Kostenmodellierung mit SystemC/SystemC-AMS

Erik Markert, Hailu Wang, Göran Herrmann und Ulrich Heinkel
Professur Schaltkreis- und Systementwurf,
Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Chemnitz
Reichenhainer Str. 70, 09126 Chemnitz
E-Mail: erik.markert@etit.tu-chemnitz.de

Kurzfassung

In diesem Beitrag wird eine Methode zur Beschreibung von Kostenfaktoren und deren Verknüpfung über Hierarchiegrenzen hinweg dargestellt. Sie eignet sich sowohl für rein digitale Systeme mit Softwareanteilen als auch für gemischt analog/digitale Systeme. Damit ist sie im Hardware-Software Codesign und im Analog-Digital Codesign zum Vergleich verschiedener Systemkompositionen anwendbar. Die Implementierung mit C++ ermöglicht neben einer Nutzung mit digitalem SystemC auch den Einsatz mit der analogen SystemC-Erweiterung SystemC-AMS und vereinfacht die Nutzung gegenüber einer vorhandenen VHDL-Implementierung. Als Anwendungsbeispiel fungieren Komponenten eines Systems zur Inertialnavigation.

1 Einleitung

Mit der fortschreitenden Entwicklung von Entwurfswerkzeugen und Bibliotheken für den Mikrosystementwurf entsteht zunehmend die Notwendigkeit, Mikrosysteme nicht nur hinsichtlich funktioneller Parameter, sondern auch hinsichtlich von Kostenfaktoren im Systementwurf zu optimieren ([1], [2]), da diese Parameter neben den funktionalen Daten wichtige Kenngrößen eines Systems darstellen. Dabei sind unter dem Begriff Kosten nicht nur fiskalische Kosten, sondern im Sinne der Terminologie der mathematischen Optimierung eine Repräsentation einer zu optimierenden Zielfunktion zu verstehen. Mit dieser Zielfunktion sollen Größen wie z.B. Latenz, Datenrate, Leistungsaufnahme, Chipfläche, Fehlertoleranz, Testbarkeit, Speicherverbrauch, Herstellungskosten u.a. gewichtet bewertet werden. Bei der Systemkomposition sind diese Werte einerseits auf das Einhalten einer vorgegebenen Grenze zu prüfen (z.B. maximal zur Verfügung stehende Chipfläche) und andererseits zu einem globalen Gütemaß zu verknüpfen. Eine Optimierung dieses globale Gütemaßes führt zu einer Verbesserung der Systemimplementierung. Die Bestimmung von Kostenwerten auf hoher Abstraktionsebene, wie z.B. in [3] dargestellt, ist nicht Bestandteil dieser Arbeit.

Um die Datenkonsistenz zwischen Modell und Kostenwerten sicherzustellen, bietet es sich an, die Kosten in einem simulationsfähigen Werkzeug mit einzubeziehen. Dafür stehen zahlreiche Systembeschreibungssprachen wie z.B. SystemVerilog, VHDL/VHDL-AMS und SystemC/SystemC-AMS zur Verfügung. Dieser Beitrag konvertiert und erweitert eine auf additive Verknüpfungen beschränkte Implementierung von hierarchischen Kostenbeschreibungen mit VHDL [4] für SystemC um neue Verknüpfungsmöglichkeiten. Eine Nutzung von [4] mit VHDL-AMS scheiterte an der mangelnden Unterstützung der IEEE-Sprachkonstrukte durch kommerzielle Simulatoren. SystemC/SystemC-AMS bietet gegenüber VHDL/VHDL-AMS mehr Möglichkeiten der abstrakten Systembeschreibung, so dass größere Systeme beherrschbar sind. Die Funktionsfähigkeit der Kostenmodellierung wird anhand von analogen Komponenten eines Inertialnavigationssystems demonstriert.

2 Implementierung der Kostenmodellierung

Die Realisierung der Kostenmodellierung in SystemC/SystemC-AMS bietet gegenüber der Implementierung mit VHDL/VHDL-AMS [4] mehrere Vorteile. In SystemC-AMS [6], der analogen Erweiterung von SystemC [5], besteht im Gegensatz zu VHDL-AMS keine Notwendigkeit, Ports als Verbindungselemente zu nutzen. Stattdessen kommt eine Listenstruktur zur Anwendung. Dies erlaubt die Nutzung des C++-Pointerkonzepts unabhängig vom Modulinterface. Somit können die Kostenwerte auch während der Laufzeit noch geändert werden, was eine dynamische Bestimmung von Kostenparametern (z.B. Stromaufnahme) ermöglicht. Außerdem ist es mit vorhandenen C++-Compilern möglich, automatisiert mehrere Implementierungsvarianten zu untersuchen. Im folgenden wird kurz die verwendete Datenstruktur sowie die zur Verfügung stehenden Funktionen erläutert.

Datenstruktur

Die Kostenwerte jeder Komponente werden in einer eigenen, durch Membervariablen einer Instanz der neuen Klasse *cost* repräsentierten, Struktur abgelegt. Um von einem Element des Designs aus das nächste zu erreichen, bietet sich eine verzweigte Listenstruktur an. Dazu enthält jede Komponente einen Zeiger auf das nächste Element auf gleicher Hierarchieebene (pNext) und einen Zeiger auf das erste Element der nächsttieferene Hierarchieebene (vNext). Zusätzlich kann die Komponente mit einer ID versehen werden. Sie enthält außerdem einen Zeiger auf sich selbst, mit der beim Parsen des Kostenbaumes der systemweit eindeutige Instanzname ausgegeben werden kann. Ein optionaler Kostengrenzvektor dient zum Festlegen der maximal von der Komponente und deren Unterkomponenten verbrauchbaren Ressourcen. Ein Flag *limit_exceeded* weist auf eine Kostengrenzenüberschreitung hin. Damit ergibt sich eine Kostenstruktur wie in Bild 1 dargestellt.

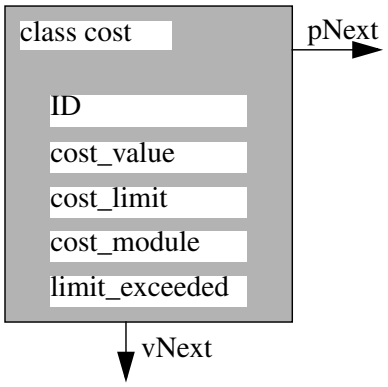

Bild 1 Kostenstruktur

Die Kostenwerte einer Komponente werden in einer Struktur als Instanz der neuen Klasse *cost* abgelegt, wobei eine verkettete Listenstruktur das Erreichen benachbarter Module ermöglicht. Im Design entsteht damit ein sog. Kostenbaum (ein Ausschnitt ist in Bild 2 dargestellt). Von dessen oberster Hierarchieebene können alle Einzelelemente durch Zeigeroperationen angesprochen werden. Ein Vektor umfasst alle Einzelkosten einer Komponente, ein weiterer Vektor gibt optional die maximal erlaubten Kosten der Komponente an. Die Bedeutung des Werts erschließt sich aus seiner Position. Tabelle 1 enthält die Elemente der neuen Klasse *cost*.

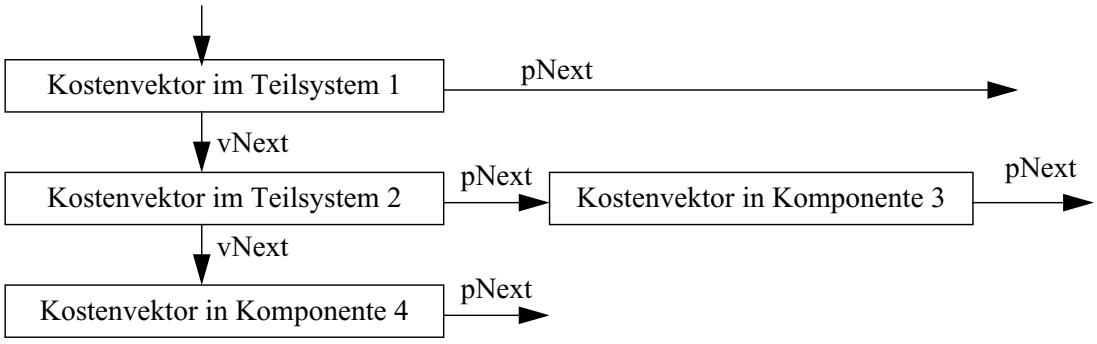

Bild 2 Teil eines Kostenbaumes

Funktionen zur Kostenmodellierung

Zur Erstellung und Analyse des Kostenbaumes muss die Klasse *cost* um Funktionen erweitert werden. Die Aufrufe zum Abarbeiten des Kostenbaumes basieren soweit wie möglich auf Rekursion. Die Klasse *cost* besitzt die in Tabelle 2 angegebenen Funktionen zum Aufbau des Kostenbaumes und zur Berechnung der Systemkosten.

Tabelle 1: Elemente der Klasse *cost*

Element	Datentyp	Beschreibung
ID	char[128]	Name
cost_value	double[]	Kostenwerte
cost_limit	double[]	Kostengrenzen
cost_module	sc_module*	Zeiger zum zugehörigen SystemC(-AMS)-Modul
limit_exceeded	bool[]	Flags für Kostenüberschreitungen
pNext	cost*	Zeiger auf das nächste Element (hierarchisch horizontal)
vNext	cost*	Zeiger auf das erste Tochterelement (hierarchisch vertikal)

Tabelle 2: Funktionen der Klasse *cost*

Funktion	Beschreibung
add(cost* a)	Fügt ein Modul auf nächstniedriger Hierarchieebene ein
set_val(double[])	Definition der Kostenwerte eines Moduls
set_limit(double[])	Definition der Kostengrenzen eines Moduls
del()	Löschen des Kostenbaumes
list_print()	Gibt die Kostenwerte aller Elemente aus, Hierarchieebenen sind eingerückt
check_system()	Prüft das System auf Kostengrenzenüberschreitungen
sum()	Berechnet die Kostenwerte über Hierarchieebenen
scale(double[])	Berechnet das globale Gütemaß

Für die Berechnung der Kostenwerte wurden neue Operatoren eingeführt. Der schon in der VHDL-Version [4] vorhandene Operator ADD summiert die Kostenwerte der Unterkomponenten. Dies ist in den meisten Fällen ausreichend, z. B. für Chipflächenverbrauch oder Leistungsaufnahme. Der MAX-Operator bestimmt das Maximum der Kosten (z.B. für die minimale Betriebstemperatur oder die minimale Taktzeit in einer Pipeline), als Gegenstück fungiert der MIN-Operator. Der vierte Operator MULT multipliziert die Kostenwerte, z.B. für die direkte Berechnung von Verstärkungen oder Zuverlässigkeitsberechnungen. Die Operatoren können für jeden Kostentyp (jedes Vektorelement) unabhängig festgelegt werden. Die Überwachung der Kostengrenzen erfordert eine Fallunterscheidung: Einige Werte wie Chipflächenverbrauch müssen die Grenze unterschreiten, während andere die Grenze überschreiten sollen (z.B. minimale Taktfrequenz). Darum stehen mit COMP_MAX und COMP_MIN zwei neue Vergleichsoperatoren zur Verfügung. Das globale Gütemaß eines Systems wird mit dem Befehl scale() bestimmt. Dieser berechnet eine gewichtete Summe der Kostenwerte des Top-level-Moduls bezogen auf die Kostengrenzen. Das Vorzeichen der Gewichte bestimmt das Modul aufgrund des Grenzoperators.

3 Anwendungsbeispiel: Auswerteeinheit für ein kapazitives Beschleunigungssensorarray

Die Funktionalität der Kostenmodellierung wurde an einem abstrakten Beispiel geprüft. Dieses umfasste vier Hierarchieebenen mit Mehrfachinstanziierungen von Komponenten. Alle Tests mit verschiedenen Kostenwerten und Kostengrenzen verliefen fehlerlos. Aus Platzgründen soll hier auf eine

Darstellung dieses Komplexbeispieles verzichtet und statt dessen eine kleinere, aber real aufgetretene Problemstellung eines Analog-ASIC betrachtet werden. Im Rahmen des SFB 379 „Mikromechanische Sensor- und Aktorarrays“ enstand ein Demonstrator zur Universellen Bewegungsanalyse (UBAS) [7]. Dieser umfasst Beschleunigungs- und Drehratesensoren sowie die analoge und digitale Verarbeitung dieser Werte zu Positionsdaten. Für die Beschleunigungssensoren kommt dabei ein Array aus sechs kapazitiv arbeitenden Einzelsensoren zum Einsatz [8].

Für dieses Array soll eine analoge Auswerteschaltung mit anschließender Analog-Digital-Wandlung entworfen werden. Die Einzelsensoren liefern bei Anlegen einer Beschleunigung einen Strom, der proportional zur Kapazitätsänderung ist. Dieser wird über sogenannte DeltaC-U-Wandler verstärkt und in eine Spannung gewandelt. Anschließend ist die Analogspannung in einen Digitalwert umzusetzen, der in einem FPGA weiter verarbeitet wird. Für A/D-Umsetzung und DeltaC-U-Wandlung sind außerdem eine Referenz- sowie eine Mittenspannungserzeugung notwendig. Da die verfügbare Fläche pro Einzelchip begrenzt ist, sollen pro Analog-IC nur die Signale von drei der sechs Einzelsensoren verarbeitet werden. Für das gesamte Array sind also zwei Analogchips notwendig.

Es stehen drei verschiedene Realisierungsvarianten zur Auswahl:

A Pro Einzelsensor wird ein DeltaC-U-Wandler und ein eigener AD-Umsetzer implementiert.

B Für das gesamte Array wird ein einzelner DeltaC-U-Wandler mit AD-Umsetzer im Zeitmultiplex genutzt, direkt am Eingang ist ein Multiplexer für das Umschalten zwischen den Einzelsensoren einzusetzen.

C Es wird für jeden Einzelsensor ein eigener DeltaC-U-Wandler implementiert. Diese übergeben ihre Werte jedoch über einen Multiplexer an einen einzelnen AD-Umsetzer.

Für jedes Systemelement werden drei Kostenwerte berücksichtigt: Chipfläche, Verarbeitungszeit und Entwurfszeit. Tabelle 3 gibt eine Übersicht über die zugewiesenen Kostenwerte in der jeweiligen Realisierungsvariante. Die Entwurfszeiten pro Element sind dabei Schätzwerte aus bereits implementierten

Tabelle 3: Übersicht der Kostenwerte der Einzelkomponenten

Element	**Variante**	**Anzahl Elemente**	**Fläche pro Element [mm^2]**	**Verarb.-zeit für drei Werte [µs]**	**Entwurfszeit pro Element [Zeiteinheiten]**
Analog-Digital-Umsetzer	A	3	2,295	0,8	3000
	B, C	1	2,295	2,5	5000
DeltaC-U-Wandler	A, C	3	0,252	0,1	300
	B	1	0,252	0,4	1000
Multiplexer	A	0	-	-	-
	B	2	0,010	0,3	100
	C	1	0,010	0,3	100
Padring	A	60 Pads	2,964	0	15
	B, C	42 Pads	2,058	0	10
Spannungserzeugung	A-C	1	0,049	0	400
globale Verdrahtung	A	-	0,050	0	600
	B	-	0,025	0	250
	C	-	0,030	0	300

Designs. Für die Pads werden Bibliothekselemente genutzt, was deren Entwurfsaufwand erheblich reduziert.

Für die Berechnung des skalaren Gütemaßes wird die Verarbeitungszeit übergewichtet, da diese maßgeblich für die spätere Systemgeschwindigkeit ist. Die benötigte Chipfläche hat bis zu einer bestimmten Grenze (6,25 mm^2) kaum Einfluss auf die Herstellungskosten, daher wird sie hier stark untergewichtet. Sie spielt jedoch als absolute Grenze eine wichtige Rolle als k.o.-Kriterium für die Realisierbarkeit. Folgende Parameter werden für die Systembeurteilung verwendet:

Grenzen: Fläche 6,25 mm^2; Verarbeitungszeit: 3,5 µs; Entwurfszeit: 7500 Zeiteinheiten

Gewichte: Fläche × 0,01; Verarbeitungszeit × 10; Entwurfszeit × 1e-3

Die Anwendung der Kostenmodellierung auf die Systemvarianten A-C erzeugt die folgenden Bildschirmausgaben:

Variante A:

```
system consists of:(cost vector: 10.654 2.7 10315 cost limit: #6.25 3.5 #7500)
  DeltaC_U      cost vector: 0.252 0.1 300      cost limit: -1 -1 -1
  DeltaC_U      cost vector: 0.252 0.1 300        cost limit: -1 -1 -1
  DeltaC_U      cost vector: 0.252 0.1 300      cost limit: -1 -1 -1
  TriGen        cost vector: 0.049 0 400        cost limit: -1 -1 -1
  Padring       cost vector: 2.964 0 15         cost limit: -1 -1 -1
  ADU   cost vector: 2.295 0.8 3000     cost limit: -1 -1 -1
  ADU   cost vector: 2.295 0.8 3000     cost limit: -1 -1 -1
  ADU   cost vector: 2.295 0.8 3000     cost limit: -1 -1 -1
WARNING: 1. cost value in module system_ana1 exceeds cost limit!
WARNING: 3. cost value in module system_ana1 exceeds cost limit!
WARNING: Design is NOT realisable with this configuration!
Weighted scaled costs for System1 7.446993
Check_System: Design is NOT realisable
```

Variante B:

```
system consists of:(cost vector: 4.674 3.2 6610 cost limit: 6.25 3.5 7500)
  DeltaC_U      cost vector: 0.252 0.4 1000      cost limit: -1 -1 -1
  TriGen        cost vector: 0.049 0 400        cost limit: -1 -1 -1
  MUX3  cost vector: 0.01 0.15 100      cost limit: -1 -1 -1
  MUX3  cost vector: 0.01 0.15 100      cost limit: -1 -1 -1
  Padring       cost vector: 2.058 0 10         cost limit: -1 -1 -1
  ADU   cost vector: 2.295 2.5 5000     cost limit: -1 -1 -1

 Weighted scaled costs for System1 9.151217

 Check_System: Design is realisable
```

Variante C:

```
system consists of:(cost vector: 5.168 3.1 6410  cost limit: 6.25 3.5 7500 )
  DeltaC_U      cost vector: 0.252 0.1 300      cost limit: -1 -1 -1
  DeltaC_U      cost vector: 0.252 0.1 300        cost limit: -1 -1 -1
  DeltaC_U      cost vector: 0.252 0.1 300      cost limit: -1 -1 -1
  TriGen        cost vector: 0.049 0 400        cost limit: -1 -1 -1
  MUX3  cost vector: 0.01 0.3 100       cost limit: -1 -1 -1
  Padring       cost vector: 2.058 0 10         cost limit: -1 -1 -1
  ADU   cost vector: 2.295 2.5 5000     cost limit: -1 -1 -1

 Weighted scaled costs for System1 8.866266

 Check_System: Design is realisable
```

Variante A scheidet wegen der Kostenüberschreitung bei den Kostenwerten „Fläche“ und „Entwurfszeit“ aus. Dies wird in der Auflistung der Kostenwerte durch ein Doppelkreuz sowie als separate Warnung angezeigt. Sowohl Variante C als auch Variante B erfüllen die gesetzten Grenzen. Variante C ist im gegebenen Anwendungsfall gegenüber der Realisierungsmöglichkeit B zu bevorzugen, da sie bei Einhaltung der Grenzen das skalare Gütemaß minimiert.

4 Zusammenfassung und Ausblick

Dieser Beitrag beschreibt eine Möglichkeit der Kostenmodellierung mit SystemC/SystemC-AMS. Als Grundlage diente eine bestehenden VHDL/VHDL-AMS Implementierung. Durch die Nutzung der Konzepte von C++ bietet die Übertragung nach SystemC neue Möglichkeiten der Datenverwaltung mittels verketteter Listen. Erweiterte Operatoren bieten neue Verknüpfungsmöglichkeiten der Kostenparameter. Die neue Implementierung in SystemC/SystemC-AMS wurde anhand eines Anwendungsbeispiels aus dem Analogentwurf verifiziert.

Durch die Listenstruktur ist es nun möglich, Kostenwerte noch zur Laufzeit zu ändern und damit dynamisch aus Simulationsläufen zu generieren. Außerdem ermöglichen vorhandene C++-Compiler, automatisiert mehrere Implementierungsvarianten zu bearbeiten, was Untersuchungen zum Hardware-/Software-Codesign und zur Partitionierung in analoge und digitale Anteile beschleunigt.

Danksagung

Die hier vorgestellten Arbeiten entstanden im Rahmen des Teilprojektes A2 „Systementwurf“ des von der DFG geförderten SFB 379 „Mikromechanische Sensor- und Aktorarrays“.

Literatur

[1] Eles, P.; Peng, Z.; Kuchcinski, K.; Doboli,A.: *System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search.* Journal on Design Automation for Embedded Systems, vol. 2, 1997, pp 5-32

[2] De Bernardis, F.; Nuzzo, P.; Sangiovanni Vincitelli, A.: *Mixed Signal Design Space Exploration through Analog Platforms*. DAC 2005, Anaheim, USA, June 2005, pp. 875-880

[3] Stammermann, A. et. al.: *ORINOCO: Verlustleistungsanalyse und Optimierung auf der algorithmischen Abstraktionsebene.* Entwurf Integrierter Schaltungen, 10. E.I.S.-Workshop, Dresden, 2001

[4] Schlegel, M.; Herrmann, G.; Mueller, D.: *Erweiterte Kostenmodellierung mit VHDL/VHDL-AMS.* GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Kaiserslautern, 2004, Berichte aus der Informatik, Shaker Verlag, Aachen, Germany 2004

[5] Open SystemC Initiative (OSCI): *SystemC 2.1 LRM*. 2004, http://www.systemc.org

[6] Einwich, K. et. al.: *White Paper SystemC-AMS Study Group.* http://www.systemc-ams.org

[7] Markert, E.; Dienel, M.; Herrmann, G.; Heinkel, U.: *SystemC-AMS assisted design of an Inertial Navigation System.* IEEE Sensors Journal: Special Issue on Intelligent Sensors, 2007 (accepted)

[8] Dienel, M. et. al.: *Development of a drift compensated acceleration sensor array.* 50. IWK Ilmenau, 19.-23. September 2005