vorhandene Verifikations- und Simulationswerkzeuge lassen sich mit Hilfe dieses Systems leicht zu flexiblen Metaverifikationswerkzeugen kombinieren, mit denen sich Verifikationsaufgaben eines Komplexitätsniveaus bearbeiten lassen, welches dem der Einzelwerkzeuge bei weitem übersteigt. Beispiele aus der praktischen Anwendung im analogen Systementwurf zeigen die Flexibilität und das Potenzial dieses neuen Verifikationsansatzes.

## 1 Einleitung

Die entwurfsseitige Sicherstellung der Funktion und Qualität einer elektronischen Baugruppe, wie z.B. eines integrierten Schaltkreises (IC), erfolgt über die Definition und die Berücksichtigung von relevanten applikationsspezifischen Randbedingungen während der Entwurfs- und Verifikationsphase. Aufgrund der rasant steigenden funktionalen und qualitativen Anforderungen an ICs und der wachsenden Komplexität moderner mikroelektronischer Fertigungstechnologien, gewinnt sowohl die möglichst vollständige und automatisierte Berücksichtigung von Entwurfsrandbedingungen als auch die Verifikation ihrer Erfüllung im Entwurfsergebnis eine immer größere Bedeutung.

Heute verfügbare Verifikationswerkzeuge für den IC-Entwurf konzentrieren sich meist allein auf die schnelle Bearbeitung weniger spezieller Verifikationsaufgaben für eine quantitativ hohe Anzahl von Verifikationsinstanzen. Insbesondere für die Verifikation von analogen, mixed-signal und RF-ICs ist es sehr wichtig, eine Vielzahl von komplex zusammenhängenden und domainübergreifenden Randbedingungen berücksichtigen zu können [4, 9, 13, 14]. Für diesen Zweck ist eine *qualitative* Vernetzung der gewonnenen Verifikationsergebnisse zur Bearbeitung komplexer Verifikationsaufgaben notwendig, welche sich aber mit bisherigen Verifikationsansätzen aufgrund deren fehlender Flexibilität und Generalität nicht realisieren lässt.

Der vorliegende Artikel beschreibt ein neuartiges software-basiertes Verifikationsrahmenwerk namens *Constraint-Engineering-System* (CES), welches auf den Ansätzen der Constraint-Logik-Programmierung (CLP) [2, 8] basiert. Mit dem vorgestellten CES ist es erstmals möglich, (1) sowohl einfache als auch komplexe Verifikationsaufgaben einheitlich und entwurfswerkzeugübergreifend beschreiben und (2) die Verifikation einfacher und komplexer Entwurfsrandbedingungen in einer vereinheitlichten Verifikationsumgebung durchführen zu können. Gegenüber herkömmlichen Verifikationswerkzeugen, erweitert das CES den Abbildungsraum für Verifikationsprobleme. Dies beruht auf dessen Eigenschaft, für die Bearbeitung komplexer Verifikationsaufgaben, die Fähigkeiten verschiedener Verifikations- und Simulationswerkzeuge problemspezifisch und flexibel miteinander kombinieren zu können.

Das nachfolgende Kapitel 2 gibt zunächst einen kurzen Überblick über den aktuellen Stand der Technik im Bereich der IC-Verifikation. Die Definition wichtiger Begrifflichkeiten erfolgt in Kapitel 3 inklusive einer Diskussion der generellen Anforderungen an die Verifikation von Entwurfsrandbedingungen. Kapitel 4 stellt das entwickelte CES und den Verifikationsablauf detailliert vor. Anhand praktischer Beispiele werden die bisher erzielten Ergebnisse in Kapitel 5 vorgestellt. Der Beitrag schließt mit einem Ausblick auf künftige Arbeiten und einer Zusammenfassung.

## 2 Stand der Technik

Innerhalb derzeit verfügbarer IC-Entwurfsumgebungen erfolgt die Überprüfung der Einhaltung einfacher und komplexer Randbedingungen (engl. constraints) ausschließlich mit Hilfe spezialisierter Verifikationswerkzeuge, für z.B. den Design-Rule-Check oder das Überprüfen von extrahierten Timingdaten. Diese hochspezialisierten Werkzeuge können dabei i.d.R. nur eine feste und bei ihrer Spezifikation und Implementierung vordefinierte Klasse von Randbedingungen berücksichtigen. Eine nachträgliche Erweiterung der Menge handhabbarer Randbedingungen zur Bearbeitung weiterer Verifikationsaufgaben kann nur über eine Abbildung des neuen Verifikationsproblems auf die vorhandenen Fähigkeiten der Werkzeuge erfolgen. Der Erweiterbarkeit von Verifikationswerkzeugen sind daher Grenzen gesetzt, wenn die erforderlichen Abbildungen des neuen Verifikationsproblems nicht eindeutig und vollständig ausgeführt werden können.

Die effizienteste und geeignetste Möglichkeit den notwendigen Verifikationsaufwand zu minimieren ist, randbedingungsberücksichtigende Entwurfsverfahren durchgängig im Entwurfsfluss einzusetzen [4, 9, 13, 14]. Dabei ist es notwendig, Entwurfsrandbedingungen geeignet speichern und verwalten zu können. Malavasi et al. stellte in [12] das Konzept und die Anforderungen an ein sog. Constraint-Management-System vor, welches vorrangig der Verwaltung von typischen Randbedingungen des IC-Entwurfs dient. Randbedingungsverwaltende Management-Systeme existieren bereits heute in proprietärer und zueinander meist inkompatibler Form in zahlreichen IC-Entwurfsumgebungen.

Das Problem der (teil-)automatischen Generierung und Transformation von Entwurfsrandbedingungen im Top-Down-Entwurf ist für die korrekte Erstellung der Verifikationsaufgaben von grundlegender Bedeutung [10, 11, 12]. Gemäß Ly et al. [10] lassen sich hierbei bessere Entwurfsergebnisse erreichen, wenn die Bottom-Up-Charakteristika kritischer Komponenten für die Steuerung der Top-Down-Spezifikation anderer Komponenten benutzt werden. Malavasi et al. beschrieb in [11] das sog. *Transformationsproblem für Randbedingungen* im Top-Down-IC-Entwurf. Die Generierung und Transformation von Randbedingungen für alle verwendeten Entwurfswerkzeuge erfolgte hierbei mit Hilfe linearisierter Modelle für das Schaltungsverhalten, welche aus Sensitivitätsanalysen abgeleitet wurden.

Die dem hier vorgestellten Constraint-Engineering-System zugrundeliegende Constraint-Logik-Programmierung (CLP) ist eine Erweiterung der logischen Programmiersprachen durch die Integration von constraint-auflösenden Methoden [1, 2, 5, 6, 8, 15, 16]. Die CLP hat sich in den letzten zwei Dekaden aus dem Bereich der Linearen Programmierung (LP) und aus den Ansätzen der künstlichen Intelligenz im Be-

reich der deklarativen Logik entwickelt. Sie definiert zudem eines der aktuellen strategischen Forschungsgebiete auf dem Gebiet der Informatik [2].

## 3 Verifikation der Einhaltung von Entwurfsrandbedingungen

Die Qualität eines Entwurfsergebnisses wird dadurch bestimmt, inwieweit das Ergebnis die angestrebten Optimierungsziele erreicht und alle gegebenen Entwurfsrandbedingungen einhält. Entwurfsrandbedingungen definieren hierbei Beziehungen zwischen einer Menge von Variablen, wobei es das Ziel einer Entwurfsaufgabe ist, eine widerspruchsfreie Belegung für alle gegebenen Variablen zu finden. Sowohl die Widerspruchsfreiheit der Randbedingungen als auch deren Einhaltung muss dabei für unabhängige und voneinander abhängige Variablen durch die Anwendung einer Verifikation des Entwurfsergebnisses sichergestellt werden.

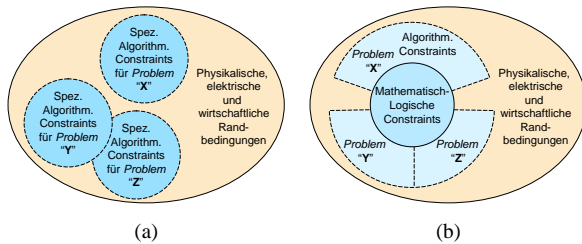
Eine Beziehung zwischen *unabhängigen* Variablen wird im Folgenden als *einfache Randbedingung* bezeichnet, wie z.B. die Forderung der Einhaltung eines max. elektrischen Leitbahnwiderstandes zwischen zwei definierten Netzterminals. Beziehungen zwischen voneinander *abhängigen* Variablen lassen sich beim Top-Down-Entwurf als Kombinationen einfacher Randbedingungen abbilden. Dies ermöglicht die Definition von Randbedingungen höherer Ordnung, welche im folgenden Text als *komplexe Randbedingungen* bezeichnet werden, wie z.B. die Forderung einer korrekten Platzierung von Floorplan-Blöcken in Abhängigkeit von Padpositionen und einer gegebenen On-Chip-Isotherme oder -Isobare.

Die notwendige Verifikation der Einhaltung einer einzelnen einfachen Randbedingung wird im Folgenden als *einfache Verifikationsaufgabe* bezeichnet. Analog dazu bezeichnet eine *komplexe Verifikationsaufgabe* die Verifikation der Einhaltung einer einzelnen komplexen Randbedingung. Eine Menge von Verifikationswerkzeugen zur Verifikation einer komplexen, jedoch nicht notwendigerweise domainübergreifenden Verifikationsaufgabe, wird nachfolgend als sog. *Metaverifikationswerkzeug* bezeichnet.

Zum Ermöglichen der automatisierten Verifikation komplexer Entwurfsrandbedingungen bedarf es

1. einer eindeutigen und vollständigen Abbildbarkeit von Entwurfsrandbedingungen,
2. einer einheitlichen, abstrahierten, maschinell verarbeitbaren und werkzeugübergreifenden Darstellung von Entwurfsrandbedingungen und Verifikationsaufgaben,
3. einer Möglichkeit, relevante Entwurfsrandbedingungen zu jedem Zeitpunkt im Entwurfsfluss adressieren zu können,
4. einer Möglichkeit zum transparenten Zugriff auf

alle relevanten Designinformationen einer Verifikationsaufgabe.



**Bild 1:** Randbedingungen in Entwurfsproblemen: (a) bisherige getrennte Berücksichtigung, (b) einheitliche und vollständige Berücksichtigung im CES.

Das Vorhandensein eines einheitlichen Formalismus zur Transformation von zu verifizierenden Entwurfsrandbedingungen zwischen aufeinander folgenden Entwurfsschritten ist neben der vollständigen Abbildbarkeit von Entwurfsrandbedingungen essentiell für eine konsistente Bearbeitung von Verifikationsaufgaben im Entwurfsfluss. Aufgrund der genannten Anforderungen ist es daher für ein Verifikationsrahmenwerk erforderlich, alle Entwurfsrandbedingungen in einer einheitlichen abstrahierten Form eindeutig und vollständig abbilden zu können (siehe Bild 1).

## 4 Constraint-Engineering-System

In diesem Abschnitt werden die für das CES notwendigen Komponenten und die Gesamtverifikation mit Hilfe des CES beschrieben. Das CES ersetzt nicht bereits existierende (formale) Verifikationslösungen. Es bietet vielmehr die Kombinationsmöglichkeit bestehender Verifikationswerkzeuge und die in diesem Abschnitt beschriebene Zusammenführung der Semantik dieser Werkzeuge auf eine abstrakte, formale Metaebene. Innerhalb des CES findet somit weder eine eigenständige Validierung noch eine eigene Datenhaltung statt. Diese Aufgaben werden von externen und für diesen Zweck spezialisierten Werkzeugen übernommen.

### 4.1 Architekturüberblick

Eines der Ziele des CES ist es, eine einheitliche und werkzeugübergreifende Darstellung von Constraint-Daten zu liefern. Zu diesem Zweck wurde ein Formalismus, basierend auf dem Horn-Kalkül [7] entwickelt, der es ermöglicht, innerhalb dieses Logik-Kalküls auf verschiedene Werkzeuge zugreifen zu können.

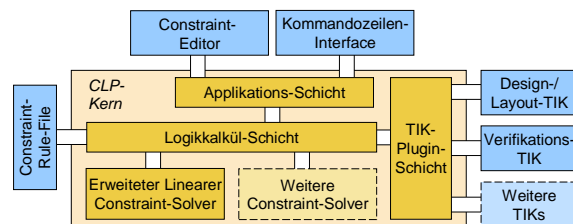
Für den IC-Entwurf kommen zurzeit kontinuierlich neue und leistungsfähigere Werkzeuge auf den Markt, die bei Bedarf ebenfalls in das CES eingebunden werden müssen. Dadurch kommen potenziell neue Constraints hinzu, die zum Zeitpunkt der Definition und Umsetzung des CES unbekannt sind. Die Architektur des CES muss demnach flexibel genug sein, um auf zukünftige Bedürfnisse eingehen zu können.

Die Grundlage des CES bildet ein CLP-Kern, wie er von Jaffar in [8] beschrieben wird (siehe Bild 2). Die Wissensbasis des CES-Logiksystems wird in Form von Horn-Klauseln bereitgestellt. Die *statische Wissensbasis* wird gebildet durch Regeln, die über verschiedene Verifikationsdurchläufe konstant bleiben. Im Gegensatz dazu liefern die Regeln aus der *dynamischen Wissensbasis* die werkzeugspezifischen, veränderlichen Daten und Constraints. Innerhalb des CES wird die Wissensbasis in drei Aspekte unterteilt:

**TIK:** Externe Werkzeuge exportieren Entwurfsdaten, Constraints und Verifikationsalgorithmen und bilden den dynamischen Aspekt der Wissensbasis. Alle Funktionen eines Werkzeuges werden in einem Tool-Integration-Kit (TIK) zusammengefasst.

**Constraint-Rule-File:** Das Constraint-Rule-File enthält die Menge aller möglichen Anfragen, die an das CES innerhalb einer bestimmten Konfiguration gestellt werden können. Diese Verifikationsanfragen werden über ein Kommandozeilen-Interface an das CES gestellt. Das Constraint-Rule-File ist ein Teil der statischen Wissensbasis des CES und ist von den bereitgestellten Regeln der externen Werkzeuge und den Support-Regeln abhängig.

**Support-Regeln:** Klauseln höherer Ordnung bilden Support-Regeln um einen einfacheren Zugriff auf die unterliegende Wissensdatenbasis aus dem Constraint-Rule-File zu erhalten. Häufig verwendete Anfragen lassen sich somit kombinieren. Die Support-Regeln gehören ebenfalls zur statischen Wissensbasis.



**Bild 2:** CES-Architektur

Allen Aspekten ist die gleiche formale Darstellung innerhalb des CES gemein. Das CES kann somit auf jeden Wissensaspekt in uniformer Weise zugreifen. Dies schafft die Grundlage für die einheitliche und werkzeugübergreifende Darstellung und Verarbeitung von Constraints.

Der Kern des CES setzt sich zusammen aus einer Logikkalkülschicht und mehreren Constraint-Solvern (siehe Bild 2). Die Logikkalkülschicht erlaubt es, mittels Unifikation [1] logische Schlussfolgerungen auf der Wissensbasis durchzuführen. Die Constraint-Solver erweitern den Anwendungsbereich der logi-

schen Schlussfolgerungen auf definierten, geordneten Mengen.

Innerhalb eines CLP-Systems wird unterschieden zwischen unterschiedlichen math. Domänen, auf denen Constraints definiert sind. Für jede Domäne verwendet das CLP-System einen anderen Constraint-Solver. Eine solche Domäne ist z.B. die Menge aller, über den Körper der natürlichen Zahlen definierten, mathematischen Ausdrücke. Es sind aber auch andere Domänen, wie z.B. die Ordnungsrelationen von Graphen denkbar.

Um auch künftigen Anforderungen gerecht zu werden, erlaubt es die Architektur des CES einen oder mehrere Constraint-Solver flexibel an die Logikschicht anzubinden. In der aktuellen Implementierung des CES ist ein Constraint-Solver, basierend auf dem Simplex-Verfahren über den reellen Zahlen, realisiert.

Durch das Simplex-Verfahren ist der Lösungsraum prinzipiell auf lineare (Un-)Gleichungen beschränkt. Um die Behandlung nichtlinearer (Un-)Gleichungen zu ermöglichen, wird die Auswertung nichtlinearer Constraints so lange verzögert, bis diese bei Verfügbarkeit weiterer Constraints linearisiert werden können. Falls während der Verifikation nicht genügend Constraints verfügbar werden, so wird als Ergebnis der nichtlineare und damit nicht aufgelöste Teil des Gleichungssystems mit ausgegeben. Die Erfüllbarkeit der durch das Gleichungssystem beschriebenen Constraints kann in diesem Fall nicht bestimmt werden.

Alternativ könnten beispielsweise zur Auswertung von polynomiellen (Un-)Gleichungen auch Gröbner-Basen eingesetzt werden [3]. Im Wesentlichen ist die Gröbner-Basis eine vereinfachte Menge von polynomiellen Gleichungen des ursprünglichen Gleichungssystems, die den Test der Erfüllbarkeit des Systems ermöglicht. Durch die Gröbner-Methode können wesentlich mehr Problemstellungen direkter abgebildet werden, als dies mit dem Simplex-Verfahren möglich ist. Allerdings besitzt dieses Verfahren eine wesentlich höhere Komplexität. Das Simplex-Verfahren bietet somit für den Einsatz in CLP eine gute Balance zwischen Leistung und Geschwindigkeit und wurde daher für den Einsatz im CES ausgewählt.

Einer der Schlüsselfaktoren des CES ist die Anbindung externer Verifikationswerkzeuge. Das CES bietet zur Anbindung eine definierte Schnittstelle, über die externe Werkzeuge in Form von Horn-Klauseln in das CES integriert werden können. Die Schnittstelle stellt einen Übersetzungsmechanismus zur Verfügung, der die Syntax und Semantik von Constraints ebenso wie die verifikationsrelevanten Daten des Werkzeuges in die Semantik des CES transformiert. Berechnungen, die nicht innerhalb des CES durchgeführt werden, werden ebenfalls durch diese Schnittstelle transformiert. So entsteht eine logische Metaebene, die alle Werkzeuge miteinander verbindet.

Die Erweiterbarkeit durch TIKs und die Integration von weiteren Constraint-Solvern erlauben es, das CES flexibel an zukünftige Bedürfnisse anzupassen. Der werkzeugübergreifende logische CLP-Formalismus erlaubt somit die Metaverifikation auch mit zukünftigen Werkzeugen.

## 4.2 Modellierung von Randbedingungen

Die Grundlage der Modellierung von Constraints innerhalb des CES bildet das Horn-Kalkül. Jedes TIK stellt dabei die spezifische TIK-Funktionalität in Form von werkzeugspezifischen Klauseln zur Verfügung.

Durch den Einsatz verschiedener Werkzeuge ergibt sich prinzipiell das Problem der doppelten Namensvergabe (Überladung). Um Klauseln mit gleichem Namen aus unterschiedlichen Werkzeugen auflösen zu können, werden alle Klauseln so definiert, dass anhand eines eindeutigen Bezeichners  $T_{ID}$  jedes spezifische Werkzeug direkt adressiert werden kann.

Bei Überladung ermittelt die Unifizierung des Logikkalküls automatisch das anzuwendende Werkzeug. Seien z.B. zwei Werkzeuge  $w_1$  und  $w_2$  gegeben, wobei  $w_1$  die Klauseln  $v_a(T_{w_1})$  und  $v_b(T_{w_1})$ , und  $w_2$  die Klauseln  $v_a(T_{w_2})$  und  $v_c(T_{w_2})$  exportiert. Eine Metaverifikation, die sowohl  $v_a$  als auch  $v_c$  instrumentiert, wird in der Form  $v_a(T_x) \wedge v_c(T_x)$  an das CES gestellt. Da für beide Verifikationsaufgaben dasselbe Werkzeug  $T_x$  verwendet wird, ermittelt das Logikkalkül mittels Unifikation  $T_x = w_2$ .

Zusätzlich stellt jedes TIK die Eigenschaftsklausel  $\Psi(T_{ID}, F)$  zur Verfügung anhand der sich die Funktionalität eines TIK ermitteln und die automatische Werkzeugauswahl des CES optimieren lässt. Die Eigenschaftsklausel ordnet jedem Werkzeug  $T_{ID}$  eine oder mehrere Eigenschaften  $F$  zu. Die innerhalb einer Verifikationsanfrage geforderten Werkzeugeigenschaften können so vor der eigentlichen Verifikation überprüft werden. Beispielsweise kann die Verifikation von  $v_a$  und  $v_b$  durch  $\Psi(T_x, v_a) \wedge \Psi(T_x, v_c) \wedge v_a(T_x) \wedge v_c(T_x)$  erweitert werden. Somit ermittelt das Logikkalkül das Werkzeug  $T = w_2$  vor den Verifikationen  $v_a$  und  $v_b$ . Die unnötige (und evtl. zeitaufwendige) Verifikation der Subaufgabe  $v_a(w_1)$  kann auf diese Weise vermieden werden.

Durch diese Methodik lassen sich komplexere Verifikationsaufgaben generisch definieren, ohne dass ein bestimmtes Werkzeug angegeben werden muss. Für die Verifikation wird durch das CES automatisch ein geeignetes, registriertes Werkzeug ausgewählt, welches, die für die Verifikation benötigte Funktionalität, bereitstellt. Generische Verifikationsaufgaben erlauben deren Anwendung auch in ähnlichen Verifikationsbereichen und ermöglichen damit deren Wiederverwendbarkeit.

## 5 Umsetzung und Ergebnisse

Die Umsetzung und Implementierung des CES erfolgte in der Programmiersprache Java. In seiner bestehenden Form wurde das CES an die kommerzielle Entwurfsumgebung DFII Virtuoso von Cadence sowie an zwei experimentelle Werkzeuge angebunden.

Die in diesem Abschnitt beschriebenen Beispiele aus dem Layoutentwurf zeigen, wie anhand der eingeführten Methodik zwei komplexe Verifikationsaufgaben durch die Instrumentierung von externen Werkzeugen innerhalb des CES durchgeführt werden können.

Die Netztopologien innerhalb eines IC-Layouts können unterschieden werden in *stern-*, *baum-* oder *maschenförmig* (siehe Bild 3). Zwischen den Pins eines Netzes  $P_n$  und  $P_m$  ( $n \neq m$ ) und dem Zentrum  $C$  tritt dabei der Leitbahnwiderstand  $R_{P_n, P_m}$  bzw.  $R_{C, P_n}$  auf.

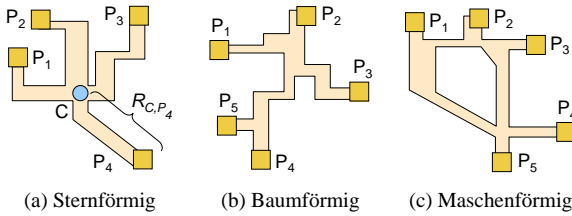


Bild 3: Netztopologien

Zur Verifikation soll das Ergebnis eines Layoutentwurfs auf die Einhaltung zweier Randbedingungen überprüft werden:

1. Für alle Leitbahnwiderstände  $R_{C, P_n}$  sternförmiger Netze soll gelten:  $R_{C, P_n} \leq R_{\max}$ ,
2. Für alle Pins  $P_n$  und  $P_m$  sternförmiger Netze soll gelten:  $|R_{C, P_n} - R_{C, P_m}| \leq \Delta R_{\max}$ .

Die Definition der Klauseln bei der Integration der benötigten externen Werkzeuge und die Beispiele der Anfragen an das CES erfolgen in der von Jaffar et al. eingeführten zu Prolog ähnlichen CLP-Notation [8].

### 5.1 Integration externer Werkzeuge

Zur Umsetzung der beiden Verifikationsaufgaben werden die folgenden Werkzeuge über TIKs in das CES eingebunden: (1) das DFII von Cadence (`dfii`) dient der Verwaltung von Layoutdaten, (2) ein Topologieextraktor (`topEx`) übernimmt die Klassifizierung der im DFII gehaltenen Netze und (3) ein Widerstandsextraktor (`resEx`) ermittelt anhand der Layoutdaten den Leitbahnwiderstand zwischen zwei gegebenen Punkten im Netz. Diese drei Werkzeuge bilden die für die Verifikation benötigte dynamische Wissensbasis. Zusätzlich werden im Folgenden auch die für die Verifikationen notwendigen Support-Klauseln als statische Wissensbasis definiert.

Das DFII exportiert als dynamische Wissensbasis Klauseln zur Identifikation von Layoutelementen (z.B.

Netze, Polygone und Pins), Eigenschaften dieser Elemente (Koordinaten) und hierarchische Zuordnungen wie z.B. die Netzzugehörigkeiten von Polygonen und Pins. Die Klauseln zur Identifizierung von Layoutelementen verwenden das im Abschnitt 4.2 dargestellte Verfahren zur Adressierung externer Werkzeuge:

```
net(dfii, NetID).
polygon(dfii, PolyID).
pin(dfii, PinID).
```

Als Eigenschaften können von Pins die Koordinaten und von Polygonen die Koordinatenliste sowie deren Netzzugehörigkeit ermittelt werden:

```
coordinate(PinID, (X, Y)).
coordinate(PolyID, [(X1, Y1), (X2, Y2), ...]).
netPin(NetID, PinID).
netPolygon(NetID, PolyID).
```

Zur Zusammenstellung der bereitgestellten Layoutelemente zu einem Tupel (`PinList`, `PolyList`) wird die Klausel `netLayout` als statische Wissensbasis innerhalb der Support-Regeln des DFII-TIKs definiert:

```
netLayout(NetID, (PinList, PolyList)) :-
    setof(Pos, (pin(NetID, PinID),
                coordinate(PinID, Pos)), PinList),
    setof(Pos, (polygon(NetID, PolyID),
                coordinate(PolyID, Pos)), PolyList).
```

Der Widerstandsextraktor ermittelt anhand von zwei gegebenen Punkten  $((X1, Y1), (X2, Y2))$  eines gegebenen Netzes den Leitbahnwiderstand:

```
resistance(resEx, (PinList, PolyList),
            (X1, Y1), (X2, Y2), Resistance).
```

Der Topologieextraktor klassifiziert ein gegebenes Netz in eine der drei Kategorien `star((X, Y))`, `tree` oder `mesh` (siehe Bild 3). Die Topologiekategorie `star` ist zusätzlich über dem Zentrum  $(X, Y)$  des Sterns parametrisiert:

```
topologyClass(topEx, (PinList, PolyList), Class).
```

### 5.2 Verifikationsanfragen

Aufgrund des einheitlichen Formalismus ist es möglich, die beiden Werkzeuge DFII und den Topologieextraktor innerhalb einer einzelnen Wissensanfrage zu kombinieren. Zur Bestimmung aller sternförmigen Netze eines Designs liefert beispielsweise die folgende Anfrage an das CES eine Liste `NL` von Bezeichnern (`NetID`) der Netze:

```
setof(NetID,
      (net(dfii, NetID), netLayout(NetID, Layout),
       topClass(topEx, Layout, star(_))), NL).
```

Zur Bearbeitung der ersten Verifikationsaufgabe  $R_{C, P_n} < R_{\max}$  innerhalb sternförmiger Netze wird die Verifikationsklausel `valStarRes` zum Constraint-Rule-File hinzugefügt und steht damit für spätere Verifikationsanfragen zur Verfügung:

```

valStarRes(NetID, PinID, Rmax) :-
  R>Rmax, net(_, NetID), netLayout(NetID, L),
  topologyClass(_, L, star(C)),
  netPin(NetID, PinID), coordinate(PinID, P),
  resistance(_, L, C, P, R).

```

Die innerhalb der Klausel zu verwendenden Werkzeuge sind in diesem Beispiel nicht spezifiziert und unter Verwendung des in Abschnitt 4.2 vorgestellten Unifikationsmechanismus automatisch aufgelöst. Die Verifikationsaufgabe ist somit formal unabhängig von den eingesetzten Werkzeugen.

Mit Hilfe der Verifikationsklausel `valStarRes` kann die Verifikation z.B. für  $R_{\max} = 5\Omega$  mittels der Anfrage `valStarRes(N, P, 5)` durchgeführt werden. Das CES ermittelt daraufhin alle resultierenden Kombinationen von Netz-ID  $N$  und Pin-ID  $P$ , für die diese erfüllt ist und somit die Ursache der fehlgeschlagenen Verifikation darstellen.

Das zweite Verifikationsbeispiel soll alle Widerstandspaare eines sternförmigen Netzes ermitteln, bei denen sich die Differenz  $\Delta R$  der Widerstände mindestens um einen definierten Wert  $|R_{C,P_n} - R_{C,P_n}| > \Delta R_{\max}$  unterscheiden. Die folgende Verifikationsklausel wird ebenfalls zum Constraint-Rule-File hinzugefügt:

```

valResDiff(NetID, PinID1, PinID2, DeltaR) :-
  isnot(P1, P2), abs(R1-R2)>DeltaR,
  netLayout(NetID, L),
  topologyClass(_, L, star(C)),
  netPin(NetID, PinID1), coordinate(PinID1, P1),
  netPin(NetID, PinID2), coordinate(PinID2, P2),
  resistance(_, Layout, C, P1, R1),
  resistance(_, Layout, C, P2, R2).

```

Zur Ermittlung aller Widerstandspaare in einem Netz mit  $\Delta R_{\max} = 2\Omega$  dient die Verifikationsanfrage `valResDiff(N, P1, P2, 2)`. Entsprechend des ersten Verifikationsbeispiels werden alle fehlgeschlagenen Kombinationen von Netz-ID  $N$  und die zu diesem Netz gehörenden Pins  $P1$  und  $P2$  zurückgegeben, die diese Entwurfsrandbedingung nicht erfüllen.

## 6 Zusammenfassung

Das in diesem Artikel vorgestellte Constraint-Engineering-System (CES) bietet erstmals die Möglichkeit einer konsistenten und entwurfswerkzeugübergreifenden Darstellung von Entwurfsrandbedingungen sowie von einfachen und komplexen Verifikationsaufgaben. Die vorhandenen Verifikations- und Simulationswerkzeuge lassen sich mit Hilfe dieses Systems leicht zu flexiblen Metaverifikationswerkzeugen kombinieren. Metaverifikationswerkzeuge sind somit in der Lage, Verifikationsaufgaben von einer Komplexität bearbeiten zu können, die die Fähigkeiten der Einzelverifikationswerkzeuge weit übersteigt. Das CES repräsentiert ein Fundament für die Automatisierung von Verifikationsaufgaben, welche bisher manuell und allein durch die Expertise von IC-Entwicklern abgesichert wurden.

Neben der Integration von weiteren Einzelverifikationswerkzeugen, liegt ein Schwerpunkt der zukünftigen Entwicklungen in der Optimierung der Abarbeitung von Subverifikationsaufgaben, durch z.B. Reihenfolgeoptimierung und parallele Bearbeitung unabhängiger Aufgaben. Ein zweiter Entwicklungsschwerpunkt basiert auf der Fähigkeit des CES, komplexe Parameter-Lösungsräume berechnen zu können. Im Zusammenhang mit einer Sensitivitätsanalyse ermöglicht dies die Entwicklung neuartiger Constraint-geführter Entwurfsalgorithmen.

## 7 Danksagung

Die vorliegende Arbeit wurde in Teilen gefördert vom BMBF-Verbundvorhaben LEONIDAS+ (Förderkennzeichen: 01M3074).

## Literatur

- [1] *Kapitel Unification Theory*. In: BAADER, F.; SNYDER, W.: *Handbook of Automated Reasoning*. Bd. 1. Elsevier Science B.V., Amsterdam, 2001. – ISBN 0-2621-8223-8, S. 445–533
- [2] BARTÁK, R.: Constraint Programming: In Pursuit of the Holy Grail. In: *Proc. of Week of Doctoral Students (WDS99), Part IV* (1999), S. 555–564
- [3] *Kapitel Gröbner Bases: An algorithmic method in polynomial ideal theory*. In: BOSE, N.K.: *Multidimensional Systems Theory and Applications*. Second Edition. Kluwer Academic Publishers, 2003. – ISBN 1-4020-1623-9, S. 89–128
- [4] CHANG, Henry ; CHARBON, Edoardo ; CHOUDHURY, Umakanta ; DEMIR, Alper ; FELT, Eric ; LIU, Edward ; MALAVASI, Enrico ; SANGIOVANNI-VINCENTELLI, Alberto ; VASSILIOU, Iasson: *A Top-Down, Constraint-Driven Design Methodology for Analog Integrated Circuits*. Norwell, MA, U.S.A. : Springer Verlag, 1999. – ISBN 0-7923-9794-0
- [5] COHEN, J.: Constraint logic programming languages. In: *Commun. ACM* 33 (1990), Nr. 7, S. 52–68. – ISSN 0001-0782
- [6] DINCIBAS, M. ; HENTENRYCK, P. van ; SIMONIS, H. ; AGGOUN, A. ; GRAF, T. ; BERTHIER, F.: The Constraint Logic Programming Language CHIP. In: *Proc. Int. Conf. on Fifth Generation Computer Systems* (1988), S. 693–702
- [7] HORN, A.: On sentences which are true of direct unions of algebras. In: *J. Symbolic Logic* 16 (1951), S. 14–21
- [8] JAFFAR, J. ; MICHAYLOV, S. ; STUCKEY, P.J. ; YAP, R.H.C.: The CLP( $\mathbb{R}$ ) Language and System. In: *ACM Trans. on Programming Languages and Systems* 14 (1992), July, Nr. 3, S. 339–395
- [9] LEENAERTS, D. ; GIELEN, G. ; RUTENBAR, R.A.: CAD solutions and outstanding challenges for mixed-signal and RFIC design. In: *Proc. Int. Conf. on Computer-Aided Design (ICCAD)*. Piscataway, NJ, USA : IEEE Press, 2001. – ISBN 0-7803-7249-2, S. 270–277
- [10] LY, T.A. ; GIRCZYC, E.F.: Constraint Propagation and Design Interaction in an Object-Oriented IC Design Environment. In: *Proc. Custom Integr. Circuits Conf. (CICC)*, 1988, S. 2.3.1–2.3.4
- [11] MALAVASI, E. ; CHARBON, E.: Constraint Transformation for IC Physical Design. In: *IEEE Trans. Semiconductor Manufacturing* 12 (1999), Nr. 4, S. 386–395
- [12] MALAVASI, E. ; CHARBON, E. ; ARSINTESCU, B. ; KAO, W.: A Constraint Management System for IC Physical Design. In: *Proc. XI. Brazilian Symp. on Integr. Circuit Design*, 1998, S. 240–243
- [13] MALAVASI, E. ; CHARBON, E. ; FELT, E. ; SANGIOVANNI-VINCENTELLI, A.: Automation of IC Layout with Analog Constraints. In: *IEEE Trans. CAD of Integr. Circuits and Systems* 15 (1996), Nr. 8, S. 923–941
- [14] RUTENBAR, R.A. ; COHN, J.M.: Layout Tools for Analog ICs and Mixed-Signal SoCs: A Survey. In: *Proc. Int. Symp. on Physical Design (ISPD)*, 2000, S. 76–83
- [15] VODA, P.: The Constraint Language Trilogy: Semantics and Computations / Complete Logic Systems. 741 Blueridge Ave, North Vancouver BC, V7R 2J5, Canada, 1988. – Forschungsbericht
- [16] WALLACE, M. ; NOVELLO, S. ; SCHIMPF, J.: ECLiPSe: A Platform for Constraint Logic Programming / IC-Parc, Imperial College. London, U.K., 1997. – Forschungsbericht