An Extensible Object-Oriented Approach to Databases for VLSI/CAD[1]

Hamideh Afsarmanesh
Dennis McLeod

Department of Computer Science
University of Southern California
Los Angeles, California 90089-0782

David Knapp
Alice Parker

Department of Electrical Engineering
University of Southern California
Los Angeles, California 90089-0781

Abstract

This paper describes an approach to the specification and modeling of information associated with the design and evolution of VLSI components. The approach is characterized by combined structural and behavioral descriptions of a component. Database modeling requirements specific to the VLSI design domain are considered and techniques to address them are described. An extensible object-oriented information management framework, the 3DIS (3 Dimensional Information Space), is presented. The framework has been adapted to capture the underlying semantics of the application environment by the addition of new abstraction primitives. An example 3DIS database for a VLSI design system is presented.

1. Introduction

The Very Large Scale Integrated circuit (VLSI) design environment is characterized by a large volume of data, with diverse modalities and complex data descriptions [Bushnell 83], [Davis 82], and [Knapp 85]. Both data and descriptions of data are dynamic, as is the underlying collection of design techniques and procedures. Design engineers, who are normally not database experts, nevertheless become the designers, manipulators, and evolvers of their databases. A final distinctive property of VLSI design environments is a requirement to model both the dynamic behavior of a circuit and its static structure.

In this paper, we characterize a class of digital VLSI design environments, describe a unified system for VLSI design, and present an object-oriented information framework appropriate to model these environments. The remainder of this section concerns digital VLSI design application domains and their specific database modeling requirements. Section 2 briefly describes an extensible object-oriented framework suitable for modeling VLSI design environments, the 3DIS (3 Dimensional Information Space), which has been extended to capture the underlying semantics of circuit structure and behavior. Section 3 describes the modeling of VLSI circuits in the ADAM (Advanced Design AutoMation) system. An example 3DIS database for the ADAM VLSI design system is presented in Section 4 of this paper.

1.1. The VLSI Circuit Design Domain

The VLSI circuit design process typically begins with a descriptive high-level specification of the design, consisting primarily of dataflow and timing graphs, which together describe the data-transformation and timing behavior of the desired hardware. Less detailed structural (i.e. schematic) and physical specifications are given, describing static properties of the target circuit. The descriptive graphs are hierarchical in that their components can be recursively decomposed into simpler components. For example, a dataflow node "multiply" can be decomposed into simpler "shift" and "add" constructs.

Several relationships might be specified among the components of a high level design specification; e.g. among specific time intervals and data operations in the timing and dataflow graphs. Various constraints can be

[1] This research was supported by the National Science Foundation, #MCS-8203485, and the Joint Services Electronics Program through the Air Force Office of Scientific Research under contract # F49620-81-C-0070.

attached to the graphs; for example, the duration of a time interval can be limited, a schematic wire can be specified to be a bidirectional bus connection, and the area of a physical bounding box can be limited. The descriptive graphical representations contain both numeric and symbolic attributes on their arcs and vertices.

The descriptive specification is usually large and complex. Many kinds of data are involved and it is in a large part recursively defined. Furthermore, the specification must be checked for completeness and consistency before the design process begins.

VLSI circuit design typically utilizes a design library, which contains components to be used in the construction of new components. It can also contain designs that are themselves under construction; these may be subparts of a larger design (e.g. the control unit for a CPU), or independent projects. Selecting the appropriate library component may be difficult. For example, if an adder is desired, there might be several components named 'adder', a few named 'ALU', and a few 'complex standard' (i.e. microprocessors). In other situations, the behavior desired may not match the stated behavior of any component in the library without some transformation being applied.

The output of the design system includes a set of graphs, relationships, and constraints similar to those of the descriptive specification, but with a much more detailed physical description.

1.2. ADAM: A Unified System for VLSI Design

The ADAM (Advanced Design AutoMation) system [Granacki 85] is envisioned to become a unified system for VLSI design, starting with a functional and timing specification and proceeding to circuit layout via automatic synthesis routines. The ADAM system describes VLSI circuits by means of four recursively defined and explicitly interrelated hierarchies. In ADAM, the representational formalisms of the input descriptive specification, the library components, and the output design are identical. This in turn facilitates the task of design verification and validation, e.g. testing the equivalence of specified and implemented dataflow graphs.

ADAM supports several major circuit design activities. These activities comprise the main part of the process by which the dataflow and timing descriptive specifications are mapped into the physical output components [Parker 84], [Director 81]. An appropriate information modeling environment for ADAM must support these tasks:

- *Algorithm Synthesis*: The dataflow graph is transformed in order to optimize speed, area, power, and other tradeoffs.

- *Partitioning*: Some part of the specification is partitioned so that the parts can be dealt with separately.

- *Floor Planning*: Given partitions and constraints, high-level chip plans can be constructed that aid in the prediction and optimization of area and performance.

- *Data Path and Control Synthesis*: Data paths are allocated hardware resources and the order of operations is fixed. Controllers are specified and synthesized. Interconnections are synthesized.

- *Built-In Test Synthesis*: Hardware is added to make the end product testable.

- *Module Selection*: Design library elements are introduced to implement operators, memories, and random logic.

- *Placement and Routing*: Modules are allocated physical positions on the layout, and interconnect wires are routed.

- *Validation and Verification*: At any step of the design process, performance and function may be validated using an appropriate simulator or formal verification tool.

1.3. Information Management Requirements of VLSI Design Environments

Given the above general characterization of the VLSI design process, the fundamental characteristics of digital VLSI design environments can be summarized as follows:

- The design data is of large volume, and of various modalities and complexities, e.g. graphical, symbolic, numeric, textual and formatted data.

- Structural information (e.g. data-description, data-interrelation, and data-classification) is complex, of large quantity and must support dynamic use. Structures must allow programs, documents, messages, constraints, and graphs to coexist.

- The end-users, design engineers and CAD application programmers, are familiar with their application environment, but are not likely to have expertise in databases or programming.

2. Information Modeling for VLSI/CAD

Much of the reported work in the VLSI database design literature describes management of design information as collections of raw data in files. Interpretation of the stored design data is completely hidden in the application programs and the users' minds. These database systems are costly to maintain and evolve. Record-oriented database models, such as the relational data model has also been applied to VLSI/CAD design environments [Wong 79], [Eastman 80]. However, these models are of limited suitability for non-database-expert VLSI designers who intend to build, use, and maintain their own databases.

Recently, the suitability of the so-called *semantic database models* as tools to help in the construction and use of design databases has been examined [Katz 82], [McLeod 83], [Batory 84], and [Dittrich 85]. Some semantic database models are object-oriented in the sense that the modeling constructs and the construct manipulators of these models are defined as objects. In such systems objects can be defined to correspond to the concepts, entities, and activities of application environments.

2.1. A Brief Summary of the 3DIS

The 3 Dimensional Information Space (3DIS) [Afsarmanesh 84], and [Afsarmanesh 85a] is a simple but extensible object-oriented information management framework. The 3DIS is mainly intended for applications that have dynamic and complex structures, and whose designers, manipulators, and evolvers are non-database experts. As a step towards addressing the modeling needs of such application environments, the 3DIS unifies the view and treatment of all kinds of information including the structural (description and classification of data) and non-structural (data) database contents, which simplifies database manipulation and modification tasks.

3DIS databases are collections of interrelated objects, where an object represents any identifiable piece of information, of arbitrary kind and level of abstraction. For example, a VLSI component, a component's attribute, a string of characters, a structural component (type), and a procedure defined on a component type are all modeled uniformly as objects in a homogeneous framework. Therefore, what distinguishes different kinds of objects is the set of structural and non-structural (data) relationships defined on them.

Each 3DIS object has a globally unique *object-id* that is an identifier generated by the system. An object can also have several user-specified surrogate *object-names* which also uniquely identify it. Objects may be referred to via their unique object-ids, object-names, or via their relationships with other objects. The 3DIS model supports the following kinds of objects:

- *Atomic objects* represent symbolic constants in databases. These objects carry their own information content in their object-ids. Atomic objects cannot be decomposed into other objects. The contents of atomic objects are uninterpreted, in the sense that they are either displayable or executable. Strings of characters, numbers, Booleans, text, messages, audio, and video objects, as well as behavioral (procedural) objects, are example atomic objects. Text objects and messages represent long character strings, while audio and video objects represent digitized voice and images. Behavioral objects represent the routines that embody database activities, representing objects that are executable. Behavioral objects accomplish modeling of data definition, manipulation, and retrieval primitives, e.g. **Insert-an-OEM-Component**[2].

- *Composite objects* describe non-atomic entities and concepts. The information content of these objects can be interpreted meaningfully by the 3DIS system through their decomposition into other objects. An example of a composite object is a component **H42paddr**. Composite objects are not displayable, except in terms of their relationships with atomic objects; for example, **Designer-names** for **H42padder** are **David** and **John**. If a composite object is related to certain other composite objects, e.g. **H42paddr** has the dataflow model **H42paddr-Dataflow**, then it may be displayed recursively in terms of the atomic

[2] **Boldface** is used to denote object-names.

objects related to those composite objects. Mapping objects are a special kind of composite objects. A mapping object is defined in terms of, and may be decomposed into, a domain type object, a range type object, an inverse mapping object, and the minimum and maximum number of the values it may return. Mappings model both the descriptive characteristics of an object, e.g. a component's name via **Component-Name**, and the associations defined among objects, e.g. a component's constituents via **Has-Link-Constituents**. Mappings also model both single and multi-valued relationships.

- *Type objects* specify classification information: a type object is a structural specification of a group of atomic or composite objects. It denotes a collection of database objects, called its *members*, together with the shared common information about these members. A type object is defined in terms of its members, a set of mappings shared by its members, the fundamental relationships between this type object and other type objects, and a set of operations shared by its members. A type object can be a *subtype* of another type object (*supertype*). Subtypes are defined by the enumeration of members of their supertypes and inherit some of their supertype's definition such as the mappings and operations shared by members (Enumeration may be accomplished through a behavioral object, i.e. a procedure defined on the supertype; this in effect supports predicate-defined subtypes). A type object can be the subtype of more than one type object. The subtype/supertype relationships among type objects can be represented by a directed acyclic graph (DAG). Examples of type objects are **In-House-Component** and **Dataflow-Model**.

Basic associations among objects in 3DIS databases are established through a set of predefined abstraction primitives. The 3DIS model has been extended to accommodate other kinds of abstractions that are useful in VLSI design applications. For example, abstraction primitives to support the definition of recursively defined entities and concepts such as sets, lists, and binary trees are included in the model. In particular, for the ADAM design database, the 3DIS supports the recursive definition of VLSI components, as described in section 4.

An integral part of the 3DIS model is its simple and multi-purpose geometric representation. This geometric framework graphically organizes both structural and non-structural database information in a 3-D representation space and supports their uniform handling. The framework reflects a mathematically founded definition for 3DIS modeling constructs in terms of the geometric components that represent them. The three axes in the space represent the domain (D), the mapping (M), and the range (R) axes. Relationships among objects are modeled by "domain-object, mapping-object, range-object" triples that represent specific points in the geometric space.

Figure 2-1 illustrates a perspective view of the geometric representation of an example 3DIS database. In this figure, **FA-1** and **FA-2** are members of the type object **Single-Node**, while they are also the **Model-Constituents** of **H42padder-Dataflow**. Figure 2-2 illustrates the right view of the geometric representation for the **H42padder-Dataflow**. Both figures have been simplified to represent only a part of the information in the database; the example is further described in Section 4.

Several geometric components such as points, lines, and planes play a meaningful role in representing certain abstractions of the data. For instance, in Figure 2-1, the vertical line emanating from the object **H42padder-Dataflow** represents all mappings defined on that object. Similarly, an orthogonal plane passing through the same object contains the information about all objects directly related to **H42padder-Dataflow**. The variety of information encapsulation supported by the geometric representation, is a unique feature of the 3DIS data model.

The geometric representation is also intended to provide a foundation for information browsing and serves as an environment for a simple graphics-based database user interface. A simple set of navigational operations is defined that consists of *viewing* and *moving* primitives. Viewing operations provide "display windows" to "information neighborhoods" of interest, such as the example views in Figures 2-1 and 2-2. Moving operations allow the information browsing and retrieval. Movements are defined between points on views in orthogonal directions, and they have unique meanings relative to their start position. However, moving in each direction has also a specific meaning that is independent of the start position. For example,

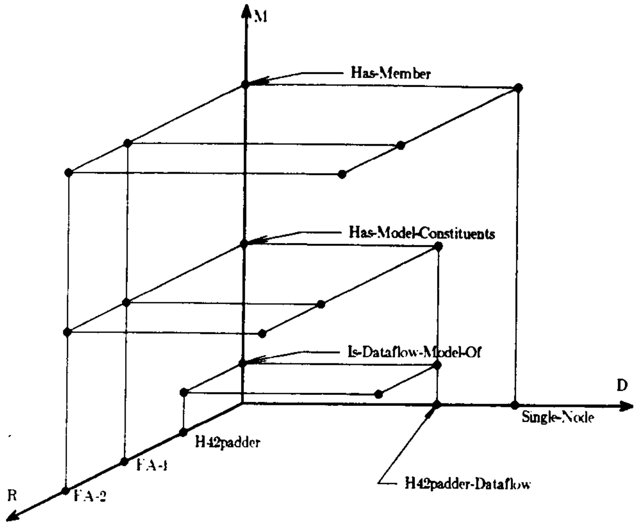


Figure 2-1: Perspective view of a part of information in a 3DIS database

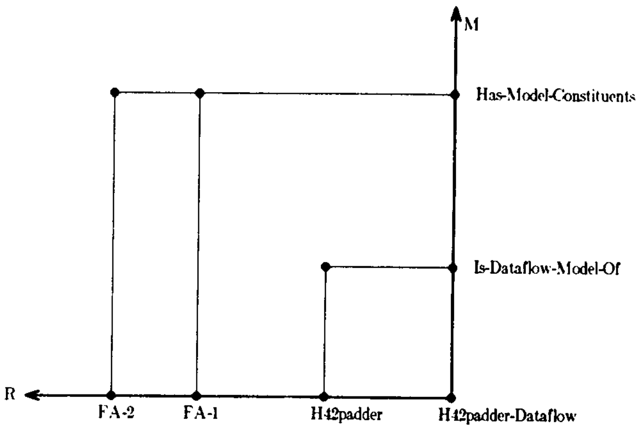


Figure 2-2: Right view of H42padder-Dataflow

moving parallel to the R-axis from any point to the next always returns the next range object for the same domain and mapping objects. Navigational operations support viewing, browsing, and retrieval of both structural and non-structural information. A further description of the 3DIS user interface is given in [Afsarmanesh 85a].

3. A 3DIS Database for VLSI

The digital circuit design process can be regarded as a search of a "design space" [Director 81] for a particular solution that meets constraints on functionality, timing, power, cost, etc. The entire design space can be broken down into subspaces which are near-orthogonal in the sense that decisions taken in one subspace affect decisions taken in another subspace weakly across some region of interest. For example, a single functional specification can be mapped into several different implementations with varying speeds, power dissipations, and costs.

The 3DIS database described below is based on four hierarchical subspaces (also called *models*) chosen for their near-orthogonality [Knapp 83] and [Knapp 85]. For example, one of the subspaces is used to represent schematic (structural) information; this subspace is a hierarchy with block diagrams at the top, registers and ALUs at the middle, gates a little lower, and transistors at the bottom. Design entities are described in terms of these subspaces and a set of relationships across them.

3.1. The Definition of Component

The fundamental structure of the ADAM 3DIS database is the **component**. A component can represent either a specification, a design in progress, or a member of the design library. A specification is represented as an incomplete component. The information that is present in the specification usually represents the operations the target must perform and the constraints it must meet.

A design in progress is also an incomplete component. The design gradually becomes more and more complete, until it can be manufactured. In the initial stages of design, the target component contains primarily dataflow and timing information; in the later stages it contains more schematic information, and finally physical layout. The original dataflow, timing, and schematic information are preserved for documentation and verification/validation purposes.

The design library is used to store both procured components (OEM components) and "In-house" components. An in-house component may be either complete or incomplete.

3.2. The Four Models of a Component

A component is described in terms of four **models** and a set of relationships (**bindings**) across the constituents of the models. The models correspond to the four subspaces of the design space. The four models are:

1. *The dataflow model* describes the data transformation operations performed by the component. Its primitives are **nodes** and **values**. Nodes represent data transformations; values represent data passed between nodes.

2. *The timing and sequencing model* describes

time-domain and branching behavior of the component. Its primitives are **ranges** and **points**[3]. A range represents a time interval during which an operation can take place; points represent infinitesimal "events", which are partially ordered because the ranges have signs as well as durations.

3. *The structural model* describes the schematic diagram of the component. Its primitives are **modules** and **carriers**. A module represents a schematic block, gate, transistor etc.; a carrier represents a schematic wire.

4. *The physical model* describes the layout, position, size, packaging and power dissipation of the component. The primitive elements are **blocks** and **nets**, which represent layout cells and interconnect respectively.

For example, the OEM-component **"74181"**, which is a 4-bit TTL ALU slice, has a dataflow model with **add**, **subtract**, **AND**, and **OR** nodes, which represent its data transformations. This component has a timing-and-sequencing model that describes its propagation delays for various combinations of inputs. It has a schematic diagram that either consists of a box with connection points or a gate diagram. It also has a physical description that signifies that its package is a 14-pin DIP.

3.2.1. Hierarchy within the Subspaces

The four models are each hierarchically structured. For example, a dataflow node is either primitive or it is defined recursively in terms of other nodes and values. Similarly, a value is either primitive or defined recursively in terms of other values. Similar recursive definitions are used in all four hierarchies.

3.2.2. Models and Links in the Four Subspaces

The generic name **Model** is used for nodes, ranges, modules, and blocks. The dataflow model of a component is therefore a **Node**, which can be recursively decomposed into **Nodes** and **Values**. The generic name **Link** is used for values, points, carriers, and nets. These too can be decomposed, with the exception of points, which represent atomic events of infinitesimal duration.

3.2.3. Relationships across Subspaces

All relationships among models and links of different subspaces are explicitly represented by means of **bindings**. There are two basic types of bindings, which are general enough to cover all of the cases of interest:

1. *Operation bindings*, which relate dataflow elements to structural elements and time ranges.

2. *Realization bindings*, which relate structural elements to physical elements.

For example, an operation binding expresses the relationship between an **add** operation (dataflow), an ALU (structure), and the time interval during which it happens. A realization binding represents the correspondence between a particular layout region and the ALU.

3.3. The Target, the Specification, and the Library

The design being constructed is the *target*. The target is functionally equivalent to the *specification*; it is composed of primitive elements and members of the *design library*. Near the top of the hierarchies, the dataflow of the target might be syntactically identical to the dataflow of the specification, but at the low levels this is unlikely. For example, suppose the specification contains a multiplication node. The definition of multiplication can be regarded as a series of shifts and conditional additions. But under a given set of timing, power, and area constraints, the dataflow actually implemented might be radically different. Therefore, the specification and the target are considered to be two completely different components. In general the relationships among constituents of the target and the specification can be complex.

4. An Example

Consider the design of a particular component, a two-bit binary adder, which can be represented as in Figure 4-1. First the schema of the component is discussed; then the dataflow model of the component is examined in detail. The timing, structural, and physical models of the component are not detailed here; for further details please see [Afsarmanesh 85b]. Finally, the way in which bindings are used to unify the four subspaces is described.

4.1. The Component

The subtype/supertype (generalization) hierarchy

[3] These points (timing events) are not to be confused with the points of the 3DIS geometric representation.

of component definitions is shown in Figure 4-2, where boxes represent type objects, the arrows represent subtype/supertype relationships, and the undirected lines that come out of the boxes lead to mappings (properties) that describe members of the types. The type **Component** has properties that denote its name, four **Models** of the component, and two sets of **Bindings**.

There are two subtypes of **Component**. The **OEM-Component** represents a component supplied by an OEM (Outside Equipment Manufacturer). As such it is characterized by the name of the manufacturer, the manufacturer's designation **(Kind)**, and a list of **Suppliers**. Other properties, such as **Price**, have been omitted from the figure in the interest of simplicity.

In-House-Component represents a component that is manufactured in-house. It may not even be a complete design; that information is captured by the **Complete-Bit**[4]. The in-house component also has a set of **Designer-Names**, denoting the people responsible for its construction, a **Process**, which identifies a particular fabrication process, and a **Guru**.

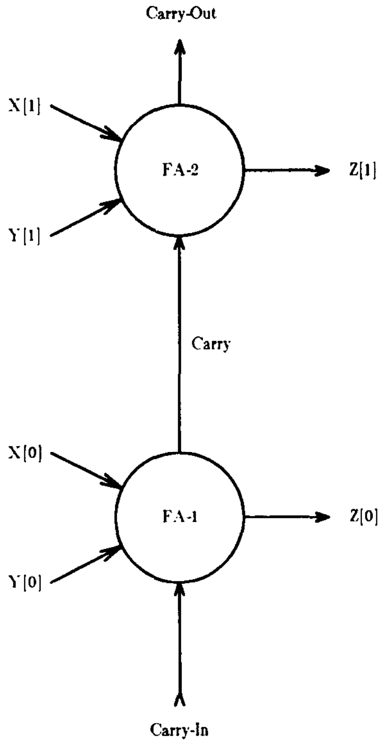


Figure 4-1: Two-bit adder example

The member of the **Component** type used for this example is shown in Figure 4-3. This **Component** is an **In-House-Component**. Its name, a property inherited from the supertype, is "H42padder". The four **Models** are similarly named; Figure 4-3 shows only the "H42padder.Dataflow" **Model** in detail, where again some mappings such as **Complete-Bit** and **Designer** have been omitted in the interest of simplicity[5]. **Operation-Bindings** and **Realization-Bindings** are also shown schematically as lists of logical references to the actual binding objects, as discussed in Section 4.3. The other properties of "H42padder" are self-explanatory. The dataflow graph of "H42padder" is given in Figure 4-1.

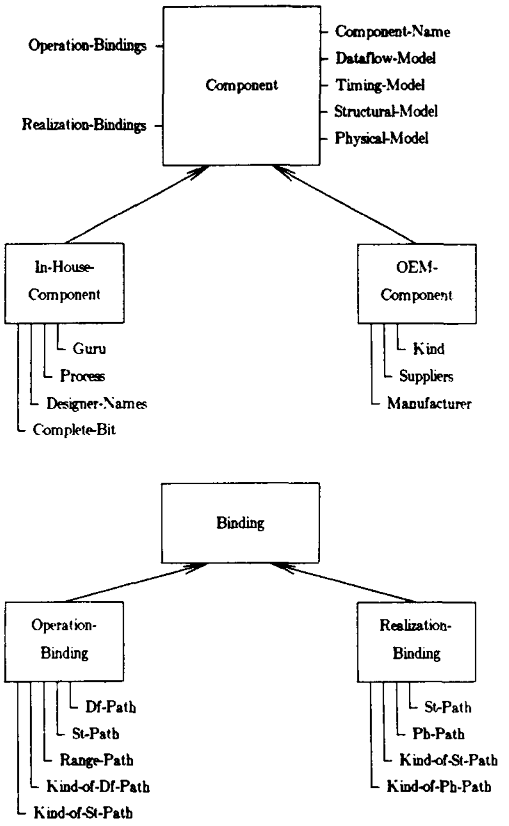


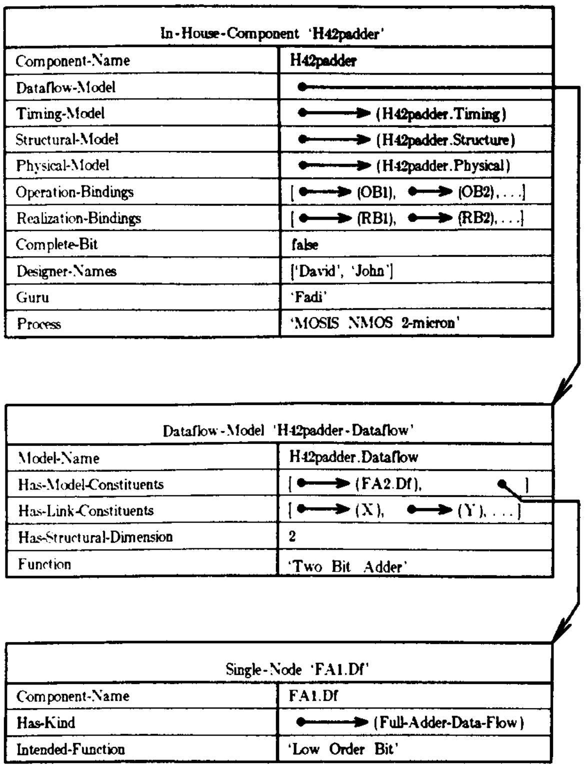
Figure 4-2: The generalization hierarchy of Components and Bindings

4.1.1. Models and Subspaces

In this example we examine only the dataflow subspace. The other models are similar to the dataflow model. The differences mainly consist of minor

[4] More complex historical information could be attached, e.g. a **Verification-History**.

[5] In figures 4-3 and 4-6, the use of parentheses ( ) denote objects whose details have been omitted in the interest of simplicity. Square brackets [ ] represent list delimiters.



In-House-Component 'H42padder'	
Component-Name	H42padder
Dataflow-Model	●
Timing-Model	●→ (H42padder.Timing)
Structural-Model	●→ (H42padder.Structure)
Physical-Model	●→ (H42padder.Physical)
Operation-Bindings	[●→ (OB1), ●→ (OB2), . . .]
Realization-Bindings	[●→ (RB1), ●→ (RB2), . . .]
Complete-Bit	false
Designer-Names	['David', 'John']
Guru	'Fadi'
Process	'MOSIS NMOS 2-micron'

Dataflow-Model 'H42padder-Dataflow'	
Model-Name	H42padder.Dataflow
Has-Model-Constituents	[●→ (FA2.Df), ●]
Has-Link-Constituents	[●→ (X), ●→ (Y), . . .]
Has-Structural-Dimension	2
Function	'Two Bit Adder'

Single-Node 'FA1.Df'	
Component-Name	FA1.Df
Has-Kind	●→ (Full-Adder-Data-Flow)
Intended-Function	'Low Order Bit'

Figure 4-3: A Component member and its partial dataflow model

attributes. For example, the structural counterpart of a dataflow **value** is the **carrier**. The carrier attribute **driver**, which describes hardware implementation attributes such as **tri-state**, **open-drain**, has no counterpart in the dataflow model. The interested reader is referred to [Afsarmanesh 85b] and [Knapp 85].

4.2. The Dataflow Subspace and Dataflow Models

The generalization hierarchy for the **Dataflow-Model** is shown in Figure 4-4. Objects of type **Model** each have a name, a **Complete-Bit** similar to that of **Components**, and a **Designer**. There are four subtypes of **Model**, one for each subspace. Shown in Figure 4-4 is the subtype **Dataflow-Model**, also called **Node** for short. The other three subtypes of **Model** are **Structural-Model**, **Timing-Model**, and **Physical-Model**.

Dataflow-Model has the following properties:

- The **Function** property indicates the overall function performed by the Node. For example, in Figure 4-3 the function of "H42padder-Dataflow" is that of "Two Bit Adder".

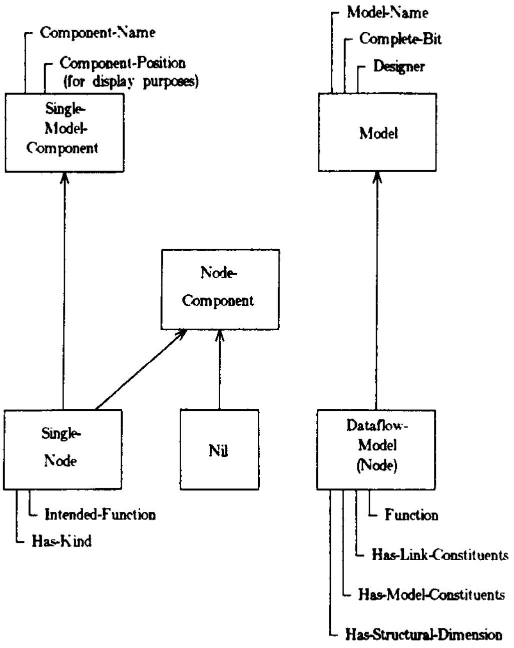


Figure 4-4: The generalization hierarchy of Dataflow Models

- The **Dimension** property specifies the bit width of the Node.

- The **Has-Link-Constituents** property indicates which links (for dataflow models, links are **Values**) are contained within the model.

- The **Has-Model-Constituents** property specifies which models (**Nodes**) are contained within the model.

The constituents of a model together express the application domain semantics of that model, thereby supporting its recursive definition. In the example of Figure 4-3, which corresponds to the two-bit adder dataflow graph of Figure 4-1, the link-constituents are the input, output and carry **Values**; and the model-constituents are the **Nodes** "FA-1" and "FA-2". The constituents of a model are represented as lists of logical references.

The objects that are logically referred to in **Has-Model-Constituents** are of type **Node-Component**, which also designates that they are either of type **Single-Node** or **Nil**[6]. If the

[6] This is accomplished via the definition of the recursion abstraction described in [Afsarmanesh 85a].

reference is to **Nil**, then the constituent is not further defined, i.e. the **Node** is either a primitive or its definition does not exist at present. In either case the recursive definition of the model ends at this point. If the reference is to a **Single-Node**, as is the case in the example, the recursive definition of the model continues through it. In the example, the **Single-Nodes** are called ▪FA-1▪ and ▪FA-2▪. ▪FA-1▪ has the **Intended-Function** ▪Low Order Bit▪; presumably ▪FA-2▪ is the high order bit of the adder. Both ▪FA-1▪ and ▪FA-2▪ could have the value ▪Full-Adder-Data-Flow▪ in their **Has-Kind** properties; that means they are both one-bit full adder nodes. ▪Full-Adder-Data-Flow▪ is itself a **Node**, and is represented by a **Dataflow-Model**; hence it is further defined in terms of its model and link constituents. This is the recursion abstraction at work: **Models** are defined in terms of other **Models**.

4.2.1. Dataflow Links

Figure 4-5 shows the subtype/supertype hierarchy of **Links** for the dataflow subspace. **Links** are more complicated than **Models**, because they bear the burden of representing connections between **Models**. A Dataflow **Link** is called a **Value**. A **Value** has a **Name**, such as ▪Carry▪, which is inherited from the supertype **Link**. It also inherits a **Complete-Bit** and a **Designer**, with meanings similar to those of the **Component**'s corresponding properties.

The reason a **Value** should have an explicitly mentioned **Designer** is that a **Value** is potentially a structured entity (for example a complex floating-point number). If the **Value** is a simple array, then the **Has-Structural-Dimension** property specifies the dimension of the array. If the **Value** is structured, then its **Has-Sublink-Constituents** property defines the structure. **Sublink-Constituents** are of type **Value-Component**, which also indicates that they are either of type **Nil**, or if they are of type **Single-Element** it signifies that they are again either of type **Single-Value** or **Sub-Value** (Figure 4-5).

For example, a floating-point number ▪Flonum▪ is a structured value consisting of two fields ▪Mantissa▪ and ▪Exponent▪. These are **Sub-Values**, which have **Has-Kind** properties of their own. The **Has-Kind** property of ▪Mantissa▪ might refer to a **Value** named ▪Long-Signed-Integer▪ and the **Has-Kind** property of ▪Exponent▪ might refer to ▪Excess-64-Integer▪.

The input ▪X▪ of Figure 4-1 is a **Single-Value**. Figure 4-6 shows ▪X▪ in more detail. The **Has-Kind** property of ▪X▪ points to the **Value** ▪Two-Bit-Integer▪. The **Value** ▪Two-Bit-Integer▪ has the **Structural-Dimension** 2. ▪Two-Bit-Integer▪ also has **Sublink-Constituents** consisting of two **Sub-Values**, named ▪High-Order-Bit▪ and ▪Low-Order-Bit▪ respectively. The **Has-Kind** properties of these bits have logical references to the primitive **Value** ▪Bit▪. The **Has-Sublink-Constituents** of the **Value** ▪Bit▪ is **nil**, so the recursive definition of ▪Two-Bit-Integer▪ ends at this point.

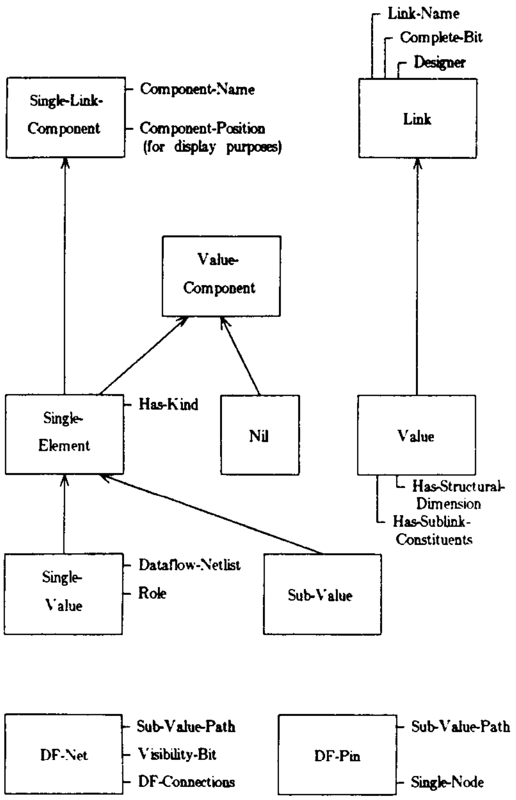
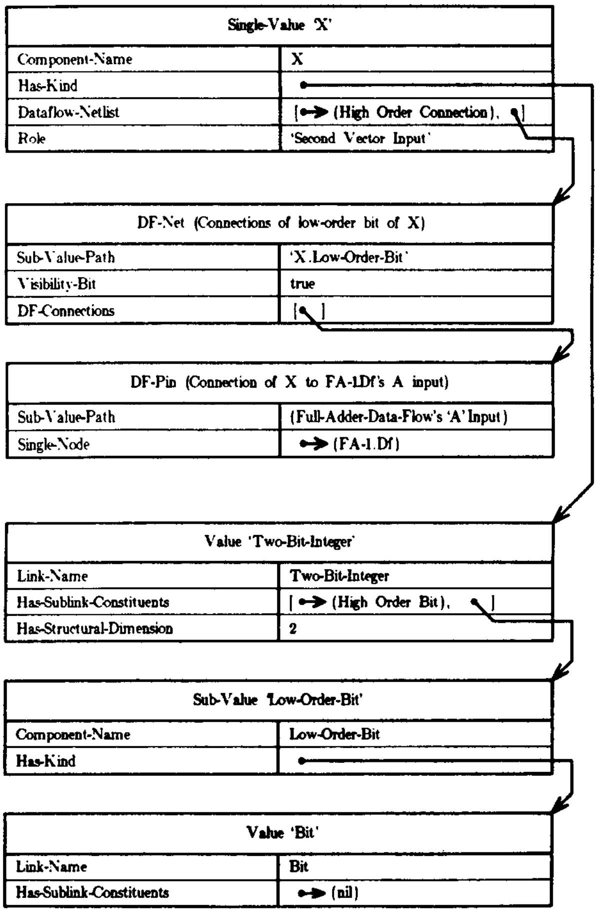


Figure 4-5: The generalization hierarchy of Dataflow Links

In addition, ▪X▪ has a **Role** which is ▪second vector input▪. Furthermore, it has connections, represented by a **Dataflow-Netlist**. The **Dataflow-Netlist** is a list of logical references to **DF-Nets**. In Figure 4-6, the two bits of the ▪Two-Bit-Integer▪ ▪X▪ are connected separately, only the connections of the ▪Low-Order-Bit▪ being shown.

The **DF-Net** has a **Sub-Value-Path**. This is a path into the structure of the value being connected. For example, if the high-order bit of the mantissa of a complex floating-point number ▪A▪ was connected individually, the path would be ▪A.Real.Mantissa.Bit63▪. In Figure 4-6, the path simply points to the low-order bit of ▪X▪.

The **DF-Net** also has a **Visibility-Bit**; this determines whether the bit can be "seen" from outside "H42padder-Dataflow". Since "X" is an input, this bit is **true** for all its **DF-Nets**. Other structured **Links** may have their visibilities determined on a field-by-field basis, which is why the visibility information is attached to the individual connections rather than to the **Single-Link** itself.



Single-Value 'X'	
Component-Name	X
Has-Kind	•
Dataflow-Netlist	[•→ (High Order Connection), •]
Role	'Second Vector Input'

DF-Net (Connections of low-order bit of X)	
Sub-Value-Path	'X.Low-Order-Bit'
Visibility-Bit	true
DF-Connections	[•]

DF-Pin (Connection of X to FA-1Df's A input)	
Sub-Value-Path	(Full-Adder-Data-Flow's 'A' Input)
Single-Node	•→ (FA-1.Df)

Value 'Two-Bit-Integer'	
Link-Name	Two-Bit-Integer
Has-Sublink-Constituents	[•→ (High Order Bit), •]
Has-Structural-Dimension	2

Sub-Value 'Low-Order-Bit'	
Component-Name	Low-Order-Bit
Has-Kind	•

Value 'Bit'	
Link-Name	Bit
Has-Sublink-Constituents	•→ (nil)

Figure 4-6: The definition and connections of the Value "X".

DF-Connections are used to describe connections in the dataflow subspace. The **DF-Connections** of a **DF-Net** are references to **DF-Pins**. A **DF-Pin** refers to a **Single-Node**, e.g. "FA-1.Df", and a **Sub-Value-Path**, which represents a connection point on that **Single-Node**. In the example of Figure 4-6, the **Sub-Value-Path** of the "X" connection point is a path to the "A" input bit of the "Full-Adder-Data-Flow" model, which is given in parentheses in Figure 4-6 (recall that "Full-Adder-Data-Flow" is the **Has-Kind** property of "FA-1.Df").

Using both the **Sub-Value-Path** of a link, as expressed in the **DF-Net**, and the **Sub-Value-Path** of a Single-Node connection point, as expressed in the **DF-Pin**, very general kinds of connections can be constructed.

For example, using both paths in their full generality would allow us to make arbitrary permutations of structured array values at connection points. If a two-bit **Value** "P" was to be connected to the "X" input of "H42padder.Dataflow", it would be possible to connect "P[1]" to "X[0]" and "P[0]" to "X[1]", thus achieving a bitwise reversal at the point of connection.

4.3. Bindings

The two binding sets represent the interrelationships between the elements of the models. **Operation-Bindings** show the relationship between an operation (or value), a structure, and a time interval; (for example, an addition, an adder, and a microcycle). Similarly, a different Operation-Binding might represent the relationship between a value, a bus or register, and a microcycle. **Realization-Bindings** are used to represent the relationships between structural elements and physical realizations (for example, between an adder's schematic and its layout).

Both kinds of **Bindings** have properties that represent paths into the four hierarchies, e.g. "St-Path" as shown in Figure 4-2. The reason paths must be used is that **Bindings** refer to unique **Single-Model-Components**. Such a **Single-Model-Component** may be deep down in the recursion hierarchy, and the only way to uniquely specify it is by giving a complete path down into the hierarchy, starting at the root **Component**.

The **Kind-of-Df-Path** property of **Operation-Binding** simply indicates whether the binding is to a **Node** or a **Value**; similarly the **Kind-of-St-Path** property indicates whether the binding is to a **Carrier** or a **Module**. These are examples of the use of a "*generic interrelation abstraction*"[7]. All combinations are permitted in the schema. Similar considerations apply to **Realization-Bindings**.

There is no **Kind-of-Range-Path** property for **Operation-Bindings** because the only valid timing element for a binding is a **Range**. **Points** have

[7] This abstraction primitive and the recursion abstraction mentioned earlier were specifically defined for the ADAM VLSI design database, and are supported by the 3DIS data model.

ınfinitesimal duration, and hence are never suitable for binding either operations or values to structural elements.

5. Conclusions

The 3DIS was introduced as an extensible information modeling framework that captures the underlying semantics of VLSI/CAD application environments, and supports requirements specific to this domain. The application of the 3DIS to the ADAM system was described. An example 3DIS database design for ADAM was presented.

The 3DIS database is object-oriented in that all data entities, relationships defined on entities, events and operations, as well as the description of data (metadata) are modeled as objects. It provides a structured, unified view of the application information that reduces the required level of expertise for database manipulation and database development/evolution. The extension of the 3DIS model to support the specific modeling requirements of engineering design environments, such as modeling recursively defined entities and concepts, simplifies the task of database design.

The representation schema is based on the idea of unifying the design data in three major structures called the specification, the target, and the library, respectively. Each of these consists of a single component or a collection of components, where all components are modeled uniformly. A component is represented in terms of four orthogonal hierarchies: dataflow, timing, structural, and physical. The four hierarchies are linked by explicit relationships called bindings.

We expect significant benefits from the presented approach in construction of the overall ADAM system. Since the design data is unified by the database, adding application packages is greatly simplified. Since non-experts can access the underlying schema easily, the designers of CAD packages need not be database experts to use the system flexibly. The representation has several advantages. It cleanly represents the data of interest. Important relationships between specification and the target are not obscured. Designer freedom is limited to the degree permitted by the specification. The same concepts and techniques can be used to analyze and construct target designs, specifications, and library components. Finally, the design details are hidden until they are needed. The unification of the database with the synthesis and analysis tools results in an automated process from algorithm specification to circuit layout. This in itself is expected to simplify the design process and enhance design correctness.

A Pascal-based graphical interface to the 3DIS, implemented on an IBM PC/XT [Afsarmanesh 85a], has been designed and implemented. This interface provides an experimental vehicle for evaluation and improvement of the browsing capabilities of the 3DIS user interface. A Pascal-based graphical editor for an older, file-oriented version of the design data structure (DDS) has also been implemented.

Future work on this project includes integrating the system into a coherent whole. In particular the data structures of a number of synthesis packages must be changed from the DDS file format to the new 3DIS-oriented database support system in order for ADAM to function as a single unified system. The implementation of the 3DIS database system and its user interface must be completed. The definition and implementation of database activities, e.g. the invocation of semantic checking routines, must also be added.

References

[Afsarmanesh 84] Afsarmanesh, H., and Mcleod, D.
A Framework for Semantic Database Models.
In *Proceedings of NYU Symposium on New Directions for Database Systems.* New York, NY, May, 1984.

[Afsarmanesh 85a]
Afsarmanesh, H.
The 3 Dimensional Information Space (3DIS): An Extensible Object-Oriented Framework for Information Management.
PhD thesis, Department of Computer Science, University of Southern California, July, 1985.

[Afsarmanesh 85b]
Afsarmanesh, H., Knapp, D., McLeod, D., and Parker, A.
An Object-Oriented Approach to Databases for VLSI/CAD.
Technical Report, Department of Computer Science, University of Southern California, April, 1985.

[Batory 84] Batory, D.S. and Kim, W.
Modelling Concepts for VLSI CAD Objects.

technical report TR-84-35, Department of Computer Science, University of Texas at Austin, December, 1984.

[Bushnell 83] Bushnell, M., Geiger, D., Kim, J., LaPotin, D., Nassif, S., Nestor, J. Rajan, J., Strojwas, A., and Walker, H.
DIF: The CMU-DA Intermediate Form.
Technical Report CMUCAD-83-11, CMU Center for Computer-Aided Design, 1983.

[Davis 82] Davis, R., Shrobe, H., Hamscher, W., Wieckert, K., Shirley, M., and Polit, S.
Diagnosis Based on Description of Structure and Function.
In *Proceedings of the National Conference on AI*, pages 137-142. AAAI, 1982.

[Director 81] Director, S.W., Parker, A.C., Siewiorek, D.P., and Thomas, D.E.
A Design Methodology and Computer Aids for Digital VLSI Systems.
IEEE Transactions on Circuits and Systems CAS-28:634-645, July, 1981.

[Dittrich 85] Dittrich, K.R., Kotz, A.M., and Mulle, J.M.
An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases.
Technical Report, Institut fuer Informatik II, Universitaet Karlsruhe, West Germany, 1985.

[Eastman 80] Eastman, C.M.
System Facilities for CAD Databases.
In *Proceedings of the 17th Design Automation Conference.* 1980.

[Granacki 85] Granacki, J., Knapp, D., and Parker, A.
The ADAM Advanced Design AutoMation System: Overview, Planner, and Natural Language Interface.
In *Proceedings of the 22nd Design Automation Conference.* 1985.

[Katz 82] Katz, R. H.
A Database Approach for Managing VLSI Design Data.
In *Proceedings of the 19th Design Automation Conference.* 1982.

[Knapp 83] Knapp, D. and Parker, A.
A Data Structure for VLSI Synthesis and Verification.
Technical Report DISC 83-6a, Digital Integrated Systems Center, Dept. of EE-Systems, University of Southern California, October, 1983.

[Knapp 85] Knapp, D. and Parker, A.
A Unified Representation for Design Information.
In *Proceedings of the 1895 Conference on Hardware Description Languages.* IFIP, 1985.

[McLeod 83] McLeod, D., Bapa Rao, K. V., and Narayanaswamy, K.
An Approach to Information Management for CAD/VLSI Applications.
In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* San Jose, California, May, 1983.

[Parker 84] Parker, A.
Automated Synthesis of Digital Systems.
IEEE Design and Test , November, 1984.

[Wong 79] Wong, S. and Bristol, W.A.
A CAD Database.
In *Proceedings of the 16th Design Automation Conference.* 1979.