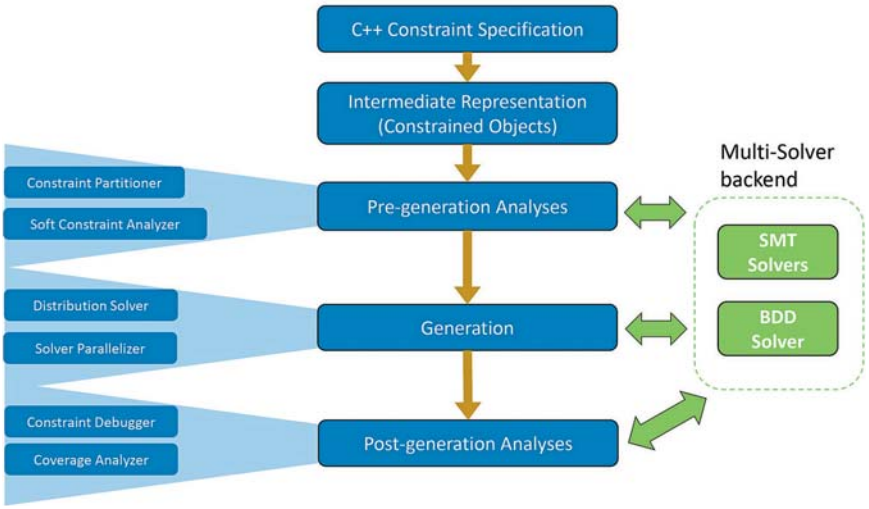
Dem Zufall überlassen

Constraint-basierte Randomisierung. Nach dem Zufallsprinzip wählen Design- und Verifikationsprogramme wie SystemVerilog mögliche Parameter für eine elektronische Schaltung aus und prüfen, ob sie mit den Constraints des Anwenders übereinstimmen. Im digitalen Bereich ist dies problemlos möglich. Das komplexe Verhalten analoger Schaltungen erfordert hingegen Anpassungen in der Programmiersprache und den Bibliotheken der Testumgebung.



1 | Verifikationsablauf: Crave analysiert Contraints, stellt die Solver für die Simulation und überprüft am Ende, ob alle Bedingungen erfüllt wurden

Die Designsprache SystemC AMS ermöglicht es, komplexe elektronische Systeme, etwa für das autonome Fahren oder IoT, auf verschiedenen Abstraktionsebenen zu modellieren. Die Anwendungsfälle reichen von schnellen virtuellen Prototypen für die frühe Softwareentwicklung und das Algorithmendesign bis hin zu Register-Transfer-Level-Implementierungen, die mit Hardwarebeschreibungssprachen wie VHDL und SystemVerilog vergleichbar sind. Der AMS-Teil von SystemC setzt sich aus Berechnungsmodellen wie Timed Data Flow und Electrical Linear Networks zusammen, die das analoge Verhalten nachbilden.

Randomisierung für das AMS-Design

Bei digitalen Schaltungen kommt für die funktionale Verifikation, etwa mit SystemVerilog, seit Langem die Constraint-basierte Randomisierung zum Einsatz. Dabei beschreibt der Anwender mögliche Eingabestimuli, der Constraint-Solver variiert die Parameter nach dem Zufallsprinzip und erzeugt somit gültige Szenarien. Zusätzlich werden Verifikationsmethoden wie Universal Verification Methodology (UVM) verwendet. Für SystemC sind externe Bibliotheken, wie SCV und Crave (siehe **Wissenskasten**), zur Constraint-Lösung erforderlich, wenn UVM implementiert werden soll. In der Praxis beschränken sich Contraint-basierte Randomisierungstechniken auf die Verifikation von digitalen Schaltungen und zwar aus folgenden Gründen:

- Es sind genaue Verhaltensmodelle von analogen Systemteilen erforderlich, da Transistornetzlisten zu langsam sind, um ein Gesamtsystem zu simulieren.
- Die Korrektheit des analogen Verhaltens automatisch zu prüfen, ist aufgrund des kontinuierlichen Zeitverhaltens der Signale schwieriger als im digitalen Bereich. Insbesondere ein einfacher Vergleich auf Äquivalenz ist nicht möglich.
- Constraints in SystemVerilog lassen sich während des Aufbaus des Designs Under Test (DUT) in der Elaborationsphase nicht lösen. Daher können sie die DUT-Parameter wie Verstärkung, Cut-Off-Frequenzen oder Impedanzen nicht verändern. Analoge Parameter unterliegen Schwankungen aufgrund von Produktionstoleranzen, die anders als bei digitalen Schaltungen nicht zu abstrahieren sind und in der Regel direkt die Funktionalität beeinflussen. Die Toleranzen in Mixed-Signal-Systemen müssen somit anderweitig berücksichtigt werden.

Um die Constraint-basierte Randomisierung auch für AMS-Designs und deren Verifikation verwenden zu können, wird in einem Paper (siehe **Online-Service**)

FAZIT

Constraints für das AMS-Design. Der Einsatz von Constraint-basierter Randomisierung im Schaltungsdesign ist nicht nur auf die digitale Verifikation beschränkt. In SystemC AMS lassen sich Constraints auch für analoge Schaltungen einsetzen. Die Crave-Bibliothek erweitert die Simulationsumgebung um Fließkommawerte und Solver für das analoge Verhalten. Für das AMS-Design eignen sich vor allem Parameter- und Runtime-Constraints, die die Rahmenbedingungen für die Parameterwahl vor der Simulation beziehungsweise für die Simulation selbst festlegen. Soll die Simulationszeit verkürzt werden, bietet sich ein lizenzfreies Framework für die Parallelisierung der Simulationen an.

Bild: Coseda/DFKI

KONTAKT

COSEDA Technologies GmbH,
Königsbrücker Straße 124,
01099 Dresden,
Tel. 0351 321 490 00,
E-Mail info@coseda-tech.com,
www.coseda-tech.com

von Coseda Technologies und dem Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) vorgeschlagen, Parameter- und Runtime-Constraints einzuführen. Parameter-Constraints verändern die Struktur des DUT und modellieren komplexe Parametervariationen. So lässt sich beispielsweise festlegen, dass die Simulationstemperatur zwischen 0 und 80 °C liegen soll, alle Elemente während eines Simulationslaufs jedoch nur eine Differenz von 10 °C aufweisen dürfen.

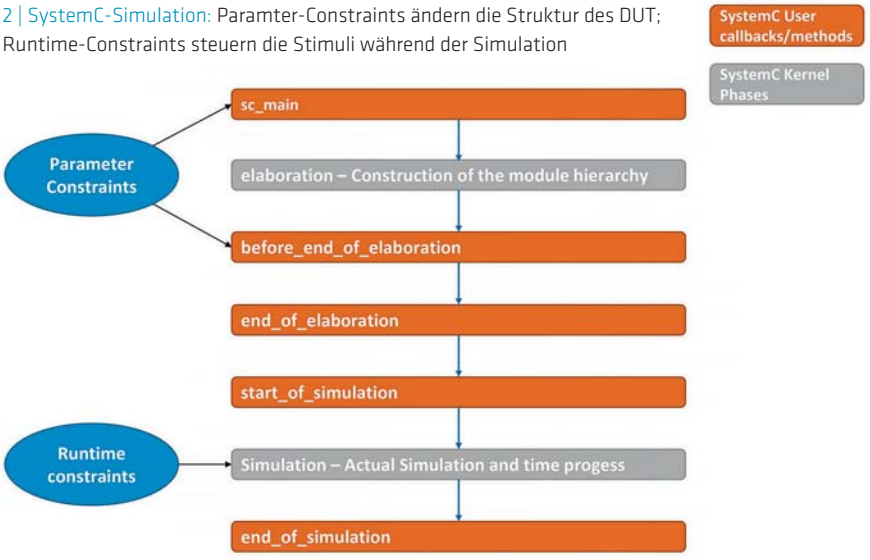
Runtime-Constraints steuern die Stimuli für das DUT während der Simulation. Sie erzeugen etwa zufällige Stimuli, ähnlich wie bei der UVM, und entsprechen dem Vorgang in SystemVerilog-Testumgebungen für die digitale Verifikation. Der Unterschied besteht darin, dass Fließkommawerte zur Beschreibung analoger Stimuli unterstützt werden müssen.

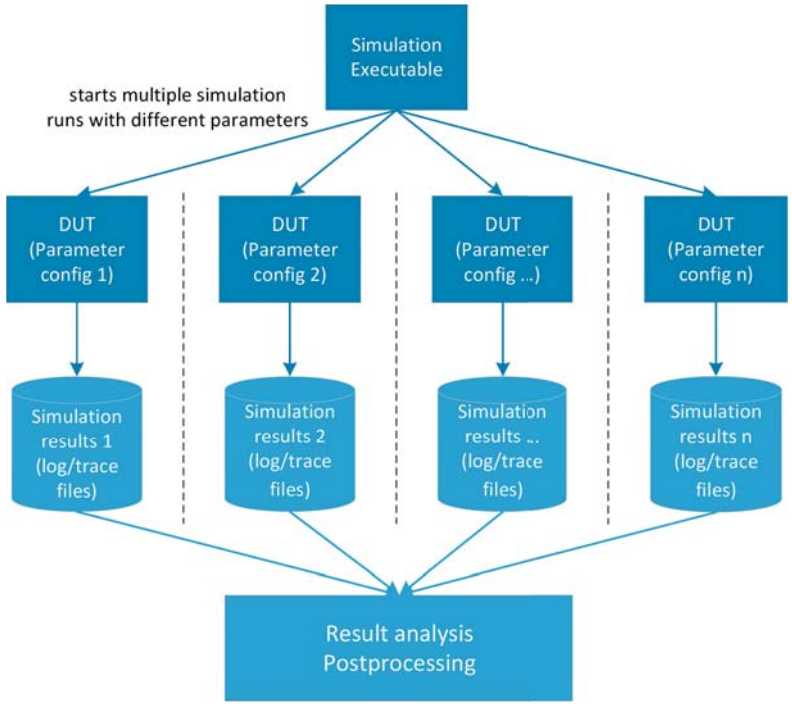
Constraints in SystemC AMS

Der Schwerpunkt der SystemC-AMS-Sprache liegt in der Erstellung von virtuellen Prototypen, die in späteren Phasen des Designprozesses wiederverwendet werden können. Dabei ist es eine wichtige Aufgabe, die Systemarchitektur zu untersuchen, denn daraus ergibt sich die Spezifikation für den weiteren Entwicklungsprozess. SystemC AMS kann darüber hinaus auch Verifikationsumgebungen mit Tests auf Systemebene erstellen. **Bild 2** zeigt, wie Parameter- und Runtime-Constraints in den SystemC-Simulationsablauf integriert werden. Die Parameter-Constraints kann man vor der Erstellung des DUT in der `sc_main` oder während des Callbacks `before_end_of_elaboration` festlegen, was immer noch eine Änderung der SystemC-Designhierarchie erlaubt (ein Beispiel in der **Codebox**).

Im Gegensatz zu Parameter-Constraints werden Runtime-Constraints während der Simulationsphase gelöst, ähnlich wie beim Randomisieren von Klassenobjekten in SystemVerilog. Immer wenn ein neuer Stimulus am DUT anliegt, wird die Randomisierung eines Stimulusobjekts ausgelöst, und es werden neue Werte zugewiesen.

2 | SystemC-Simulation: Paramter-Constraints ändern die Struktur des DUT; Runtime-Constraints steuern die Stimuli während der Simulation



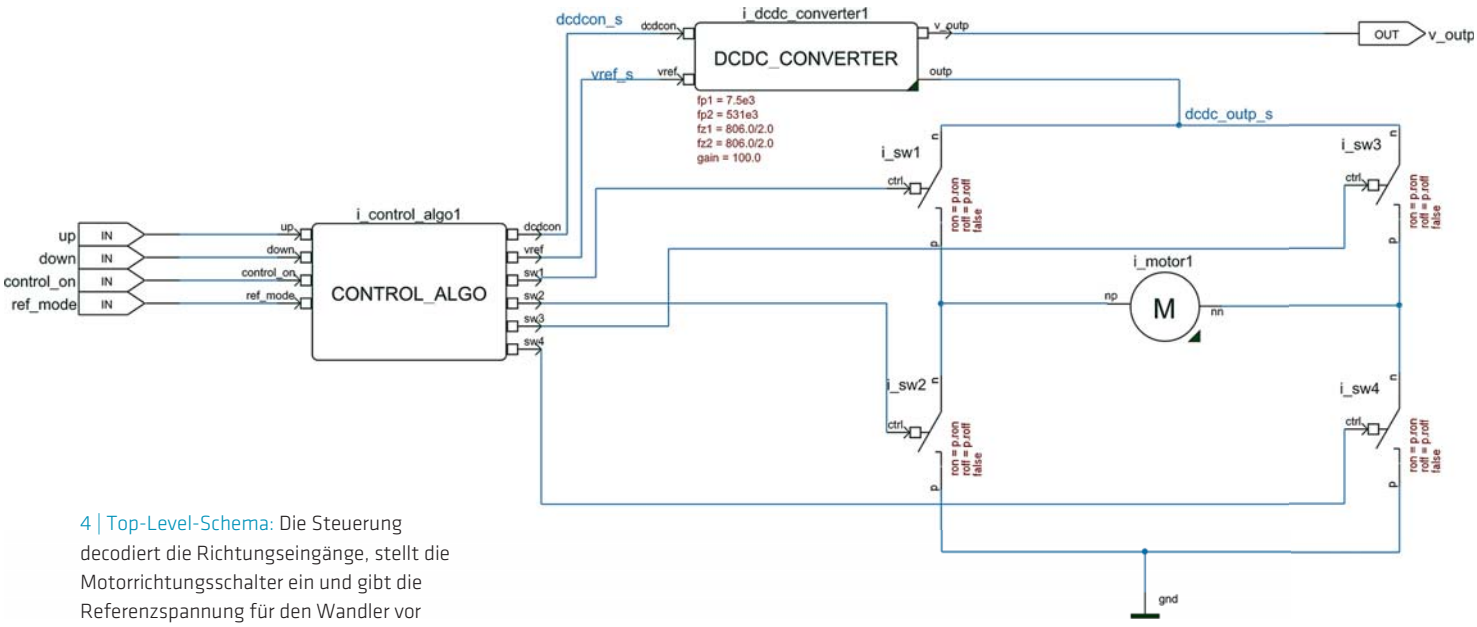


3 | Parallelisierung: Mehrere Simulationen lassen sich parallel mit verschiedenen Parameterkonfigurationen durchführen

Paralleles Simulations-Framework

Da die Prozesse im ereignisgesteuerten Kernel von SystemC eng gekoppelt sind, ist es schwierig, die Simulationszeit durch Parallelisierung zu verkürzen. Die Alternative besteht darin, mehrere Simulationen parallel für verschiedene Parameter oder Eingangsstimuli durchzuführen. Dafür haben Coseda und DFKI ein paralleles Simulations-Framework für verschiedene Testfälle entwickelt. In **Bild 3** ist dargestellt, wie die Simulationen über einen Fork-Mechanismus auf mehrere Anwendungsprozesse verteilt werden. Sie laufen jeweils mit einem eigenen SystemC-Kernel und verwenden einen anderen Startwert (Seed) für die Zufallszahlen. Durch verschiedene Seeds werden auch unterschiedliche zufällige Parameter oder Stimulisets erzeugt, die zu anderen Ergebnissen führen.

Bilder: Coseda/DFKI



4 | Top-Level-Schema: Die Steuerung decodiert die Richtungseingänge, stellt die Motorrichtungsschalter ein und gibt die Referenzspannung für den Wandler vor

Beispielschaltung

Um die Methodik zu demonstrieren, soll eine DC/DC-Wandlerschaltung betrachtet werden. Crave erstellt für dieses Beispiel das DUT mit verschiedenen Parameterkonfigurationen, UVM-SystemC dient der Verifikation und das parallele Simulationsframework beschleunigt die Simulation. Die schematische Darstellung auf oberster Ebene ist in **Bild 4** zu sehen. Die Anwendung besteht aus einer digitalen Steuerung, einem Motor und einem DC/DC-Wandler.

Die Steuerung decodiert die Richtungseingänge, stellt die Motorrichtungsschalter I_{Sw1} bis I_{Sw4} und damit die Drehrichtung des Motors ein. Zusätzlich gibt sie die Referenzspannung für den Wandler vor und schaltet diesen ein. Die DC/DC-Wandlerschaltung **(Bild 5)** basiert auf dem Prinzip eines Abwärtswandlers. Eine Sägezahngeneratorquelle erzeugt ein PWM-Signal, das eine Spannungsquelle ein- und ausschaltet. Wenn diese angeschlossen ist, steigt der Strom durch die Spule und erzeugt eine Gegenspannung. Um die Ausgangsspannung zu steuern, wird diese Spannung ständig gemessen, gefiltert und mit dem gewünschten Wert V_{Ref} verglichen.

Es wurde eine UVM-SystemC-Testumgebung erstellt, welche zufallsbasiert gültige Eingaben für die Steuerung generiert und verschiedene Spannungen einstellt. Die Inputstimuli bestimmen die Richtung des Motors und schalten das System ein und aus. Mithilfe der Enumerationstypen aus der Crave-Bibliothek werden die möglichen Eingangsspannungen ausgewählt. Zusätzlich überprüft ein UVM-Monitor die Stabilität der erzeugten Ausgangsspannung. Ein Mixed-Signal-Checker-Framework achtet auf eine korrekte Anstiegszeit und ausreichende Stabilität. Die darin enthaltenen Regressionen testen auch das Verhalten analoger Signalformen.

Eine Besonderheit der Schaltung ist, dass bei einer Richtungsänderung des Motors eine Spannungsspitze am Ausgang (V_{Outp}) auftritt. Ihre Höhe hängt vom Lastwiderstand $I_{Sca\text{-}R1}$ ab. Er beeinflusst aber auch die Zeit bis zum Erreichen von V_{Ref}. **Bild 6** veranschaulicht dieses

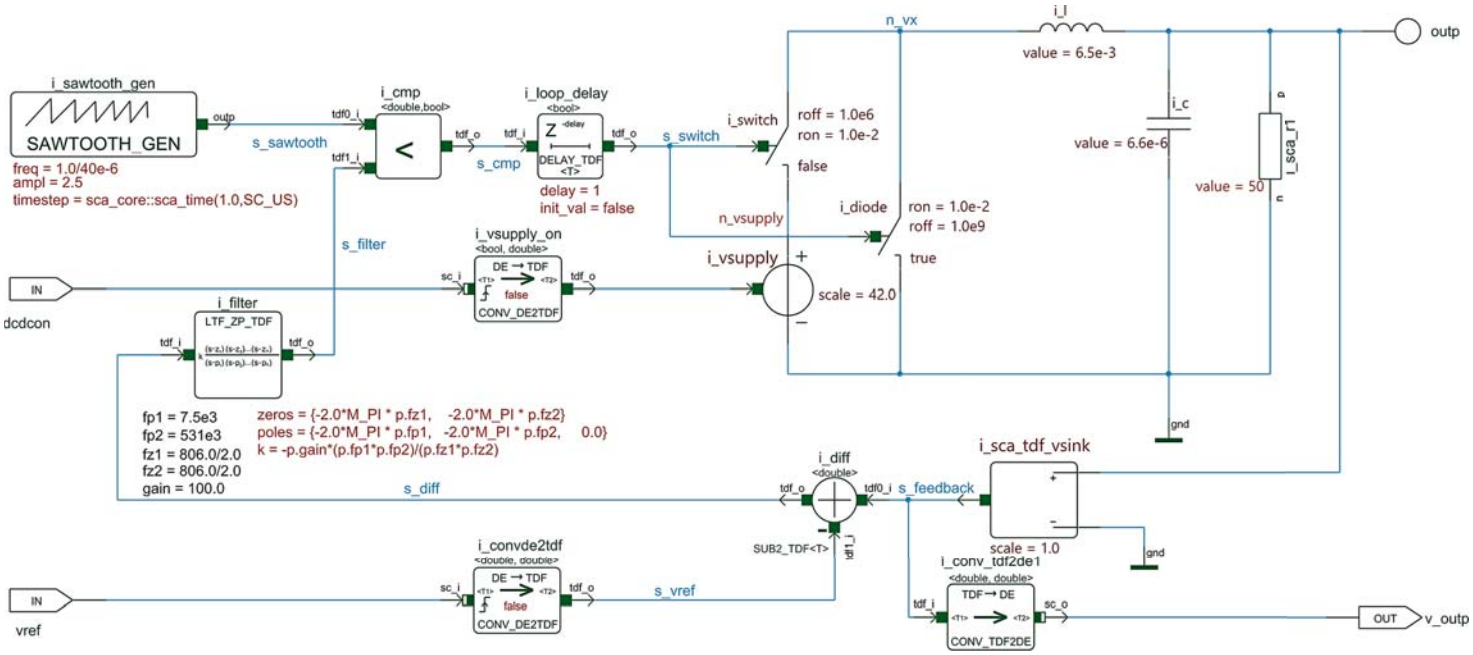
WISSENSWERT

Constraints in der Crave-Architektur. Crave ist eine Open-Source-Umgebung für die Constraint-basierte Verifikation mit SystemC. Im Vergleich zu SCV bietet sie einige Vorteile, wie eine bessere API für die Spezifikation und Handhabung von Constraints sowie ein automatisches Constraint-Debugging. Der wichtigste Unterschied besteht darin, dass Crave moderne SMT-Solver (Satisfiability Modulo Theories) enthält, die sich besser skalieren lassen als binäre Entscheidungsdiagramme (Binary Decision Diagrams, BDDs). Außerdem werden unter anderem Distributions- und Soft-Constraints sowie Constraint-Partitionierung unterstützt. Die Erweiterung um Fließkommawerte adressiert analoge und Mixed-Signal-Systeme.

Die Crave-Architektur ist in **Bild 1** dargestellt. Zunächst müssen die Constraints spezifiziert in eine Zwischendarstellung umgewandelt werden. Ein Partitioniermodul prüft sie auf gegenseitige Abhängigkeiten und trennt sie gegebenenfalls in unabhängige Sets, um die Performance zu erhöhen. Als nächstes analysiert das Programm die Eingaben: Wenn es sich um Hard-Constraints handelt, müssen diese immer erfüllt sein. Soft-Constraints können bei Konflikten mit Hard-Constraints ignoriert werden. Ihre wichtigste Anwendung ist es, das Normalverhalten des DUT zu beschreiben, das sich später mithilfe von Klassenspezialisierungen anpassen lässt. Den Soft-Constraints wird eine eindeutige Priorität zugewiesen, entsprechend der Reihenfolge der Erstellung. Wenn nämlich ein Widerspruch entsteht, hebt sich der niedrigstpriorisierte Constraint auf.

Während der Generierungsphase prüft Crave, ob eine benutzerdefinierte Abweichung vorliegt. Die Verteilungen spiegeln die Deckung des Lösungsraums wider. Anschließend führt eine Multithreading-Umgebung einen BDD-und einen SMT-basierten Constraint-Löser parallel aus. Die Ergebnisse des schnellsten Solvers zählen als Lösungsraum. Im letzten Schritt kontrolliert der Constraint-Debugger, ob alle Bedingungen erfüllt sind, und der Coverage-Analyzer, ob die Deckungsziele erreicht wurden.

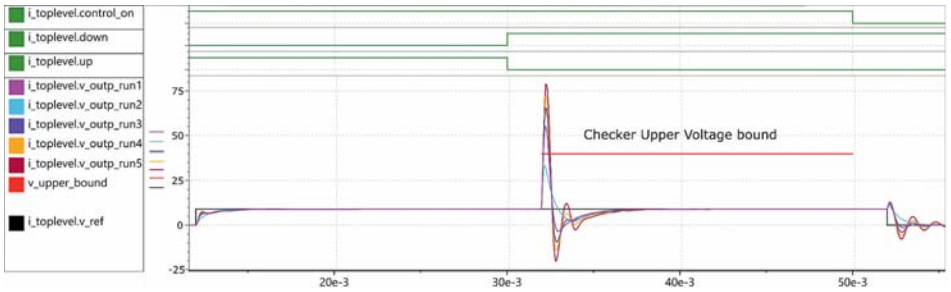
Bild: Coseda/DFKI



5 | Abwärtswandler: Die Sägezahngeneratorquelle (links oben) erzeugt ein PWM-Signal, das die Spannungsquelle $I_{Vsupply}$ ein- und ausschaltet

Verhalten bei Laständerung. Parameter-Constraints legen den Bereich des Lastwiderstands auf 10 bis 100 Ω fest. Im Checker-Framework lässt sich die Spannung auf einen Maximalwert begrenzen. Dadurch findet man automatisch zulässige Werte für $I_{Sca\text{-}R1}$.

Die UVM-Sequenz wurde auf dem parallelen Simulations-Framework mit mehreren Modi und für 100 verschiedene DUT-Parametersätze ausgeführt. Das dauerte etwa 68 Sekunden auf einem Standardvierkern-Laptop mit acht Threads und 3 GHz Taktfrequenz. Mit nur einem Kern dauerte es 255 Sekunden, das entspricht einer Beschleunigung um das 3,75-Fache, liegt also nah bei der maximal möglichen Beschleunigung von 4. mey



6 | Ausgangsspannung für verschiedene Parametersätze: Bei einer Richtungsänderung des Motors tritt eine Spannungsspitze auf, deren Höhe vom Lastwiderstand abhängt

Online-Service

Das Paper ‚Using Constraints for SystemC AMS Design and Verification' wurde auf der DVCon Europe 2018 veröffentlicht und steht online zum Herunterladen bereit. Die diesjährige DVCon findet am 29. und 30. Oktober in München statt.

Autoren

Thilo Vörtler ist Senior Development Engineer und Karsten Einwich ist CEO bei Coseda Technologies. Muhammad Hassan und Daniel Große sind Wissenschaftler des Deutschen Forschungszentrums für Künstliche Intelligenz (DFKI).

www.elektronik-informationen.de/85013

Bilder: Coseda/DFKI

CODEBOX

Parameter-Constraints festlegen. In diesem Codebeispiel werden Constraints verwendet, um die zulässigen Widerstandswerte in einer SystemC-AMS-Simulation zu modellieren und deren Korrelation zu definieren. Die Widerstände R1 und R2 sind auf einen bestimmten Wertebereich beschränkt, und `R_overall` begrenzt die Summe der Widerstandswerte. In jedem Durchlauf entsteht aus dem Ergebnis des Constraint-Lösers ein neuer Parametersatz, der das Verhalten des DUT beeinflusst. Dabei sind auch Fließkommawerte zu berücksichtigen, da die meisten Parameter in AMS-Systemen auf Analogwerte angewiesen sind.

```
class parameter_constraints : public crv_sequence_item {
crv_variable<double> R1; //Resistor 1
crv_variable<double> R2; //Resistor 2
crv_constraint  R1_val{ 1 < R1 (), R1 () < 3.3  }; // 1KOhm
to 3.3Kohm
crv_constraint R2_val{ 0.5 < R2(), R2() < 4.7 }; // 0.5Kohm
to 4.7Kohm
  crv_constraint R_overall{ R2()+R1() < 4.7e3 }; // Restrict
sum
parameter_constraints(crv_object_name) {}
};
```