관계형 데이터베이스에 기반한 버전이 지원되는 VHDL 모델의 관리 기법*

박휴찬**

A Methodology for Management of Version Supported VHDL Models Based on Relational Database

Hyu Chan Park

Abstract

VHDL has been widely used in modeling and simulation of hardware designs. However, complex relationship between components of the designs makes the VHDL modeling problem very difficult. Furthermore, after the initial creation of VHDL models, they evolve into many versions over their lifetime. To cope with such difficulties, this paper proposes a new methodology for the management of VHDL models supporting versions. Its conceptual bases are system entity structure and relational database. Within the methodology, a family of hierarchical structures of a design is organized in the form of VHDL model structure. It is, in turn, represented in the form of relational tables. Once the model structure is built in such a way, a specific simulation model which meets design objective is pruned from the model structure. The details of VHDL codes are systematically synthesized by combining it with the primitive models in a model base. These algorithms are also defined in terms of relational algebraic operations.

Key Words: VHDL, Database, Models Management, System Entity Structure

* 본 연구는 한국과학재단 목적기초연구(R05-2001-000-00851-0) 지원으로 수행되었음.

** 한국해양대학교 기계·정보공학부

1. Introduction

VHDL(Very high speed integrated circuit Hardware Description Language) is the most popular hardware description language that has been used as a modeling and simulation tool to develop hardware designs. The designs may have such information of components, interconnections among components, attributes characterizing components, and relationships among components. The fact that they are related in a complex manner makes the VHDL modeling problem very complex [1, 2, 7, 15].

To cope with the complexity, the structural information need to be separated from the behavior one and then managed hierarchically. In the hierarchical management, a complex model can be constructed just by interconnecting input and output ports of component models. The complex model may itself be employed as a component of yet more complex, hierarchical models. It is also desirable to systematically organize the models into a unified representation and then extract a specific one from the representation.

Furthermore, VHDL models may evolve into many versions over their lifetime. After the initial creation of a model, new versions of the model may be derived from it, and other new versions can be in turn derived from them, and so on. According to the kind of evolution, versions are classified into revisions and variants. Sequential versions that evolve along the time dimension are called revisions. They are created to fix bugs or perform other enhancements. Parallel or alternative versions coexisting at a given time are called variants. They are used concurrently in alternative configurations [3, 8, 16]. A mechanism must be devised to support the creation and destruction of the versions.

This paper proposes a new methodology as a means of managing VHDL models with versions. Its conceptual bases are the *system entity structure* (SES) formalism [5, 13, 17, 18, 19, 20] and relational database [4, 6, 14], which have been applied successfully to the models management problem [10, 11, 12].

We adapt the success to the VHDL models management problem. In this adaptation, a family of versioned VHDL models is organized in the form of *VHDL Model Structure with Versions* (VMS/V), a representation scheme derived from the SES formalism. The VMS/V itself, in turn, can be formulated in terms of relational tables which can be saved in and retrieved from a relational database. Furthermore, algorithms on the VMS/V, such as the pruning [17], can be defined in terms of relational algebraic operations which can be realized with database language. Therefore, the methodology can exploit the power of relational database. It can manage large amounts of VHDL models and provide sharable repository for many designers and fast queries.

The methodology supports a hierarchical and structural management of versioned VHDL models as follows. Primitive components of a system are assumed to be coded as behavioral models and saved in a model base. The structures that can be constructed from the primitive components are represented as a *model structure*. The model structure, here, serves as a compact representation for organizing a family of hierarchical structures of the system. This separation of the structures from their behaviors may reduce the modeling complexity of the system. Furthermore, the model structure may evolves into several versions over its lifetime.

Once the model structure is organized, a

specific simulation model can be constructed. To construct a simulation model which meets design objectives, the pruning algorithm is used to reduce the model structure to a so-called *pruned model structure*. This pruned model structure, in turn, can be synthesized into VHDL codes by combining with primitive models in the model base.

2. Preliminaries

This section first review the constituents of VHDL and then SES briefly.

2.1 VHDL

VHDL is a hardware description language at several abstraction levels from gates to systems. It supports modeling and simulation of hardware designs and documentation of design data. There are two alternative ways to describe the designs in VHDL. One is behavioral description which models the behavior of the designs as a whole. The other is structural description which decomposes the designs hierarchically and then construct structural models through the composition of primitive components. In this paper, we focus on the second approach because the VHDL modeling problem can be managed systematically by this way. Fig. 1 shows a simplified example of VHDL model, called the DLX proposed by Patterson [9].

The main constituents of VHDL concerned in this paper are the followings: 1) *Entity,* such as *dlx, alu, reg_file,* and *controller* in Fig. 1, represents a real world object and provides the external view. It describes what can be seen from the outside, including generics and ports. 2) *Architecture,* like *dlx_rtl*

```
entity dlx is
  generic (Tpd_clk: time);
  port (phi, phi2: in bit; reset: in bit; halt: out bit;
        a: out dlx_address; ...);
end dlx;

architecture dlx_rtl of dlx is
  signal s1_bus, s2_bus: dlx_word_bus;
  signal dest_bus: dlx_word;
  signal reg_s1_addr: dlx_reg_addr;
  signal reg_file_out1: dlx_word;
  signal reg_write: bit;
  ...
begin
  the_alu: alu
    generic map (Tpd=>4ns)
    port map (s1=>s1_bus, s2=>s2_bus,
              result=>dest_bus, ...);
  the_reg_file: reg_file
    generic map (Tac=>4ns)
    port map (a1=>reg_s1_addr, q1=>reg_file_out1,
              write_en=>reg_write, ...);
  the_controller: controller
    generic map (Tpd_clk_ctrl=>4ns)
    port map (phi1=>phi1, phi2=>phi2, reset=>reset, ...);
  ...
end dlx_rtl;

entity alu is
  generic (Tpd: time);
  port (s1, s2: in dlx_word; result: out dlx_word; ...);
end alu;

entity reg_file is
  generic (Tac: time);
  port (a1: in dlx_reg_addr; q1: out dlx_word;
        write_en : in bit);
end reg_file;

entity controller is
  generic (Tpd_clk_ctrl: time);
  port (phi1, phi2: in bit; reset: in bit; ...);
end controller;
...
```

Fig. 1 Example of VHDL model

in Fig. 1, is an implementation of the entity and considered as an object that contains the internal descriptions of the entity. There may

exist several alternative architectures of an entity. 3) *Component,* such as *the_alu, the_reg_file,* and *the_controller* in Fig. 1, is an entity that is instantiated to structurally compose an architecture. 4) *Generic,* like *Tpd_clk_out* of the *dlx* in Fig. 1, is a constituent of entity that is a way of passing parameters to the entity. 5) *Port,* such as *phi1, phi2,* and *reset* of the *dlx* in Fig. 1, is also a constituent of entity that is a way of connecting the entity with the outside. 6) *Signal,* like *s1_bus, s2_bus,* and *dest_bus* of the *dlx_rtl* in Fig. 1, is a constituent of architecture that interconnects ports of the architecture's components. 7) *Generic map,* such as *Tpd=>4ns* of the *dlx_rtl* in Fig. 1, is the assignment of values to the generics. 8) *Port map,* like *s1=>s1_bus* and *s2=>s2_bus* of the *dlx_rtl* in Fig. 1, is the interconnections among ports of components that compose the architecture.

The use of VHDL enables designers to model and verify their designs effectively. However, the advantages offered by the richness and power of VHDL may make it difficult for less experienced designers to code the details. Therefore, a methodology to reduce the complexity of VHDL modeling problem is necessary.

2.2 System Entity Structure

The system entity structure (SES) formalism [17, 19], proposed by Zeigler, is an unified structural knowledge representation scheme which systematically organizes a family of possible structures of a system. Such a family characterizes decomposition, coupling, and taxonomic relationships among entities. The entity represents a real world object. The decomposition concerns how an entity may be broken down into sub-entities, and the concept of coupling is to specify how these sub-entities may be combined to reconstitute the entity. The taxonomic relationship concerns admissible variants of an entity.

The SES is represented as a labeled tree with attached attributes. There are three types of nodes in the tree. *Entity* node represents a real world object. There are two types of entity, namely composite entity and atomic entity. Composite entity is defined in terms of other entities which may be either atomic or composite, while atomic entity can not broken down into sub-entities. Each entity may be attached with and characterized by *variables.* It may have several aspects and/or specializations. *Aspect* node is connected by a single vertical line from an entity and represents one decomposition of the entity. The children of the aspect are entities, called *sub-entities,* distinct components of the decomposition. Associated with each aspect is *coupling* specifications. *Specialization* node is connected by a double vertical line from an entity. It defines the taxonomy of the entity and represents the way in which the entity can be categorized into *specialized-entities.*

3. Representation of VHDL Models with Versions

In this section, we first propose a simple representation scheme, called VHDL Model Structure (VMS), to represent the structural information of VHDL models. We then propose an extended scheme, called VHDL Model Structure with Versions (VMS/V), to support the versioning concept. The conceptual basis of the representation schemes is the SES formalism.

3.1 VHDL Model Structure

A representation scheme, called *VHDL Model Structure* (VMS), organizes structural information of VHDL models. It supports hierar -chical and structural management of VHDL models. It is depicted as a labeled tree with attached attributes. Fig. 2 shows an example of VMS for the VHDL model of Fig. 1.

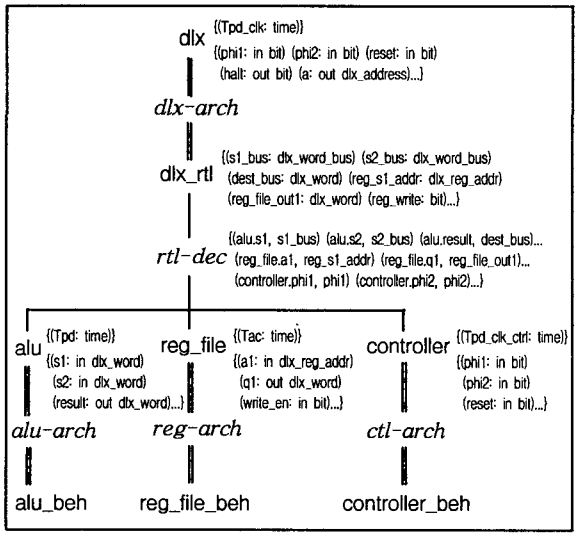

Fig. 2 A VHDL model structure

There are four types of nodes in the tree. *Entity* node, like *dlx* and *alu* in Fig. 2, represents the entity of VHDL. It may be attached with and characterized by the *generic* and *port* of VHDL. *Architecture* node, like *dlx_rtl* and *alu_beh* in Fig. 2, represents the architecture of VHDL. It may be attached with the *signal* of VHDL. *Entity-architecture relationship* node, like *dlx-arch* and *alu-arch* in Fig. 2, is connected by a double vertical line from an entity. It links an architecture with the entity, and represents the relationship in which the architecture is an implementation of the entity. *Architecture-entity relationship* node, like *rtl-dec* in Fig. 2, is connected by a single vertical line from an architecture. It represents a decomposition of the architecture. Its children are entities, distinct components of the decomposition. Associated with it is *port map* specifications.

The correspondence between the constituents of VMS and SES is summarized in Table 1.

Table 1. Correspondence between VMS and SES

VMS	SES
entity	entity
architecture	specialized-entity
relationship between entity and architectures	specialization
component	sub-entity
relationship between architecture and components	aspect
generic, port, signal, generic map	variable
port map	coupling

3.2 VHDL Model Structure with Versions

An extended representation scheme, called *VHDL Model Structure with Versions* (VMS/V), organizes structural information of VHDL models which evolve into several versions over their lifetime. According to the kind of evolution, versions are classified into revisions and variants. Sequential versions that evolve along the time dimension are called revisions. They are created to fix bugs or perform other enhancements. Parallel or alternative versions coexisting at a given time are called variants.

We first consider the revisions. They may change the attributes attached with entity and architecture nodes. These changes may be local in the nodes at one extreme, or propagate downward and/or upward to all the corresponding nodes at another extreme.

For the case of addition or deletion of generics of an entity, the changes need to propagate only to the corresponding architec -tures of the entity. For example, the deletion of the generic *Tpd* of the entity *alu* results in a new reversion of *alu:1*, and then need to propagate to the architecture *alu_beh*. This propagation modify the codes related with the generic *Tpd* in the architecture, and results in a new reversion of *alu_beh:1*. These new reversions are attached to the original nodes, and depicted as arrows in Fig. 3.

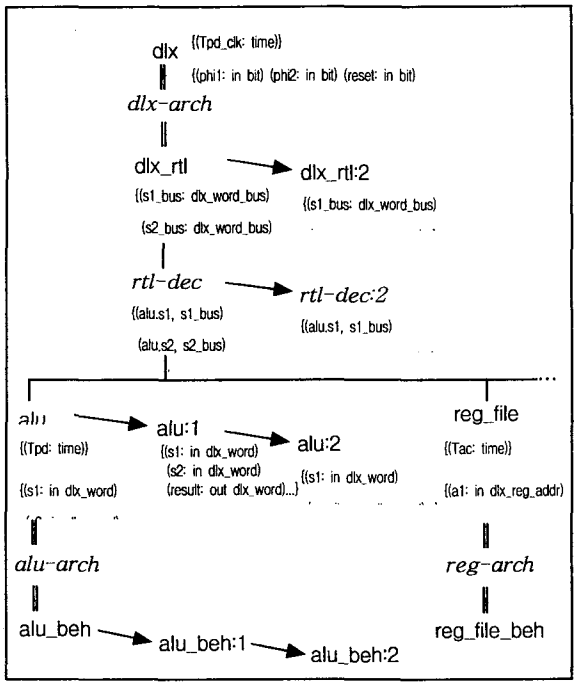

Fig. 3 A VHDL model structure with revisions

For the case of addition or deletion of ports of an entity, the changes must propagate downward to the corresponding architectures of the entity, and upward to the architectures using the entity as a component. For example, the deletion of the port *s2* of the entity *alu* must modify the codes of the architecture *alu_beh*, the port map of the decomposition *rtl-dec*, and the signals of the architecture *dlx_rtl*. These modifications result in new revisions, and depicted as revision number 2 in Fig. 3.

For the case of changes on signals of an architecture, they need to propagate only to the port map of the corresponding decomposition of the architecture. For the case of changes on the port map of a decomposition, they are local in the decomposition, and therefore do not need to propagate to any other nodes. These kinds of revisions can be depicted in a simple version hierarchy as shown in Fig. 3.

We next consider the variants. We will treat significant changes that influence the model structure itself as variants. These changes may include additions and deletions of entities and architectures themselves.

For the case of addition of an architecture, this architecture should be attached under the corresponding entity. For example, the addition of an architecture *dlx_beh* for the entity *dlx* can be attached under the entity-architecture relationship *dlx-arch*. The new architecture *dlx_beh* is assumed as a primitive architecture in Fig. 4. If a new architecture has sub-structures, then its decomposition should be represented as a new architecture-entity relationship.

For the case of addition of an entity as a component of an architecture, a new architecture-entity relationship can be attached under the architecture. For example, the addition of an entity *mux* as a component of the architecture *dlx_rtl* can be represented as the new architecture-entity relationship *rtl-mux* under the architecture *dlx_rtl*. These kinds of variants can be depicted in alternatives as shown in Fig. 4.

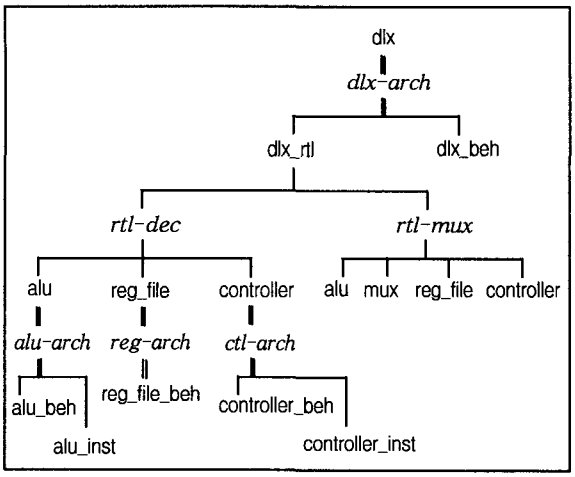

Fig. 4 A VHDL model structure with variants

The two kinds of versions may be combined. That is, each variant of nodes can have another variant, and the variant can, in turn, have revisions. Futhermore, any change resulting in an inconsistent state during the propagation must be rejected and then recovered to the original state. This recovery may be easily implemented by virtue of the transaction support of database system.

4. Database Supported Management of VHDL Models with Versions

Although VMS/V has been visually represented as a treelike structure, it can be transformed into other forms which can coherently convey the information it bears. In this section, we present a relational formalization of VMS/V in which its constituents are represented as relations, and the algorithms are defined by relational algebra.

4.1 Relational Representation

The constituents of VMS/V can be easily represented in the form of relations. The relations can be viewed as tables, where each row is tuple and each column has distinct name called attribute. The relations for VMS/V are defined as follows.

1) ENAR[*ent*, *enar*, *arch*]: contains relationships between entities and architectures. *ent* is an entity, *arch* is one of the architectures of the entity, and *enar* is an entity-architecture relationship node which relates the entity with its architectures. This means that *arch* is an architecture of the entity *ent* under the relationship *enar*.

2) AREN [*arch*, *aren*, *ent*]: contains relationships between architectures and entities. *arch* is an architecture, *ent* is one of the entities to be instantiated as components of the architecture, and *aren* is an architecture-entity relationship node which relates the architecture with its entities (components). This means that *ent* is an entity of the architecture *arch* under the relationship *aren*.

3) REV[*node*, *rev*]: contains revisions of nodes. *node* is a node to be revised, and *rev* is a revision number of the node. This relation is a reference one for the management of revisions.

4) ATTR[*node*, *rev*, *attr*, *class*, *type*]: contains attributes attached with nodes. *node* is an entity or architecture node, and *rev* is the revision number of the node. *attr* is an attribute attached with the node. *class* is the classification of the attribute and can be one of PORT, GENERIC, and SIGNAL. *type* is the type of the attribute. This means that *attr* with auxiliary information (*class* and *type*) is attached with *node*.

5) PMAP[*aren*, *rev*, *ent*, *port*, *signal*]: contains port maps attached with architecture-entity relationship nodes. *aren* is

an architecture-entity relationship node, and *rev* is the revision number of the relationship. *ent* is an entity (component) of the relationship, *port* is a port of the entity, and *signal* is a signal of the architecture of the relationship. This means that *port* of the entity *ent* is connected to *signal*.

The relations are so general as to represent any model structures with versions. For example, a part of relations for the DLX is listed in the following relational tables of Fig. 5.

[ENAR]

ent	*enar*	*arch*
dlx	dlx-arch	dlx_rtl
dlx	dlx-arch	dlx_beh
alu5	alu-arch	alu-beh
reg_file	reg-arch	reg_file_beh
controller	ctl-arch	controller_beh
controller	ctl-arch	controller_inst
...	...	...

[AREN]

arch	*aren*	*ent*
dlx_rtl	rtl-dec	alu
dlx_rtl	rtl-dec	reg_file
dlx_rtl	rtl-dec	controller
dlx_rtl	rtl-mux	alu
dlx_rtl	rtl-mux	mux
dlx_rtl	rtl-mux	reg_file
dlx_rtl	rtl-mux	controller
...	...	...

[REV]

node	*rev*
dlx_rtl	0
dlx_rtl	2
rtl-dec	0
rtl-dec	2
alu	0
alu	1
alu	2
...	...

[ATTR]

node	*rev*	*attr*	*class*	*type*
dlx	0	Tpd_clk	GENERIC	time
dlx	0	phi1	PORT	in bit
dlx_rtl	0	s1_bus	SIGNAL	dlx_word_bus
dlx_rtl	0	s2_bus	SIGNAL	dlx_word_bus
dlx_rtl	0	dest_bus	SIGNAL	dlx_word
dlx_rtl	2	s1_bus	SIGNAL	dlx_word_bus
dlx_rtl	2	dest_bus	SIGNAL	dlx_word
alu	0	Tpd	GENERIC	time
...	...	...	...	...

[PMAP]

aren	*rev*	*ent*	*port*	*signal*
rtl-dec	0	alu	s1	s1_bus
rtl-dec	0	alu	s2	s2_bus
rtl-dec	0	alu	result	dest_bus
rtl-dec	2	alu	s1	s1_bus
rtl-dec	2	alu	result	s2_bus
...	...	...	...	...

Fig. 5 Relational tables of model structure

4.2 Relational Algebraic Algorithms

When VMS/V is represented in the form of relations, algorithms can be defined by the following relational algebraic operations. Fundamental operations are projection(π), selection(σ), union($\cup$), difference(—), and cartesian product($\times$). There are additional operations, such as intersection($\cap$), natural join($\bowtie$), theta join($\bowtie_\theta$), and aggregation(G), that can be defined in terms of the fundamental operations [14].

As described, model structures are represented in the form of relational tables. Once they are built in such a way, VHDL simulation models which meet design objectives can be constructed by relational algebraic pruning and synthesis algorithms.

The *pruning* is to extract a sub-structure, called pruned model structure, from the model structure. Actually, the pruning process is a

sequence of selections from the alternatives such as variants and revisions. We propose a relational algebraic pruning algorithm as shown in Algorithm 1.

```
Algorithm Pruning
  Input: model structure with versions
  Output: pruned model structure
begin
  Ent ← {root_entity};
  do until Ent = ∅
    ▷ step1: selects a revision of the next entity,
          and extracts attributes attached with it.
    next_ent ← next(Ent);
    Rev ← σ node=next_ent REV;
    selected_rev ← select(Rev);
    Pruned_ATTR ← Pruned_ATTR ∪
                  ( σ node=next_ent∧rev=selected_rev ATTR);
    ▷ step2: selects a revision of an architecture,
          and extracts attributes attached with it.
    Arch ← π arch( σ ent=next_ent ENAR);
    selected_arch ← select(Arch);
    Rev ← σ node=selected_arch REV;
    selected_rev ← select(Rev);
    Pruned_ATTR ← Pruned_ATTR ∪
                  ( σ node=selected_arch∧rev=selected_rev ATTR);
    Pruned_ENAR ← Pruned_ENAR ∪
                  ( σ ent=next_entity∧arch=selected_arch ENAR);
    ▷ step3: selects a revision of a decomposition,
          and extracts port maps attached with it.
    Aren ← π aren( σ arch=selected_arch AREN);
    selected_aren ← select(Aren);
    Rev ← σ node=selected_aren REV;
    selected_rev ← select(Rev);
    Pruned_PMAP ← Pruned_AREN ∪
                  ( σ aren=selected_aren∧rev=selected_rev PMAP);
    Pruned_AREN ← Pruned_AREN ∪
                  ( σ arch=selected_arch∧aren=selected_aren AREN);
    ▷ step4: proceeds into entities (components) of the
          selected decomposition.
    Ent ← Ent ∪
                π ent( σ arch=selected_arch∧aren=selected_aren AREN);
  end do
end
```

Algorithm 1. Pruning algorithm

The algorithm starts from the root entity of a model structure. Step1 selects a revision of the given entity, and extracts the attached attributes such as generics and ports. Step2 first selects an architecture from the alternatives hanging from the given entity. Note that an entity may have several architectures. The step then selects a revision of the selected architecture, and extracts attached attributes such as signals. Step3 selects a decomposition from the alternatives hanging from the selected architecture. Also note that an architecture may have several decompositions. The step also extracts attributes such as port maps attached with the selected revision of the decomposition. In step4, the selection process proceeds into next entities through a breadth-first traversal.

In pruning, not all choices may be selected independently. Once an alternative is chosen in one place, some alternatives are rejected or enabled in other places.

```
Algorithm VHDL Synthesis
  Input: pruned model structure
  Output: VHDL code
begin
  ▷ step1: constructs entity declarations.
  for each tuple E of ENAR relation
    construct entity-declaration for entity (E[ent]);
    retrieve generics ( σ node=E[ent]∧class=GENERIC ATTR)
      and construct interface-list of generic;
    retrieve ports ( σ node=E[ent]∧class=PORT ATTR)
      and construct interface-list of port;
    if architecture (E[arch]) is leaf
      retrieve architecture-body from model base;
  end for
  ▷ step2: constructs architecture bodies.
  for each tuple A of AREN relation
    construct architecture-body for architecture (A[arch]);
    retrieve signals ( σ node=A[arch]∧class=SIGNAL ATTR)
      and construct signal-declaration;
    for each component of the architecture
      construct component-instantiation;
      retrieve generics ( σ node=A[ent]∧class=GENERIC ATTR)
        and construct association-list of generic map;
      retrieve port map ( σ aren=A[aren]∧ent=A[ent] PMAP)
        and construct association-list of port map;
    end for
  end for
end
```

Algorithm 2. VHDL synthesis algorithm

A pruned model structure is synthesized into VHDL descriptions by combining it with primitive models in the model base through the following *synthesis* process. The details of the synthesis algorithm are shown in Algorithm 2.

Every entity node of the pruned model structure is transformed into an entity declaration of VHDL. Every architecture node is transformed into an architecture body of VHDL. For each leaf architecture, the architecture body that has behavioral descriptions is just retrieved from the model base. For each non-leaf architecture node, it is transformed into structural VHDL descriptions by composing components (entities) according to the port maps attached with the corresponding architecture-entity node. Other informations, such as generics and signals, are also transformed into appropriate descriptions of VHDL.

4.3 VHDL Models Management System

The proposed VHDL models management methodology may be developed on relational database systems, such as the *Oracle*. It supports the proven reliability, efficiency, and the potential for a rapid implementation. The development relies on the embedded SQL, the *Pro*C*, in which the database language SQL is embedded into the host programming language C. The system may provide a supporting tool to systematically organize a family of model structures according to the principles of VMS/V. It also provides a guiding tool to extract a specific simulation model from the model structures.

As shown in Fig. 5, the core is database components whose responsibility are to store data. There are three databases. *Model Structures* and *Pruned Models* contains model structures and pruned model structures, respectively. *Model Base* contains a set of primitive models. Several management modules are built on top of the database components. They are responsible for managing model structures, pruned model structures, and primitive models supported by the database components.

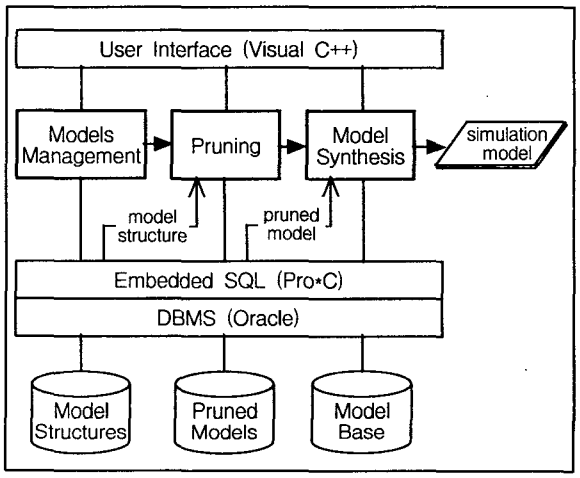

Fig. 5 VHDL models management system

Models Management module provides several facilities such as construction and modification of model structures. *Pruning* module is an implementation of the pruning algorithm. *Model Synthesis* module is to synthesize simulation model by combining pruned mode structure with primitive models in the model base. The system can manage large amounts of VHDL models and provide a sharable repository for many designers by virtue of the capability of database.

5. Conclusions

VHDL modeling requires a hierarchical management of model structures and their

versions. To cope with the problem, this paper proposed a unified representation scheme, called the VHDL Model Structure with Versions (VMS/V), whose conceptual basis is the system entity structure. The scheme, in turn, was represented in the form of relations and saved in a database. Furthermore, algorithms are defined in terms of relational algebraic operations and realized with database language.

The methodology provides an integrated approach within which one may systematically manage VHDL models and their versions. Because VHDL models are, to a great extent, constructed logically, designers can be relieved from the burden of writing and debugging the details of VHDL codes. It also provides such features as the consistencies among modeling components. These features help in creating correct models.

We need to enhance the methodology to support more complex version concepts and advanced databases such as the objected oriented database.

References

[1] Augustin, L. M. et al., *Hardware Design and Simulation in VAL/VHDL,* Kluwer Academic Publishers, Boston, 1991.

[2] Baraona, P., Penix, J., and Alexander, P. "VSPEC: A Declarative Requirements Specification Language for VHDL", in *High-Level System Modeling: Specification Languages,* Kluwer Academic Publishers, Berge, J. M., Levia, O., and Rouilard, J. eds., pp.51-75, 1995.

[3] Conradi, R. and Westfechtel, B. "Version Models for Software Configuration Management", *ACM Computing Surveys,* Vol.30, No.2 (1998), pp.232-282.

[4] Date, C. J. *A Guide to The SQL Standard,* Addison-Wesley, Massachusetts, 1989.

[5] Kim, T. G., Lee, C., Christensen, E. R., and Zeigler, B. P. "System Entity Structuring and Model Base Management," *IEEE Trans. Systems, Man, and Cybernetics,* Vol.20, No.5 (1990), pp.1013-1024.

[6] Lee, T. T. and Lai, M. Y. "A Relational Algebraic Approach to Protocol Verification", *IEEE Trans. Software Engineering,* Vol.14, No.2 (1988), pp.184-193.

[7] Lipsett, R., Schaefer, C. F., and Ussery, C. *VHDL: Hardware Description and Design,* Kluwer Academic Publishers, Boston, 1989.

[8] Miles, J. C. et al. "Versioning and Configuration Management in Design using CAD and Complex Wrapper Objects", *Artificial Intelligence in Engineering,* Vol.14, No.3 (2000), pp.249-260.

[9] Patterson, D. A. and Hennesy, J. L. *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann Publishers, San Mateo, 1990.

[10] Park, H. C. and Kim, T. G. "Relational Algebraic System Entity Structure for Models Management", *IEE Computers and Digital Techniques,* Vol.32, No.1 (1996), pp.49-54.

[11] Park, H. C., Lee, W. B., and Kim, T. G. "RASES: A Database Supported Framework for Structured Model Base Management", *Simulation Practice and Theory,* Vol.5, No.4 (1997), pp.289-313.

[12] Park, H. C. and Kim, T. G. "A Relational Algebraic Framework for VHDL Models Management", *Transactions of the Society for Computer Simulation International,* Vol.15, No.2 (1998), pp.43-55.

[13] Sevinc, S. and Zeigler, B. P. "Entity Structure Based Design Methodology: A LAN Protocol Example", *IEEE Trans.*

Software Engineering, Vol.14, No.3 (1998), pp. 604-611.

[14] Silberschatz, A., Korth, H. F., and Sudarshan, S. *Database System Concepts,* 3rd ed., McGraw-Hill, New York, 1997.

[15] Vahid, F., Narayan, S., and Gajski, D. D. "SpecChart: A VHDL Front-End for Embedded Systems", *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems,* Vol.14, No.6 (1995), pp.694-706.

[16] Westfetchel, B., Munch, B. P., and Conradi, R. "A Layered Architecture or Uniform Version management", *IEEE Trans. on Software Engineering,* Vol.27, No.12 (2001), pp.1111-1133.

[17] Zeigler, B. P. *Multifacetted Modelling and Discrete Event Simulation,* Academic Press, London, 1984.

[18] Zeigler, B. P., Luh, C. J., and Kim, T. G. "Model Base Management for Multifacetted Systems", *ACM Trans. Modeling and Computer Simulation,* Vol.1, No.3 (1991), pp.195-218.

[19] Zeigler, B. P., Praehofer, H, and Kim, T. G. *Theory of Modeling and Simulation,* 2nd ed., Academic Press, San Diego, 2000.

[20] Zhang, G. and Zeigler, B. P. "The System Entity Structure: Knowledge Representation for Simulation Modeling and Design", in *Artificial Intelligence, Simulation and Modeling,* John Wiley & Sons, New York, Widman, L. E., Loparo, K. A., and Nielsen, N. R. eds., pp.47-73, 1989.

● 저자소개 ●

박휴찬

1985 서울대학교 공과대학 전자공학과 학사
1987 한국과학기술원 전기및전자공학과 석사
1995 한국과학기술원 전기및전자공학과 박사
1987~1990 금성반도체 연구원
1997~현재 한국해양대학교 기계·정보공학부 조교수
관심 분야 : 데이터베이스, 모델링 방법론, VHDL