

# Systemverifikation im Designprozess heterogener mikroelektronischer Systeme

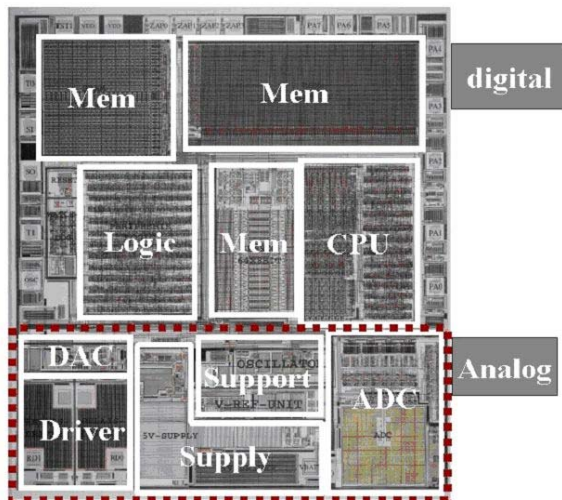
M. Pistauer, S. Kajtazovic CISC Semiconductor Design+Consulting GmbH, Klagenfurt, Austria, office@cisc.at  
 I. Pill, Ch. Steger Institut für Technische Informatik, Technische Universität Graz, Austria, steger@iti.tu-graz.ac.at

## Kurzfassung

In diesem Beitrag wird eine einfach zu handhabende Methodik zur heterogenen System-Simulation vorgestellt. Die steigende Komplexität eingebetteter Systeme und der Konkurrenzdruck für die Hersteller fordert kostengünstigere und schnellere Entwicklungsmethoden (Hardware/Software Codesign) für heterogene Systeme. Die einzelnen Entwurfsschritte werden durch eine Cosimulation verifiziert. An Hand eines Beispiels einer Kopplung der Tools bzw. Sprachen Simulink(Matlab), Modelsim(VHDL) und SystemC wird die Methodik vorgestellt.

## 1 Einführung

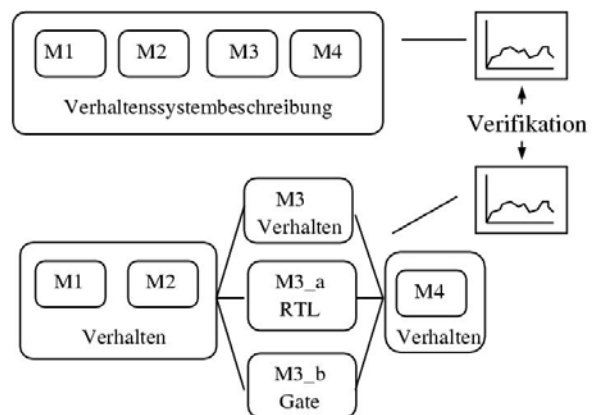
Typische Mixed-Signal-IC Designs vereinen aufgrund ständig steigender Integrationsdichten und Verfeinerung vorhandener CMOS-Prozesse immer mehr verschiedene Funktionseinheiten auf einem Chip. Die Funktion des Gesamtsystems wird oft erst durch zusätzliche Softwarekomponenten realisiert. Abbildung 1 stellt z.B. einen ASIC im Automotiv-Bereich dar, der verschiedene Speicher, Logik, einen CPU-Kern und ein Signalverarbeitungsfrontend mit einem Aktuator vereint.



**Bild 1** Layout Mixed-Signal IC

Um die Gesamtfunktionalität des Systems garantieren zu können bedarf es Methodiken das Gesamtsystem durchgängig über alle Abstraktionsebenen (Verhalten, Register-Transfer Ebene(RTL), Gate-Level) unter Einbindung aller Komponenten (inkl. Mechanik usw.) simulieren und verifizieren zu können (Abb. 2). Die Systemmodellierung erfolgt dabei grundsätzlich entweder in einer einheitlichen Umgebung, oder

getrennt mit verschiedenen Tools und Sprachen [3], [8]. Eine homogene Umgebung bietet zwar den Vorteil der einfacheren Handhabung, birgt jedoch den Nachteil der Inflexibilität und des beschränkten Funktionsumfangs in den einzelnen Domänen.



**Bild 2** Moduldesign und Verifikation auf verschiedenen Abstraktionsebenen

Die Verwendung mehrerer Tools bedeutet jedoch einen höheren Koordinationsaufwand, da entweder alle Teil-Module zur Gesamtsimulation in eine Sprache zu übersetzen oder verschiedene Simulatoren zu koppeln sind [8], [3].

Inhalt dieses Beitrages ist eine Methodik basierend auf Simulatorkopplungen, bei der durch geeignete Systematiken der Koordinationsoverhead gering gehalten wird. Es wurden speziell Simulink (Matlab) [5] VHDL/Modelsim [1] und SystemC [10] in einer UNIX/Linux Umgebung betrachtet.

## 1.1 HW/SW Cosimulation

Da eine formale Verifikation von heterogenen Systemen nicht möglich ist, wird die Verifikation durch eine Cosimulation durchgeführt, also eine gemeinsame Simulation von Hardware- und Softwarekomponenten bzw. analogen/digitalen Komponenten. Natürlich kann man in diesem Fall die Möglichkeit wählen, eine gemeinsame Sprache zu finden, die in *einem* Simulator abgearbeitet werden kann. Meist ist diese Methode mit Blickrichtung auf die eigentliche Implementierung aber zu ineffizient (Informationsverlust). Daher wird in vielen Fällen eine Aufteilung der Simulation auf verschiedene Simulatoren vorgenommen, wobei Kommunikation und Synchronisation einen wichtigen Faktoren darstellt.

Edwards et al. [7] gibt eine kurze Übersicht über verschiedene Cosimulationen. So wird z.B. eine Kopplung Hardware-Simulator - Softwareprozess angeführt, bei der die Synchronisation durch die vom Betriebssystem zur Verfügung gestellten Interprozesskommunikation (z.B. RPC) durchgeführt wird. Es gibt in der Literatur ([2], [9], ...) eine Vielzahl von implementierten Systemen zur Cosimulation, welche meist für eine bestimmte Anwendungsdomäne (Prozessor-Design, Steuerungssysteme, Signalverarbeitung, ...) entwickelt wurden.

Sehr viele in der Literatur gezeigten Beispiele laufen auf eine kombinierte Simulation der Sprachen VHDL und C hinaus. VHDL ist für die Beschreibung von Hardwarekomponenten sehr effizient und es sind verschiedene Simulatoren verfügbar. VHDL ist auch durch die Möglichkeit verschiedene Abstraktionsebenen einzuführen beliebt. C ist die Standard-Programmiersprache und auf jeder Computerplattform verfügbar.

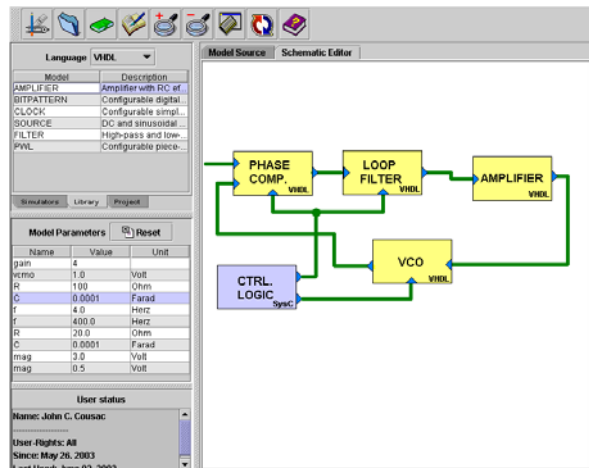
## 2 Cosimulation im System-design

Die mikroelektronischen Systeme werden immer komplexer. Sie beinhalten sowohl digitale als auch analoge Komponenten. Eine gemeinsame Simulation von analogen und digitalen Komponenten gleichzeitig ist erwünscht. Jedoch ist die Erstellung von Simulatorkopplungen komplex und erfordert gute Kenntnisse über einzelne Simulatoren sowie vorhandene Schnittstellen, die von einzelnen Simulatoren unterstützt werden.

Um das Systemdesign effizienter zu machen, kann die Erstellung von Simulator-Schnittstellen automatisiert werden. Die Methode der Simulatorkopplung, die hier vorgestellt ist, wird im Rahmen eines Projekts benutzt, in dem eine Methode entwickelt wird, welche eine

Automatisierung der Erstellung von Simulator-Schnittstellen ermöglicht. Es handelt sich um eine Methode, die sowohl das *Top-Down* System-Design als auch die *Bottom-Up* Validierung unterstützt. Das Projekt umschließt:

- einen *Schematic-Editor*, mit dem die Partitionierung durchgeführt wird (Abb. 3)
- den *Automatic Simulation Framework Generator*, mit dem die Simulationsumgebung generiert wird



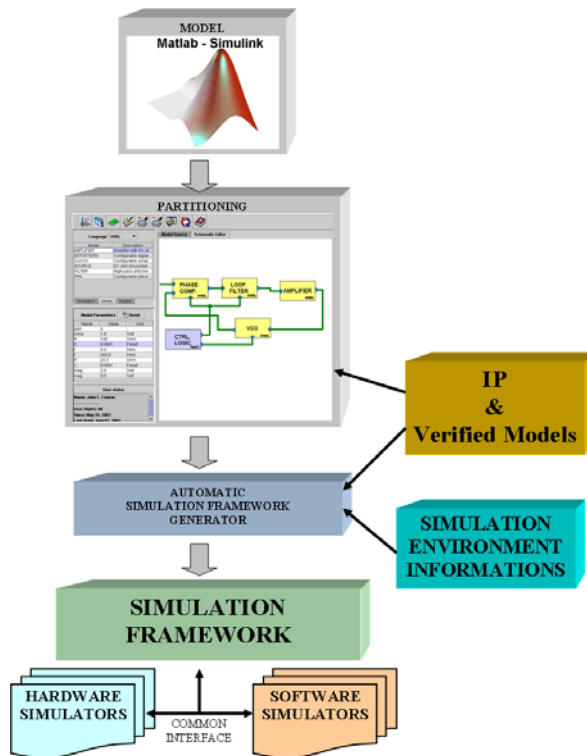
**Bild 3** Partitionierung mit dem *System Design Editor*

## 2.1 Generierung einer Cosimulationsumgebung

Abbildung 4 zeigt den Datafluß der Generierung von einer Simulationsumgebung anhand eines Matlab/Simulink Modells und vorgenommener Partitionierung.

Als *System Top-Level Design - Entry* wird hier das Matlab/Simulink Modell benutzt. Das Modell wird durch Partitionierung in die bestimmten VHDL bzw. SystemC Modelle transferiert.

Um die Transformation von Spezifikationen und Systemcharakteristik zwischen zwei Design-Levels fehlerfrei zu realisieren, werden die Top-Level Beschreibungen durch die vorhandenen, *verifizierten* Modelle in bestimmter Sprache ersetzt. Das Matlab/Simulink Modell wird mit gleicher Methode durch die entsprechenden VHDL bzw. SystemC Modelle ersetzt. Ein umgekehrter Prozess ist ebenso möglich.



**Bild 4** Erstellung einer Simulationsumgebung

Anhand von erstellten Partitionierungsdaten, VHDL bzw. SystemC Modellen und vorhandenen Simulationsumgebung können dann die Interfaces zwischen den Modellen generiert werden. Dieser Prozess wird vollständig automatisch durchgeführt. Dabei werden die Modelle modifiziert, sodass die Daten zwischen einzelnen Simulatoren getauscht werden können.

Abhängig von dem zu simulierenden mikroelektronischen System und vorhandenen Simulatoren werden womöglich die *Peer-to-Peer*, aber auch die Client/Server Netzwerke von Simulatoren benutzt. Die Synchronisation von beteiligten Simulatoren spielt bei der Schnittstellenerstellung eine wichtige Rolle.

### 3 Cosimulationsumgebung

#### 3.1 Simulatorkopplungen

Für eine Simulator-Kopplung maßgeblich sind einerseits die zur Verfügung stehenden Schnittstellen der einzelnen Simulatoren und andererseits das Simulatorverhalten selbst. Beide Faktoren beeinflussen die Interaktionsmöglichkeiten wesentlich.

Alle drei Simulation-Engines (Matlab, Modelsim, SystemC) bieten eine linear fortlaufende Simulation. Die Berechnung eines einzelnen Time-Steps erfolgt jedoch unterschiedlich, wodurch die

Interaktionspunkte (Datenaustausch/Synchronisation) zwischen die einzelnen *Time-Steps* gelegt wurden und keine In-Step-Kommunikation vorgesehen ist (Abb. 5).

Dadurch sind innerhalb eines *Time-Steps* keine Rückkopplungen über einen externen Pfad möglich, worauf beim Design geachtet werden muß. Gegebenenfalls ist das Modul in zwei Submodule aufzuspalten, die während der Simulation mit dem externen Modul in Reihenfolge der Datenflußrichtung abgearbeitet werden.

Rückkopplungen, die eine *Algebraic Loop*<sup>1</sup> erzeugen, sind grundsätzlich zu vermeiden, da hier die Berechnung der Lösung eines Differentialgleichungssystems entspricht. Dies ist systematisch bedingt zwischen Simulatoren nicht möglich bzw. bieten auch die Simulatoren selbst meist keine Möglichkeiten diese exakt zu lösen<sup>2</sup>.

Ist dennoch eine *Algebraic Loop* zwischen den einzelnen Modulen vorhanden, muß zur Simulation ein Startmodul (inkl. Vorgabewerte der Eingänge) bestimmt werden. Die Berechnung erfolgt anschließend in Datenflußrichtung, wobei pro *Time-Step* nur ein Durchlauf der Schleife erfolgt und das Simulationsergebnis dementsprechend zu bewerten ist. Die bei Simulatorkopplungen verwendeten Verfahren der dezentralen Bindung und der zentralen Backplane [2] bieten jeweils Vor- und Nachteile [8], [3] die anwendungsspezifisch zu bewerten sind. Für den System-Designer wäre somit ein Hybrid, bei dem einfache Schnittstellen zu den Simulatoren existieren und diese dann je nach Bedarf und Anwendung untereinander oder über eine "zentrale Einheit" verbunden werden können, ein willkommener Kompromiss.

#### 3.2 Simulationsprinzip

Die für den Anwender wichtigen Eigenschaften der Flexibilität und einfachen Handhabungen können durch den Zugriff auf einen möglichst benutzerfreundlich strukturierten Quellcode der Simulation(s-Steuerung) erreicht werden, wodurch sich folgende Vorgaben ergeben:

- Synchroner Modelle; alle Modelle benutzen die selbe Taktrate oder ein ganzzahliges Vielfaches einer Grundtaktrate
- Das dem Anwender zugängliche Simulationssteuerungsprotokoll soll möglichst einfach sein.

<sup>1</sup> In Simulink werden Rückkopplungen die eine Schleife erzeugen als *Algebraic Loop* bezeichnet [5]

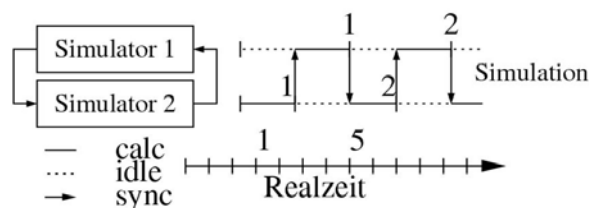
<sup>2</sup> Simulink selbst kann u.U. *Algebraic Loops* iterativ lösen, jedoch nicht über generierten Code

- Das Kommunikationsmedium soll zuverlässige, bidirektionale In-Order-<sup>3</sup> Verbindungen ermöglichen

### 3.2 Implementierung

Um sowohl die Datenkommunikation als auch die Steuerung der einzelnen Simulatoren einfach aber effektiv halten zu können wird ein datengetriebenes System verwendet. Die Synchronisation (Abb. 5) zwischen den einzelnen *Time-Steps* gliedert sich in zwei Schritte:

- erteilen der Ergebnisse des vorherigen *Time-Steps*
- warten auf alle notwendigen Eingangsdaten, danach wird die Berechnung automatisch fortgeführt



**Bild 5** Synchronisation / Datenaustausch

Werden für die Verbindungen ein zuverlässiges Medium und Verbindungsprotokoll gewählt, können sowohl das Simulationsprotokoll als auch Sende- und Empfangsmechanismen einfach gehalten werden, da somit einerseits kein Fehlermanagement im Protokoll notwendig ist und andererseits sowohl die Größe als auch die Struktur der versendeten *Datagrams* konstant und bereits vor der Kompilierung bekannt sind. So enthalten die versendeten Datenpakete nur einen Zeitstempel und die Signalwerte, jedoch keine zusätzlichen Informationen.

Für ein dezentrales System wäre aufgrund des datengetriebenen Modells und der In-Order-Kommunikation auch die Zeitinformation nicht zwingend notwendig, wird jedoch in Hinsicht auf eine mögliche zentrale Einheit verwendet, die somit für Steueraufgaben eine Zeitinformation über den Simulator erhält.

Eine notwendige Bedingung für diese Systematik sind allerdings die in den Vorgaben erwähnten speziellen Eigenschaften der Kommunikationskanäle zwischen den Simulatoren.

#### 3.3.1 Kommunikationskanäle

Für die Kommunikationskanäle bieten sich Stream-Sockets an, da sie eine sehr einfache Möglichkeit darstellen I/O Aufgaben durchzuführen und eine bereits etablierte, ausgereifte Technologie darstellen.

<sup>3</sup> Die versendeten Daten kommen in der richtigen Reihenfolge beim Empfänger an

Stream-Sockets bauen im Gegensatz zu Datagram-Sockets auf TCP auf, wodurch die erforderlichen zuverlässigen In-Order-Verbindungen ermöglicht werden. Darüber hinaus bieten Sockets im Gegensatz zu vielen anderen Kommunikationsmechanismen den Vorteil der integrierten Netzwerkfähigkeit und ermöglichen somit über eine entsprechende Adressierung<sup>4</sup> eine über ein Netzwerk verteilte Simulation.

#### 3.3.2 Simulatorschnittstellen

Die Interfaces selbst folgen zwar dem beschriebenen Gesamtsimulationsprinzip, sind jedoch auf die entsprechenden Möglichkeiten der anzubindenden Simulatoren angewiesen und unterscheiden sich somit in ihrem Konzept sehr stark. So wird z.B. bei Simulink die Simulations-Engine so angepasst, daß ein manueller Eingriff in den Ablauf möglich ist, wogegen bei Modelsim ein zusätzlicher Prozeß in die VHDL-Simulation eingebracht wird.

#### Simulink-Interface

Simulink bietet über den RealTime-Workshop bzw. dessen Target-Language-Compiler die Möglichkeit ein *Stand-Alone-Executable* der Simulation inklusive C-Quellcode für verschiedene Plattformen zu erzeugen. Zwar gibt es hier einige Einschränkungen gegenüber der Simulationsumgebung, so ist z.B. die Auflösung von *Algebraic loops* nicht möglich, allerdings bietet die Codegenerierung den Vorteil der höheren Performance und die Möglichkeit manuell Anpassungen an der Simulation vorzunehmen [4].

Zielplattform ist das *Generic-Real-Time* Format mit einer diskreten Zeitbasis, wobei eine Beschränkung auf *single-rate* Modelle erforderlich ist. Simulink sieht (zwecks Speicherreservierung für das Data-logging) eine fixe Endzeit vor, die entsprechend der Gesamtsimulationszeit zu wählen ist.

Durch anschließendes Ersetzen der Simulations-Hauptroutine kann der Fortschritt (Initialisierung, Step by Step Betrieb, Cleanup) manuell gesteuert werden. Dabei kann auf als externe In- und Outputs des Modells zugegriffen werden, womit alle Voraussetzungen für eine Simulatorkopplung erfüllt sind.

#### VHDL Interface

Der verwendete Simulator Modelsim bietet zwar keine integrierten Funktionen zur externen Steuerung, jedoch ist es über das FLI-Interface<sup>5</sup> möglich externen

<sup>4</sup> Sockets vom Typ AF\_INET werden per IP- und Portnummer adressiert, im Gegensatz zu lokalen UNIX-Sockets (AF\_UNIX) die über einen lokalen Pfad angesprochen werden.

<sup>5</sup> Foreign Language Interface; Bestandteil der SE Version von Modelsim

C-Code in die Simulation einzubinden. Ein genaues Studium des FLI-Interfaces [6] zeigt, daß es möglich ist, Prozesse mit bestimmten Prioritäten zu erzeugen und zu schedulen. Durch Verwendung eines Prozesses mit entsprechender Priorität können sowohl die noch gültigen Resultate des letzten *Time-Steps* über bestimmte Funktionen ausgelesen, als auch die erforderlichen Input-Werte für den nächsten *Time-Step* über Signal-Treiber in den Simulationsablauf eingebracht werden.

### SystemC Interface

SystemC [10] ist eine noch sehr junge Entwicklung und ist im wesentlichen eine Simulations-Bibliothek [11] die man in C(++)-Programmen nutzen kann. Da SystemC im Quellcode zur Verfügung steht und es ohne weiteres möglich ist "fremden" C-Code einzubinden werden die Bibliotheken oft von den Entwicklern nach eigenen Bedürfnissen angepasst und erweitert. Insofern ist eine starre Schnittstelle eher unpraktikabel, aber auch nicht notwendig, da aufgrund der Systematik von SystemC die Kommunikationsmechanismen, unter Rücksichtnahme auf die in Kap.3.1 vorgestellte Interaktionssystematik zwecks Datenkonsistenz, direkt in den Simulationscode integriert werden können.

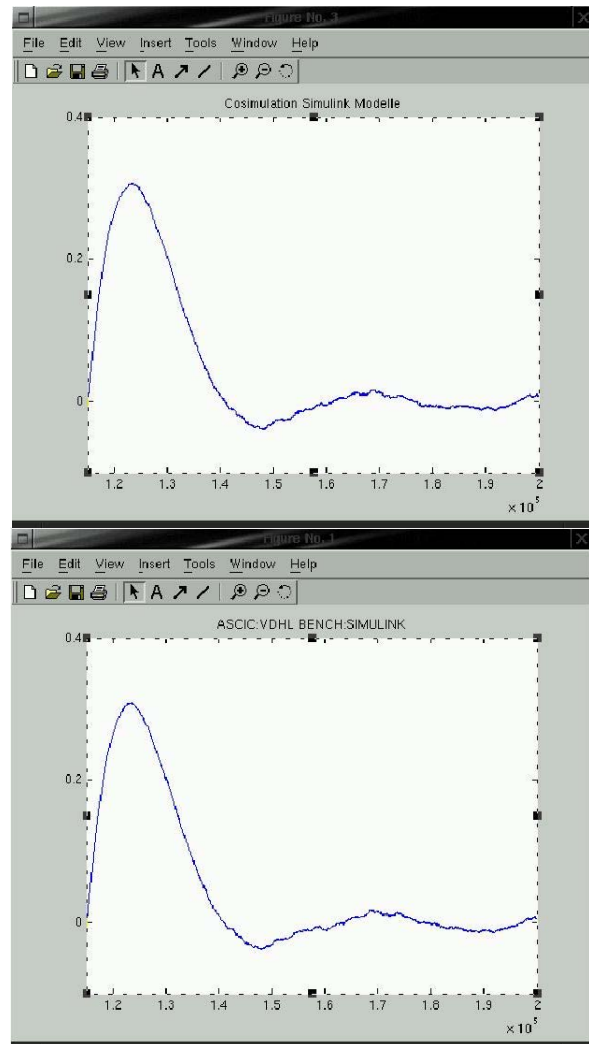
### 3.3.3 Kommunikationsknoten

Ähnlich dem Backplane-Verfahren kann ein Kommunikationsknoten eingeführt werden. Das Protokoll sieht zwar keine Steuer-Befehle vor, jedoch können die Simulatoren indirekt durch Zurückhalten von Daten angehalten werden. Der Zeitstempel der Datenpackages ermöglicht eine eindeutige Identifikation selbiger und ein entsprechendes Management.

Bei Bedarf kann das Protokoll um Backplane-Befehle erweitert werden, dies bedarf jedoch einer Anpassung der Kommunikationsmechanismen. So wäre z.B. die Integration von zwei Werten zur Unterscheidung zwischen Nutzdaten und Befehlssequenzen bzw. der Bestimmung der Package-Länge eine mögliche Vorgehensweise.

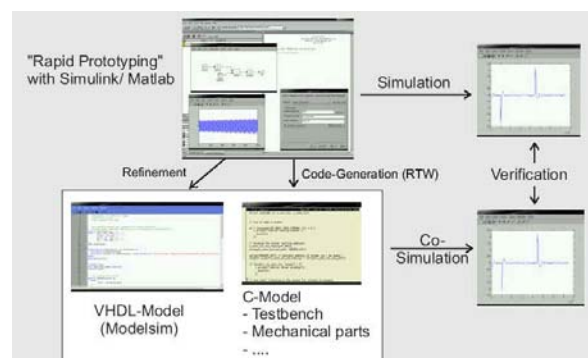
## 4 Experimentelle Versuche

Zur Verifikation des Systems wurde ein komplexes Simulink Modell bestehend aus einem ASIC<sup>6</sup> und einer zugehörigen Testbench in zwei Submodule partitioniert und die Simulationsergebnisse (Abb. 6) verglichen.



**Bild 6** Signalausschnitt bei Gesamtsimulation(oben) und partitioniert (unten)

Die Cosimulation einer VHDL-Beschreibung des ASICs mit dem Simulink-Modell der Testbench (Abb. 7) brachte ebenfalls eine qualitative Äquivalenz. Die Tests bestätigen sowohl die Funktion des Systems, als auch die Validierungsfunktion über verschiedene Abstraktionslevel.



**Bild 7** Verifikationsvorgang

<sup>6</sup> Application Specific Integrated Circuit

## 5 Zusammenfassung und Ausblick

Es wurde eine sehr flexible Methode vorgestellt, welche eine verteilte Cosimulation mit den Tools Simulink, SystemC und Modelsim (VHDL) ermöglicht. Die noch andauernden Tests belegen sowohl die Funktion als auch die praktische Einsatzfähigkeit des Systems.

Für den Anwender bieten sowohl das Vorliegen der Bibliothek in der Standardsprache C als auch das zu Grunde liegende Konzept neue Freiheitsgrade bei der Modellierung und bieten dem Designer somit neue weitreichende Möglichkeiten.

Aktuelle Arbeiten beschäftigen sich mit der Entwicklung eines Tools zur Modellierung heterogener Systeme. Dieses Tool dient zur Systemmodellierung, Steuerung der einzelnen Simulatoren und zur automatischen Interfacegenerierung.

## 6 Literatur

- [1] G.Lehmann B. Wunder M.Selz. *Schaltungsdesign mit VHDL*. Number ISBN 3-7723-6163-3. Franzi's Verlag, 1994.
- [2] L.Kriaa W.Youssef G.Niculescu S.Martinez S.Levitan J.Martinez T.Kurzweg A.A. Jerraya B.Courtois. *SystemC-Based Cosimulation for Global validation of MOEMS*. Technical Report ISRN TIMA-RR-02/01-1-FR, TIMA Laboratory, Grenoble, France / Department of Electrical Engineering, University of Pittsburgh, Pennsylvania, USA, 2002.
- [3] Heiko Schubert. *A Survey of HW/SW Cosimulation Techniques and Tools*. Master's Thesis, Royal Institute Of Technology, Schweden, 1998.
- [4] Michael M. Madden. *An Object-Oriented Interface for Simulink Models*. Technical Report AIAA-2000-4391, NASA Langley Research Center, Hampton, USA, 2000.
- [5] The MathWorks Inc., Natick, USA. *Simulink Model-Based and System-Based Design; Using Simulink*, 5 edition.
- [6] Model Technology, Portland, USA. *Modelsim:Foreign Language Interface*, 5.5f edition, August 2001.
- [7] E.A.Lee A.Sangiovanni-Vincatelli S.Edwards L.Lavagno. *Design of Embedded Systems: Formal Models, Validation and Synthesis*. Technical Report Proceedings of IEEE, Vol. 85, No. 3. March 1997, S.366 ff.
- [8] C.Simhart, *Implementierung einer HW/SW Cosimulations-Schnittstelle für die Simulation heterogener Systeme*. Master's thesis, Technische Universität Graz, Österreich, 1998.
- [9] K.D.Müller-Glaser S.Schmerler, Y.Tanurhan. *A Backplane for Mixed-Mode Cosimulation*. Technical Report EUROSIM95, S.499 ff.
- [10] SystemC Konsortium. *Functional Specification For SystemC 2.0, Final*, 2001.
- [11] W.Müller J.Ruf D.Hoffmann J.Gerlach Th.Kropf W.Rosenstiehl. *The Simulation Semantics of SystemC*. Technical Report, Universitäten von Paderborn und Tübingen, Paderborn/Tübingen, Deutschland.