

# A Data Model for VHDL Databases \*

Satish Venkatesan<sup>†</sup>

Karen C. Davis<sup>‡</sup>

Database Systems Laboratory  
University of Cincinnati  
Cincinnati, OH 45221-0030

## Abstract

*Database support for VLSI CAD is proposed to facilitate data sharing and tool interoperability. A methodology for database development that can utilize standard specification languages and formats is described here. The methodology is applied to developing a database language for the structural semantics of VHDL (described in EXPRESS). This research provides a foundation for developing semantic mappings between information models and for integrating VHDL design management with database technology.*

## 1 Introduction

Engineering databases are specialized database management systems (DBMSs) designed for managing engineering data. Objectives of an engineering database for computer-aided design (CAD) are to effectively manage data from the application domain and to provide a framework for integrating various CAD software tools. The database fulfills the latter objective by providing a shared repository for data and a language to manipulate the data.

Traditional database systems (such as relational database systems) were developed for business applications and are not well suited for engineering data management. This is primarily because of a semantic gap between the application domain and the data model, that leads to poor performance. Traditional database systems also lack the expressive power necessary to model the complex nature of design data and capture the dimensionality of the design process. Recently a number of database systems based on complex-object data models have emerged that cater to the specific needs of CAD application domains

[9, 15, 14, 18, 21, 27, 22, 11]. These are better suited to engineering applications than traditional models as they strive to reduce the semantic gap between the application and the data model, but several questions remain to be answered:

1. Since capabilities and concepts vary from system to system, how does a user know what functionality to expect of a CAD database in general?
2. How can various data sources/formats be integrated to facilitate sharing/exchanging of data?
3. How can interoperability between various CAD tools be accomplished?

These questions are addressed by our research, that has the general goal of providing database support for electronic CAD (ECAD) software environments. Database support for CAD provides a common data repository and a language to manipulate data, allowing data to be shared between tools and multiple users. This research proposes to facilitate interoperability of tools and enhance cooperation between designers by developing database view technology for ECAD databases. Views provide a conceptual framework for unifying a CAD software environment; both the input and output of a tool are views of a design. The focus of this paper is on information modeling for VLSI design database systems. A novel approach outlined in this paper is the use of hardware description languages, specifically VHDL [12], as the database definition language for an ECAD database.

Section 2 presents background research in ECAD data modeling and introduces an information modeling methodology for the ECAD domain. Section 3 describes the hierarchy of information models developed using this methodology. A data model for VHDL is presented in Section 4. Conclusions and future research directions are discussed in Section 5.

\*Research supported by NSF Grant IRI-9210200.

<sup>†</sup>e-mail: satish.venkatesan@uc.edu

<sup>‡</sup>e-mail: karen.davis@uc.edu

## 2 Data Modeling for the ECAD domain

A CAD framework is a software backbone that enables CAD tools to cooperate with other CAD tools. Framework research issues include tool integration/communication [23, 16], and data modeling and management (discussed below). CAD Framework Initiative [5, 8] efforts are underway to address standardization of programming interfaces in these areas. The Ptolemy project also addresses these issues [4]. Ptolemy is a system that allows heterogeneous subsystems to be defined and integrated in a rapid prototyping environment. The Ptolemy project's goals are not the same as those of frameworks, but it addresses issues such as data modeling and tool integration.

Our focus is on data modeling and management. Several ECAD database systems and data models have been proposed [20, 22, 9, 15, 11, 14, 18, 21, 27]. One approach uses specialized data modeling (e.g., [20, 22, 27]) and another uses existing standards (e.g., [11, 14, 17]). In this work, we emphasize the second approach. We extend the notion of a hierarchy of information models [28] by developing and partitioning a data model for VHDL and specifying database views on the model. Standardization efforts for a VHDL Intermediate Form had the same goal of providing tool interoperability [3], but differ from our research; the information model hierarchy is intended to accommodate other information models as well. Database view technology has been applied to the CAD domain by Rundensteiner [19]; the approach does not specifically address VHDL as we do here. The VHDL Information Model [7] follows a layered approach and covers structural, behavioral, elaboration, and execution aspects of VHDL. Their goal is to develop a complete semantics of VHDL in EXPRESS; our goal is to create database view technology for ECAD objects while supporting relationships between information models, with VHDL as an example.

In this section we outline a general data modeling methodology. This methodology is then applied to the ECAD domain and we identify specific models that have been developed. These models are then described in detail in Sections 3 and 4.

### 2.1 Methodology

Data modeling for the ECAD domain is complicated by the diversity of design producers (design editors, hardware description languages, etc.) and their singularities. Also, the variety of design consumers (structure verification tools, simulators, etc.) implies

that different design tools need to view different aspects of the design. An approach to a solution is to define multiple associated information models, each depicting the design producer/consumers perspective.

An information model describes the objects in the domain of interest, their attributes, constraints and relationships between objects. Wilkes and Scholz [28] propose using different levels of information models, related by specialization/generalization relationships, each tailored to the specific needs of applications. This facilitates interoperability and reuse of existing models; semantic mappings between models permit data exchange. We apply similar techniques to the ECAD domain with emphasis on hardware description languages, specifically VHDL, as design producers.

Data manipulation across different information models (on the same database) can be performed by specifying database views on the models. A view is a query that defines a suitably limited amount of data for an application [6], CAD tools in this research. The input to a tool is a materialized view, a stored portion of the database, that is updated to reflect changes to the underlying database. The output of a tool can update the database, also through the view. A view can contain data from different models that specify different producers, thus facilitating interoperability of formats. At the same time a view provides logical data independence for CAD tools by insulating them from the underlying representation. View mechanisms are under development [24].

We have identified three modeling levels, depicted in Figure 1. These levels represent the evolution from domain concepts to a database model for the domain. The ovals in the figure denote information models while the arrows denote the direction of specialization. The arrows also denote the sequence in which the models are specified; the dashed arrows denote the transition from model layers to model partitions; the dotted arrows denote the transition from model layers or model partitions to database views. A description of the information model levels follows:

1. Level 1: Shown in Figure 1 (a), are multiple layers of information models. Each layer captures some aspect of the domain that is of interest to the developer. IM1 is the most general model; IM2, IM3 and IM4 are specializations of IM1.
2. Level 2: In Figure 1 (b), the information models can be partitioned into subsets (not necessarily disjoint) based on tool requirements or user perspectives. IM2 is partitioned into IM2.1 and IM2.2. The shaded region represents the overlap

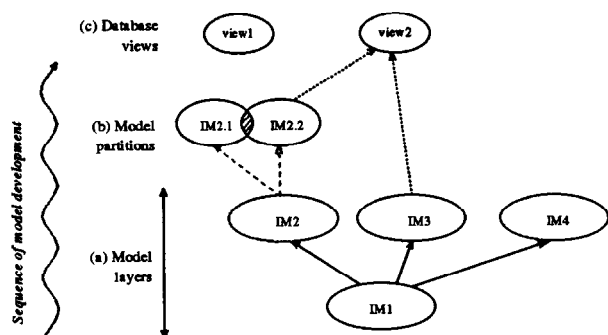


Figure 1: Information Model Levels

between IM2.1 and IM2.2. The models at this level are snapshots of parts of the model at the lower level. Partitioning a model has the advantage that, since a partial model of the domain can suffice for specifying database views, incremental development of a domain model is possible. Extensions to the model will not necessitate changes to existing partitions or tools operating on them.

3. Level 3: Figure 1 (c), is the database view level. Database views specify a subset of both data types in the schema and data stored in the database.

In the next section, this modeling approach is applied to the ECAD domain.

## 2.2 Application to ECAD

Figure 2 shows the information modeling concepts outlined in Section 2.1 applied to the ECAD domain with special emphasis towards VHDL. The information models, depicted by ovals, are described in detail in Sections 3 and 4. Figures 2(a, b) contain two layers of information models at Level 1. They are termed the primary and producer layers respectively. The primary layer is further subdivided into two layers: the lower layer contains the *Object Model* and the upper layer contains the *Core ECAD Model* and the *Version Model*. The producer layer is composed of information models for various design producers. Semantic mappings can be defined between producers to facilitate translation between producer formats or to potentially allow mixed descriptions, for example, a Verilog component in a VHDL design or vice-versa. Mixed descriptions are complicated by the fact that it might not be possible to depict some features of one producer in

another. For example, EDIF does not have an equivalent notion for the package concept in VHDL. We have developed a generalized core model; this facilitates identifying the existence of semantic mappings between design producers, this topic is currently under investigation. Say, there are two entities, E1 and E2, in two producer models, P1 and P2, that are specializations of the same entity in the core model. We can automatically infer that a semantic mapping exists between E1 and E2.

Figure 2(c) shows Level 2, the partition layer. The VHDL model is partitioned into structural and behavioral segments. Since the VHDL concepts of architecture and entity, among others, are common, there will be an overlap between the two partitions.

Level 3, the database view level, is depicted in Figure 2(d). An example of a view is a VHDL configuration that binds component instances to entities.

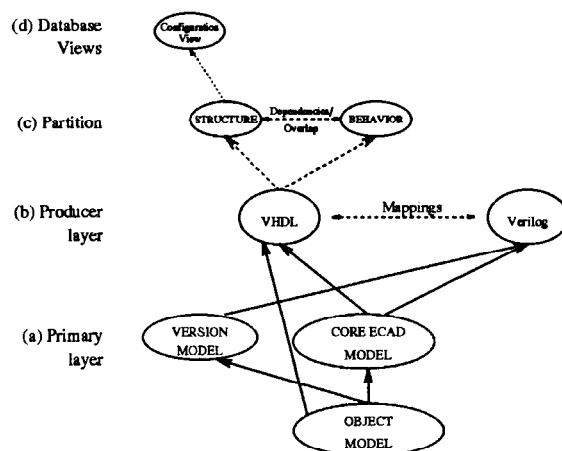


Figure 2: Modeling the ECAD Domain

The Object Model and Core ECAD Model are described in detail in Section 3. We do not propose a Version Model here; but the methodology is flexible enough to accommodate a general versioning mechanism. The VHDL Model is detailed in Section 4.

## 3 ECAD Data Models

In the previous section we described an information modeling methodology and identified a hierarchical set of models that define the concepts in the ECAD domain. This section will describe the Object and Core ECAD models in detail. The information modeling language EXPRESS [13] and its graphical counter-

part EXPRESS-G [13] have been used to document the models. EXPRESS has become a de facto standard for describing information models in the ECAD domain (e.g., CFI [5], DASSY [10], EDIF [1]). It provides for specification of objects belonging to a universe of discourse, the information units pertaining to those objects and the constraints on those objects [13]. The motivation for using EXPRESS is to:

1. define the model in a computer readable format,
2. facilitate comparison to other information models defined using EXPRESS, and
3. partially automate identification of commonalities to other models and specify semantic mappings to them.

### 3.1 Object Model

An EXPRESS-G diagram of the object model is shown in Figure 3. In EXPRESS-G notation: solid boxes denote entity<sup>1</sup> types (*Object*); dashed boxes denote user types (*ObjectId*); dashed boxes with a line on the right are enumerated user types (*ObjectType*). Rounded boxes are off-page connectors. Thick lines denote inheritance (is-a relationship; *NamedObject* inherits the attributes of *Object*); relationships are depicted using thin lines (*Property* is an attribute of *Object*). The small circle at the end of relationship lines denotes its direction.

Every entity in the database is modeled as an *Object*; *Object* is the root of the is-a hierarchy. All objects have an object identifier and an object type. The object type is one of the predefined primitive types or it can be user-defined. All objects can have a set of properties. A property can be of any valid data type (including references to complex objects); the Object Model does not define any properties nor does it place any restriction on allowed properties. Valid data types are Integer, Real, Boolean, String, and references to other object types in the model. Types can also be user defined. Aggregate types and composition of types are also supported.

In the ECAD domain most objects have a name associated with them, hence we define a *NamedObject* entity type, a subtype of the *Object* entity type. It has a *name* attribute in addition to the *ObjectId* and *ObjectType* attributes it inherits from *Object*.

<sup>1</sup>To avoid confusion between the VHDL keyword entity and the EXPRESS modeling unit entity we use the term VHDL-entity to refer to the VHDL concept. The words entity and entity type are used interchangeably in this paper to refer to the EXPRESS concept.

We also define an abstract entity type *WrappedObject*, a subtype of *Object*. *WrappedObject* serves as a supertype for objects whose internal structure is unknown (or only partially known). Figure 4 shows the EXPRESS text for a *WrappedObject* entity type. The *ext\_id* attribute is a pointer to some external means of storing the object. The *otype* attribute is redefined to be of type *WrapperType*.

```

ENTITY WrappedObject
SUBTYPE OF (Object)
ABSTRACT SUPERTYPE OF
  (ONEOF (Process, Expression, Subprogram));
  ext_id: String;
  otype: WrapperType;
END_ENTITY;
```

Figure 4: *WrappedObject* Entity Type

The three primary entity types in the model, *Object*, *NamedObject*, and *WrappedObject*, are all abstract types and are never instantiated directly. They are used for classification purposes only.

The above model is quite similar to the CFI DR 1.0 Base Object Model [5] and can be substituted by it with a few minor changes and extensions. An important distinction, however, is that the CFI model does not support user-defined types, while ours does.

### 3.2 Core ECAD Model

The core ECAD model developed for this research consists of mechanisms for describing ECAD objects and relationships between objects that arise as a result of the design methodology and the design process.

Design objects are modeled as molecular objects [2]. A molecular object consists of an *interface* and an *implementation*. The interface description specifies the communicating ports of the object. The implementation description provides the internal view of the object, i.e., its component objects and their interconnections. Other terms, such as complex objects [15] and composite objects are also used in the literature to express similar concepts.

The mechanisms used to structure design objects are briefly outlined below. A detailed description can be found elsewhere [25, 24].

1. **Hierarchy:** The complexity of an ECAD design is managed by hierarchically structuring the design; a design is decomposed into several smaller designs, which in turn can be decomposed into

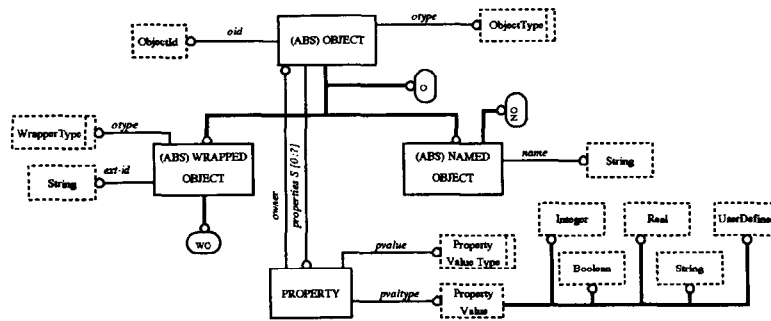


Figure 3: Object Model

simpler objects and so on down the hierarchy until further decomposition is not possible. Objects at the leaves of the hierarchy tree represent the atomic primitives available to a designer.

2. Template/Instance: It consists of an interface specification for a design cell. Templates are instantiated and the instances can be bound (possibly delayed) to implementations.
3. Multiple Representations: An inherent feature of ECAD design is the multiple levels of abstraction of a design. Designs may be transformed to different objects at different representation levels, usually by some CAD tool.
4. Alternatives: One aspect of a design is its function. This functionality may be realized by different design structures, each valid, subject to different constraints. *Alternatives* provide a means of relating objects that perform the same function; each alternative represents a design decision made at a particular phase of the design process.
5. Versions: VLSI design is an evolutionary process; objects and relationship types evolve during various phases of design development. Versioning tracks the evolution of an object's state through time. Each successive version represents additional work on its predecessor. Maintaining the derivation history provides traceability and the option to backtrack in time.
6. Configurations: A configuration represents binding information for a design hierarchy.
7. Library: A library is a collection of design cells.

The core ECAD model is shown in Figure 5. It captures different aspects of design representation: hi-

erarchical design, design library membership, configurations, and electrical connectivity. Libraries and configurations are kept structurally simple with the details being left to design producer models. This is primarily because various hardware description languages, the focus of this research, already have their own concepts of libraries and configurations.

The shaded region represents part of the object model outlined in § 3.1. The basic unit of abstraction is a *design cell*. A design cell is a molecular object; *Port* objects comprise the interface specification; implementation consists of *CellInstance* and *Net* objects.

Cell Instances are described by CellTemplate objects. A CellTemplate object has its own interface specification. *Terminal* objects are instances of *Port* that are owned by CellTemplate. *Port/Terminal/Net Bus* objects are modeled as subtypes of the corresponding entity type.

## 4 VHDL Model

The previous section described a generalized data model for the ECAD domain. This section presents an outline of a VHDL data model<sup>2</sup> that is a specialization of the core ECAD model. The model specifies the semantics of design objects in VHDL and relationships between them. The model will be partitioned into structural and behavioral subsets, and the motivation for partitioning presented. The structural subset, the emphasis of the current model, will be described and a mechanism by which behavioral descriptions can also be stored in the database will be presented. A brief outline of allowed types will be included.

The objectives for developing a data model are

<sup>2</sup>The VHDL language reference manual [12] is used as the source of the language description.





VHDL i.e., electrical connectivity and configurations. In the following, we describe some of the aspects of the structural model.

#### 4.1 Structural VHDL

Entity types in the structural partition of the model are described here.

**Design Interface** is an abstract supertype for `VHDLentity` and `Block`. It consists of Formal Ports and Generics and encapsulates the interface between a `VHDLentity` or `Block` and its environment.

**Port/Generic** are abstract superclasses for Formal and Local Ports/Generics. Formals and Locals are separated to model the distinction in permissible ownership (Formals belong to `VHDLentities` or `Blocks`; Locals belong to `ComponentTemplates`) and to facilitate type checking in rules associated with port/generic maps.

**Design Implementation** is an abstract supertype for `Architecture` and `Block` that consists of `ComponentTemplate` and `ComponentInstance` and, optionally, `BoundComponentInstance` and `BoundComponentTemplate`. The bound objects reflect the option that VHDL provides for binding instances/templates inside an `Architecture/Block`.

**Declarative Object** is an abstract supertype for `Package`, `Entity`, and `DesignImplementation` (`Architecture/Block`). It captures the notion that all subtypes of `Declarative Object` may have user defined attributes, constants, types, and subtypes, declare signals and may import objects via VHDL's use clause. By factoring out these commonalities, we have eliminated the redundancy that would be present in a pure syntax model.

**VHDLentity** is a model of an entity declaration in VHDL. It has an attribute *Implementations* which is a list of `Architectures` that implement the `VHDLentity`. Thus, the model makes explicit the relationship between a `VHDLentity` and an `Architecture`. Another attribute *Alternatives* relates it to other entities that might perform the same function but have a different interface.

**Architecture** is the equivalent of an architecture declaration in VHDL. An attribute *Implements* is an inverse of the *Implementations* attribute in `VHDLentity`. It has attributes *VersionOf* and

*VersionId* outside of the language syntax that enable versioning for architectures. An attribute *ArchType* (an enumeration of `Structure`, `Behavior`, `Mixed`) denotes the nature of its contents.

**Library** is an implementation dependent storage, in this case, the database.

**Port/Generic Map** are modeled as two-dimensional arrays of `AssociationElementType`, an enumeration of `Local/Formal Port/Generic` and `Signal`.

**Configuration** is the equivalent of a VHDL configuration statement. It establishes bindings for component instances and port/generic associations.

#### 4.2 Behavioral VHDL

Behavioral constructs in VHDL are modeled as subtypes of `WrappedObject`. This enables the associated information to be encapsulated in an external wrapper. It can then be stored and accessed in the database as a unit. This enables existing VHDL designs that contain both behavioral and structural objects to be easily incorporated into the database.

Some attributes of a behavioral object are extracted from it and made explicit so that database queries can be expressed on them. For example, a designer can query a `Process` entity (see Figure 7) in a design to see what signals it is sensitive to.

An EXPRESS definition of a VHDL process entity type is shown in Figure 7. The sensitivity list of the process, and attributes, types, subtypes, and constants are extracted from a VHDL process statement and modeled explicitly. The remaining parts of a process statement (some declarative items and sequential statements) are encapsulated in a wrapper.

```

ENTITY VHDLprocess
SUBTYPE OF (WrappedObject);
  name:          String
  sensitive_to:  LIST [0:?] OF signal;
  user_attribs:  OPTIONAL SET [1:?] OF valued_attrib;
  user_types:   OPTIONAL SET [1:?] OF user_type;
  user_subtypes: OPTIONAL SET [1:?] OF user_subtype;
  constants:    OPTIONAL SET [1:?] OF constant;
END_ENTITY;
```

Figure 7: Process Entity Type

Examples of other `WrappedObjects` are subprograms (functions and procedures) and expressions. Further research is needed on modeling behavior.

### 4.3 Types in VHDL

The model also defines permissible types. These include both language types (categories that are provided by VHDL) and user defined types (those that are declared by the user/tool using VHDL's `type_declaration` construct. Currently, scalars and composites are the only language types allowed. File and access types are not supported in our VHDL subset. All user defined types constructed from valid language types are permitted. An EXPRESS-G diagram of allowed types is shown in Figure 8. A detailed description is excluded due to space constraints.

## 5 Conclusions

A methodology for ECAD data modeling has been proposed. This methodology has been applied to the ECAD domain and an Object Model and a Core Model for the ECAD domain have been presented. A data model for a VHDL database has been developed (as a specialization of the Object and Core ECAD models). This model provides the foundation for implementing a database that can store VHDL design descriptions in the form of component libraries. The Object and Core ECAD models that have been developed also facilitate the definition of models for other hardware description languages, for example, EDIF and Verilog. Integrating models for other languages into the system and specifying semantic mappings to them from the VHDL model can potentially permit heterogeneous design descriptions, for example, using EDIF components in a VHDL description or vice-versa. Identifying the existence of semantic mappings has been simplified by specifying a core model that is a generalization for language specific models. The methodology can accommodate specification of database views on domain information models. This facilitates specification of data manipulation semantics; operations can be defined at the view level. We now address some of the issues raised in Section 1 in the following:

1. Different tools are designed for different purposes, each tuned to specific applications, and each having its own perspective of the data. Tool interoperability is facilitated by providing a common database, and by using database view technology to provide each tool with its perspective of the data.
2. Integration of data sources and formats is made possible by sharing information models and creat-

ing application specific models as specializations of generic core models.

The data model also serves as a semantic domain for a formal query language. Query operators have been identified and their semantics are being developed for the VHDL model [24]. A formal query language will:

- serve as the foundation for view materialization/updating research,
- provide a formal setting for operations defined in a Procedural Interface (PI),
- permit querying component libraries for design components that meet the desired constraints,
- facilitate design browsing, and
- aid in investigating constraint satisfaction research.

An implementation of the model is currently underway using an object-oriented database. Its compatibility with existing tools is also under investigation.

## References

- [1] Electronic Industries Association. *Electronic Data Interchange Format 2 0 0*, 1989.
- [2] D. S. Batory and W. Kim. Modeling Concepts for VLSI CAD Objects. *ACM Transactions on Database Systems*, 10(3):322-346, 1985.
- [3] M. Brown. VHDL Intermediate Form Standardization: Process, Issues, and Status. In *EURO-VHDL92*, 1992.
- [4] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *International Journal of Computer Simulation, special issue on Simulation Software Development*, 1992.
- [5] CFI. *Design Representation Programming Interface: Electrical Connectivity V1.0.0*. CAD Framework Initiative, Inc., 1993.
- [6] K. C. Davis. Object-Oriented View Materialization. In *Intl. Workshop on Foundations of Models and Languages for Data and Objects: Modeling Database Dynamics*, Germany, October 1992.
- [7] Christian A. Giumale. An Information Model of VHDL. Technical Report ECIP2/UM/015-1, University of Manchester, 1993.
- [8] A. Graham. Unbundle the Framework. *The Initiative*, 1(1), January 1994.

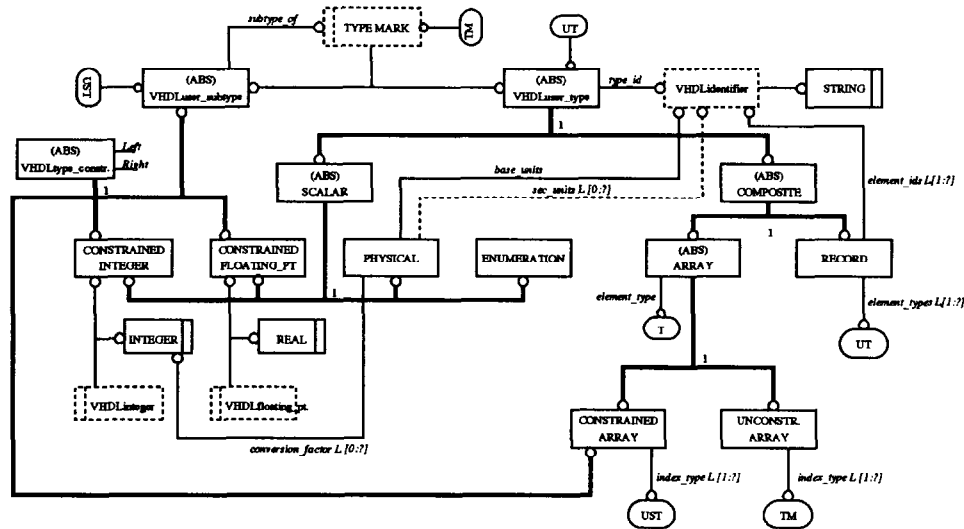


Figure 8: Types in VHDL

- [9] R. Gupta et al. The Development of a Framework for VLSI CAD. In *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*, chapter 13, pages 237–260. Prentice-Hall, 1991.
- [10] W. Heijenga, U. Jasnoch, and E. Radeke. DaDaMo - A Conceptual Data Model for Electronic Design Applications. In *EDAC-92*, 1992.
- [11] F. Hinchliffe 2nd et al. Testing and Integrating a Federated Electrical Engineering Information Model. In *2nd IFIP WG 10.2 Workshop on Electronic Design Automation Frameworks*, 1991.
- [12] *IEEE Standard VHDL Language Reference Manual*. New York, NY, 1988.
- [13] ISO TC184/SC4/WG5 Document N14. EXPRESS Language Reference Manual, April 1991.
- [14] S. Kim and M.J. Chung. Version Control in a Constraint-Driven, Object-Oriented VLSI Design Environment. In *2nd IFIP WG 10.2 Workshop on Electronic Design Automation Frameworks*, 1991.
- [15] W. Kim, J. Banerjee, Hong-Tai Chou, and J. F. Garza. Object-oriented database support for CAD. *Computer Aided Design*, 22(8), 1990.
- [16] N. Kraft. Embedded Tool Encapsulation. In *2nd IFIP WG 10.2 Workshop on Electronic Design Automation Frameworks*, 1991.
- [17] R. A. J. Marshall and H. J. Kahn. A Structural Information Model of VHDL. In *VHDL for Simulation, Synthesis, and Formal Proofs of Hardware*, pages 207–225. Kluwer Academic Publishers, 1992.
- [18] Tapas K. Nayak et al. VLODS: A VLSI Object Oriented Database System. *Information Systems*, 16(1):73–96, 1991.
- [19] E. A. Rundensteiner. Design Tool Integration Using Object-Oriented Database Views. In *ICCAD*, 1993.
- [20] O. Schettler and A. Bredendfeld. Handling Schema Information in the DASSY Data Model. In *3rd IFIP WG10.2/WG10.5 Workshop on Electronic Design Automation Frameworks*, 1992.
- [21] E. Siepmann and G. Zimmermann. Object-Oriented datamodel for the VLSI design system PLAYOUT. In *Proc. 26th DAC*, pages 814–817, 1989.
- [22] T.G.R. van Leuken et al. Standardization Concepts in the NELSIS CAD Framework. In *2nd IFIP WG 10.2 Workshop on Electronic Design Automation Frameworks*, pages 57–70, 1991.
- [23] I. Vedeira and H. Sarmento. Tool integration made easier. In *3rd IFIP WG10.2/WG10.5 Workshop on Electronic Design Automation Frameworks*, 1992.
- [24] Satish Venkatesan. *Query and Update Semantics for a VLSI CAD Database*. PhD thesis, Elect. and Comp. Engg., Univ. of Cincinnati, 1994. In progress.
- [25] Satish Venkatesan and Karen C. Davis. A Formal VLSI CAD Data Model. Technical Report TR 149/5/93/ECE, University of Cincinnati, 1993.
- [26] Satish Venkatesan and Karen C. Davis. EXPRESS models for ECAD and VHDL. Technical report, University of Cincinnati, 1993.
- [27] Flávio R. Wagner et al. Design Version Management in the STAR Framework. In *3rd IFIP Intl. Workshop on EDA Frameworks*, 1992.
- [28] W. Wilkes and G. Scholz. Different Levels of Information Models for Use in Frameworks. In *3rd IFIP WG10.2/WG10.5 Workshop on Electronic Design Automation Frameworks*, pages 21–40, 1992.