Anforderungen an VHDL-AMS-Simulatoren

(Entwurf vom 1. Juli 2003)

Bearbeiter: Joachim Haase

Fraunhofer-Institut für Integrierte Schaltungen
Außenstelle EAS Dresden

Zeunerstr. 38
01069 Dresden

Tel.: 0351 - 4640 736
Fax: 0351 - 4640 703
E-Mail: Joachim.Haase@eas.iis.fhg.de

Versionen:		
	25. Juni 2003	1. Entwurf
	1. Juli 2003	Überarbeitung unter Berücksichtigung der Hinweise von H. Gall und D. Warning

1. Ziel

Zum gegenwärtigen Zeitpunkt sind noch nicht alle standardisierten VHDL-AMS-Sprachkonstrukte [1] in verfügbaren Simulatoren vollständig implementiert. Trotzdem wird beabsichtigt, für Anwendungen in der Fahrzeugtechnik mit der Erstellung einer VHDL-AMS-Modellbibliothek zu beginnen. Um die entstehenden Modelle in unterschiedlichen Simulationsumgebungen einsetzen zu können, wird im folgenden versucht, eine (minimale) Auswahl für die Modellentwicklung unbedingt notwendiger VHDL-AMS-Sprachkonstrukte zusammenzustellen. Diese Konstrukte sollten in den einzusetzenden Simulatoren verfügbar sein. Unterschiedliche Prioritäten für die geforderte Verfügbarkeit von Sprachkonstrukten ergeben sich aus dem beabsichtigten Vorgehen beim Aufbau der VHDL-AMS-Modellbibliothek.

2. Modellierungsansätze

Typische Ansätze für die mathematische Modellbildung in der Fahrzeugtechnik (siehe z. B. [2], Abschnitt 4.1.2) sind:

- Netzwerkmodelle
- Signalflußmodelle
- Digitalmodelle (insbesondere Automatengraphen)
- Gemischt analog-digitale Modelle

Im Anhang A1 sind wichtige VHDL-AMS-Sprachkonstrukte zusammengestellt, die für die Implementierung entsprechender Modelle erforderlich sind. Es ist dabei versucht worden, die Wichtigkeit der Konstrukte für die Bibliothekserstellung zu werten. Zu Beginn sollten auf alle Fälle die Konstrukte mit der Priorität 1 verfügbar sein. Die Anforderungen sollen im folgenden kurz begründet werden.

Netzwerkmodelle

Netzwerke können zur Modellierung elektrischer und nichtelektrischer Systemkomponenten verwendet werden. Netzwerke werden durch Knoten, Zweige und die Beziehungen zwischen Fluß- und Differenzgrößen der Zweige beschrieben. Eventuell sind dazu noch zusätzliche Unbekannte in das beschreibende Gleichungssystem aufzunehmen (in VHDL-AMS entsprechen diesen Größen die *free quantities*). Die Beziehungen zwischen den Fluß- und Differenzgrößen der Zweige und den zusätzlichen Größen werden mit algebraischen Gleichungen und/oder gewöhnliche Differentialgleichungen ausgedrückt. Netzwerkmodelle werden im VHDL-AMS-Standard auch als konservative Modelle bezeichnet.

Unbedingt verfügbar sein müssen folgende Sprachkonstrukte und Packages:

- Deklaration skalarer Klemmen und Knoten (`TERMINAL`)
- Verfügbarkeit des `STANDARD` Packages der `STD` Bibliothek
- Unterstützung der voraussichtlich Ende 2003 standardisierten IEEE-Packages mit den Deklarationen für elektrische und nichtelektrische `NATURE`s (Packages `ELECTRICAL_SYSTEMS`, ...)
- Deklaration von (skalaren) *branch quantities* und (skalaren und vektoriellen) *free quantities*

- Verfügbarkeit der *simultaneous statements* (*simple simultaneous statements*, `IF ...` USE-Statement, CASE-Statement, ...) mit Ausnahme des `PROCEDURAL` *simultaneous statements*
- Unterstützung der auf *quantities* anzuwendenden Attribute (`'DOT, 'INTEG, 'LTF, 'ZTF, ...`)
- Unterstützung der Mehrfachanwendung von Attributen, insbesondere von `'DOT` (d. h. `'DOT'DOT, ...`)
- Verfügbarkeit des IEEE-Packages `MATH_REAL` mit elementaren mathematischen Funktionen und Konstanten
- Unterstützung der Verwendung von (`pure` und `impure`) Funktionen zur Beschreibung von Algorithmen und der erforderlich *sequential statements* (`IF... THEN`-Statement, `LOOP`-Schleifen, `WHILE`-Schleifen, ...) und Deklarationen (`VARIABLE`-Deklaration, ...)
- Unterstützung der Arbeit mit Packages und logischen Bibliotheken (`work` *library* und *resource libraries*)

Teilweise liegen Netzwerkbeschreibungen auch als Spice-Netzlisten vor. Die Einbeziehung von Spice-Netzlisten ist nicht Bestandteil des VHDL-AMS-Standards. Trotzdem ist es wünschenswert, Spice-Unternetzwerke (SUBCircuits) oder Spice-Primitive in geeigneter Form in Modellbeschreibungen einzubeziehen. Die entsprechenden Lösungen werden beim augenblicklichen Stand aber simulatorspezifisch sein. Hier besteht ein Bedarf nach Standardisierung.

Zu einem späteren Zeitpunkt erscheint die Unterstützung der Beschreibung von Netzwerken mit vektoriellen Klemmen sinnvoll zu sein. Dann ist zusätzlich die Verfügbarkeit folgender Sprachkonstrukte und Packages erforderlich:

- Deklaration vektorieller Klemmen und Knoten (zunächst wenigstens von *constrained arrays*, später eventuell auch von *unconstrained arrays*)
- Unterstützung der Arbeit mit mehrdimensionalen Feldern, insbesondere reellwertigen Matrizen
- Möglichkeit zum Überladen von Operatoren (z. B. “*“ und “+“ für Matrix-Vektor-Operationen)
- Verfügbarkeit des IEEE-Packages `MATH_COMPLEX`
- Ggf. Unterstützung von `ALIAS` zur einfacheren Formulierung von Matrix-Vektor-Operationen)

Aus heutiger Sicht sollte die Verwendung von `RECORD-NATURE`s zunächst bei der Formulierung von Modellen vermieden werden.

Signalflußmodelle (Blockdiagramme)

Signalflußdiagramme dienen z. B. der Beschreibung von Regelungssystemen. Signalflußmodelle werden im VHDL-AMS-Standard auch als nichtkonservative Modelle bezeichnet.

Signalflußblöcke lassen sich in der Regel auch unter Verwendung idealer gesteuerter Spannungsquellen modellieren. Trotzdem ist es wünschenswert, auch mit Blick auf eventuell nach VHDL-AMS umzusetzende MATLAB/Simulink-Beschreibungen, die Modellierung nichtkonservativer System zu unterstützen.

Verfügbar sein sollten zusätzlich zu den Sprachkonstrukten für die Netzwerkmodellierung zunächst:

- Deklaration (skalarer) Klemmen (`QUANTITY`)
- Die Anwendbarkeit von Attributen auf Klemmengrößen erscheint wünschenswert, aber nicht zwingend erforderlich.

Zu einem späteren Zeitpunkt sollte die Modellierung von Blockmodellen mit vektoriellen Klemmen unterstützt werden. Derartige Blockmodelle sind in MATLAB/Simulink-Beschreibungen verbreitet. Damit ergibt sich die Notwendigkeit zur Unterstützung der

- Deklaration vektorieller `QUANTITY`-Klemmen (zunächst wenigstens von *constrained arrays*, später eventuell auch von *unconstrained arrays*).

Digitalmodelle (insbesondere Automatengraphen)

Wünschenswert ist, daß in den verwendeten VHDL-AMS-Simulatoren der volle Umfang von VHDL zur Verfügung steht. Nur dadurch ist es gesichert, daß vorhandene VHDL-Beschreibungen (z. B. für digitale Steuerungen) in die VHDL-AMS-Simulation aufgenommen werden können.

Weit verbreitet ist die Beschreibung von digitalen Modellen mit Automatengraphen. Erforderlich sind in diesem Zusammenhang wenigstens folgende Anweisungen:

- Deklaration digitaler Klemmen und Signale (`SIGNALs`)
- Deklation von *enumerated types* (zur Beschreibung von Automaten-Zuständen)
- Unterstützung des `PROCESS`-Statements und speziell der sequentiellen Signalwertzuweisungen sowie der übrigen *sequential statements*
- Signalwertzuweisungen mit *concurrent statements* (`INERTIAL` *delay* unter Verwendung von `REJECT`, `TRANSPORT` *delay*)
- Verfügbarkeit des IEEE-Packages `STD_LOGIC_1164`

Gemischt analog-digitale Modelle

Wesentlich für VHDL-AMS-Anwendungen ist die Möglichkeit zur Simulation des gemischt analog-digitalen Verhaltens. Es sind wenige Sprachkonstrukte, die den Datenaustausch zwischen analogen und digitalen Werten unterstützen. Diese sollten im wesentlichen unterstützt werden.

Die Verfügbarkeit folgender Sprachkonstrukte ist erforderlich:

- Attribute `'RAMP`, `'SLEW` (angewendet auf Signale), `'ABOVE`
- `BREAK`-Statement zum Hinweis auf Unstetigkeitsstellen (`BREAK ON ...`)

Eine Verwendung des vollständigen BREAK-Statements mit der Möglichkeit zum Setzen von Anfangswerten und dem Setzen neuer Werte einzelner *quantities* nach Unstetigkeitsstellen sollte zunächst bei der Modellerstellung vermieden werden.

3. Modellierungsstil

Parameterprüfung mit ASSERT Statement

Der Gültigkeitsbereich der aktuellen Parameterwerte sollte im VHDL-AMS-Modell unter Verwendung einer oder mehrerer ASSERT *statements* getestet werden können.

Dazu ist die Unterstützung folgender Sprachkonstrukte erforderlich:

- Zulässigkeit passiver *concurrent statements* in einer ENTITY-Deklaration
- Ermöglichung von Wertangaben in der Zeichenkette einer REPORT-Anweisung. Das erfordert die Möglichkeit zur uneingeschränkten Verwendbarkeit des 'IMAGE-Attributes

Modellverfeinerung - Modelle unterschiedlicher Abstraktionsniveaus

Einer ENTITY-Beschreibung müssen mehrere ARCHITECTUREn zugeordnet werden können. Dazu müssen die entsprechenden organisatorischen Möglichkeiten von VHDL-AMS unterstützt werden. Unbedingt erforderlich ist die

- Möglichkeit zur direkten Instanziierung einer Design-Entity, gegeben in der Form *entity*_name (*architecture*_name).

Mit Blick auf die Weiterverwendung vorliegender digitaler VHDL-Beschreibungen erscheint die Unterstützung von CONFIGURATIONen zweckmäßig, zunächst aber nicht unbedingt erforderlich zu sein.

Verwendung generischer Modelle (Parameterangaben)

Eine Reihe von Modellen sollen durch Variation der Parameter an unterschiedliche Gegebenheiten angepaßt werden können (z. B. Batterie und Leitungsmodelle). Parameter sollen auch von Dateien gelesen werden können. Ebenso muß das Lesen der Werte von Signalquellen aus Dateien unterstützt werden können (z. B. für Fahrprofile).

Dazu sind erforderlich:

- Verfügbarkeit und Unterstützung des STD-Packages TEXTIO - Insbesondere sollte auch die File-Deklaration mit einer Zeichenkettenkonstanten parametrisierbar sein.
- Unterstützung von Attributen, die auf Felder angewendet werden können ('LOW, 'LEFT, 'HIGH, 'LENGTH, 'RANGE,...
- Unterstützung der Arbeit mit Packages und logischen Bibliotheken (work *library* und *resource libraries*), um Approximationsfunktionen (z. B. *lookup table functions*) einfach bereitstellen zu können

Verwendung generischer Modelle (Strukturbeschreibungen)

Variable Strukturbeschreibungen sind unter Verwendung des GENERATE *statements* möglich. Die Anwendung von GENERATE ist in Verbindung mit *concurrent* und *simultaneneous statements* erlaubt. In der ersten Phase der Modellerstellung sollte auf die Verwendung von GENERATE verzichtet werden.

4. Simulationsarten

Von den im VHDL-AMS-Standard [1] festgelegten Simulationsarten (siehe auch [2], Abschnitt 4.1.1) sind unbedingt erforderlich:

- Arbeitspunktberechnung (`DOMAIN = QUIESCENT_DOMAIN`)
- Zeitbereichssimulation (`DOMAIN = TIME_DOMAIN`)
- Frequenzbereichssimulation (`DOMAIN = FREQUENCY_DOMAIN`, entsprechend AC-Simulation in Spice)

Die Unterstützung der Kleinsignalrauschsimulation (entspricht der NOISE-Simulation in Spice) ist wünschenswert. In der Anfangsphase der Erstellung der Modellbibliothek sollte aber auf sie verzichtet werden können.

Zur Überprüfung, ob Grenzen für elektrische, thermische und mechanische Größen bei der Simulation einer Anordnung eingehalten werden (*stress analysis*) ist die

- Verfügbarkeit des `'ABOVE`-Attributes erforderlich

5. Simulatoranforderungen

Der Funktionsumfang der zu verwendenden Simulatoren ist nicht nur durch den implementierten VHDL-AMS-Sprachumfang festgelegt. Folgende Merkmale sind zusätzlich wünschenswert:

- Einbeziehung von Spice-Netzlisten in Modellbeschreibungen
- Einbeziehung von C-Programmbeschreibungen (z.B. über das `FOREIGN` Language Interface)
- Einfache Wiederholung von Arbeitspunktberechnungen für unterschiedliche Werte von Fluß- und Differenzgrößen („DC-Sweep“)
- Einfache Wiederholung von Simulationen für unterschiedliche Netzwerkgrößen und Parameter ohne Neustart des Simulators und neue Compilierung der VHDL-AMS-Beschreibungen. Das entspricht
  - "Alter"-Anweisung mit anschließender Simulation
  - "Parameter-Sweep"; Sicherung einer entsprechenden Funktionalität wie in marktgängigen Spice-Simulatoren

- Wiederanlauf von Simulationen nach Abbruch (Start und Restart)
- „Automatische“ Beeinflussung des Ablaufs von Simulationen in Abhängigkeit von Ergebnissen vorangegangener Simulationen (z. B. durch Verfügbarkeit einer Simulationssteuersprache)

6. Literatur

[1] IEEE Std 1076.1-1999 -
IEEE Standard VHDL Analog and Mixed Signal Extensions, 1999.

[2] Model Specification Process Standard (SAE Standard J2546). February 2002.

Anhang A1: Prioritäten für die Bereitstellung von VHDL-AMS-Statements

1. Design Units

ENTITY Declaration
 	
entity *entity*_name **is**
  [**generic (**parameter_list**);**]
  [**port (**port_list**) ;**]
  {declaration _part}
[**begin**
  {*passive*_concurrent_statement}]
end [**entity**] [*entity*_name] **;**
	
	Priority 1
	
	Priority 3: For declaration _part
	
	Priority 3: For *passive*_concurrent_statements (general)
	Priority 1: For concurrent ASSERT statement

ARCHITECTURE Body	Priority 1
Package Header **package** *package*_name **is** declaration { , ...} **end** [**package**] [*package*_name] **;**	 Priority 1 Priority 1: Declaration of types, functions, procedures, constants (also deferred constants), global terminals
Package Body **configuration**	Priority 1 Priority 3
context_clause ::= { **library** *library*_name_list **;** } { **use** selected_name {, selected_name}**;** }	Priority 1: Usage of resource libraries

2. Declarations

2. 1. Type, Subtype, and Nature declarations

type *type*_name **is** type_definition **;**
type_definition ::=
  scalar_type_definition |
  array_type_definition |
  record_type_definition |
  access_type_definition |
  file_type_definition
	
	Priority 1
	Priority 2: Esp. real matrices
	Priority 4
	Priority 5
	Priority 3: For user-defined file type

subtype *subtype*_name **is** *type*_name [constraint] [tolerance_aspect] **;**	Priority 2

nature *scalar*_nature_name **is**
  *real_acrosstype*_name **across**
  *real_throughtype*_name **through**
  *reference_node*_name **reference ;**
 	
nature *array*_nature_name **is**
  array (index_range**) of** nature_name **;**
 	
nature *record*_nature_name **is record**
  nature_element_declaration
  { nature_element_declaration }
end record ;
	Priority 1: To allow recomiltaion of "NATURE"-Packages from different EDA vendors
	
	Priority 2
	
	Priority 4

2. 2. Object declarations

constant name_list **:***type_or_subtype*_name **:=** expression **;**	Priority 1
[**shared**] **variable** name_list **:***type_or_subtype*_name [**:=** expression] **;**	Priority 1 Priority 3: For shared variables
signal name_list **:** *type_or_subtype*_name [**:=** expression] **;**	Priority 1
file name_list **:** *file*_type [[**open** open_kind] **is** file_logical_name **];**	Priority 1: For *file*_type defined in STD.TEXTIO string-constant should be allowed as file_logical_name
terminal name_list**:** nature_name**;**	Priority 1
quantity name_list **:** *real_type*_name [1)] [**:=** expression] **;**	Priority 1
quantity [across_aspect] [through_aspect] terminal_aspect **;**	Priority 1
quantity name_list **:** *real_type*_name [1)] **spectrum** *magnitude*_expr, *phase*_expr **;**	Priority 1
quantity name_list **:** *real_type*_name [1)] **noise** power_expr **;**	Priority 3

limit quantity_list **:** *quantity*_type **with** *real*_expression;	Priority 3

2. 3. (Port) Interface declarations

interface_signal_declaration ::= [**signal**] name_list **:** [**in** | **out** | **inout** | **buffer**] type_name [**:=** static_expression]	Priority 1
interface_terminal_declaration ::= **terminal** name_list **:** nature_name	Priority 1
interface_quantity_declaration ::= **quantity** name_list **:** [**in** | **out**] *real_subtype*_name [**:=** static_expression]	Priority 1
port (interface_declaration { **;** interface_declaration } **) ;**	Priority 1

2. 4. Component declaration

component identifier [**is**] [**generic (**generic_list**) ;**] [**port (**port_list**) ;**] **end component** [identifier] **;**	Priority 3

2. 5. Subprograms

procedure *procedure*_name [**(** formal_parameter_list **)**] **is** { declaration_part } **begin** { sequential_statement } **end** [**procedure**] [procedure_name] **;**	Priority 2

[**pure** | **impure**] **function** *function*_name **return** type | subtype **is** { declaration_part } **begin** { sequential_statement } **end** [**function**] [*function*_name] **;**	Priority 1

4. Concurrent statements

[*process*_label **:**] **process** [**(** signal_name { , ...} **)**] [**is**] { *process*_declaration } **begin** { sequential_statement } **end process** [*process*_label] **;**	Priority 1
[label **:**] *procedure*_name [**(**actual_parameter_list**)**] **;**	Priority 2
[label **:**] **assert** *boolean*_expression [**report** string_expression] [**severity** severity_level] **;**	Priority 1
[label **:**] *signal*_name **<=** [[**reject** *time*_expression] **inertial**] waveform**;**	Priority 1
[label **:**] *signal*_name **<= transport** waveform**;**	Priority 1
[label **:**] *signal*_name **<=** [delay_mechanism] { waveform **when** boolean_expr **else** } waveform [**when** boolean_expr];	Priority 1 ... 2
[label **:**] **with** expression **select** *signal*_name **<=** [delay_mechanism] { waveform **when** choice { **|** choice} **,** } waveform **when** choice { **|** choice} **;**	Priority 1 ... 2

label **:** **entity** *entity*_name [**(***architecture*_name**)**] [**generic map (**generic_association_list**)**] [**port map (**port_association_list**)**] **;**	Priority 1
label **:** [**component**] *component*_name [**generic map (**generic_association_list**)**] [**port map (**port_association_list**)**] **;**	Priority 3
[label **:**] **break** [break_list] [sensitivity_list] [**when** condition] **;**	Priority 1: **break on** signal Priority 3: Full BREAK statement

5. Simultaneous statements

[label **:**] expression **==** expression [tolerance_aspect]**;**	Priority 1
[label **:**] **if** condition **use** { simultaneous_statement } { **elsif** condition **use** { simultaneous_statement } } [**else** { simultaneous_statement }] **end use** [label] **;**	Priority 1
[label **:**] **case** expression **use** **when** choice { **|** choice } **=>** { simultaneous_statement } { **when** choice { **|** choice } **=>** { simultaneous_statement } } **end case** [label] **;**	Priority 2 ... 3
[label **:**] **procedural** [**is**] { declaration_part } **begin** { sequential_statement } **end procedural** [label] ;	Priority 4

6. Sequential statements

[label **:**] **wait** [**on** signal_list] [**until** condition] [**for** *time*_expression] **;**	Priority 1
[label **:**] **assert** *boolean*_expression [**report** string_expression] [**severity** severity_level] **;**	Priority 1
[label **:**] **report** string_expression [**severity** severity_level] **;**	Priority 1
[label **:**] *signal*_name **<=** [[**reject** *time*_expression] **inertial**] waveform**;**	Priority 1
[label **:**] *signal*_name **<= transport** waveform**;**	Priority 1
[label **:**] *variable*_name **:=** expression **;**	Priority 1
[label **:**] *procedure*_name [**(**actual_parameter_list**)**] **;**	Priority 2
[label **:**] **if** condition **then** { sequential_statement } { **elsif** condition **then** { sequential_statement } } [**else** { sequential_statement }] **end if** [label] **;**	Priority 1
[label **:**] **case** expression **is** **when** choice { **|** choice } **=>** { sequential_statement } { **when** choice { **|** choice } **=>** { sequential_statement } } **end case** [label] **;**	Priority 1: For state transition diagrams
[label **:**] **loop** { sequential_statement } **end loop** [label] **;**	Priority 1

[label **:**] **while** condition **loop** { sequential_statement } **end loop** [label] **;**	Priority 1
[label **:**] **for** identifier **in** range **loop** { sequential_statement } **end loop** [label] **;**	Priority 1
[label **:**] **break** [break_list] [**when** condition] **;**	Priority 3

7. Attributes

7. 1. Attributes on types

*type*_name**'LOW**	Priority 2
*type*_name**'HIGH**	Priority 2
*type*_name**'LEFT**	Priority 2
*type*_name**'RIGHT**	Priority 2
*type*_name**'POS(**x**)**	Priority 3
*type*_name**'PRED(**x**)**	Priority 3
*type*_name**'SUCC(**x**)**	Priority 3
*type*_name**'VAL(**n**)**	Priority 3
*type*_name**'VALUE(**s**)**	Priority 3
*type*_name**'IMAGE(**x**)**	Priority 1
*real_type*_name'**TOLERANCE**	Priority 4

7. 2. Attributes on arrays

*array*_name**'LOW** [**(**n**)**]	Priority 1
*array*_name**'HIGH** [**(**n**)**]	Priority 1
*array*_name**'LEFT** [**(**n**)**]	Priority 1
*array*_name**'RIGHT** [**(**n**)**]	Priority 1
*array*_name**'RANGE** [**(**n**)**]	Priority 1
*array*_name**'REVERSE_RANGE** [**(**n**)**]	Priority 3
*array*_name**'ASCENDING** [**(**n**)**]	Priority 3
*array*_name**'LENGTH** [**(**n**)**]	Priority 1

7. 3. Attributes on signals

*signal*_name**'DELAYED** [**(**T**)**]	Priority 1
*signal*_name**'STABLE** [**(**T**)**]	Priority 1
*signal*_name**'EVENT**	Priority 1
*signal*_name**'LAST_EVENT**	Priority 1
*signal*_name**'LAST_VALUE**	Priority 1
*real_signal*_name**'RAMP (**trise, tfall**)**	Priority 1
*real_signal*_name**'SLEW (**up, down**)**	Priority 1

7. 4. Attributes on quantities

*quantity*_name**'DOT**	Priority 1
*quantity*_name**'INTEG**	Priority 1
*quantity*_name**'DELAYED** [**(**T**)**]	Priority 1
*quantity*_name**'SLEW** [**(**maxrise [**,**maxfall]**)**]	Priority 1
*quantity*_name**'LTF (**num **,** den**)**	Priority 1
*quantity*_name**'ZOH (**Tsampl [**,** init_delay] **)**	Priority 1
*quantity*_name**'ZTF (**num**,** den**,** t [**,**init_del]**)**	Priority 1
*quantity*_name**'ABOVE (**level**)**	Priority 1
*quantity*_name**'TOLERANCE**	Priority 4

7. 5. Attributes on natures and terminal

*nature*_name'**ACROSS**	Priority 3
*nature*_name'**THROUGH**	Priority 3
*terminal*_name'**REFERENCE**	Priority 2
*terminal*_name'**CONTRIBUTION**	Priority 3

8. Miscellaneous

now	Priority 1
frequency	Priority 1
domain	Priority 1

label**: for** parameter_specification **generate** [{block_declarative_item} **begin**] { simultaneous_statement | concurrent_statement } **end generate** [label] **;**	Priority 4
- - comment	Priority 1
Packages STD.STANDARD STD.TEXTIO IEEE.MATH_REAL IEEE.MATH_COMPLEX IEEE.STD_LOGIC_1164	 Priority 1 Priority 1 Priority 1 Priority 3 Priority 1
Consideration of standard set of **nature** declarations (Proposal of Standard Packages Working Group; IEEE ballot will start 2003; http://www.vhdl.org/analog/)	Priority 1

Anhang A2: Testprogramme

Die folgenden VHDL-AMS-Programme können verwendet werden um zu prüfen, ob bestimmte Sprachkonstrukte von einem VHDL-AMS-Compiler ausgewertet werden. Sie dienen nicht zur Prüfung der korrekten Arbeitsweise eines Simulators.

Sprachkonstrukte zur Beschreibung von Design Units

```
-- ------------------------------------------------
-- File:      Part1.vhd
-- Contents:  Syntax check (design units)
-- ------------------------------------------------

library IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;

entity comparator is
   generic (level : REAL    := 2.5);
   port      (terminal a, ref : ELECTRICAL;
              signal     d        : out BIT);
begin
   assert level > 0.0
      report "Level must be > 0.0"
      severity ERROR;
end entity comparator;


architecture simple of comparator is
   quantity v across i through a to ref;
begin
   v == 1.0E6*i ;
   d <= '1' when v'ABOVE(level) else '0' ;
end architecture simple ;


package my_functions is
   function boolean2bit (b : BOOLEAN)  return BIT;
end package my_functions;

package body my_functions is
   function boolean2bit (b : BOOLEAN)  return BIT is
   begin
       if b = TRUE then return '1'; else return '0'; end if;
   end function boolean2bit;
end package body my_functions;
-------------------------------------------------------
entity reg_array is end entity reg_array;

architecture structural of reg_array is
   component inv
      port (signal i, o : bit);
   end component;
   component reg
      port (p : bit);
   end component;
   signal a, b, c, d, e, f : bit;
begin
UUT: inv port map (a, b);
UU1: inv port map (c, d);
UU2: inv port map (e, f);
UU3: reg port map (a);
end architecture structural;

entity inverter is
   port (inp, outp : bit);
end entity inverter;

architecture str of inverter is
begin
end architecture str;

architecture bhv of inverter is
begin
end architecture bhv;

entity my_register is
   port (p : bit);
end entity my_register;
-------------------------------------------------------
library WORK;
    use WORK.all;

configuration conf1 of reg_array is
 for structural
for UUT : inv   use entity  inverter(bhv)
port map (inp => i, outp => o);
end for;
for others : inv use entity  inverter(str)
port map (inp => i, outp => o);
end for;
for all : reg  use entity WORK.my_register ; end for;
 end for;
end configuration conf1;
```

Sprachkonstrukte für Typdeklarationen

```
-- ------------------------------------------------
-- File:      Part2.vhd
-- Contents:  Syntax check (type declarations)
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

library STD;
    use STD.STANDARD.all;

package types is

    type INTEGER is  range -2147483648 to 2147483647;
    type REAL is range -1.0E38 to 1.0E38;
    type word_index is range 31 downto 0 ;

    type BIT is ('0', '1') ;      -- enumeration type definition
    type BOOLEAN is (TRUE, FALSE) ;

    type BIT_VECTOR is array (NATURAL range <>) of BIT;
    type word is array (31 downto 0) of BIT ;
    type truth_table is array (BIT, BIT) of BIT ;

    type COMPLEX is record
            RE, IM: REAL ;
    end record COMPLEX;

    type word_index_ptr is access word_index ;
    type TEXT is file of STRING ;

    subtype byte_index is INTEGER range 0 to 7 ;
    subtype VOLTAGE is REAL tolerance "default_voltage" ;

    -- ------------------------------------------------
    subtype CURRENT is REAL tolerance "default_voltage" ;
    -- ------------------------------------------------

    nature ELECTRICAL is
         VOLTAGE across
         CURRENT  through
         ELECTRICAL_REF reference;

    nature ELECTRICAL_VECTOR is
         array (NATURAL range <>) of ELECTRICAL ;

    nature electrical_bus is record
        strobe : ELECTRICAL ;
        data   : ELECTRICAL_VECTOR (7 downto 0);
    end record ;

end package types;
```

Sprachkonstrukte für Objektdeklarationen

```
-- ------------------------------------------------
-- File:      Part3.vhd
-- Contents:  Syntax check (object declarations)
-- ------------------------------------------------

library IEEE, IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;
    use IEEE.MATH_REAL.all;
    use STD.TEXTIO.all;

entity object_test is end entity object_test;

architecture bench of object_test is

    -- terminals -----------------------------------------------------
    terminal a, b, inp, inp_ctrl, ref : electrical;
    terminal anode, cathode : electrical;
    -- --------------------------------------------------------------

    -- TEXT ----------------------------------------------------------
    type TEXT is file of string;
    -- --------------------------------------------------------------


    constant clk_period : TIME := 20 ns ;
    constant data  : BIT_VECTOR    := B"1010_0010" ;
    constant coeff : REAL_VECTOR := (2.0, 3.1) ;
    signal s : BIT_VECTOR (15 downto 0) ;
    signal clk : BIT := '1';
    signal reset, strobe, enable : BOOLEAN ;
    file file4 : TEXT open WRITE_MODE is "intdata";
    terminal t1, t2 : ELECTRICAL ;
    quantity q1, q2 : REAL ;
    quantity qv : REAL_VECTOR (1 to 4) := (2 => 1.0, others => 0.0) ;
    quantity v across i through a to b;
    quantity vin across inp to ref;
    quantity vctrl across inp_ctrl;
    quantity vd across id, ic through anode to cathode;
    quantity ac : REAL spectrum 1.0, 90.0;
    quantity ac_in : REAL spectrum 1.0, 0.0;

    -- constants -----------------------------------------------------
    constant af : real := 1.0;
    constant kf : real := 0.0;
    -- --------------------------------------------------------------

    quantity fl_noise_pow : REAL noise kf*id**af/FREQUENCY ;

    -- quantities ----------------------------------------------------
    quantity v_t1 across i_t1 through t1;
    quantity v_t2 across i_t2 through t2;
    -- --------------------------------------------------------------

    -- limit ---------------------------------------------------------
    subtype VOLTAGE is REAL tolerance "default_voltage" ;
    constant cf : real := 1.0e3;
    quantity v_limit : VOLTAGE;
    -- --------------------------------------------------------------
    limit v_limit : VOLTAGE with 0.05/cf;

begin

    process is
        variable sum : REAL := 0.0 ;
        variable state_a, state_b  : BOOLEAN ;
    begin
        wait;
    end process;

    q1 == 1.0;
    q2 == 2.0;
    qv == (0.0, 0.0, 0.0, 0.0);
    v  == 1.0;
    vin == 1.0;
    id == 1.0;
    ic == 1.0;
    -- --------------------------------------------------------------
    v_t1 == 1.0;
    v_t2 == 1.0;
    id   == 1.0;
    v_limit == 1.0;

end architecture bench;
```

Sprachkonstrukte für ENTITY-Deklarationen und concurrent statements

```
-- ------------------------------------------------------
-- File:      Part4.vhd
-- Contents:  Syntax check (Interfaces/Concurrent Statements)
-- ------------------------------------------------------

library IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;

entity resistor is
    generic (res_value : real := 1.0);
    port    (terminal p, m :electrical);
end entity resistor;

architecture simple of resistor is
begin
end architecture simple;


library IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;
    use WORK.all;

entity test_port is
port (         clk, d : in BIT ; terminal p, m : ELECTRICAL ;
      quantity qi     : in REAL );
end entity test_port;

-- Compilation of Procedures/Functions ----------------------------

architecture a1 of test_port is
    component flipflop
       generic ( JtoQ_delay : TIME) ;
       port (j, k : in BIT; q, q_bar : out BIT) ;
    end component ;

    procedure check_setup (signal clk, data : in BIT) is
    begin
       if clk = '1' then
          assert data'LAST_EVENT >= 3 ns
                 report "Setup time violation" severity NOTE;
       end if;
    end procedure check_setup ;

    function f1 (x1, x2, x3 : BIT) return BIT is
       variable y : BIT;
    begin
       y := ( (not x1) or (x2 and x3)) ;
       return y ;
    end function f1 ;
begin
end architecture a1;

-- Concurrent statements ------------------------------------------

architecture a2 of test_port is

    component flipflop
       generic ( JtoQ_delay : TIME) ;
       port (j, k : in BIT; q, q_bar : out BIT) ;
    end component ;

    procedure check_setup (signal clk, data : in BIT) is
    begin
       if clk = '1' then
          assert data'LAST_EVENT >= 3 ns
                 report "Setup time violation" severity NOTE;
       end if;
    end procedure check_setup ;

    signal a, b, phi, data, res, eoc, y1, y2, y3, z :bit;
    signal d0, d1, s1, s2, z1, in_1, in_2, out_1 : bit;
    signal s, enable : bit;
    signal sel : bit_vector (1 to 2);

    terminal t1, t2 : electrical;
    quantity q : real;

-- ---------------------------------------------------------------

begin

    p1a : process (a, b)
    begin
       s <= a  xor b;
```

```
    end process p1a;

    p1b : process is
    begin
       s <= a  xor b;
       wait on a, b;
    end process p1b;

    check_setup (clk => phi, data => enable) ;

    assert  not (res = '1' and eoc = '1')
           report "res and eoc can't be 1 at the same time"
           severity WARNING ;

    y1 <= not a ;
    y2 <= reject 2 ns inertial not a after 5 ns ;

    y3 <= transport not a after 5 ns ;

    z <= d0 when s1 = '0' and s2 = '0' else
         d1 when s1 = '0' and s2 = '1' else
         d1 when s1 = '1' and s2 = '0' else
         d0 when s1 = '1' and s2 = '1' ;

    sel <= s1 & s2 ;
    with sel select
         z1 <= d0 when "00" | "11" ,
               d1 when others;

res1 : entity resistor (simple)
               generic map (res_value => 1.0E3)
               port map (p => t1, m => t2) ;

C : flipflop
              generic map (20 ns)
              port map (in_1, in_2, out_1, open) ;

break for q use q => 2.0 on s ;

-- ---------------------------------------------------------------

q == 1.0;

end architecture a2;
```

Sprachkonstrukte für simultaneous statements

```
-- ------------------------------------------------
-- File:      Part5.vhd
-- Contents:  Syntax check (Simultaneous statements)
-- ------------------------------------------------

library IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;

entity test_port is
port (clk, d: in BIT ; terminal p, m : ELECTRICAL ;
        quantity qi : in REAL );
end entity test_port;


architecture a3 of test_port is

    terminal n : electrical;
    quantity v1 across i through n;
    quantity f across x through n;
    quantity vin across n;
    quantity v : real;
    quantity verr : real;
    quantity vout : real;

    type my_logic is ('0', '1', 'X', 'Z');
    constant mass, k, ron, roff, vmax, vlo, vhi, vx  : real := 1.0;
    constant res_value : real := 1.0;
    constant inp : real_vector (1 to 4) := (others => 1.0);
    signal din : my_logic;

begin

    v1 == res_value * i ;
    f  == mass*x'DOT'DOT + k*x ; -- tolerance "mechanical_mst";

    if      vmax = REAL'RIGHT use verr == vin ;
    elsif  vin > vmax         use verr == vmax ;
    elsif  vin < -vmax        use verr == -vmax ;
    else                          verr == vin ;
    end use ;

    case din use
       when '0'  => v == ron * i + vlo;
       when '1'  => v == ron * i + vhi;
       when 'X' =>  v == ron * i + vx;
       when 'Z'  => v == roff * i + vx;
    end case ;

    procedural is
       variable sum : real := 0.0 ;
    begin
       for i in inp'RANGE loop
           sum := sum + inp(i);
       end loop;
--     vout := sum;
    end procedural ;

end architecture a3;
```

Sprachkonstrukte für sequential statements

```
-- ------------------------------------------------
-- File:      Part6.vhd
-- Contents:  Syntax check (sequential statements)
-- ------------------------------------------------

library IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;
    use STD.TEXTIO.all;

entity test_port is
port (clk, d: in BIT ; terminal p, m : ELECTRICAL ;
      quantity qi : in REAL );
end entity test_port;


architecture a4 of test_port is
     constant res_value : real := 1.0;
     signal y1, a, y2, y3 : bit;
     signal q_bit : bit;
     quantity q1 :real;
     quantity q2 :real;

     file in_file  : TEXT open READ_MODE is "intdata";
     constant name : string := "outdata";
     file out_file : TEXT open WRITE_MODE is name;

begin

process is
     variable count : integer := 0;
     variable vmax  : real := 1.0;
     constant a_real , b_real : real := 1.0;
     variable int : integer := 1;
     variable v : integer;
     type integer_vector is array (integer range <>) of integer;
     variable vector : integer_vector (1 TO 5);

     procedure report_max_and_sum (samples:real) is
     begin
     end procedure;
     variable samples : real;

     variable in_line : line;

begin
-- ---------------------------------------------------------------------------
     wait until clk = '1' for 1.0E-3;
     assert  res_value >= 100.0
           report "res_value is too small"
           severity ERROR ;
     report "Entering function f1" ;
     y1 <= not a ;
     y2 <= reject 2 ns inertial not a after 5 ns ;

     y3 <= transport not a after 5 ns ;
     count := count + 1 ;
     report_max_and_sum (samples) ;

     max_ab : if a_real > b_real then
          vmax := a_real ;
     else
          vmax := b_real ;
     end if max_ab ;

     case int is
        when 0           => null;
        when 1 | 2 | 7   => v := 6;
        when 3 to 6      => v := 8;
        when others      => v := 0;
     end case;
     L : loop
        wait until clk = '1';
        q_bit <= d after 5 ns ;
        exit L when NOW > 100 ms ;
     end loop L;

     while not ENDFILE (in_file) loop
         READLINE (in_file, in_line);
         WRITELINE (out_file, in_line) ;
     end loop;

     for i in 15 downto 0 loop
          vector(i) := i * 2;
     end loop ;
```

```
     break for q1 use q1 => 2.0;

end process;


q1'dot == 1.0;
q2'dot == 0.5;

-- initial values

break q1 => 1.0;
break q2 => 1.8;

-- break on signal;

break on clk;

-- new initial values after discontinuity

break q2 => 3.0 on clk when d='1';

end architecture a4;
```

Verwendung von GENERATE

```
-- ------------------------------------------------
-- File:      Part7.vhd
-- Contents:  Syntax check (generate)
-- ------------------------------------------------

library IEEE_proposed;
    use IEEE_proposed.ELECTRICAL_SYSTEMS.all;

entity test_port is
port (clk, d: in BIT ; terminal p, m : ELECTRICAL ;
      quantity qi : in REAL );
end entity test_port;

architecture a5 of test_port is

quantity q : real_vector (1 to 4);
constant a : real_vector (1 to 4) := (others => 1.0);

begin

c : for i in 1 to 4 generate
         q(i)'DOT + a(i)*q(i) == q(i-1);
     end generate ;

end architecture a5;
```