

INFORMATION BASED DESIGN ENVIRONMENT

Ole Bentz, David B. Lidsky and Jan M. Rabaey

Electrical Engineering & Computer Sciences

University of California

Berkeley, CA 94720-1772, U.S.A.

Abstract - An information based system design environment is presented. The environment is information based because it helps users collect and manage information in a uniform fashion, independent of abstraction levels or implementation platforms. The environment is particularly aimed at providing feedback, design guidance, and support for design exploration. The concepts are demonstrated in a prototype called the "Design Agent."

INTRODUCTION

Designers of modern electronic systems face increasingly complex problems. Systems usually consist of numerous heterogeneous entities whose design requires highly specialized skills. The available design tools and frameworks can facilitate the design process, but they offer little in terms of design exploration and guidance. Designers must make decisions that significantly affect the outcome of the design process, but they are forced to gather information to guide the decisions by complex tool invocations and data operations. As a consequence, it is often too time consuming to explore more than a few design alternatives.

Consider the design of an FIR filter. How does a designer answer questions such as "How large would an ASIC implementation be?" or "Would a TMS320 meet my real time constraint?". The answers often involve performing tedious spreadsheet-type calculations, reading data sheets and documentation, and perhaps some complex tool invocations. Furthermore, such answers often require specialized procedures at each level of abstraction (e.g. algorithm, architecture, or circuit level). For example, while estimating power consumption at the architectural level requires one sequence of tool-specific commands, different tools and commands are required at the gate level.

Traditional design tools and frameworks are "task based" because the main mode of interaction is based on performing such tasks as executing tools and manipulating data. In contrast, in "information based" design, the main mode of interaction is based on the request and manipulation of information in the form of predictions, feedback, guidance, etc. An information based design environment provides ways to collect and manage information in a uniform fashion, independent of abstraction levels or implementation platforms.

In the proposed environment, the focus is to help a designer collect and manage relevant information at any point in the design trajectory using a uniform interface.

The environment relieves a designer of having to know how information is obtained, and may employ many different techniques for finding information. The techniques include the search of previous designs, manipulation of data (e.g. by extrapolation or interpolation), estimation, simplified or full synthesis, and translation of the obtained data into relevant numbers. The designer interacts with the design environment by directly asking for specific feedback rather than performing a number of tasks to indirectly obtain the desired results.

The design environment requires knowledge about the context in which a designer works. This context, specified using a construct called the *design domain*, determines how information requested by a designer is obtained.

These concepts have been gathered in a prototype called the “Design Agent.” This name was chosen because the environment acts as a personal assistant for the designer. The Agent interacts with the designer, invoking tools and accessing data on the designer’s behalf (Figure 1).

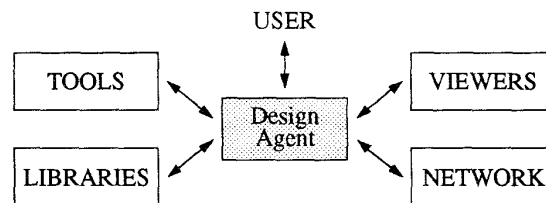


Figure 1. The user interacts directly with the Design Agent. The Agent interacts with tools, libraries, viewers and the network.

DESIGN DOMAINS

For the design environment to gather relevant information, it is important to have a flexible way to specify what data can be provided and which actions can be performed. The specification must be an abstraction of the real world such that the environment can use it to provide meaningful data regarding a design. In the Design Agent such an abstraction is called a *domain*.

A domain is a bounded design space template, defined by a set of relevant description methods, views, and actions, and is characterized by a set of parameters and constraints.

The meaning of the five elements of a domain is briefly discussed below.

Description Methods: The relevant languages or representations that are used to specify design entities in this domain. For example, C, C++, VHDL, state charts, and data flow graphs are viable description methods.

Parameters: Variables used in parameterizable design specifications (e.g. the taps

of an FIR filter, or the bitwidth of an ASIC datapath), and variables used to bound constraints (e.g. real time sample rate to bound the execution time of an algorithm or die size to bound ASIC chip area).

Constraints: Inequalities that quantify the bounds of a design, such as ASIC chip area \leq MaxArea, supply voltage \leq 5 V. Constraints are also used to define which domains are compatible with each other.

Views: Information about a design is obtained through a view. A view abstracts the information inside an entity into a simple representation (a number, a formula, a text file, etc.). Examples of views include design descriptions (e.g. source code, layout) as well as data that requires some form of analysis (e.g. power, area).

Actions: Specific design actions that are meaningful within the domain. This includes optimizations, synthesis and design refinement.

Domains can contain a wide range of information depending on the design context that they model. However, many domains are similar and can often be modeled as a subset of one another. For example, a “real time FIR filter” domain models a subset of a “real time filter” domain. Therefore, it is convenient to derive a number of specific domains from more general ones, such that the specific domains *inherit* the properties of the general domain, and add or modify the distinguishing information. The environment has three root domains built in that represent Gajski’s three dimensions of design: behavioral, structural, and physical [1]. All other domains are user-defined using the inheritance scheme.

Domains encourage extensive reuse. They can be reused as is or form a basis for more specific domains. They are portable and can be shared between designers and organizations.

Example: A designer who works on digital filter algorithms can define a domain which abstracts the digital filter design space (Figure 2). The relevant description methods are Silage and C. A parameter is defined to constrain the critical path of the algorithm, which is measured in number of operations. In this simple example three different views are considered relevant: The first view causes the source code of the algorithm to be displayed in a text editor. The two other views are the total operation count and the number of operations on the critical path. When the designer requests these views, the Design Agent invokes the tools that are necessary to analyze, extract, and present this information. Two types of actions are possible: Simulation of the behavior of the algorithm, and optimization of the algorithm (e.g. by behavioral transformations).

The designer can derive a more specific domain called the FIR filter domain, by *inheriting* the digital filter domain, then modifying or adding new information to it. Thus, the description methods and the actions remain identical to those in the parent domain, while a new parameter, *taps*, is introduced to represent the number of taps in FIR filters. The FIR domain is specific enough to allow a number of views to be modeled by symbolic equations. This creates an alternative to finding the quan-

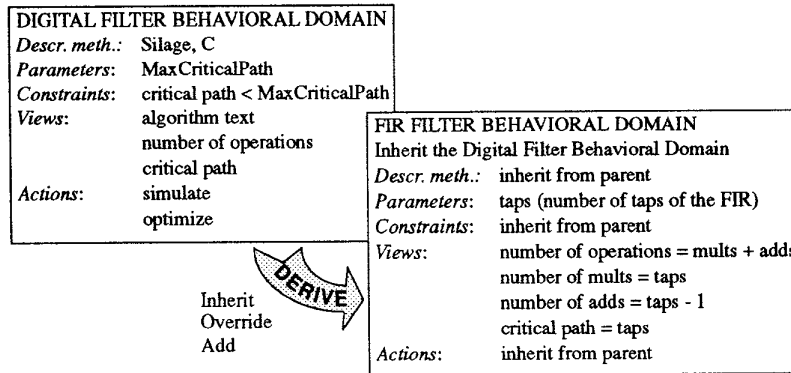


Figure 2. Inheritance makes it easy to define new, more specific domains.

tity ‘critical path’; rather than analyzing a file, a parameterized equation is evaluated. This makes it possible to provide information about filter objects before a precise formulation of the algorithm has been developed. It is conceivable to model the complexity of most DSP algorithms such as filters, FFTs, DCTs and data encoders in specific domains.

Domains form a template which represents what is common to a class of designs. To model a specific design project, a *design object* must be constructed.

Design Object: A design object represents a specific design project. It keeps track of design files, versions, configurations, history etc. A design object can either be hierarchically composed of other design objects, or it can be a “flat” entity.

Design Context: A designer establishes a *design context* for a design object by selecting a combination of domains. There are limitations on which domains can be selected for a design object. For example an FIR domain would not be appropriate for an IIR filter. Some of these limitations are captured in the *description methods* field in the domain.

The effect of placing a design object in a context can now be seen. The domains lend their parameters, constraints, views, and actions to the design object, which in turn assigns specific values to the parameters if necessary. The designer can now get information about an object by simply requesting a view. In addition, actions can be invoked to perform transformations, optimizations, refinement, etc.

Example: A 128-tap FIR filter must be implemented using an ASIC library in a 1.2 μm technology. A specific design instance is defined either by a number of parameters or by a code specification. To put this design object in context, the FIR domain is selected (Figure 2). The FIR domain models behavioral features of the filter, but does not give any implementation-specific information. Implementation aspects can be provided by selecting an ASIC (behavioral) library domain (that provides information on the cost and performance of the available computational modules), and an ASIC structural domain that models the cost of interconnecting the modules.

In addition, a physical domain is selected which defines technology-dependent issues such as the minimum feature size (1.2 μm), and the scaling of delay and power models with the supply voltage. In this context, feedback about maximum clock rate, maximum sample rate, min and max bounds for area, etc. are obtained. The accuracy of the results depends on the accuracy of either the available models (equations) or the available tools.

Suppose instead a designer wants the FIR filter implemented on a DSP processor. The design context representing this scenario can be established by keeping the same FIR domain, replacing the ASIC domain with a DSP processor domain, and selecting a physical domain that defines technology-related parameters (e.g. clock speed and power). The FIR filter is placed into a completely different context, yielding instantaneous feedback on the projected real-time execution rate and code size.

INFORMATION GATHERING USING DOMAINS

Before a designer requests information about a design, the Design Agent has to decide which views are available in the current context. The Agent uses the list of relevant views from the domains and the available design descriptions (files) from the design object to determine which views can be made available.

The designer can get information by requesting one of the available views. The Design Agent probes the domains in the design context to find out how to satisfy the request. The Agent then runs the required tools, performs the necessary data operations, and presents the extracted information to the designer.

If the designer is not satisfied with the way information was gathered, new methods can be captured interactively and incorporated into the domains.

THE UNDERLYING SYSTEM

The proposed environment has been incorporated into a prototype called the "Design Agent." The Agent has several capabilities that support information gathering. Ideas presented in the design methodology literature have influenced the development of these capabilities. Kleinfeldt *et al.* give an excellent overview of this field [2].

Domains define how views are obtained and actions performed by specifying a methodology and a list of file types. A methodology is a sequence of tool executions or data manipulations that are specified in a textually based extension language. Methodologies can be nested, and allow conditional constructs. The list of file types specify the types of files that are appropriate as arguments to a methodology in a given design context. Examples of file types include C, VHDL, and EDIF files. Each type corresponds to a powerful, object oriented construct that allows for arbitrary customization of file usage and handling. Object oriented representation of design data has been used in environments such as the Engineering Information System (EIS) [3].

The Design Agent processes a user's request by executing each of the statements in the definition of a methodology. When encountering a statement that requires a tool to be executed the Agent will look for a tool encapsulation that defines how and where to invoke the tool. The encapsulations are similar in scope to the CAD Framework Initiative's Tool Encapsulation Specification (CFI TES) [4], with additional constructs to allow for remote execution of tools. Further extensions that deal with the interactive nature of many modern CAD tools are planned.

Before the Agent runs a tool, it checks the file history and time stamps to avoid unnecessary executions. If an execution is necessary, the Agent sends a command to the tool's host computer. The Agent has communications capabilities that provide access to computational and storage resources distributed across the network created from the communications and networking software in the "Henry" environment [5].

The Agent can assist users with file management tasks, such as creating and deleting files, keeping track of different versions of the design, and maintaining backup copies. This capability is necessary to support design exploration where several versions of the design need to coexist.

The graphical user interface provides access to the internal data structures of the design environment. Menus, representing views and actions, are adjusted dynamically to reflect the current state of the design context. For example, the option to view "area" may not be available until an architectural domain has been selected.

Implementation: The Design Agent has been implemented in an object oriented extension to the Tcl language [6] called "incrTcl" [7]. The object oriented programming style is suitable for representing several of the concepts discussed. For example, domains are defined as classes, and more specific domains can therefore easily be derived by inheritance. Tcl is suitable as an extension language, enabling users to write powerful customizations of the environment. Figure 3 shows how the FIR filter domain is specified in incrTcl. Figure 4 shows the UI of the Design Agent.

<pre># FIR Filter Domain -- # DOMAIN FIR_Filter_Domain { inherit Digital_Filter_Domain constructor {} { # Desc. Methods are given on 'cmdTags' lines parameter { Label = "Number of taps" Name = taps VarType = string } # Constraints are inherited from parent domain. view { Label = "Number of operations" Name = operations cmdTags = opCount {silage-file dataflow-file} model = "mults + adds" } view {</pre>	<pre> Label = "Number of multiplications" Name = mults cmdTags = multCount {silage-file dataflow-file} model = "taps" } view { Label = "Number of additions" Name = adds cmdTags = addCount {silage-file dataflow-file} model = "taps-1" } view { Label = "Critical Path" Name = criticalpath cmdTags = critPath {silage-file dataflow-file} model = "taps" } # Actions are inherited from parent domain. } public taps 0 }</pre>
---	--

Figure 3. Specification of the FIR filter domain.

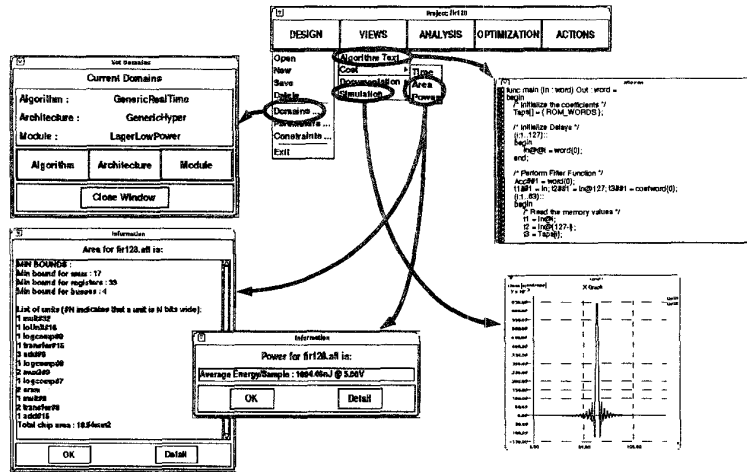


Figure 4. The Design Agent's User Interface.

EXAMPLE

This example describes a design flow from a high level behavioral specification to a layout. Salient features of the Design Agent are highlighted throughout the example. The focus is on getting predictions and feedback about the power consumption of the design, a 128-tap FIR filter.

Before investing time in writing a full behavioral specification of the filter, the designer can get early predictions about the design. By selecting the FIR domain (Figure 2), and by specifying the number of taps, predictions of power consumption can be obtained. The design context is purely behavioral, so it is impossible to directly estimate power. However, behavioral characteristics can help predict relative values, enabling comparisons between design alternatives (e.g. various FIR mutations). The Design Agent uses equations specified by the domain to calculate these quantities, since a full FIR specification has not yet been written (Figure 5).

Operation Type	Operation Count
Memory Reads	256
Memory Write	127
Multiplications	128
Additions	127
Total	638

Figure 5. Operation counts for the FIR algorithm.

FIR Version	Op. Count	Crit. Path
Direct Loop	638	384
Symmetric Loop	510	320
Loop Pipelined	638	256

Figure 6. Comparison table.

Operation Type	Operation Count	Switching Cap.	Total Cap.
Memory Reads	256	80 pF	20 nF
Memory Write	127	87 pF	11 nF
Multiplications	128	65 pF	8 nF
Additions	127	4 pF	0.5 nF
Total	638		40 nF

$$Power = \frac{TotalCap \times Voltage^2 \times f_{clock}}{ClocksPerSample} = \frac{40nF \times 5V^2 \times 20MHz}{384} = 52mW$$

Figure 7. The result of requesting “power consumption” in the ASIC cell library domain.

To continue the design exploration, the designer writes a behavioral specification (in either C or Silage) of the filter. The exploration is supported by the environment in two ways: design files are managed such that different versions can exist simultaneously, and the feedback obtained about versions can be presented in a table or graph for comparison (Figure 6). Different versions of the FIR filter can be generated either automatically by behavioral transformation tools, or manually. For each version, the Design Agent extracts the relevant information, obtaining more detailed feedback than the equations used before. The designer chooses another domain to make the design context more specific. The chosen domain represents an ASIC cell library that has been characterized for switching capacitances. A measure for power can be obtained by simple spreadsheet calculations performed by the Agent. Initially, a number of defaults are used in this domain to calculate power, including supply voltage and clock frequency. The designer can override each of the defaults. The Agent’s calculations are shown in Figure 7.

The ASIC library domain models individual cells, but not the effects of combining them. By adding an architectural domain to the design context, more accurate power estimates can be obtained. In this case, the “Hyper domain” is selected. Estimates of power come from estimators and includes contributions from active area, routing and control. Signal correlations are assumed that of white noise. After setting the sample rate and the supply voltage to desired values, the designer can obtain power estimates.

The designer creates an architecture for the filter, either manually or using a framework such as the Hyper high level synthesis system [8]. The design context is automatically expanded to include an architectural description. At this level of abstraction, power can be estimated more accurately. However, the designer requests power in the same fashion as before. In this case, estimates come from an architectural power analysis tool SPA [9], which utilizes simulations to obtain data correlations.

The designer generates the layout for the architecture, either manually or using a

silicon compilation system such as the Lager system [10]. The design context is expanded to include a layout description. Power can now be obtained through the Agent, from either IRSIM-CAP [11] or SPICE [12] simulations.

Throughout the design process, the Design Agent maintains summaries of the collected data such that the designer can always review the status and history of the design.

STATUS AND FUTURE WORK

The prototype of the Design Agent has demonstrated the feasibility of the information based design concept. The Agent incorporates the features described in this paper, but some features are being refined or expanded to make the environment more flexible.

The domains are currently being extended to allow specification of constraints (design bounds and compatible domains). Once in place, we are planning to explore the usefulness of letting the Design Agent search through a design context's compatible domains to find answers to user requests that may not be obtainable in the current context.

Management of constraints across levels of hierarchy is an important feature. A global system constraint translates into different constraints on composing elements. For instance, heat at the system level means power at the circuit level. Therefore, constraints and costs may have to be "translated." This translation process is specific to a domain and has to be modeled in the domain specification. Also, constraints may have to be broken up over the composing elements. There is no unique solution but rather a grid of points in the design space. One approach is to present the design space open to the designer through spreadsheets. The constraint management problem will receive attention in the future.

The prototype will also be extended to provide design guidance by suggesting steps of actions to take. The domains will be expanded to capture guidance information in the form of statements conditional upon views and parameters.

CONCLUSIONS

The concept of *information based design* has been presented and encapsulated in the Design Agent. The Agent frees a designer from having to know the details of how to obtain information, allowing the designer to focus on creating the design. By providing an abstract way to model design contexts using domains, the Design Agent also encourages the designer to capture and reuse information from current and previous designs.

ACKNOWLEDGMENTS

We would like to thank Mario Silva, Lisa Guerra and Asawaree Kalavade for fruitful discussions of this topic.

REFERENCES

- [1] D. D. Gajski and R. H. Kuhn, "New VLSI Tools," *IEEE Computer Magazine*, Vol. 16, No. 12, December, 1983, pp. 11-14.
- [2] S. Kleinfeldt *et al.*, "Design Methodology Management," *Proc. of the IEEE*, Vol. 82, No. 2, February, 1994.
- [3] "EIS Phase II Final Design Review Proc.," San Francisco, California, March 19-21, 1991.
- [4] CAD Framework Initiative, Inc., "Tool encapsulation specification standard," CFI Doc. DMM-91-G-1, 1991.
- [5] M. J. Silva and R. H. Katz, "Active Documentation: a New Interface for VLSI Design," *Proc. of the 30th ACM/IEEE Design Automation Conf.*, Dallas, Texas, June 1993, pp. 654-660.
- [6] J. K. Ousterhout, "Tcl and the Tk Toolkit," *Addison-Wesley Publishing Company*, Reading, Massachusetts, 1994.
- [7] M. J. McLennan, "[incr Tcl]: Object-Oriented Programming in Tcl," *Proceedings of the Tcl/Tk Workshop*, University of California at Berkeley, June 10-11, 1993.
- [8] C. Chu *et al.*, "Hyper: An Interactive Synthesis Environment for High Performance Real Time Applications," *Proc. IEEE ICCD Conf.*, Boston, November 1989.
- [9] P. Landman and J. Rabaey, "Power Estimation for High Level Synthesis," *Proc. of EDAC-EUROASIC '93*, Paris, France, February 1993, pp. 361-366.
- [10] R. W. Brodersen, editor, "Anatomy of a Silicon Compiler," *Kluwer Academic Publishers*, Boston, Massachusetts, 1992.
- [11] P. Landman and J. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method," *Transactions on VLSI Systems*, June 1995, pp. 173-187.
- [12] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *Technical Report ERL-M520*, University of California, Berkeley, 1975.