# Regularization of Hierarchical VHDL-AMS Models using Bipartite Graphs

Jochen Mades
Infineon Technologies AG
Munich, Germany
Jochen.Mades@vhdl-ams.de

Manfred Glesner
Institute of Microelectronic Systems
Darmstadt University of Technology, Germany
glesner@mes.tu-darmstadt.de

## ABSTRACT

The powerful capability of VHDL-AMS to describe complex continuous systems in form of differential algebraic equations (DAEs) often leads to problems during numerical simulation. This paper presents a discrete algorithm to analyze unsolvable DAE systems and to correct the underlying hierarchical VHDL-AMS description automatically in interaction with the designer, avoiding time-consuming manual error correction.

## Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph algorithms

## General Terms

Algorithms, Design

## Keywords

DAEs, VHDL-AMS, Regularization, Structural Solvability, Bipartite Graphs

## 1. INTRODUCTION

VHDL-AMS is established as an IEEE standard (1076.1) designed to describe discrete as well as continuous behavior in a unique language. VHDL-AMS is very valuable for the simulation of complex mixed Systems-on-Chip (SoC), due to the capability to describe mixed-signal systems in one common language and the possibility to simulate the SoC with a single simulator. Therefore, the time consuming coupling of discrete simulators with continuous simulators gets unnecessary.

Apart from the simulation of SoCs, VHDL-AMS is also of great interest in the world of analog design, since it allows the description of analog systems or subsystems on a very high level of abstraction, which can lead to enormous simulation speed-ups (factor 10 to 100).

In contrast to SPICE-based simulators the intention of VHDL-AMS and especially of VHDL-AMS simulator packages is not to have only a variety of basic library elements, which are interconnected by topology descriptions like structural VHDL or SPICE-like netlists to get the final circuit behavior. The power of VHDL-AMS is to have the possibility to describe the behavior of a complex system by formulating directly the corresponding differential algebraic equations (DAEs) of the system.

However, this universality implies that standard topological tests (based on simulator libraries) like the ones utilized to check DC-solvability are not sufficient anymore. Thus, for VHDL-AMS new diagnostic tools have to be developed to identify problems caused by unsolvable systems.

For the purpose of supporting designers using VHDL-AMS to describe complex systems on a high level of abstraction, we developed an algorithm that localizes critical equations in a DAE system. Furthermore, if it is not solvable, the algorithm checks whether the system can be regularized by a special VHDL-AMS language construct - the break statement. In that case the description of the DAE system is automatically regularized in interaction with the designer, insuring that the corrected VHDL-AMS source code then leads to a solvable system.

Section 2 initially gives an overview over different types of equations appearing in VHDL-AMS. Then Section 3 introduces standard analysis methods for structural solvability before in Section 4.1 a short overview of hierarchical modeling in VHDL-AMS is given. Section 4.2 introduces the *break statement* which is used in Section 5 where the algorithm for the regularization of structural unsolvable DAE systems, described in hierarchical VHDL-AMS, is presented. Finally, the algorithm will be illustrated in Section 6 on an abstract description of a ΣΔ-analog to digital converter and first results of a prototype implementation will be shown in Section 7.

## 2. EQUATIONS IN VHDL-AMS

During the pre-simulation phase of a VHDL-AMS model, the so-called elaboration [1], the DAE system needed for numerical simulation is created. Therefore, different VHDL-AMS language constructs like *simple simultaneous statements*, *component instantiations* and *quantity declarations* have to be evaluated and corresponding characteristic expressions have to be generated.

The VHDL-AMS standard differentiates between three different sets of equations. The first set, the *explicit set*, contains all the expressions explicitly formulated in the model. The characteristic expressions of the second set, the *structural set*, are generated implicitly. These so-called *structural expressions* represent the conservative laws KCL and KVL in case of *terminal ports*, while they represent the equality of internal and external quantities in case of *quantity ports*. The third set, the *augmentation set*, provides characteristic expressions defining the explicitly used derivatives of the VHDL-AMS model. For each derivative of the VHDL-AMS model there is exactly one characteristic expression in the *augmentation*


*DAC 2002,* June 10-14, 2002, New Orleans, Louisiana, USA.

*set* and the derivative is called the *tag* of the characteristic expression. During DC-Analysis these expressions set the derivatives to zero (*quiescent state augmentation set*).

# 3. STRUCTURAL SOLVABILITY

A necessary condition for the solvability of a system of equations is the so-called *structural solvability*. The *structural solvability* focuses on the structure of the matrix (Jacobian), considering only matrix entries unequal to zero. This structure is normally represented by a bipartite graph $B$ where the nodes of the first partition of $B$ represent the rows (expressions) and the nodes of the second partition of $B$ represent the columns (quantities) of the matrix [2, 3, 4, 5]. An edge between a node $i$ of the first partition and a node $j$ of the second partition exists if the entry $(i, j)$ of the matrix is unequal to zero. If there exists a *perfect matching* in $B$, which can be determined by efficient matching algorithms [6], then the matrix is structural solvable.

In the following we denote by a *structural DC-solvable DAE system*, a DAE system for which the corresponding DC system is structural solvable.

# 4. PROPERTIES OF VHDL-AMS

## 4.1 Hierarchical Modeling

VHDL-AMS allows to model in a hierarchical manner, meaning that subsystems, the so-called *design units*, can be developed independently. A *top-level description* connects all the subsystems together to a complete *design.*

Each declaration in such a *design unit* has its own *scope.* A scope is the area inside the VHDL-AMS source code, where a declaration is visible and can be used.

The quantities of the characteristic expressions of the *explicit set* belong to the same scope, in contrast to the quantities of the *structural set* belonging to different scopes. Therefore, the presented algorithm distinguishes between nodes in the bipartite graph representing *structural expressions* and others.

## 4.2 The Break Statement

The *break statement* is an advanced VHDL-AMS language construct which allows to initialize the DAE system at the beginning of a transient simulation and to mark discontinuities during such a transient simulation. In either case a DC-Analysis of the system is performed.

The application of a *break statement* substitutes a characteristic expression of the *current augmentation set* (see Section 2) selected by the *selector quantity* of the *break statement*[1] by an expression

$$Q_{break} = |E|, \tag{1}$$

where $Q_{break}$ denotes the *break quantity* and $|E|$ the evaluated *expression* of the *break statement*. The *selector quantity* and the *break quantity* are restricted to belong to the same scope.

In the bipartite graph of the DAE system, the characteristic expression of the *augmentation set* is represented by a node of the first partition. The substitution of this characteristic expression by the application of the *break statement* is reflected in the replacement of the edge in the bipartite graph, which is incident to the node representing the characteristic expression of the *augmentation set* with an edge between that node and the node of the second partition of the bipartite graph representing the *break quantity* $Q_{break}$ of the *break statement*.

[1]There is a one to one correspondence between the *selector quantity* and the *tag* of the characteristic expressions of the augmentation set [1].

# 5. REGULARIZATION-ALGORITHM

The presented algorithm analyzes a structural DC-unsolvable DAE system described in VHDL-AMS and checks if the DAE system is regularizable, meaning if it is possible to transform it into a structural DC-solvable DAE system. To this end, a first bipartite graph $B_1 = \{V_1^1 \cup V_1^2; E\}$ of the DAE system is set up as described in Section 3. The nodes of the first partition of $B_1$ are divided into three different categories belonging to the type of the characteristic expression they represent:

$$V_1^1 = V_e \cup V_s \cup V_a$$

The nodes $v \in V_e$ represent the characteristic expressions of the *explicit set*, nodes $v \in V_s$ characteristic expressions of the *structural set* and nodes $v \in V_a$ represent characteristic expressions of the current *augmentation set* (see Section 2).

If the analyzed DAE system is structural DC-unsolvable, then the *maximal matching* in $B_1$ is not a *perfect* one (see Section 3). The aim of the presented algorithm is to replace specific edges in $B_1$ in a way that a *perfect matching* in $B_1$ can finally be determined. This edge-replacement can be performed with *break statements* (see Section 4.2). To formulate adequate *break statements* the algorithm analyzes $B_1$ in order to detect valid *break-* and *selector quantities* in the following manner.

First, the algorithm removes all nodes $v \in V_s$ from $B_1$ representing *structural expressions*, since the *break-* and *selector quantities* are restricted to belong to a common scope and because of *structural expressions* may contain unknowns from different scopes (see Section 2). Edges incident to these nodes are also removed from $B_1$. Then the *connected components* [7] of the modified bipartite graph are considered and standard matching algorithms [6] determine a *maximal matching* in every connected component in a way that no *alternating path* from an *unmatched* node $v \in V_e$ representing a characteristic expression of the *explicit set* to a *matched* node $v \in V_a$ representing a characteristic expressions of the *augmentation set* in $B_1$ exists. An *alternating path* is a path in a bipartite graph with edges belonging alternately to a given matching $M$ starting with an edge $e \notin M$ [8]. Furthermore the matching algorithm avoids *alternating paths* from *unmatched* nodes $v_u \in V_1^2$ representing quantities which are not contained in *structural expressions* to *matched* nodes $v_m \in V_1^2$ representing quantities which are contained in *structural expressions*.

In the next step the algorithm considers *matching edges* incident to nodes $v \in V_a$. If there exists an *alternating path* from the node $v \in V_1^2$, incident to such a *matching edge*, to a node representing a quantity which is contained in a *structural expression* then the *matching edge* is removed from the *connected component.* This removement leads to an *unmatched* node in both partitions of the *connected component.* The reason for that is to give the algorithm a higher degree of freedom in detecting valid *selector-* and *break quantities* as described below.

In the following step the algorithm starts building a second bipartite graph $B_2 = \{V_2^1 \cup V_2^2; E\}$ by analyzing the nodes of the *connected components* of $B_1$. Each *unmatched* node $v \in V_a$ of the first partition of the analyzed *connected component* serves for the determination of a valid *selector quantity*. Therefore, for each determined *unmatched* node $v \in V_a$ a corresponding node $sel \in V_2^1$ of the first partition of $B_2$ is created. If the algorithm detects an *unmatched* node $v \in V_e$ in the first partition of the *connected component* then the algorithm suspends without having a chance to regularize the system by specifying *break statements*. Additionally, the algorithm creates a node $brk \in V_2^2$ in the second partition of $B_2$ for every *unmatched* node $v \in V_1^2$ of the second partition of the analyzed *connected component*. The nodes $brk \in V_2^2$ serve for the determination of valid *break*

*quantities* needed for the specification of regularizing *break statements* as described later. Furthermore the nodes $v \in V_s$ representing the *structural expressions* removed at first from the first bipartite graph $B_1$ are represented by nodes $sce \in V_2^1$ of the first partition of the second bipartite graph $B_2$.

The edges of the second bipartite graph $B_2$ between nodes of the first partition and nodes of the second partition are created in the following manner. A Node $sel \in V_2^1$ is connected via an edge to a node $brk \in V_2^2$ if the *unmatched quantity* represented by the node *brk* belongs to the same scope as the *selector quantity*[2] represented by the node *sel*. A node $sce \in V_2^1$ of the first partition of $B_2$ representing a *structural expression* is connected to a node $brk \in V_2^2$ of the second partition of $B_2$ if either the *unmatched quantity* represented by *brk* is contained in the *structural expression* or if there exists an *alternating path* in the first bipartite graph $B_1$ from the node representing the *unmatched quantity* to a node representing a *quantity* which is contained in the *structural expression*.

In the next step standard matching algorithms determine a *maximal matching* in $B_2$. If this matching is a *perfect matching*, then the structural DC-unsolvable DAE system is regularizable by the specification of different *break statements*. These *break statements* are added to the VHDL-AMS source code in order to regularize the structural DC-unsolvable DAE system.

The *break statements* are determined by the evaluation of the *matching edges* of the second bipartite graph $B_2$ incident to nodes $sel \in V_2^1$. The evaluation of such a *matching edge* is performed in the following way. First the algorithm determines all nodes $v \in V_a$ in the first bipartite graph $B_1$ which can be reached by *alternating paths* in $B_1$ starting from the *unmatched* node $v \in V_a$ represented by the node *sel*. The *tags* of these determined nodes together with the *tag* of the *unmatched* node represented by the node *sel* are the valid *selector quantities* for the regularizing *break statement*. Then the algorithm evaluates the node $brk \in V_2^2$ incident to the *matching edge* in order to detect the valid *break quantities* of the *break statement*. The valid *break quantities* are the *quantities* represented by nodes $v \in V_1^2$ of the second partition of $B_1$ which can be reached on *alternating paths* starting from the *unmatched* node represented by *brk* and the *quantity* represented by that *unmatched* node itself.

In the next step the algorithm interacts with the designer asking to choose one of the determined valid *selector quantities* as well as one of the determined valid *break quantities* and to specify an expression for the corresponding *break statement* (see Section 4.2).

Then the first bipartite graph $B_1$ has to be updated in the following manner. The edge incident to the node $v \in V_a$ in $B_1$ representing the *characteristic expression tagged* by the chosen *selector quantity* is replaced by an edge between that node $v \in V_a$ and the node $v \in V_1^2$ representing the chosen *break quantity*. If either the node $v \in V_a$ selected by the *selector quantity* or the node $v \in V_1^2$ representing the *break quantity* is a *matched* node in $B_1$, then the *alternating path* from the *unmatched* node represented by the node *sel* or *brk* of $B_2$ to the selected node has to be *augmented* [8], meaning that the edges $e \in M$ of the *alternating path* are removed from the given maximal matching $M$ and the edges $e \notin M$ are added to the maximal matching $M$. This enlarges the maximal matching of $B_1$, which was originally not a perfect one. Then the algorithm continues evaluating the next *matching edge* in $B_2$.

Finally, after having evaluated all the *matching edges* the matching in the first bipartite graph $B_1$ is a *perfect matching* which represents the structural DC-solvability of the DAE system. The proof and a more detailed description of the algorithm are given in [9].

[2]The derivative of the *selector quantity* is the *tag* of the *characteristic expression* of the current *augmentation set* represented by the *unmatched* node $v \in V_a$ in the first bipartite graph $B_1$.

# 6. EXAMPLE

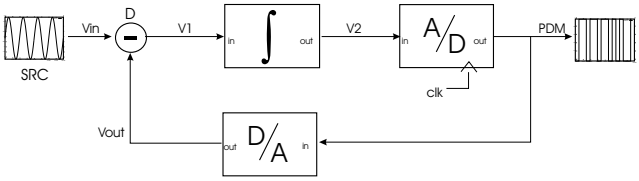


**Figure 1: Sigma-Delta Analog to Digital converter**

Figure 1 shows the block diagram of a sigma-delta analog to digital converter (ΣΔ-ADC). Such a mixed signal system can conveniently be described in VHDL-AMS. To demonstrate how the presented algorithm works we described the ΣΔ-ADC hierarchically. The continuous behavior of the sinus source *SRC*, the difference building part *D*, the integrator *I* and the 1-Bit digital to analog converter *DA* can be described with equations (e1) to (e4), respectively.

$$SRC = 2.0 * sin(2.0\pi * 5.0 \cdot 10^7 * t) \quad \text{(e1)}$$
$$D_{out} = D_{in1} - D_{in2} \quad \text{(e2)}$$
$$\frac{d}{dt} I_{out} = I_{in} \quad \text{(e3)}$$
$$DA_{out} = \pm V_{dd} \quad \text{(e4)}$$

The *structural expressions* (s1) to (s8) result from the elaboration of the complete hierarchical design of the ΣΔ-ADC.

$$V_{in} = SRC \quad \text{(s1)} \qquad V_1 = I_{in} \quad \text{(s5)}$$
$$V_{in} = D_{in1} \quad \text{(s2)} \qquad V_2 = I_{out} \quad \text{(s6)}$$
$$V_{out} = D_{in2} \quad \text{(s3)} \qquad V_2 = AD_{in} \quad \text{(s7)}$$
$$V_1 = D_{out} \quad \text{(s4)} \qquad V_{out} = DA_{out} \quad \text{(s8)}$$

For the calculation of the DC operating point the *quiescent state augmentation set* [1] is determined setting the derivatives of the DAE system to zero. In our example there is just one derivative appearing in the description of the Integrator (see equation (e3)):

$$\frac{d}{dt} I_{out} = 0.0. \quad \text{(a1)}$$

The subsystem formed by equation (s6) and (s7) leads to an unsolvable DAE system, because of the three unknowns $V_2$, $AD_{in}$ and $I_{out}$ are not contained in further equations. Figure 2(a) shows the corresponding bipartite graph with a maximal matching. Due to the maximal matching being not a perfect one - $s_2$ as well as $AD_{in}$ are unmatched - the structural unsolvability of the DAE system is confirmed.

The presented algorithm starts analyzing the VHDL-AMS description of the structural DC-unsolvable DAE system by removing the nodes representing *structural expressions* from $B_1$ (see Figure 2(b)) and by determining a *maximal matching* for each *connected component.* Then the algorithm the *matching edges* incident to nodes of the current *augmentation set* are removed under the conditions explained in Section 5. The next step of the algorithm creates the second bipartite graph $B_2$ (see Figure 3[3]) by evaluating the *unmatched* nodes of the *connected components.* There is one *unmatched* node $a_1$ representing the characteristic expression (a1) of the *augmentation set*. This node is represented as node $sel_1$ in the first partition of $B_2$ (shadowed node). Together with the eight *structural*

[3]First partition - upper lane, second partition - lower lane.

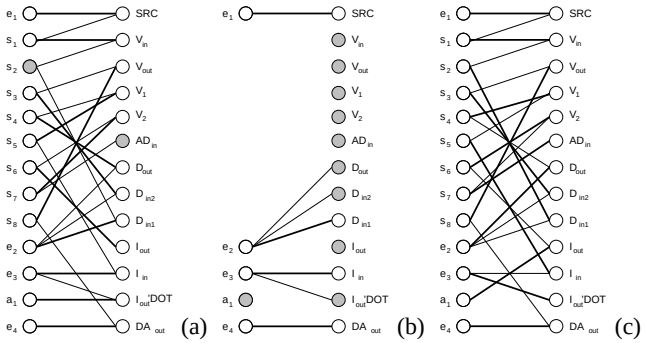


**Figure 2: Bipartite graphs of DAE Systems**

*expressions* $sce_1$ to $sce_8$ this node completes the first partition of the second bipartite graph $B_2$, while the second partition is formed by nodes representing the *unmatched* nodes of the second partition of $B_1$.

To determine whether the structural DC-unsolvable DAE system is

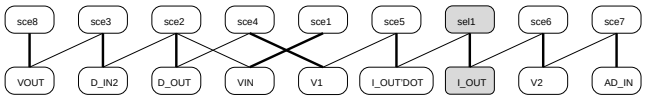


**Figure 3: Second bipartite graph used for regularization**

regularizable with *break statements* the algorithm determines a maximal matching. Because of the existence of a *perfect matching*[4] in $B_2$, the DAE system of the ΣΔ-ADC is regularizable.

The evaluation of the *matching edge* $e = \{a_1, I_{out}\}$ leads to the following *break statement*. Because there is no *alternating path* in $B_1$ from the node representing (a1) to another node representing a characteristic expression of the *augmentation set*, no choice for the *selector quantity* can be given. Therefore the *selector quantity* $I_{out}$ is determined from the *tag* of (a1). Furthermore in our example there is no *alternating path* in $B_1$ from the node representing the quantity $I_{out}$. Due to that, the *break quantity* in our example is $I_{out}$. Finally the algorithm interacts with the designer to specify an expression $|E|$ for the following *break statement*:

```
break for Iout use Iout => |E|;
```

The application of this *break statement* finally will substitute the equation (a1) with equation

$$I_{out} = |E|. \tag{2}$$

This substitution can be seen by the replacement of the edge being incident to the node representing (a1). Figure 2 (c) shows the bipartite graph corresponding to the regularized DAE system described in VHDL-AMS. The perfect matching shows the structural DC-solvability.

# 7. EXPERIMENTAL RESULTS

The presented algorithm was implemented to extend our VHDL-AMS Simulation Framework [10] to support the diagnostic of hierarchical VHDL-AMS models. Analyzing the ΣΔ-ADC example from Section 6, the algorithm interacts with the designer who specifies e.g. $|E| = 2.3 \cdot 10^{-3}$ for the *expression* of the proposed *break statement*. Then the *break statement* is added to the VHDL-AMS source code of the integrator (see Figure 4).

[4]Matching edges are illustrated bold.

```
entity integrator is
   port (quantity I_in : in  real;
 quantity I_out : out real);
end integrator;

architecture behave of integrator is
begin  -- behave

  I_out'DOT == I_in;

end behave;
```

```
entity integrator is
   port (quantity I_in : in  real;
 quantity I_out : out real);
end integrator;

architecture behave of integrator is
begin  -- behave

  I_out'DOT == I_in;
  -- inserted by BA (05.12.2001)
  break for I_OUT use I_OUT => 2.3e-3;
end behave;
```

**Figure 4: Original and corrected VHDL-AMS description**

Finally the algorithm asks to recompile the design unit and to restart the simulation with the corrected design unit of the integrator.

# 8. CONCLUSION

The presented algorithm supports the development of high level models of mixed-signal systems using VHDL-AMS. Unsolvable DAE systems resulting from VHDL-AMS descriptions are analyzed and the designer is supported in correcting the underlying VHDL-AMS source code.

A first prototype of the algorithm extends the environment of the Infineon Technologies inhouse circuit simulator TITAN [11] and was efficiently used at the Technical University of Darmstadt in the summer course "High-Level Modeling of Mixed-Signal VLSI Circuits".

# 9. REFERENCES


[1] The Institute of Electrical and Electronics Engineers, *IEEE Standard 1076.1-1999. VHDL Language Reference Manual*, IEEE, 1999.

[2] Kazuo Murota, *Matrices and Matroids for Systems Analysis*, Algorithms and Combinatorics 20. Springer Verlag, 2000.

[3] Andras Recski, *Matroids Theory and its Applications in Electric Network Theory and in Statics*, Springer Verlag, 1989.

[4] J. Unger, A. Kröner, and W. Marquardt, "Structural analysis of differential-algebraic equation systems - theory and application," in *Comput. Chem. Engng.*, 1995, vol. 19.

[5] B. Petersen, "Investigating solvability and complexity of linear active networks by means of matroids," in *IEEE Transactions on Circuits and Systems*, 1979, vol. 26.

[6] B. V. Cherkassky, A. V. Goldberg, P. Martin, J. C. Setubal, and J. Stolfi, "Augment or push? a computational study of bipartite matching and unit capacity flow algorithms," in *Technical Report 98-036R.* NEC Research Institute, Inc., 1998.

[7] O. Melnikov, R. Tyshkevich, V. Yemelichev, and V. Sarvanov, *Lectures on Graph Theory*, B.I. Wissenschaftsverlag, 1994, ISBN 3-411-17121-9.

[8] A. V. Aho, J. E. Hopcroft, and J. D. Ullmann, *Data structures and algorithms*, Addison-Wesley, 1983.

[9] Jochen Mades, *Structural Solvability of hierarchical VHDL-AMS Models*, PhD. Thesis, Darmstadt University of Technology, 2002, (in preparation).

[10] T. Schneider, J. Mades, M. Glesner, A. Windisch, and W. Ecker, "An Open VHDL-AMS Simulation Framework," in *2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation (BMAS 2000)*, Orlando, Florida, USA, October 16-18 2000, (pages missing).

[11] G. Denk, U. Feldmann, C. Hammer, M. Kahlert, R. Neubert, G. Reissig, and A. Windisch, "Erweiterung eines Standard-Schaltkreissimulators in Richtung VHDL-AMS," in *Proceedings of the 5. ITG/GMM-Diskussionssitzung, Analog 99.* Informationstechnische Gesellschaft im VDE, 1999.