Databases and Cell-Selection Algorithms for VLSI Cell Libraries

Simon Y. Foo, Florida State University/Florida A&M University

Yoshiyasu Takefuji, Case Western Reserve University

Designing a system with library cells resembles putting one together with off-the-shelf vendor ICs. Library cells have different specifications ranging from I/O buffer options to internal gates with varying degrees of drive capability. For example, a function can be implemented in various structures, with a choice of buffers and I/O port locations. These essential implementation alternatives allow designers to explore new chip architectures using different cell structures with varying design constraints, such as speed, chip area, and power consumption.

As the complexity of VLSI circuits continues to grow, the need for a common, shared cell library is becoming inevitable. This is particularly true because of the diverse technologies that have evolved during the computer age — for example, the bipolar junction transistor (BJT), n-channel metal-oxide semiconductor (NMOS), complementary MOS (CMOS), gallium arsenide (GaAs), and bipolar MOS (BiMOS).

Bipolar technology, particularly the advanced low-power Schottky (ALS) series, is still very popular due to its very high-speed switching capability (as in the case of emitter-coupled-logic (ECL)) and the gradual reduction in power consumption due to improved processing techniques. MOS technology is most appropriate for prototyping VLSI circuits due to its high-density packing capability, relatively simple processing steps, and low power consumption (as in the case of CMOS). GaAs technology is used primarily in military applications due to its extremely high-speed switching, high temperature operation, low internal power dissipation, and antiradiation properties. But, as improved GaAS chip fabrication processes bring costs down, commercial applications will become feasible.

This framework for capturing design data is based on semantic networks. It is well suited for application-specific ICs, yet general enough for other CAD/CAM environments.

Recently, processes have been successfully mixed on a single silicon wafer — for example, BiCMOS technology, where the goal is to approach the high-speed switching of BJT and low power consumption of CMOS. To have the greatest possible utility, a shared cell library will allow the complete spectrum of functions, implementation alternatives, and process technologies.

Issues

Although there has been considerable interest in using commercial database management systems for managing VLSI CAD data, the complex nature of VLSI design limits their suitability. Most general-purpose DBMSs are simply too slow, expensive, and inefficient for VLSI CAD applications.[1] Thus, current research on design data management in CAD environments focuses on extensions or modifications to support

(1) efficient spatial searching,
(2) design hierarchies and multilevel representations,
(3) design alternatives and version control,
(4) interface to simulation tools,
(5) common interface between cell libraries from different IC vendors, and
(6) efficient cell selection based on given design constraints.

0018-9162/90/0200-0018$01.00 © 1990 IEEE

Spatial searching. In human-machine design interactions, the human verification process depends on machine spatial searching to display selected portions of the VLSI design. Efficient spatial searching is essential to high performance and fast turnaround. Popular layout editors, such as UC Berkeley's Kic and Magic, use a geographic bin structure for spatial indexing. Existing commercial DBMSs do not support such capabilities and, consequently, must rely on exhaustive search.

Design hierarchies. Most CAD environments require hierarchical levels of abstraction. Typically, VLSI designs can be described at the functional, logical, circuit, and layout levels. System architects, operating at the top level in the design hierarchy, rarely work with transistors and device models; at the bottom level, layout experts hardly ever access system-level descriptions. However, these tasks must be well coordinated by a group manager supervising the entire design process from concept to silicon.

Figure 1 shows a typical flow of operations in an IC design cycle based on a top-down methodology. *Behavioral* representations, in text or equation form, describe a circuit's function. For example, procedural descriptions, such as "If clock is high then increment counter," and Boolean expressions, such as "Y = !A + B," are functional representations; they do not specify details about implementation. *Structural* representations describe the composition of circuits in terms of cells (components) and interconnections among components. Such descriptions are usually hierarchical due to the composite nature of VLSI circuits. Block diagrams, circuit schematics, and netlists of logic gates are typical examples of structural descriptions. *Physical* representations show the circuit's geometric layout on semiconductor layers (polysilicon, diffusion, etc.) but do not convey information about functionality. Hence, a snapshot of the design process at each level of abstraction requires a multilevel representation.

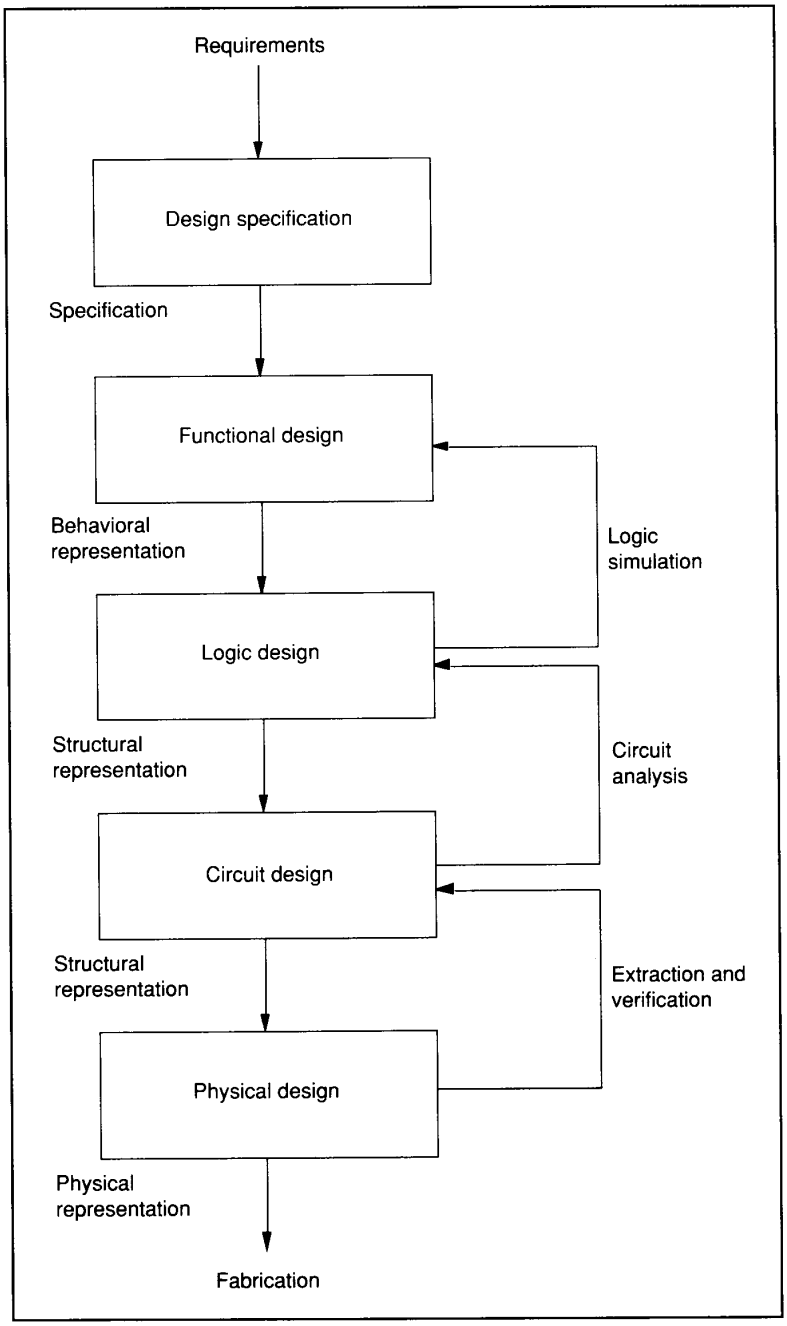

Figure 1. Phases of an IC design system based on a top-down methodology.

Version control. To achieve fast turnaround (rapid prototyping) of high-performance VLSI chips, it's often desirable to consider many alternative designs. Much of VLSI design is evolutionary or exploratory, so the database must support design versions and alternatives. Versions are improvements or modifications to a design object, while alternatives are different implementations of the same object with varying performance characteristics. For example, the process of partitioning or synthesizing a function into a structure of components is not unique; usually several alternatives can be considered. Consider the following alternatives: the next-state function in a control unit can be implemented as a one-way or two-way branch; multiple inputs to a functional unit can be made via a bus or a multiplexer; an adder can be designed as a ripple-carry, carry-save, or carry-look-ahead adder; a digital IC chip can use a one- or two-phase clocking scheme; I/O ports can be on different sides of a cell; a cell layout can have different aspect ratios of transistors, and so forth.

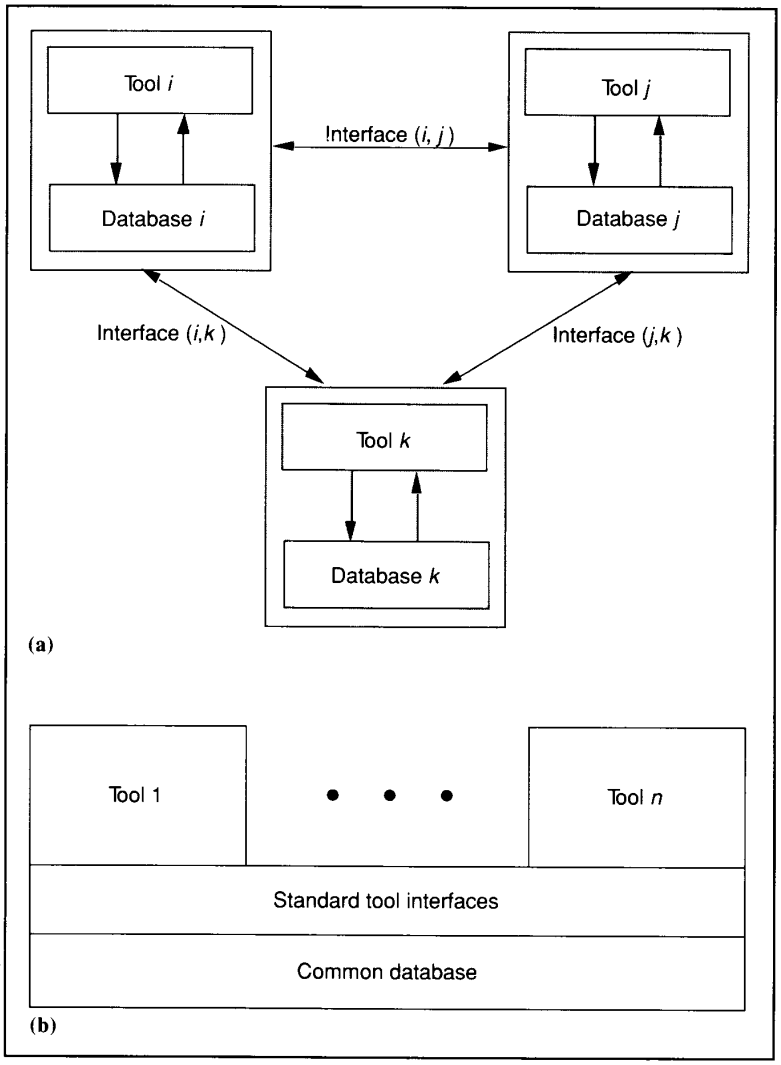

Figure 2. Contrasting CAD systems: (a) independent nonstandard; (b) integrated.

Tool interface. Besides representing domain-specific data, a dedicated VLSI database system should interface with other CAD tools. There is a need to represent cell specifications/characteristics extracted from circuit simulators (such as UC Berkeley's SPICE (Simulation Program with Integrated Circuit Emphasis)) and timing analyzers (such as UC Berkeley's Crystal) in a common database. Currently, simulations are performed independently, and the results are loosely organized in a rather ad hoc manner. UC Berkeley's Oct tools exemplify recent attempts to solve this problem. The Oct package includes an object-oriented database manager for VLSI CAD and interfaces with other popular VLSI tools readily available to universities.

Library interface. Sharing cell libraries helps prevent duplicate development efforts and promotes exchange of ideas for new cell architectures. Recent attempts include an NMOS library[2] and a CMOS3 (three-micron CMOS technology) cell library[3] consisting of contributions from various sources. The fabricated cells are functionally tested and characterized, then catalogued in a manner similar to IC data books. However, differences in cell specifications, from I/O-buffer options to internal gates, have often discouraged the sharing of cell libraries.

A standard format for documenting cells or a common interface for interlibrary communications is needed so that IC designers can concentrate on the design phase. One solution is offered by design languages, such as AHPL, that formalize the design process by interconnecting small-, medium-, and large-scale-integration subsystems. AHPL offers a convenient, unambiguous means for expressing and communicating a design at the register-transfer level. Recently, there has been much pressure on commercial IC vendors to adopt the Electronic Design Interchange Format (EDIF) as the industry's standard design interchange language. In contrast to AHPL, EDIF resembles a symbolic layout language. It supports design description at several levels of abstraction, including mask and symbolic layout levels. Recent EDIF activities include UC Berkeley's project to support the EDIF 2.0.0. standard. The EDIF Software Project's goal is a translator-building toolkit that will provide facilities needed for EDIF translations.

Cell selection. The last issue concerns the familiar problem of combinatorial optimization. Cell selection involves choosing from a list of possible candidates an implementation that meets system requirements. The choice is often influenced by design constraints such as speed, area, power, fan-out, complexity, architecture, and reliability. Selection algorithms can help prune the design search space, thereby providing near-optimum decisions on cell selection. This process is particularly important in the design of application-specific ICs, where a high degree of performance and optimization is desirable.

In general, our article represents a survey of approaches addressing four of the six issues just defined. That is, we cover design hierarchies and multilevel representations, design alternatives and version control, common interface between cell libraries, and efficient cell selection based on given design constraints.

Background

Consider an independent nonstandard CAD system (Figure 2a) where each application tool has its own unique database. Communication between tool *i* and tool *j* is via interface (*i*, *j*). The total number of

interfaces required for n tools is $n(n-1)/2$, that is, the number of interfaces required to implement a complete CAD system grows exponentially as the number of tools increases. In contrast, an integrated CAD system (Figure 2b) consists of a set of tools tightly coupled to a common database through a standard set of tool interfaces. Besides the advantages of a common database, the integrated CAD system is also modular, that is, application tools can be "plugged" in or taken out without reconfiguring the common database.

The desire for such an integrated CAD system brings us to the concept of CAD frameworks. Basically, a CAD framework is a software system into which separate application tools, such as schematic editors, simulators, and IC or printed-circuit-board layout tools, can be plugged. A CAD framework also includes front-end software that manages a tool's appearance on the computer user's terminal, back-end software that tracks the enormous data required to design a complex VLSI system, and key utilities that help manage multiple teams of designers.

Most VLSI chip manufacturers offer their own in-house CAD framework, but standards are now being developed to support the concept of painlessly plugging different application tools into frameworks from different manufacturers. The CAD Framework Initiative (CFI) is an industry-sponsored effort to standardize interfaces between VLSI CAD tools. CFI is seeking a set of standards for design-automation tools that would allow end-users to "mix and match" their own design systems. Such standards would give IC designers rapid access to new CAD software technology from multiple vendors.

The need for standardization — and its potential benefits — was succinctly stated in the September 1989 *SIGDA Newsletter*:

> . . . The need for industry-compatible CAD frameworks has become more acute in recent years with the emergence of the commercial CAD industry and the increasing complexity of designs. As new IC designs approach one million transistors and electronic systems are ten to a hundred times more complex, the CAD support environments have become slow and cumbersome. Many observers feel the standardization of CAD environments is the single most important factor in providing the design productivity required for these next-generation efforts.

The CFI board, representing more than 40 major IC design companies, has since set up seven committees to study VLSI CAD issues ranging from design data-management techniques to user interfaces.

Data models and design data management

Data models. The database models applied in today's database systems generally fall into one of the following categories:

- hierarchical,
- network,
- relational, or
- semantic.

A hierarchical schema consists of a collection of record types and a collection of link types that specify connections between the record types. The restriction is that each record type in a given tree (with the exception of the root type) must be the child type of a single parent record type. Hence, a hierarchical database consists of a "forest" of trees, conforming to the structure defined in the schema.

The network data model is similar to the hierarchical model, except for two distinct features. First, it permits multiple link types between record types. Second, a given record can have multiple parent records. Consequently, the network data model is more flexible than the hierarchical model.

In contrast, a relational database consists of n-ary relations, where each relation may be viewed as a table with a fixed number of named columns and a variable number of rows. The rows in each table are called *tuples*, and the columns are referred to as *attributes*. The relationships between the data records are called *sets*. Relational operations on the database, such as

- form subsets of rows or columns,
- form the union, difference, or intersection of sets, and
- combine relations by concatenating sets,

can be performed using a data manipulation language. Data manipulation languages for the relational data model are typically derived from a relational calculus or a relational algebra. The relational data model overcomes the record-oriented limitation of the hierarchical and network models by allowing many-to-many relationships without pointer overhead.

A fundamental problem with the hierarchical, network, and relational data models is their limited semantic expressiveness. Semantic data models, on the other hand, use static constructs to represent object-object, type-type, and type-object relationships. One of the major features of the semantic data model is the ability to inherit attributes from types to objects. The object-oriented model and the semantic (frame-based) network model are variations of the semantic data model.

Object-oriented programming systems (OOPSs), such as Smalltalk, C++, and the Common Lisp Object System, are based on the object-oriented data model. Central to the object-oriented data model is the concept of a *class* (sometimes referred to as type), which is a collection of data items defining the state of an *instance* (or object) of the class, and a set of functions to manage that state. An instance is a unique copy of the collection of data items belonging to a particular class. Each data item is called an *instance variable*, and the functions of the class are referred to as *methods*. The primary feature of an OOPS is the concept of inheritance, which allows us to specify a new class by defining only the differences between it and another class (called its superclass). For example, we could specify a class "half-adder" by declaring it a subclass of "full-adder" and describing the differences. Then, full-adder becomes the superclass of half-adder.

In frame-based systems, instances are frames (or generalized property lists) and instance variables are called slots. Frame systems often include runtime support for expressing relationships between instance records. For example, there are built-in facilities that search the frame networks for sets of instances that resemble but do not exactly match each other. Such support is often built on top of a conventional OOPS.

Design data management. Currently, there are four major approaches to effective management of data in VLSI CAD applications:

(1) developing a special-purpose design DBMS (DDBMS),
(2) enhancing a current DBMS by adding new capabilities and functions,
(3) building a layer of software on top of a current DBMS to compensate for deficiencies, and
(4) using a special-purpose file manager that views the DBMS as an application.

Ketabchi and Berzins[4] provide an example of the first approach. They proposed an object-oriented data model as a frame-

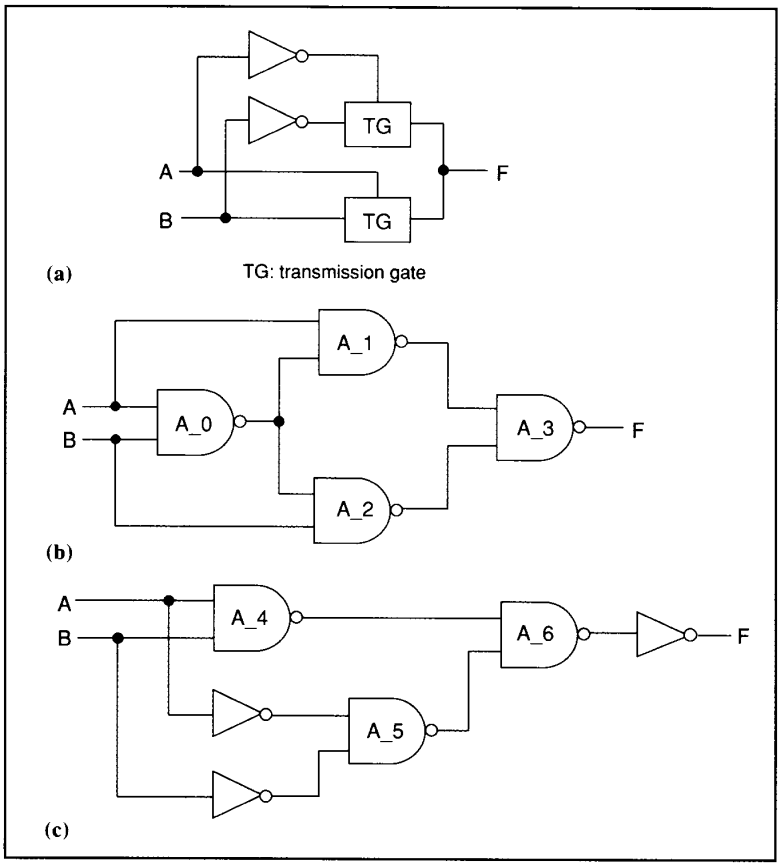

Figure 3. Three possible architectures for the XOR function: (a) based on transmission gates; (b) gate-array implementation; (c) unoptimized combinational logic circuit.

work for achieving effective management of versions (refinements) and alternatives of composite objects in CAD applications.

The second approach requires extensive enhancements to the current DBMS, as reported by Lorie and Plouffe[5] and Stonebraker.[6] Lorie and Plouffe discussed extensions to System R, while Stonebraker worked on enhancements to Ingres based on the applications of abstract data types and abstract indices. However, this approach is not very satisfactory because its extra overhead slows down the DBMS.

The third approach is attractive because it runs user-application programs on top of a general-purpose DBMS and thus becomes operational rapidly. For example, Ingres provides user interfaces through high-level programming languages such as C, Pascal, and Fortran. However, this approach is inefficient because user programs must be linked to all Ingres database functions, but only a few features are used frequently. Furthermore, Ingres requires enormous memory space during program runtime.

Katz et. al.[7] used the fourth approach. Their version server organizes design data and provides mechanisms, such as object check-in and check-out, for controlling design evolutions. Design objects are connected using version, configuration, and equivalence relationships that form orthogonal, hierarchical name spaces for logically grouping design objects. This approach, however, does not take advantage of capabilities, such as data encapsulation, provided by high-level data models and database technology.

Some researchers claim that object-oriented database management systems (OODBMSs) offer several aids to rapid prototyping of VLSI designs. For example, Batory and Kim[8] introduced an object-oriented Smalltalk-style model where generic objects and version objects become types and instances, respectively. Instances can inherit attributes from their individual types, similar to generalization hierarchies in semantic data models. Gupta et. al.[9] described their use of Cbase, a VLSI CAD framework developed at the University of Southern California. Cbase uses OODBMS concepts and provides a platform for creating, displaying, manipulating, and maintaining digital VLSI designs.

Among other approaches is a data model, introduced by McLellan,[10] that groups cells into libraries and establishes a search path based on an ordered list of libraries. In Neuman's[11] generalized approach to CAD databases, the version server explicitly tracks versions but does not use timestamps to identify invalid equivalences.

Version management in software development environments has been a matter of concern for some time, but capturing the semantics of design objects and version control has not yet been dealt with. For example, the Unix Source Code Control System maintains versions of files but has no knowledge of how they map onto a hierarchical configuration of program modules. Versions are named according to creation times. One of the shortcomings of SCCS is that it does not associate versions of component modules with the module that incorporates them. Some VLSI layout editors, such as Magic from UC Berkeley, incorporate a kind of version control in the form of a timestamp. Mismatches in the timestamps are reported when the composite (leaf) cells of a root cell are altered, with or without the designer's knowledge.

As a case study of the special-purpose DDBMS approach, we'll consider a frame-based model. We chose the semantic (frame) model because its hierarchical data structures support attribute inheritance and are well suited for representing VLSI design data, which require hierarchical decomposition and multiple levels of abstraction. The balance of this article discusses our frame-based database system.

Frame-based model

In our VLSI CAD environment, the design database can be modeled as a collection of *atomic* and *composite* objects. Primitives such as inverters, transmission gates (pass transistors), and two-input NAND and NOR gates are classified as atomic objects. Composite objects are made up of smaller components (atomic or composite) interconnected in a hierarchy

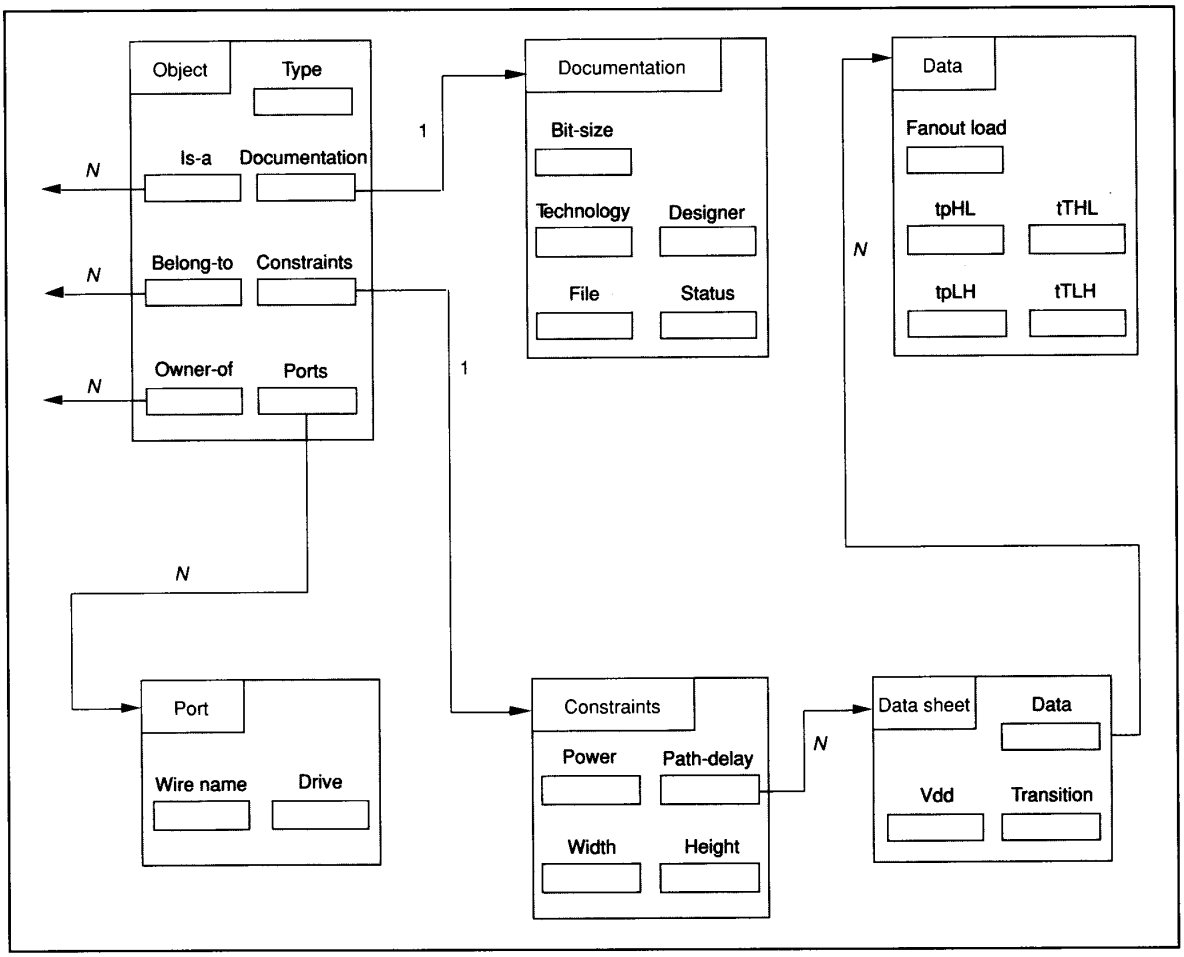

Figure 4. A frame-based schema for a design object.

to perform a particular function. The leaves of the hierarchy are atomic objects, and the internal (intermediate) nodes are composite objects. A composite object is described only once at the type level; when the object is instantiated, it becomes an instance (or cell).

We adopted a variation of Magic's technique of labeling cell names, where each instance is denoted by

<cell-type>_<tag>_<id>

The identifier cell-type specifies circuit function (NAND, for example), tag identifies a particular implementation (or version) of cell-type, and id is the instantiation number, a function of the number of times an object type is instantiated in the global database. The concatenation of identifiers cell-type, tag, and id uniquely identifies an instance in a common database.

For example, consider the circuit implementation of the exclusive-OR (XOR) function. Figure 3 shows a set of possible implementations of the function F = AB′ + A′B. The list of cell-types used in a composite object can be derived by taking the uniqueness of cell-types used in a list of instances. For the XOR gate in Figure 3b, the complete list of instances used in the composite object is

{nand_A_0, nand_A_1, nand_A_2, nand_A_3}

where the cell-type used is "nand" gate and the tag is "A."

Each design object is described by a set of *views* (documentation, switching characteristics, I/O interface, etc.). In a semantic network, each *frame* represents a particular view. Therefore, a design object can be represented by a hierarchical structure of frames, as shown in Figure 4. The dependency between the views is represented by the hierarchical structure of the corresponding frames.

The root frame may contain slots specifying the *specialization* and *aggregation* of the instance. Specialization is represented by an is-a or AKO (a-kind-of) link, while aggregation (or parts relationship) is represented by parent-of (or owner-of) and child-of (or belong-to) links. The type slot is used to specify a particular implementation of a function (for example, carry-save adder) or a unique characteristic of a cell (for example, up-down counter).

The documentation frame describes the legal aspects of a cell. The file slot specifies the directory path for accessing the layout file. The state of the design process,

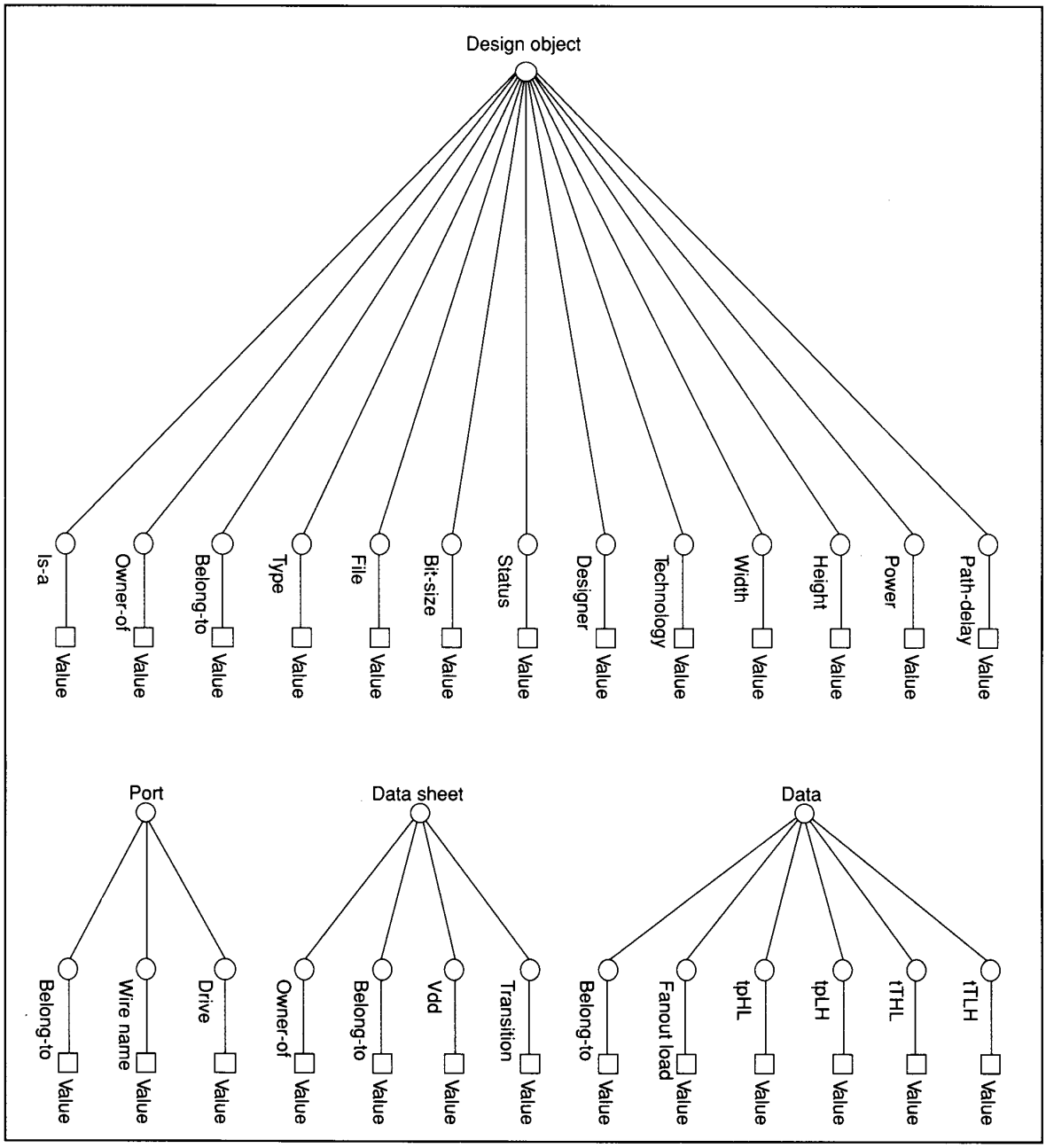

Figure 5. Tree representations of a design object. Notice the additional slots, "belong-to" and "owner-of," containing pointers to other trees.

whether it has been functionally simulated or its timing analyzed, is indicated in its status slot. In the case where the design has been fabricated and parametric testing performed on the IC chip, such progress should also be noted in the status slot.

Each port frame describes the interface of one cell to other cells. The wirename slot indicates wire types (for example, signal, power, or ground), while the drive slot specifies fan-out capabilities. This information is essential for checking the I/O compatibility of each port before global routing or channel routing.

The constraints frame typically specifies the physical dimensions of the Manhattan-style cell layouts, in terms of width and height, and other equally important considerations, such as power consumption and propagation (or path) delay.

Besides functional specifications, IC

designers are interested in dynamic and static cell characteristics (for example, propagation delay, fan-out, maximum clock frequency, and test conditions). Therefore, the propagation delay of each I/O node is represented by a data-sheet frame of the cell's switching characteristics. For example, delay time and rise/fall time are specified under the following conditions: worst-case processing parameters, 5 volts, 125 degrees Celsius, and a typical loading of 1.5-kilohms lumped-series resistance and 1.5-picofarad load capacitance with an input transition time of 10 nanoseconds. The worst-case delay information is intended to be used for quick cell-to-cell performance comparisons. For each data sheet under a particular test condition, there is one set of switching characteristic data for each fan-out load. Hence, there is a separate data frame containing the rise/fall times of a cell driving a particular fan-out load.

Mapping design objects onto tree structures. Based on the above schema, we defined a set of rules for mapping design objects from the frame-based representation (shown in Figure 4) to tree representations such that the information about the design objects can be stored or manipulated. There are two major rules:

(1) Attributes of one-to-one (1:1) relationships between a parent frame and a child frame are merged into a single tree with the parent's node as the root.

(2) In the case of one-to-many (1: *N*) relationships between a parent and its child frames, a separate tree is created for each child frame. Each child frame is connected to its parent through the belong-to slot and each parent to its child frames via the owner-of slot.

Figure 5 shows the corresponding tree representation of the frame-based schema (shown in Figure 4) after the mapping process. Attributes of the documentation frame, along with the area and power attributes of the constraints frame, are merged onto the parent tree since these attributes are one-to-one relationships. However, subtrees are created for ports since each cell has more than one I/O port. Separate trees are also established for the data sheets since each cell has at least one set of timing data. Also note that additional slots, belong-to and owner-of, containing pointers to other trees are created as a result of the second mapping rule.

Table 1. Switching characteristics of an XOR cell. (Vdd = 5 V; input transition time = 5 ns)

Parameter		Fan-out load = 1 (typical)	Fan-out load = 10 (typical)
Propagation delay (high to low)	tpHL	7.5	11.5
Propagation delay (low to high)	tpLH	7.0	19.0
Output fall time	tTHL	7.0	12.5
Output rise time	tTLH	12.5	40.5

Manipulating the design database

Following the conversion of frames to trees, information can be appended to the cell database through an operation

```
fput(<object>; <constraint> <value>;
   ... )
```

where fput is a procedure for performing the append operation, the object argument specifies the identifier of a frame or node, the constraint slot specifies an attribute, and one or more values are assigned to that attribute. For example, consider the switching characteristics of a tested XOR cell from a library,[12] as shown in Table 1. The timing information is appended to the design database through the following append operations:

```
fput(XX7486;
   is_a "XOR";
   bit_size 2;
   owner_of "datasheet1";
   power 0.2;
   status "SPICEd, Crystaled";
   technology "CMOS";
   file "library/cells/XX7486.mag";
   designer "UW/NW-VLSI")

fput(datasheet1;
   belong_to "XX7486";
   owner_of "data1, data2";
   vdd 5;
   input_transition_time 5)

   fput(data1;
   belong_to "datasheet1";
   fanout_load 1;
   tpHL 7.5;
   tpLH 7.0;
   tTHL 7.0;
   tTLH 12.5)

fput(data2;
   belong_to "datasheet1";
   fanout_load 10;
   tpHL 11.5;
   tpLH 19.0;
   tTHL 12.5;
   tTLH 40.5)
```

Pointers under slots owner_of and belong_to link the parent and child nodes. Units of transition, voltage, and DC power consumption are assumed to be in nanoseconds, volts, and milliwatts, respectively. Notice that the popular TTL (transistor-transistor logic) catalog numbering of cells is still applicable in a CMOS library; that is, in XX7486 above, XX is the manufacturer's standard prefix, 74 denotes manufacturer's specifications to meet commercial applications (54 would denote military specifications), and 86 is a TTL number for XOR logic gate. Also, tpHL and tpLH stand for time of propagation from high to low and low to high, respectively; tTHL and tTLH stand for time of transition from high to low (output fall time) and low to high (output rise time), respectively.

Daemons. Functions automatically activated in response to the need for a value, or its placement or removal, are called daemons. Therefore, these functions are called if-needed, if-added, and if-removed daemons. An example of a demand-driven

Query#1:	fget(*)
Returns:	A list of all instances and classes available in the database, where wild-card indicator "*" means "match all."
Query#2:	fget(*; *; *)
Returns:	All existing information available in the database.
Query#3:	fget(74HC74; *; *)
Returns:	All existing information about instance "74HC74."
Query#4:	fget(74HC74; *)
Returns:	A list of attributes associated with "74HC74."
Query#5:	fget(74HC74; function; status; technology)
Returns:	A list of requested information about "74HC74."
Query#6:	fget(register; where bit_size = 4 & designer = "foo")
Returns:	A list of 4-bit registers designed by "foo."
Query#7:	fget(*; where designer = "foo")
Returns:	A list of all instances designed by "foo."
Query#8:	fget(comparator)
Returns:	A list of all instances of object type "comparator."
Query#9:	fget(adder; where bit_size = 4 & path_delay> = 5.0 & path_delay <= 10.0)
Returns:	A list of all 4-bit adders with path-delay ranging from 5.0 to 10.0 ns.
Query#10:	fget(*; where parent_of = "74HC74")
Returns:	A list of composite objects where "74HC74" has been instantiated.
Query#11:	extract("74HC74," complete)
Returns:	A complete list of atomic and composite objects belonging to "74HC74."
Query#12:	extract("74HC74," unique)
Returns:	A list of unique objects (atomic and composite) belonging to "74HC74."

Figure 6. Example queries demonstrating language features.

if-needed daemon is the area-calculating function, which can be applied to all instances with Manhattan-style geometries. Given the width and height of a cell, for example,

fput(74HC74; width 210; height 196)

the function "area = width × height" will return a value of 41,160 if queried. The database does not store this value because it can be automatically computed if needed. This area-calculating function is always "lurking" inside the program, hence the term "if-needed daemon."

The expertise of IC designers can be captured with artificial intelligence techniques. This expertise can take the form of procedural knowledge, for example, a set of procedures to calculate the passive component values for a differential amplifier, given specifications of gain, output impedance, common-mode rejection, etc.

Due to the nature of VLSI CAD data, simple expertise for manipulating domain-specific data can be captured in the form of procedural knowledge. This knowledge, represented by a set of rules, is useful for handling incomplete or plausible information through deductive inferences on the design database. These rules can be implemented as daemons, for example,

for each instance
  if function is digital comparator
    then number of outputs is 3;
  if status is tested
    then all subcells are tested;
  if technology is CMOS
    then all subcells use CMOS;
  and so forth.

The above rules may seem trivial, but they are useful in enforcing integrity constraints on the design database. Since they are invoked when information is appended, the rules are implemented as if-added daemons. If certain instances are deleted from the database as part of the design process, links between the parent cell and its child cells are also deleted through the if-removed daemon.

Query language for relational operations. Information can be retrieved using a query language capable of performing relational operations in a frame-based system. The syntax for the query operation used in our frame-based system is

fget(<object> {;<conditional expressions> | <target slots>})

where function fget returns a list of information that satisfies the conditional expressions or is stored in the target slots. Conditional expressions are lists of constraints (numerical or symbolic) with relational arithmetic operators (for example, =, ≤, and ≥) and logical connectives (for example, AND, OR, and NOT), while target slots are object attributes. In general, conditional expressions are used for selection queries, while target slots are used for projection queries. Numerical range can be specified by a conjunction of two numerical constraints as follows

<slot><relational operator><value1>
& <slot><relational operator><value2>

where "&" is the conjunction operator. For example, the constraint "20.00 ≤ path-delay ≤ 30.00" is represented as

path_delay ≥ 20.00
& path_delay ≤ 30.00

The example queries in Figure 6 demonstrate a few features of the query language. Queries 1 through 5 are called projections, while queries 6 through 10 are selection operations as used in relational databases. In addition to the fget function, there is an

extract function that recursively traverses the component cells of a particular instance and returns a list of all subcells (nodes). For example, queries 11 and 12 will traverse the tree of instance 74HC74, from the root node down to the leaves. It is also possible to traverse the first layer of intermediate nodes beneath the root. Reserved key identifiers, "complete" and "unique," specify whether the returned list is a complete tally of all instances or a list of unique object types only.

Cell selection

The query language described above can aid IC designers in choosing appropriate cells from a frame-based VLSI cell library. Choosing cells for VLSI circuit design generally depends on four major categories: function, propagation delay, area, and power consumption. These categories can be subdivided into sets of attributes. For example, attributes bit-size and type supplement the category function, where type is used to specify a particular implementation of the function. Category propagation-delay is usually described by a set of attributes such as the minimum, maximum, and typical delay associated with a particular fan-out load.

To aid cell selection, IC designers currently use verification tools for functional simulation and timing analysis. Previous work in cell selection includes the LSMS (logic synthesis and module selection) step of Carnegie Mellon University's Design Automation Project, which used a set of predictors to estimate achievable bounds of cost, delay, and power parameters.

Our approach to cell selection is based on the following sequence of tasks:

(1) Normalize and rank the constraint data.
(2) Apply the search algorithm.

Normalizing design constraints. Statistics on area and power consumption can be retrieved directly from the cell database. The value for path-delay is assumed to be the worst-case propagation delay, where

worst-case propagation delay
= maximum(tpHL, tpLH)

In the XX7486 example (with the timing data shown in Table 1), the path-delay value is derived by traversing the XX7486 tree using pointers in the owner-of and belong-to slots. Consequently,

path-delay[XX7486]
= maximum(11.5, 19.0)
= 19.0 ns for fan-out load of 10

For each cell, statistics on path-delay, area, and power consumption are quantized (normalized) and ranked on a scale of 0.0 to 10.0 (least to most critical) by

$$\text{rank} = \begin{cases} \text{scale} \times (1.0 - ((a[i] - \min)/(\max - \min))) & \text{if } \max \neq \min \\ \text{scale} & \text{if } \max = \min \end{cases}$$

where $a[i]$ is a constraint value for cell i. The value for scale is arbitrarily set at 10.0, while min and max are the global minimum and maximum values, respectively.

The following example illustrates the ranking of design constraints:

Instance	Path-delay (ns)
cell_i	20.0
cell_j	25.0
cell_k	15.0

From the above list of candidates, the following parameters are computed:

max = 25.0
min = 15.0
rank of cell_i = 5.0
rank of cell_j = 0.0
rank of cell_k = 10.0

Therefore, cell_k is the best candidate among the three cells in terms of path-delay.

Prioritizing rules. Criteria for resolving design trade-offs in terms of speed, area, and power can be obtained by specifying a weight for each constraint. This strategy of prioritizing the selection rules is simple but effective, as shown by a rule-base example:

```
fput(rule1; path_delay 10.0;
     area 8.0; power 1.0)
fput(rule2; path_delay 10.0;
     area 5.0; power 1.0)
fput(rule3; path_delay 10.0;
     area 3.0; power 1.0)
fput(rule4; path_delay 8.0;
     area 8.0; power 1.0)
```

The constraint weights (ranging from 0 to 10) are arbitrarily set by the user to specify their relative importance in the selection process. In the above rules, for example, path-delay is most critical and power is least important to rule1. The rules are applied in ascending order. For example, if there exists among the candidates a cell_k with path-delay ranked 10.0, area ≥ 8.0, and power at least 1.0, then rule1 is fired and cell_k is selected. If rule1 is not fired, the next rule is applied and the process repeated until success is reported or the list of rules is exhausted.

```
(1) Group candidates into a preliminary cell-list P(f)
        /* P contains a list of possible candidates that satisfy the design
        specification (e.g., bit-size, etc.) for each function f */
(2) for each empty preliminary list, announce failure for f.
(3) Create a working list W consisting of the first instances
    from each nonempty preliminary list.
(4) for each instance in W,
      apply rule-set;
      if any rule is fired then
        append instance to S;
          /* S = { final selected cells } */
      else the function of the particular instance is tagged "problem,"
        and the "problem" function is appended to F.
          /* F = { problem functions } */
(5) for each "problem" function of F,
      if preliminary list is empty then announce failure and exit;
      else choose the next instance from the preliminary list and update W,
        and goto step 4.
          /* Backtrack and choose new instance */
(6) Return S.
```

Figure 7. Cell selection scheme based on a backtracking algorithm.

```
SelectCells(function-list F)
{
    Initialize selected cell-list S;
    for each function f in F, group cells which satisfy the design specifications
        into a cell choice list.
    for each empty choice list, announce failure for f;
    for each nonempty choice list,
        rank and sort instances under each constraint;
        apply rule-set to the first instance of the sorted list;
        if at least one rule is fired then
                append instance to S;
                notify user of the rule fired;
        else announce failure for f;
    return(S);
}
```

Figure 8. Pseudocode (in C-like notation) of the non-backtracking algorithm.

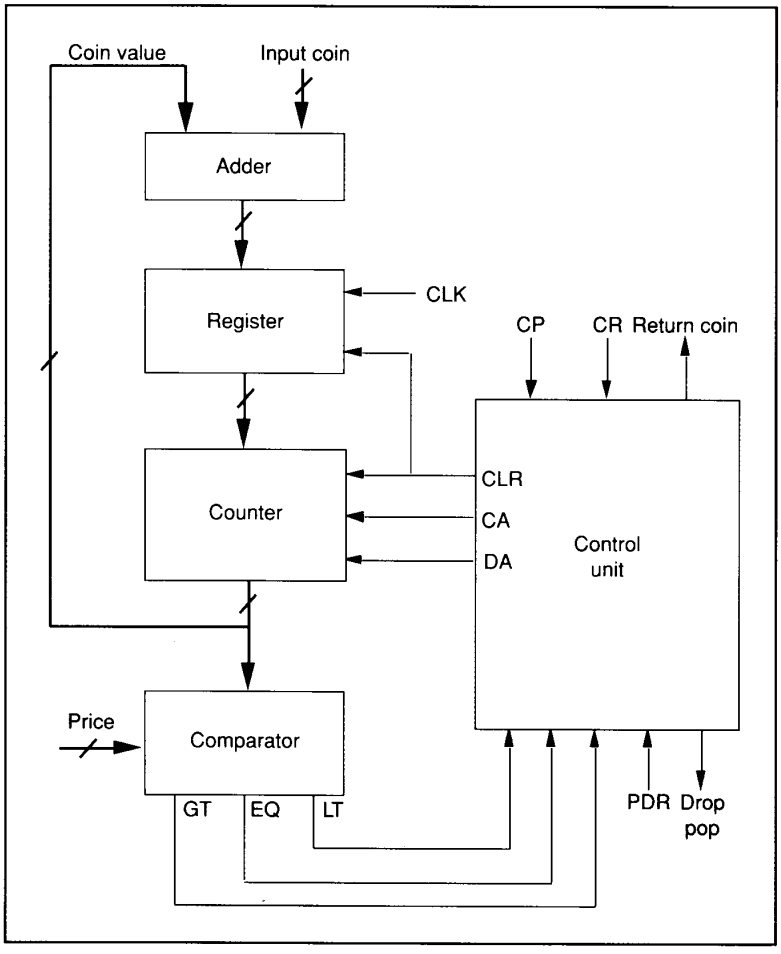

Figure 9. A vending machine system controller.

Design trade-offs are resolved by emphasizing the importance of each constraint in a selection rule. For example, if area is critical, then more weight is placed on that attribute. The ordering of rules, along with their weight factors, allows user control of cell selection from a list of possible candidates and resolves simple design trade-offs.

Backtracking algorithm. One approach to the overall cell-selection scheme is based on the backtracking algorithm outlined in Figure 7 (preceding page). Essentially, the backtracking algorithm is an exhaustive search. It quits when the first solution is encountered or when the instance choice-list for a particular function is exhausted. The biggest drawback of this backtracking approach is that it does not optimize the solution (the selected cell set). Since candidates are not sorted, cells at the top of the lists are evaluated first; better candidates deep in the list may not be considered at all. However, for small lists and cases where optimized solutions are not important, this approach can be quite fast.

Non-backtracking algorithm. The non-backtracking algorithm closely resembles the backtracking approach. The major difference is that the non-backtracking algorithm relies on a sort routine that ranks and sorts possible candidates according to design constraints. Since the candidates are sorted, backtracking is not necessary. User-specified rules are applied successively, in descending order, to the list of candidates. Hence, unlike the backtracking approach, the non-backtracking approach employs some form of optimization.

The pseudocode (in C-like notation) of the non-backtracking algorithm appears in Figure 8. Notice that each instance may satisfy more than one rule in a given rule set. In such cases, only the highest prioritized rule is reported to the user.

Test example

We used the design of a prototype vending machine to benchmark our selection algorithms, which ran on top of the frame-based database system. Figure 9 shows the vending machine's system controller. Basically, it consists of an operation unit and a control unit. The operation unit consists of an adder, register, counter, and comparator. The control unit is imple-

mented using a programmable logic array (PLA) with input and output registers. The mnemonics for the control signals are

CP:	coin present
CR:	changer ready
CA:	clear accumulator
DA:	decrement accumulator
PDR:	pop drop ready
GT:	accumulated coin value greater than price
EQ:	accumulated coin value equal to price
LT:	accumulated coin value less than price.

Assuming a four-bit data bus, the design specifications of the components implementing the system controller can be expressed as

```
fget(adder; where bit_size = 4
  & technology = "cmos")
fget(register; where bit_size = 4
  &type = "parallel-in/parallel-out"
  & technology = "cmos")
fget(counter; where bit_size = 4
  & type = "synchronous, up/down"
  & technology = "cmos")
fget(comparator; where bit_size = 4
  & technology = "cmos")
```

Notice that the cell database does not track the control unit, since it can be automatically generated using finite-state machine or PLA compilers.

We evaluated performance of the selection algorithms using a test database of dummy entries with randomly generated specifications. Figure 10 compares the backtracking and non-backtracking algorithms in terms of CPU access time on a VAX-11/780 minicomputer. The programs are coded in C, and hash tables are used in both cases. Generally, the speed of the backtracking algorithm depends on the selection rules, that is, tougher criteria require more backtracking until a solution satisfies all the conditions in the rules. On the other hand, the speed of the non-backtracking algorithm depends on the number of entries in the preliminary choice lists, that is, more entries mean a longer sorting time. With the preliminary choice lists sorted, the selection rules are applied only once, and backtracking is not required.

Figure 10's approximately linear curves confirm the algorithms' exhaustive-search nature. The speeds can be dramatically improved if the parallelism inherent in our selection algorithms is exploited and the database has sophisticated access methods.

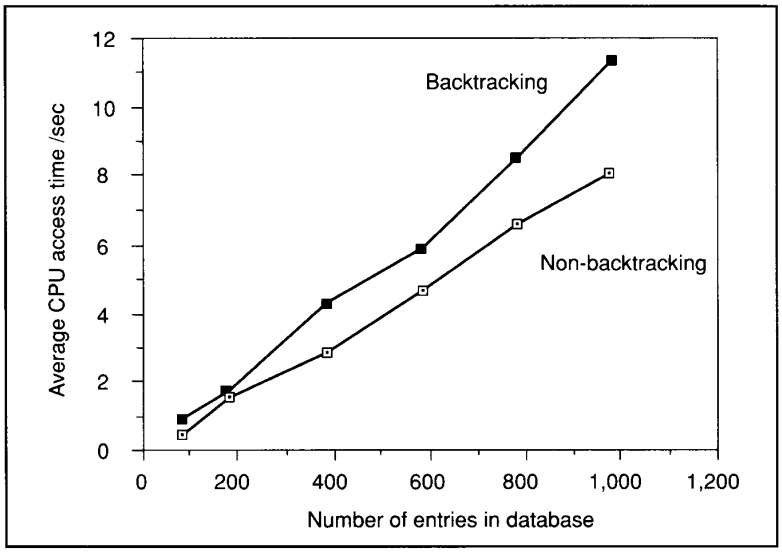

Figure 10. Performance analysis of the selection algorithms.

Overall, our case study supports the need for cell-selection mechanisms in integrated CAD or DA systems. The characteristics of our prototype frame-based database system, especially the strategy of specifying a weight factor for constraining selection rules, make it a valuable tool for ASIC design. Furthermore, the framework is general enough for other CAD-related applications.

However, the selection algorithms have two limitations. First, they do not consider cell shapes and I/O positions, criteria that may be crucial for global routing of selected cells. Second, because of shape, the selection of one cell may affect the selection of others. This interdependency needs to be properly identified and addressed. It is also possible that a particular cell's ranking can be made dependent on the number of times it's instantiated in the design database.

In the frame-based system, using pointers to parent-child relationships between frames requires a lot of extra memory. The approach is also prone to collapse of data integrity in the case of a fault in the pointer representations. In this aspect, relational databases such as Ingres are preferable because the relational data model uses a concatenation of keys instead of the single pointer used by the frame-based data model.

On the other hand, the frame-based cell library for digital circuits can also be applied to analog circuits. Information on analog circuits, such as gain and frequency response, and SPICE parameters, such as the transconductance parameter, channel length modulation parameter, threshold voltage, bulk threshold parameter, and channel surface mobility, can be encoded in a similar manner.

Finally, our case study is not intended to favor independent CAD systems. Rather, it is an attempt to highlight major issues, ones we can all agree on, involved in building a standardized, integrated CAD/CAM system. ■

Acknowledgment

The authors wish to thank Lisa R. Anderson for her help with the figures and Harold Szu of the Naval Research Laboratory for his encouragement and support. Also, special thanks go to several anonymous referees whose thoughtful and interesting comments helped improve the quality of this article.

References

1. R.H. Katz, "Computer-Aided Design Databases," *IEEE Design & Test*, Vol. 2, No. 1, Feb. 1985, pp. 70-74.

2. J. Newkirk and R. Matthews, *VLSI Designer's Library*, Addison-Wesley, Reading, Mass., 1983.

3. *CMOS3 Cell Library*, D.V. Heinbuch, ed., Addison-Wesley, 1988.

4. M.A. Ketabchi and V. Berzins, "Modeling and Managing CAD Databases," *Computer*, Vol. 20, No. 2, Feb. 1987, pp. 93-102.

5. R. A. Lorie and W. Plouffe, "Complex Objects and Their Use in Design Database," *Proc. ACM SIGMOD/IEEE Eng. Design Applications*, ACM, New York, 1983.

6. M. Stonebraker, B. Rubenstein, and A. Guttman, "Application of Abstract Data Types and Abstract Indices to CAD Data Bases," *Proc. ACM SIGMOD/IEEE Eng. Design Applications*, ACM, New York, 1983, pp. 107-113.

7. R.H. Katz et. al., "Design Version Management," *IEEE Design & Test*, Vol. 4, No. 1, Feb. 1987, pp. 12-22.

8. D.S. Batory and W. Kim, "Modeling Concepts for VLSI CAD Objects," *ACM Trans. Database Systems*, Vol. 10, No. 3, Sept. 1985, pp. 322-346.

9. R. Gupta et al., "An Object-Oriented VLSI CAD Framework: A Case Study in Rapid Prototyping," *Computer*, Vol. 22, No. 5, May 1989, pp. 28-37.

10. P. McLellan, "Effective Data Management for VLSI Design," *Proc. 22nd Design Automation Conference*, CS Press. Los Alamitos, Calif., Order No. 635 (microfiche only), 1985, pp. 652-675.

11. T. Neuman, "On Representing the Design Information in a Common Database," *Proc. ACM SIGMOD/IEEE Eng. Design Applications*, ACM, New York, 1983, pp. 81-87.

12. *CMOSPW Standard Cell Library*, UW/NW VLSI Consortium, Univ. of Washington, Seattle, Wash., 1984.

Simon Y. Foo is an assistant professor in the Electrical Engineering Department of Florida State University and Florida A&M University, Tallahassee. His research interests include VLSI CAD, microelectronics, and artificial neural networks.

Foo received his BSEE, MSEE, and PhD degrees in electrical engineering from the University of South Carolina in 1983, 1984, and 1988, respectively. He is a member of Eta Kappa Nu, the IEEE Computer Society, ACM SIGDA, and the International Neural Network Society.

Readers may contact Foo at the Dept. of Electrical Engineering, FAMU/FSU College of Engineering, Florida State Universtiy, Tallahassee, FL 32316.

Yoshiyasu Takefuji is an assistant professor of electrical engineering at Case Western Reserve University in Cleveland, Ohio. Before joining Case Western in 1988, he taught at the University of South Florida and the University of South Carolina. His current research interests focus on neural networks and parallel processing. He is the author of more than 40 technical papers and two books published in Japanese, *Digital Circuits* (Ohmsha, 1984) and *Neural Computing* (Baifu-kan, 1990)

Takefuji received his BS, MS, and PhD degrees in electrical engineering from Keio University, Japan, in 1978, 1980, and 1983. He is a member of the IEEE Computer Society, ACM, and the Neural Network Society.