VHDL-AMS Subset for REAL-TIME

Eduard Moser, Robert Bosch GmbH, moser@fli.sh.bosch.de

August 13, 1999

0 Introduction

The following syntax and additional comments describe the VHDL-AMS (IEEE 1076.1) language subset for REAL-TIME (in short VHDL-AMS-RT). All items which are not supported are deleted (~~item~~). All items which can be ignored are underlined (item).

The VHDL-AMS-RT is based on the [1] and the section numbering corresponds to this document.

This VHDL-AMS-RT definition does not include an explanation of the semantical concepts of VHDL-AMS.

This document is a result of the EC funded projects TOOLSYS[1] and ODECOMS[2].

The VHDL-AMS-RT can only include language concepts which would do not contradict the following precondition:

> *The required calculation time for each time step must stay below some predictable maximum time.*

0.1 Procedural Code

Most semantic constructs of the language have corresponding constructs in regular procedural programming languages (e.g. C). The code for these constructs can be easily converted. This includes the subprograms (procedures and functions) and most of the sequential code.

Functions within the VHDL-AMS-RT are always pure, they have no side effects (the only exception is the function `now` of the package STD). Within subprograms wait statements, signal assignment statements and most attributes are not supported (supported attributes within subprograms see §14.3).

[1] Brite-Euram Project BE3249
[2] Briet-Euram Project BPRP-CT97-567

0.2 Libraries and Packages

The libraries and packages are the primary concepts to achieve modularisation and readability and should be supported. It would be very useful to define a REAL-TIME-package which includes all the necessary definitions (constants, data structures, functions).

0.3 Time

VHDL-AMS allows different means to influence the simulation related to time. First of all the function NOW (returning the time) can be used within any expressions.

Besides the use of time within expressions, the VHDL-AMS-RT supports only two further concepts to influence the simulation. For discrete VHDL the wait statement within a process can be used (e.g. wait for 10 ms). For analog VHDL time is expressed using the derivative of a quantity.

0.4 Discrete VHDL

The VHDL-AMS-RT of the discrete VHDL is derived from [2] with additional constraints and one additional feature. This pragmatic approach was chosen, since it fits most of our needs. The discrete descriptions written according to the draft standard will be reusable for synthesis and therefore, code generation (here C code) can easily be performed.

There are some minor additional constraints (e.g., no resolution functions [§4.2]). All the other REAL-TIME specific restrictions are described in the following subsections.

The purely syntactical constraints regarding the optional extension of the keyword **end** with the corresponding keywords (e.g., **entity**) were relaxed for VHDL-AMS-RT in order to keep the language more consistent.

0.4.1 Sequential Statements (§8)

Further restrictions concerning the sequential statement part. No assertion statement, report statement [§8.2] are allowed. One additional feature is added to the subset definition [2], this feature concerns the wait statement.

Assertions (including the reporting mechanism) are very well suited for monitoring the models during simulations, but the additional time and the required input/output mechanism would it seems not appropriate for real-time applications.

The signal assignment statement does not support the delay mechanism [§8.4]. The delay mechanism has to be substituted by explicitely using wait statements (this substitution is not equivalent).

In [2] the wait statement is restricted to only one statement within each process. For the modelling within hydraulics and mechanics, it is convenient to relax this restriction. (E.g., for modelling stick/slip friction, one has to describe a finite state machine with two states. This can easily be done using one process with two wait statements.)

The sequential break statement is discussed later.

0.4.2 Concurrent Statements (§9)

Process statements, component instantiation statements, concurrent signal assignment statements and concurrent break statements are allowed as concurrent statements, all others are not supported.

The process statement allows the definition of finite state machines. The component instantiation statement is necessary for structured modelling.

The concurrent procedural call statement can be modeled as a process with the sequential procedural statement call.

Block statements are not supported.

Generate statements are very helpful to describe large systems of the same components, this feature is of little use for the modelling within hydraulics and mechanics.

0.4.3 Discrete Communication: Signals

Signals are the basic unit for communication between processes. They allow to communicate events including attached information. Signals have to be supported. The restrictions are according to [2]. The restrictions are mainly defined within the predefined environment [§14.1] and do not allow to use any history information of the signal. This makes the handling of signals much easier.

For the reasons explained below (section 0.6) no communication via signals is supported between components. Therefore, the PORT SIGNAL is not supported.

0.5 Analog and Mixed-Signal VHDL

The analog and mixed-signal part, meaning the additional features which are introduced by VHDL-AMS, describe the necessary features for modelling hydraulics, mechanics and electronics. Only very few concepts can be deleted.

0.5.1 Quantities

Quantities are continuous wave forms whose behaviour is determined by simultaneous statements [§15] (some of the quantities are state variables). Their use is restricted

according to section 0.5.2.

This document calls some quantities “to-be-determined-quantities”, these quantities are either free quantities or interface quantities of mode out (PORT (QUANTITY q: OUT real)).

0.5.2 Simultaneous Statements

The behaviour of the quantities is given by the simultaneous statements [§15].

All simultaneous statements are based on simple simultaneous statements - equations. These simple simultaneous statements are restricted to one of the following forms (expressed in mathematical notation):

Assigned Equations (ASE):

$$q_i = f_i(\ldots, q_j, \ldots)$$

Ordinary Differential Equations (ODE):

$$\dot{q}_i = g_i(\ldots, q_j, \ldots)$$

Each to-be-determined-quantity q_i is exactly on the left-hand-side of one equation of either ASE or ODE.

$f_i,\ g_i$ can depend on quantities, signals and time (function now), but f_i is independent of q_i. The functions on the right hand side are “expressions” in VHDL-AMS terminology, and represent anything which could be written as a function of the above mentioned arguments (e.g., $f(force, vel, c)$ = `force - sign(vel) * c`).

Furthermore, the derivative (attribute) ($\dot{q}$ = `q'dot`) can only appear as left-hand-side of ODE. The derivative are not allowed as argument on the right-hand-side in any equation, the integral (attribute `q'integ`) is not allowed at all.

The model is within the VHDL-AMS-RT only if the set of ASE has the following properties:

1. Each equation in the set $q_i = f_i()$ is indexed as eq_i.
2. A total order $\leq_T$ can be established such that if q_i appears in eq_j follows $eq_i \leq_T eq_j$.

(These properties guarantee that all of these equations can be evaluated as a sequence of assignment statements using the established total order $\leq_T$ starting with the smallest element - the equation eq_i for which no j exists such that $eq_j \leq_T eq_i$).

All simultaneous statements are compositions of simple simultaneous statements. For simultaneous if and case statements, the number of simultaneous statements for each branch has to be the same.

To be discussed: It seems reasonable to allow only the same equation types and same quantities on the left-hand-side for each branch.

The simultaneous procedural statement was excluded from the VHDL-AMS-RT but the rejection seem to be a short sighted decision. In many cases modelling becomes more intuitive using this form. The use of the simultaneous procedural statement should be restricted to ASE and ODE. The implementation of this additional statement with the given restrictions is straight forward.

0.5.3 Analog Communication

Only one communication mechanism is supported:

The communication is handled using interface quantities (PORT QUANTITY) which carry always their respective information direction.

Remark: Energy conserving communication is not supported within the current VHDL-AMS-RT, since its simulation needs the resolution of algebraic equations and can not always be resolved without numerical iteration. Without energy conserving communication neither terminals nor natures have to be supported.

0.5.4 Analog - Discrete Interaction

The break statement [§8.14, §9] allows the discrete kernel to reinitialise the analog kernel and to make a discontinuous resetting of individual quantities. The break statement is essential for realizing discontinuities (e.g., stick/slip friction).

The predefined attribute q'above(expression) [§14.1] is an implicit signal and will generate an event if the value of the condition (q >= expression) changes. This signal allows (like any other signal) the initiation of some action within the discrete part (e.g., wait until q'above(0.0), break on q'above(0.0)).

0.6 Simulation Cycle - Maximum Number of Delta Cycles

The maximum number of delta cycles should be predictable for each architecture.

To be discussed: Some relaxation might be necessary since delta cycles due to analog interaction are probably not predictable.

References

[1] IEEE, New York. *IEEE Standard VHDL Language Referenz Manual (Integrated with VHDL-AMS changes), Draft*, 1997.

[2] IEEE, New York. *IEEE P1076.6/D1.12 Draft Standard For VHDL Register Transfer Level Synthesis*, 1998.

1 Design entities and configurations

1.1 Entity declarations

entity_declaration ::= [§1.1]
entity identifier **is**
entity_header
~~entity_declarative_part~~
~~[**begin**~~
~~entity_statement_part]~~
end [**entity**] [entity_simple_name] ;

entity_header ::= [§1.1.1]
[formal_generic_clause]
[formal_port_clause]

generic_clause ::= [§1.1.1]
generic (generic_list) ;

port_clause ::= [§1.1.1]
port (port_list) ;

generic_list ::= generic_interface_list [§1.1.1.1]

port_list ::= port_interface_list [§1.1.1.2]

~~entity_declarative_part~~ ::= [§1.1.2]
{ entity_declarative_item }

~~entity_declarative_item~~ ::= [§1.1.2]
subprogram_declaration
| subprogram_body