Kostenmodellierung mit VHDL/VHDL-AMS

Michael Schlegel, Göran Herrmann, Dietmar Müller
TU Chemnitz, Fakultät für Elektrotechnik und Informationstechnik,
Professur Schaltungs- und Systementwurf,
09107 Chemnitz,
michael.schlegel@infotech.tu-chemnitz.de

Kurzfassung

Dieser Beitrag beschreibt einen methodischen Ansatz zur Beschreibung von Kostenfaktoren[1] digitaler oder heterogener System zusammen mit dem Verhalten des Systems. Durch die Abbildung der Kostenfaktoren auf funktionelle Komponenten des Modells kann der Entwerfer sein System während der Verhaltenssimulation bezüglich nichtfunktionaler Parameter überprüfen und optimieren. Desweiteren macht dieses Verfahren die Kostenfaktoren eines Systems für Softwaretools zur Verhaltensoptimierung zugänglich, wodurch auf extra Werzeuge zur Kostenoptimierung verzichtet werden kann.

Die Bestimmung von Kostenwerten auf hoher Abstraktionsebene, wie z. B. in [1] und [2], ist nicht Bestandteil dieser Arbeit.

1 Motivation

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, da diese Parameter neben den funktionalen Daten wichtige Kenngrößen eines Systems darstellen. Erste Ansätze dazu sind z. B. in [3] dokumentiert.

Bisher werden vorrangig Tools verwendet, die die Kostenparameter parallel zum Modell erfassen und vom Modell getrennt auswerten und optimieren. Daraus ergeben sich jedoch zwei Nachteile:

- für die Optimierung des Verhaltens und für die Optimierung der Kosten werden zwei Werkzeuge mit ähnlicher Grundfunktion benötigt,
- bei Änderungen am Modell können die zugehörigen Kostenfaktoren u. U. vergessen werden und damit die Konsistenz der Daten verloren gehen.

Ziel des hier vorgestellten Ansatzes ist es daher, diese Kostenfaktoren auf funktionelle Bestandteile des Verhaltens des Modelles abzubilden und gleichzeitig mit der Simulation auszuwerten und zu optimieren, wobei vorhandene Werkzeugen für die Optimierung der Funktionalität, z. B. [4] und [5], prinzipiell zur Anwendung kommen können.

1. Der Begriff *Kosten* wird hier im Sinne der Terminologie der mathematischen Optimierung verwendet und repräsentiert das Ergebnis einer zu optimierenden Zielfunktion, mit der Größen wie Leistungsverbrauch, Chipfläche, Fehlertoleranz, Testbarkeit, Speicherverbrauch, Herstellungskosten u. a. gewichtet bewertet werden [6].

2 Realisierung

Die Modellierung der Kosten als Bestandteil des Verhaltens soll für die Hardwarebeschreibungssprache VHDL bzw. VHDL-AMS realisiert werden, da diese Sprache die Möglichkeit zur Modellierung- und Simulation eines breiten Spektrums an heterogenen Systemen bietet.

Als Randbedingungen für die Entwicklung des methodischen Ansatzes sollen gelten:

- leichte Anwendbarkeit ohne große Einarbeitungszeiten,
- minimaler Modellierungs- und Berechnungsmehraufwand,
- Sicherstellung der Datenkonsistenz zwischen Modell und Kostenfaktoren.

2.1 Datenstrukturen

Für die Integration der Kostenfaktoren in das Modell müssen zuerst geeignete Datenstrukturen geschaffen werden, die die Informationen aufnehmen können.

```
--Kostenfaktoren
TYPE criterion IS ( develop_cost, product_cost,
                    space, power, testability,
                    error_safety);

TYPE criterion_val IS RECORD
                        val:real;  --Kostenwert
                        lim:real;  --Kostengrenze
                    END RECORD;

--Kostenvektor
TYPE criterion_vect IS ARRAY (criterion) OF
                                    criterion_val;
```

Der Datentyp `criterion` ist eine Aufzählung von Kostenfaktoren, die optimiert werden sollen. Der Inhalt dieses Typs kann dabei auf die im jeweiligen Entwurf erforderlichen Parameter angepaßt werden. Der Typ `criterion_val` ermöglicht die Angabe eines Wertes für einen Kostenfaktor und die Definition

einer Grenze für diesen Wert. Wird für die Grenze der Wert `-1` eingesetzt, so wird diese Kostengrenze bei späteren Berechnungen ignoriert. Dies gilt ebenso, wenn für den Kostenwert `-1` angegeben wird, wenn z. B. die Größe des Kostenfaktors nicht ermittelt werden kann.

Der Typ `criterion_vect` ist ein Vektor, der alle Kostenwerte und Kostengrenzen einer Komponente zusammenfaßt. Dieser Datentyp kann auf Konstanten (`CONSTANT`) oder Signale (`SIGNAL`) angewendet werden. Die Benutzung von Konstanten hat den Vorteil, daß die Berechnungen zur Kostenmodellierung nur zu Beginn der Simulation einmal ausgeführt und die Simulationsgeschwindigkeit dadurch kaum beeinträchtigt wird. Die Kostenwerte können so auch per `GENERIC` parametrisiert werden, sie sind zur Simulationslaufzeit aber nicht mehr änderbar. Ist dies erforderlich, so läßt sich der Datentyp `criterion_vect` auf ein Signal anwenden, dessen Wert sich simulationsabhängig ändert. Ergibt sich der Kostenwert aus einer `QUANTITY` als Ergebnis der Berechnung eines *simultaneous statements* (Differentialgleichung), so ist dafür Sorge zu tragen, daß die `QUANTITY` bei der Zuweisung auf das `SIGNAL` auch ein *event* auslöst (z. B. mittels des Attributs `'above`). Durch die Verwendung eines Signals kann sich allerdings, je nachdem wie oft sich dessen Wert ändert, die Simulationsgeschwindigkeit verringern.

Für die getrennte Definition der Vektoren für Kostenwert und -grenze werden zusätzlich die Datentypen `criterion_limit_vect` und `criterion_val_vect` definiert.

2.2 Definition der Kostenparameter und Kostengrenzen im Modell

Neben der im vorigen Abschnitt beschriebenen Verwendung von Konstanten oder Signalen für die Kostenvektoren besteht weiterhin die Möglichkeit, die Werte für Kostenwert und -grenze gemeinsam oder in getrennten Definitionen zu setzen. Die gemeinsame Definition bietet sich an, wenn für ein Modell nur eine Architekturvariante zur Verfügung steht. Sind dagegen für eine Komponente mehrere Architekturvarianten verfügbar, so kann in der `ENTITY` die Definition der Grenzen erfolgen, die dann für alle Architekturalternativen (`ARCHITECTURE`) gültig sind. In der jeweiligen `ARCHITECTURE` werden dann die für diese Variante gültigen Kostenwerte gesetzt.

2.3 Verbinden der Kostenvektoren über Hierarchieebenen

Im nächsten Schritt müssen die Kostenvektoren der einzelnen Komponenten im übergeordneten Strukturmodell zu einem neuen Kostenvektor, der die Kosten z. B. einer Baugruppe repräsentiert, zusammengefaßt werden. Dabei muß die Anzahl der Komponenten *n* dieser Struktur variabel sein und darf nicht durch die Funktion, die die Kostenvektoren zusammenfassen soll, eingeschränkt werden.

$$K_{j_{val,\,limit}} = \sum_{i=1}^{n} K_{i,\,j_{val,\,limit}} \qquad \forall j(criterion(j)) \quad \text{(GL 1)}$$

Normale Funktionen haben aber eine feste Anzahl von Parametern und sind somit nicht zur Zusammenfassung einer variablen Anzahl von Kostenvektoren geeignet. Als Lösung bieten sich hier die VHDL *resolution functions* an. Diese Funktionen dienen dazu die Konflikte, die entstehen wenn mehrere Treiber auf ein *resolved signal* schreiben, aufzulösen.

Die Kostenvektoren der einzelnen Komponenten werden über Signal-Ports mit dem Richtungsattribut `out` in die nächsthöhere Hierarchieebene übertragen und dort mit einem *resolved* Signal vom Typ `criterion_resolve_vect` verbunden. Die für die Zusammenfassung der einzelnen Kostenvektoren notwendige spezielle *resolution function* addiert im einfachsten Fall die Kostenwerte und Kostengrenzen für alle Kostenfaktoren. Ist die Kostengrenze eines Kostenfaktors in einer Komponenten auf `-1` gesetzt, so werden diese Grenze und der dazugehörige Kostenwert ignoriert. Betragen die Kostengrenzen für einen Kostenfaktor in allen Komponenten `-1`, so erhält der entsprechende Faktor im resultierenden Kostenvektor ebenfalls die Grenze `-1`. Die *resolution function* ermittelt ebenfalls, ob ein Kostenwert einer Komponente seine Kostengrenze übersteigt und gibt ggfs. eine Meldung aus. Dabei kann vom Anwender festgelegt werden, ob die Meldung ein Hinweis, eine Warnung oder eine Fehlermeldung sein soll.

Darüber hinaus wurden weitere Funktionen geschaffen, mit deren Hilfe der von der *resolution function* erzeuge Kostenvektor bearbeitet werden kann. Mit diesen Funktionen ist es möglich, die Werte von Kostenfaktoren zu ändern bzw. die Kostengrenzen neu festzulegen.

2.4. Bestimmung eines skalaren Gütemaßes

Die Optimierung der Kostenfaktoren erfordert es im allgemeinen, aus den Kostenvektoren ein skalares Gütemaß *S* zu bestimmen [6].

$$S = \sum_{\forall j(criterion(j))} \frac{K_{j_{val}}}{K_{j_{lim}}} \cdot W_j \qquad \text{(GL 2)}$$

Dazu existiert im Rahmen der Methode der Kostenmodellierung ebenfalls eine Funktion, die auf der Basis der Kostenwerte K_{val}, der Kostengrenzen K_{lim} und eines Wichtungsvektors *W* einen skalaren Wert ermittelt, der ein Gütemaß für die Einhaltung der Kostengrenzen darstellt (Zielfunktion). Beträgt der Wert oder die Grenze eines Kostenfaktors `-1`, so wird dieser Faktor bei der Berechnung ignoriert. Sind sowohl Kostenwert als auch -grenze `0`, so wird als Quotient `1` angenommen.

3 Anwendungsbeispiel

Die Anwendung dieser Methode war ursprünglich beim Entwurf eines Vibrationsmeßsystems mit frequenzselektivem mikromechanischen Vibrationssensor-Array [7] geplant. Da aber in den getesteten VHDL-AMS Simulatoren (AdvanceMS, hAMSter, SMASH) verschiedene, zur Kostenmodellierung notwendige Statements nicht implementiert sind, ist eine Anwendung beim Entwurf heterogener Systeme noch nicht möglich. Deshalb soll hier die Anwendung der Kostenmodellierung an einem abstrakten System unter VHDL demonstriert werden.

3.1 Ausgangsdaten

Das abstrakte, fiktive System soll folgende Struktur besitzen:

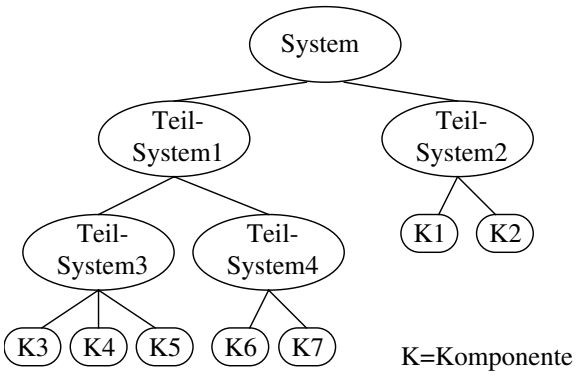

Bild 1 Systemstruktur

Die zu optimierenden Kostenfaktoren (Kriterien) sind:

- Anschaffungskosten
- Stromaufnahme
- mittlere Ausfallzeit (MTBF)
- Platzbedarf

Anschaffungskosten, Stromaufnahme und Platzbedarf der gesamten Struktur sind zu minimieren, die mittlere Ausfallzeit zu maximieren. Die gleichzeitige Maximierung und Minimierung von Kriterien als Ziel der Optimierung ist mit der aktuellen Version des vorgestellten Ansatzes nicht möglich. Die mittlere Ausfallzeit muß daher durch Reziprokwertbildung in eine Ausfallwahrscheinlichkeit umgewandelt werden, die dann zu minimieren ist. Tabelle 1 zeigt die für die einzelnen Komponenten des Systems angesetzten Grenzen und Tabelle 2 die Kostenwerte für die Komponenten 1 bis 4. So stehen z. B. für Komponente K1 drei verschiedene Alternativen zur Verfügung, bei der die erste die Vorgabe bezüglich der Stromaufnahme überschreitet. Für die zweite und dritte Komponente steht jeweils nur eine Architektur zur Verfügung, wobei bei der dritten die Kostenwerte noch nicht feststehen. Für die weiteren Komponenten des Systems gelten ähnliche Vorgaben. Weiterhin sollen im Teilsystem 3 z. B. durch die Montage der Komponenten weitere Kosten hinzukommen, ohne daß sich die Kostengrenzen erhöhen.

3.2 Modellimplementierung

Die folgenden Modellausschnitte zeigen an Beispielen die Anwendung des Ansatzes beim Entwurf. Alle in Abschnitt 2 genannten Datentypen und Funktionen sind im Package `cost_support` beschrieben, das bei Bedarf auf den jeweiligen Anwendungsfall leicht angepaßt werden kann (zu optimierende Kriterien, Reaktion auf Kostenüberschreitung).

Komponente 1, Entity mit Definition der Kostengrenzen:

```
USE work.cost_support.ALL;
ENTITY Komponente1 IS
    --funktionale Generics und Ports des Modells
    -- ...
```

	K1	K2	K3	K4	K5	K6	K7
Anschaffungskosten [EURO]	100	50	80	5	1000	20	10
Stromaufnahme [mA]	10	5	100	-1	500	5	-1
Ausfallwahrscheinlichkeit [1/Std]	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
Flächenbedarf [cm^2]	5	10	10	50	200	20	20

Tabelle 1 Kostengrenzen der einzelnen Komponenten

	K1 Variante 1	K1 Variante 2	K1 Variante 3	K2 Variante 1	K3 Variante 1	K4 Variante1	K4 Variante2	K4 Variante3
Anschaffungskosten	5	30	120	50	-1	5	5	5
Stromaufnahme	11	3	1	5	-1	0	0	0
Ausfallwahrsch.	1e-4	1e-6	1e-10	1e-4	-1	1e-4	1e-5	1e-6
Flächenbedarf	3	3	2	10	-1	10	20	30

Tabelle 2 Kostenfaktoren der Komponenten 1 bis 4

```
    --Port für Kostenvektor
    PORT(cost:OUT criterion_vect:=
                            (OTHERS=>(-1.0,-1.0)));
    --Definition der Kostengrenzen
    CONSTANT cost_limit:criterion_limit_vect:=
              (Anschaffung    =>100.0,   --EURO
               Stromaufnahme  =>10.0,    --mA
               Fehlerrate     =>1.0e-4,  --1/Stunde
               Groesse        =>5.0);    --Einheiten
END;
```

Parallel zu dem Port `cost` liegen bei Anwendung in einem realen Modell die normalen funktionalen Ports des Modells. Der Port `cost` muß mit dem Defaultwert `-1` für Kostenwert und -grenze versehen werden, um falsche Kostenüberschreitungsmeldungen zu vermeiden.

Komponente 1, Architekturvariante 1 mit Definition ihrer Kostenwerte:

```
ARCHITECTURE Version1 OF Komponente1 IS
    --Definition der Kostenwerte
    CONSTANT cost_val:criterion_val_vect:=
              (Anschaffung   =>5.0,
               Stromaufnahme =>11.0,
               Fehlerrate    =>1.0e-4,
               Groesse       =>3.0);
BEGIN
    --Modellgleichungen
    -- ...
    --Zusammenfügen von Kostengrenze und -wert
    cost<=cost_limit & cost_val;
END;
```

Um die getrennt definierten Vektoren für Kostenwert und -grenze zu vereinigen, wurde im Package `cost_support` der `"&"` Operator entsprechend überladen.

Komponente 1, Architekturvariante 2 mit Definition ihrer Kostenwerte:

```
ARCHITECTURE Version2 OF Komponente1 IS
    --Definition der Kostenwerte
    CONSTANT cost_val:criterion_val_vect:=
              (Anschaffung   =>30.0,
               Stromaufnahme =>3.0,
               Fehlerrate    =>1.0e-6,
               Groesse       =>3.0);
BEGIN
    --Modellgleichungen
    -- ...
    --Zusammenfügen von Kostengrenze und -wert
    cost<=cost_limit & cost_val;
END;
```

Komponente 3, gemeinsamme Definition von Kostengrenzen und -werten:

```
USE work.cost_support.ALL;
ENTITY Komponente3 IS
    --funktionale Generics und Ports des Modells
    -- ...
    --Port für Kostenvektor
    PORT(cost:OUT criterion_vect:=
                          (OTHERS=>(-1.0,-1.0)));
END;
ARCHITECTURE Version1 OF Komponente3 IS
    --Definition der Kostenwerte
    CONSTANT cost_val:criterion_vect:=
    (Anschaffung     =>(val=>-1.0, lim=>80.0),
     Stromaufnahme   =>(val=>-1.0, lim=>100.0),
     Fehlerrate      =>(val=>-1.0, lim=>1.0e-4),
     Groesse         =>(val=>-1.0, lim=>20.0));
BEGIN
    --Modellgleichungen
    -- ...
    cost<=cost_val;
END;
```

4 Resultate

Die Simulation wurde mit dem VHDL Simulator „Modelsim“ von Mentor Graphics durchgeführt. Der Ansatz der Kostenmodellierung hat dabei eine sehr gute Stabilität bewiesen. Die zur Berechnung der Kosten notwendige Rechenzeit ist vernachlässigbar. Die Kostenüberschreitung bei der Stromaufnahme der Version 1 der Komponente 1 wurde zuverlässig erkannt.

Das Bestimmen der optimalen Konfiguration erfolgte durch eine systematische Simulation der möglichen Systemkonfigurationen. Bei der Anwendung auf ein reales System müssen zur Bestimmung der Funktion ebenfalls viele Kombinationen aus verfügbaren Komponenten simuliert werden, so daß sich für die Bestimmung der Kosten kaum ein Mehraufwand ergibt.

Als kostenoptimales System haben sich bei einem Wichtungsvektor

```
CONSTANT Kostenwichtung: criterion_weight_vect:=
                          (Anschaffung   =>0.4,
                           Stromaufnahme =>0.3,
                           Fehlerrate    =>0.2,
                           Groesse       =>0.1);
```

die Konfigurationen aus

	K1	K4	K5	K7	skalares Gütemaß
Variante	2	2	1	1	0,61
Variante	2	3	1	1	0,61

ergeben. Für die Komponenten 2, 3 und 6 steht jeweils nur eine Architektur zur Verfügung.

Die „worst case“ Konfiguration für dieses System besteht aus:

	K1	K4	K5	K7	skalares Gütemaß
Variante	3	1	2	2	0,83

Es zeigt sich, daß trotz größerer Wichtung auf Anschaffungskosten im System für Komponente 1 nicht die billigste Version gewählt wird, da diese z. B. eine sehr hohe Stromaufnahme aufweist. Da zur Berechnung des skalaren Gütemaßes das Verhältnis von Kostenwert zu Kostengrenze ermittelt wird, spielt die absolute Größe eines Kostenwertes keine Rolle, sondern nur deren Relation.

Die interaktive Bestimmung des Optimums ist – insbesondere wenn viele Architekturalternativen zur Auswahl stehen – u. U. nicht mehr sinnvoll. Mit der hier vorgestellten Methode ist nun auch eine automatische Optimierung möglich. Dazu können die gleichen Tools eingesetzt werden, die auch zur Optimierung des Verhaltens Anwendung finden, vorausgesetzt diese sind in der Lage, die `CONFIGURATION` eines VHDL-Modells zu ändern und den entsprechenden VHDL-Compiler zu

starten. Da derzeit leider kein solches Tool zur Verfügung steht, konnte die automatische Optimierung nicht getestet werden.

Solange kein Kostenparameter von einem `SIGNAL` oder von einer `QUANTITY` abhängt, verursacht die Optimierung der Kosten gegenüber der reinen Optimierung des Verhaltens keine nennenswerte Verlängerung der Simulation, da in diesem Fall die Berechnungen zur Kostenoptimierung nur zum Zeitpunkt 0 s ausgeführt werden.

Ein weiterer Punkt der bei digitalen Modellen berücksichtigt werden muß, ist die Synthese des Modells. Es ist darauf zu achten, daß die für die Kostenoptimierung verwendeten Objekte nicht mit in die Synthese einbezogen werden, da diese nicht synthetisierbar sind. Dies ist compilerspezifisch möglich, bei Tools der Firma Synopsys z. B. mittels:

```
--synopsys translate_off
--Definition eines Kostenvektors
--synopsys translate_on
```

5 Zusammenfassung

Die Kostenmodellierung wurde implementiert und anhand eines abstrakten Beispiels getestet. In diesem Beispiel zeigte sich die sehr gute Funktionalität des Ansatzes. Negative Auswirkungen auf Rechenzeit oder Simulationsstabilität traten nicht auf. Für die Modellierung heterogener Systeme mit VHDL-AMS konnte diese Methode allerdings noch nicht angewandt werden, da die aktuellsten Versionen der zur Verfügung stehenden VHDL-AMS Simulatoren „AdvanceMS“ (Mentor Graphics) und „hAMSter“ (Ansoft) Teile der für diese Methode notwendigen Typdefinitionen noch nicht beherrschen.

Der hier aufgeführte methodischen Ansatz ermöglicht die interaktive Optimierung der Kostenparameter während der Simulation des Systems. Ebenso wird die Grundlage für ein automatisiertes Arbeiten mit Werkzeugen zur Verhaltensoptimierung geschaffen.

Durch die gemeinsame Beschreibung von Kosten und Verhalten der Komponente wird außerdem die Konsistenz dieser Daten zueinander gewährleistet.

6 Ausblick

Der methodische Ansatz der Kostenmodellierung ist weitestgehend fertiggestellt und getestet. Die Anwendung dieser Methode im Systementwurf heterogener System mit VHDL-AMS steht wegen der zuvor genannten Einschränkungen der Simulatoren noch aus. Größere Probleme sind dabei jedoch nicht zu erwarten.

Der nächste Entwicklungsschritt ist die Übertragung und Anwendung der Methode der Kostenmodellierung auf die Modellierung und Simulation mit SystemC bzw. SystemC-AMS.

Danksagung

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

Literatur

[1] Stammermann, A.; u. a.: *ORINOCO: Verlustleistungsanalyse und Optimierung auf der algorithmischen Abstraktionsebene*. Entwurf Integrierter Schaltungen, 10. E.I.S.-Workshop, Dresden, 2001

[2] Schmitt, H.: *High-Level Bewertung der Leistungsaufnahme sequentieller Schaltungen*. Entwurf Integrierter Schaltungen, 10. E.I.S.-Workshop, Dresden, 2001

[3] Heuschen, F.; Grimm, C.; Waldschmidt, K.: *Modellierung des Implementationsraumes im Analog/Digital Co-Design*. 3. ITG/GI/GMM-Workshop. Forschungs-Report Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Systemen und Schaltungen, Frankfurt am Main, 2000

[4] Schneider, A.; Schneider, P.; Bastian, J.: *MOSCITO-Ein modulares, internetbasiertes Programmsystem für die Optimierung von Mikrosystemen.* Statusseminar zum Verbundprojekt OMID Optimierung von Mikrosystemen für Diagnose- und Überwachungsanwendung, Bremen, 2001

[5] Peters, D.; Bolte, H.; Laur, R.: *Designoptimierung von Mikrosystemen mit MODOS*. Statusseminar zum Verbundprojekt OMID Optimierung von Mikrosystemen für Diagnose- und Überwachungsanwendung, Bremen, 2001

[6] Teich, J.: *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. Springer Verlag, 1997

[7] Schlegel, M.; Herrmann, G.; Müller, *D.:* *Ein System-Level Modell in VHDL-AMS eines mikromechanischen Vibrationssensor-Arrays*. Scientific Reports, Journal of the University of Applied Science Mittweida, 15th International Scientific Conference Mittweida, Nr. 10, 2002, Mittweida, Germany, 07.-11. November 2002