# Design Process Management for CAD Frameworks

Margarida Jacome, Stephen Director

EDRC 18-27-92

# Design Process Management for CAD Frameworks*

Margarida F. Jacome and Stephen W. Director
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15217

**Abstract**

*We introduce a new mechanism for planning and managing the VLSI design process. This mechanism significantly enhances the capabilities of CAD frameworks, relieving designers from dealing with low level details, thereby allowing them to concentrate on the more innovative aspects of design. A model for representing design processes is described. A prototype design process manager, called Minerva, that uses this model is presented.*

## 1 Introduction

VLSI design has become a complex undertaking that involves a variety of activities and the use of a large, diverse set of CAD tools. During the course of design many data files are generated and the designer is faced with the immense task of keeping track of myriad details. In order to make the design process itself more tractable, CAD frameworks have been developed. Among other things, the purpose of these frameworks, from the user's point of view, has been to provide basic mechanisms for tool and data management. A few frameworks also provide some higher level facilities to help designers manage the design processes. Typical of such facilities are means for executing static design flows and capture of design history. From the point of view of tool developers and framework maintainers, frameworks should provide convenient mechanisms for tool and data encapsulation in order to facilitate the integration of new CAD resources. These issues have also been addressed in several framework implementations. Unfortunately, CAD frameworks still fall short in the area of true design process management. Consequently, designers still expend a significant amount of time dealing with cumbersome and error prone management details. In this paper we describe an approach to enhance CAD frameworks that can significantly aid the designer in managing the entire VLSI design process.

As a means for facilitating the discussion that follows, it is convenient to view the services that can be provided by a framework in terms of four basic levels of abstraction [1]. In this view, the *component level* comprises the CAD tools and databases available to the framework, exactly as they come from their developers or distributors. Designers "working" at the component level must therefore be familiar with all the details associated with specific idiosyncrasies of each particular tool.

At the next higher level of abstraction, called the *resource level*, all component level tools and data are encapsulated, thereby providing them will well defined and consistent interfaces. Such interfaces allow resources to interact with each other through a framework. While designers working at this level are still basically manipulating individual CAD tools and data objects, the specifics of each resource are hidden by a stable, and consistent "front-end", provided by the encapsulation.

In general, CAD tools are developed as a means for performing specific *design functions* or *CAD tasks* (e.g. synthesis, verification and optimization). CAD tasks constitute therefore the first real "design management abstraction" to be implemented in a framework. Accordingly, the next abstraction level is the *CAD task level*. At this level, the semantics of specific design functions are actually modelled and represented, independent of specific CAD resources, and are further dynamically mapped, by the *CAD Task Manager*[2], to these resources. Thus, the designer interacting with the framework at the CAD task level is allowed to request execution of abstract tasks such as "verify circuit performance" rather than directly requesting tool invocations such as "execute SPICE".

Although working at the CAD task level relieves designers from low level details concerned with CAD tool invocation, the *design process itself is* still not represented. Thus, while a CAD Task Manager provides a CAD framework with the capability to perform design functions, the actual reason for the need to perform these functions is not explicitly known by the framework. In other words, a framework whose highest level of abstraction is the CAD task level "knows how to do things" but does not know "why these things are being done". So, designers interacting with the framework directly at the

* This work is supported in part by the Engineering Design Research Center, Carnegie Mellon University, under contract no. EDC-8943164.

CAD task level are still required to map the "design problem" being solved onto the specific collection of CAD tasks available in the framework. Further, the designer must still manually guarantee consistency among decisions made during the course of design; has to sequence the execution of CAD tasks; must manage all generated data objects; and has to coordinate activities with the group of designers working simultaneously on the same project.

The highest level of abstraction is the *design process level* (or *problem level*). At this level the designer carries out design directly in terms of *design problems*, such as "design an operational amplifier to meet a set of specifications", or "verify the performance of this ALU". The *Design Process Manager* is responsible for automatically and dynamically mapping this "natural" (designer oriented) representation of problems to executable CAD tasks. In other words, it is responsible for planning the design process in the sense that it will derive the set of CAD tasks (seen as atomic operators) that have to be executed, together with the data objects involved on such executions, in order to solve a given design problem. Since the Design Process Manager relieves designers from all design process *planning* and *data management* activities, the obvious advantage of interacting with a framework at the proposed abstraction level is precisely to allow designers to concentrate their effort on the creative and exploratory aspects of design. Exploring alternative implementations, choosing appropriate specifications, and investigating trade-offs among performances while deriving those components, are examples of such creative/exploratory activities to which designers working at the design process level focus their attention.

In our view, the planning and managing capabilities provided by a Design Process Manager are useful to both the novice and experienced designers, so it should not in anyway limit the designer's flexibility or autonomy in implementing a design. Thus, the Design Process Manager must be able to cope with non-routine design problems, which are typically loosely defined at the beginning of the design process, as well as support any design methodology (e.g., bottom-up, top-down).

In order to demonstrate the suitability of our design process model, a prototype design process manager, called *Minerva*, has been developed. While Minerva can be used as a stand alone meta-tool, its true benefit is only fully realized when it is incorporated into a CAD framework, such as the *Odyssey* framework [1], that supports resource and CAD task management.

Before proceeding it is worthwhile to place our work within the context of other related work. Knapp's Design Planning Engine (DPE) [3] generates plans for invoking CAD tools to realize design functions. Thus, DPE employs planning at the *design task level*, rather than at the *design problem level*. Harjani, et. al., [4] employs hardcoded plans for encoding multi-variable numerical optimization techniques in an analog circuit synthesis framework. These hardcoded plans are conceptually similar to complex CAD tasks (they statically encode a given design function), and therefore [4] also focuses on the design task level. Dewey's *conceptual design* [5] work, developed to aid the designer in the decision making process that occurs during the early stages of the design process, is most closely related to the work described here. However, Dewey did not attempt to explicitly represent design problems or to map the set of design decisions developed during conceptual design into an actual executable design plan.

The remainder of this paper is organized as follows. We first describe the Minerva Design Process Representation, or MDPR, which is our formal model for representing VLSI design processes. Then, in Section 3, we describe the dynamic modeling of the design process implemented by the Design Process Manager. Minerva, the prototype design process manager, is described in Section 4. Some conclusions are given in Section 5.

## 2 Minerva Design Process Representation (MDPR)

We begin this section by briefly reviewing the main phases of a typical VLSI design process. Of special interest are the strategies employed by designers in order to control complexity, since it is these strategies, and how they are used, that actually define a "design methodology".

The initial phase of design is *problem definition*. During this phase, the designer develops detailed specifications of the system or circuit to be designed. Such specifications may include the desired behavior and the functionality of the circuit, the general timing constraints, as well as environmental conditions such as the temperature range over which the circuit is expected to operate. As the specifications stabilizes, the next phase, conceptual design, begins.

*Conceptual design* involves the investigation of alternative, often high level, design options in terms of their consequences on performance, without actually undertaking design and fabrication steps [5]. During this phase, designers may need assistance in the form of qualitative advice and/or quantitative information that can be used to discriminate among the various options.

Upon completion of conceptual design, designers are typically faced with two options. If the size of the design problem is such that it can be solved using an available CAD tool suite, the designer may choose to attack the design problem as formulated. Alternatively the designer may employ some type of decomposition strat-

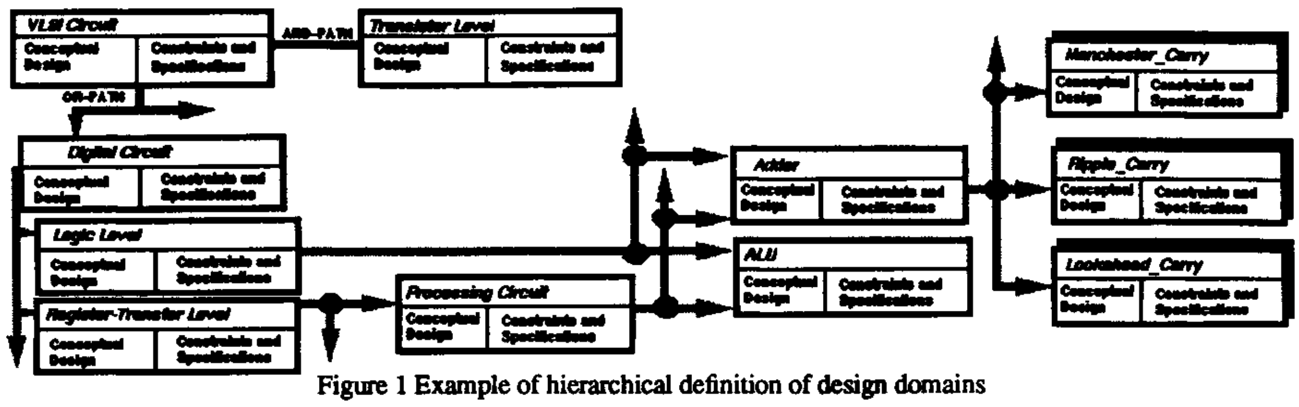

Figure 1 Example of hierarchical definition of design domains

egy (either structurally or functionally), in order to create a number of smaller design problems. Specifications for each of these design sub-problems must then be developed, reflecting the restrictions imposed on the original circuit by the original specifications. Conceptual design is then performed for each of these sub-problems taking into account decisions made for the original circuit. Note that, because of the nearly decomposable nature of VLSI design problems, different designers may be simultaneously working on each of the sub-problems. Depending on the complexity of the resultant sub-problems, the designer may choose to undertake further decomposition. An important observation is that different design domains may lend themselves to different design methodologies or patterns of design activity. For instance, within the domain of analog circuits, parametric yield optimization procedures may be employed, while in the domain of digital circuits such techniques may not be necessary.

Let us now introduce a formal means for representing the VLSI design process.[1] We begin by defining the VLSI design problem in terms of three components: *design domains*, *design objectives* and *design specifications*. In what follows, we discuss each of these components and discuss the role that decomposition plays in controlling complexity.

## 2.1 Design Domains

A *design domain* is a self contained design context that may be broad, such as digital signal processing systems, or more limited, such as operational amplifiers, multipliers, adders, etc. Design domains can be defined in terms of *a static hierarchy* of *design domain knowledge templates* where each template contains two basic categories of knowledge: conceptual design knowledge and domain constraint knowledge.

Conceptual design knowledge includes: the set of specifications relevant for the particular domain; the set of *design issues* and related *design options* relevant to the particular design domain; *ordering* and *consistency constraints*[2] between design issues and related design options, respectively; and the set of relevant *discriminating factors* that are used to differentiate among the design options. Domain constraint knowledge includes: definition of formal languages for describing behavioral and structural constraints on the domain object; and actual constraints on behavioral, structural and physical characteristics for the domain of interest.

Figure 1 illustrates a hierarchy of design domain templates. In this hierarchy, descendent templates *inherit* knowledge (e.g. specifications, design issues, etc.) from their parent templates. Leaf templates that have predefined constraints describing their behavior are called *closed design domains* while other templates are called *open design domains*. Note that the knowledge contained in this representation of design domains[3] allows designers to specify, during run time, the behavior of circuits, or to specify functional or structural decompositions of circuits. This feature supports the development of new (non-routine) systems and circuits, treated as open domains.

In order to facilitate dynamic decomposition of design domains during the design process, as is often done to control the complexity of the circuit being designed, design domain knowledge templates can contain a *decomposition design issue* whose design options correspond to the possible decompositions, expressed in terms of sub-domains available to the designer. Thus, for example, a digital filter may be designed by designing multipliers, adders, registers, etc. When a design domain is an option associated with the decomposition design

1. While we are specifically focusing on the domain of VLSI design processes in this paper, our means of representing the design process may be generalized to many other domains as well.

2. Ordering constraints guide the designer towards addressing the most pervasive design issues first. Consistency constraints prevents the designer from choosing a design option that is inconsistent with previous design decsions.

3. Namely, the description languages defined in the design domain knowledge templates.

issue of an higher level design domain, ordering and consistency constraints can be stated between design issues and related design options of both domains. These constraints are included in the knowledge template of the parent design domain.

## 2.2 Design Objectives

*Design objectives* refer to, and characterize, the general types of problems that are typically addressed during the design process, e.g. "design", "verify", "synthesize", "conceptual design" and "optimize." Design objectives have varying degrees of complexity. In order to control the complexity of the design problem itself, designers often decompose design objectives into a set of sub-objectives. For example, the objective "design" may be decomposed into the sub-objectives "synthesize&optimize" and "verify". Similarly, the objective "synthesizeA-optimize" may be decomposed into the sub-objectives "conceptual design", "synthesize" and "optimize".

Ignoring for the moment design objective decomposition, realization of a design objective involves typically four steps: *design plan generation, design plan validation, design plan execution* and, finally, *execution validation.*

*Design plan generation* transforms a design problem into a partially ordered sequence of CAD tasks that reflects the design decisions made during conceptual design, the specifications imposed by the designer, and the design objective itself. This step is carried out in the *planning space*, viz. the space of all possible task sequences that can be generated to solve the same basic VLSI design problem.

The *design plan validation* step determines the existence of resources required by the individual tasks in the plan. If the plan cannot be validated there are two alternatives: backtracking, by reactivating the conceptual design objective in order to re-evaluate basic design decisions, or design objective decomposition, as discussed below.

*Design plan execution* is performed in the *design space*, viz. the space of all possible designs that totally or partially satisfy a set of design constraints with each point in the space representing a different trade-off between performance characteristics. Successful execution of a design plan results in an actual candidate solution to the design problem. Validation of the solution must then occur. If performance is unacceptable, again several different strategies may be pursued. Backtracking can be performed or, alternatively, the problem currently being addressed (or others) can be reactivated. Finally, a design plan may $lso fail because the selected tasks (or the related tools) failed during execution. In this case backtracking can be performed, or alternatively, the design objective can be decomposed.

As indicated above, design objective decomposition. may occur for several reasons. In order to achieve decomposition, the objective has to generate an appropriate set of sub-objectives, as well as a possible set of ordering restrictions for addressing these sub-objectives. It is important to note that there is no unique decomposition associated with a VLSI design problem. Therefore our model of the design process has to capture and represent expressively the nature of such diversity. Thus, when objectives are defined and represented, their decomposition can be made design domain dependent and/or dependent on *specifications defined for the objective itself*, as would be the case for enforcing a particular *design methodology.*

As was the case for design domains, each design objective can be defined in terms of a knowledge template that contain information such as: the set of specifications that can be defined for the objective; the steps during which the design objective is to be accomplished, and what to do in case of success and/or failure; possible decompositions for the objective, conditions required to activate and sequence the sub-objective, and what to do in case of sub-objective failure;

## *2.3* Design Specifications

After a given design objective is selected, within a particular design domain, the problem description is completed by a statement of *design specifications* (or a priori constraints) to be satisfied by the final solution. Design specifications can either be design objective dependent or design domain dependent. Examples of design objective dependent specifications are minimum area and minimum dissipated power, which may be associated with an optimization objective, and design methodology, that may be associated with a "design" objective. Examples of design domain dependent specifications are gain, bandwidth and slew rate, which may be associated with the domain of operational amplifiers.

Specifications can be defined in terms of *specification knowledge templates,* that contain the following information: indication of whether the specification is design domain dependent or design objective dependent; references to templates of corresponding design domains or design objectives; and restrictions or bounds on design specification assignments.[1]

Whenever problem decomposition occurs, specifications related to the original problem have to be propagated to the newly generated sub-problems. In order to accommodate this, our problem model contains *propagation constraint knowledge templates* that properly transform high level constraints into constraints associated with each sub-problem.

**1. Assignments may be values, behavioral descriptions, etc.**

## 3 The Design Process Manager

We now describe how a design process can be planned and managed given the representation of design problems described above. The design process begins with the definition of a specific VLSI design problem, called the *target problem*. Towards this end, the designer selects an appropriate design domain from among the entire set of design domains represented in the knowledge base. This choice causes the creation of an *instance* of the selected design domain knowledge template. The designer then selects a primary design objective and an *instance* of the associated design objective knowledge template is also created. Finally the designer chooses an appropriate set of specifications and assigns values to each specification in the set.

After the *target problem* has been defined, the design process can commence. The design process is basically a repetitive (quasi-cyclic) process involving two basic steps: *sub-problem selection* and *sub-problem solution*. We use a *decision tree* to represent the state of the design process. Decomposition relations among instances of design domain templates are implicitly represented by the branches of the tree. Thus, the decision tree represents the dynamic hierarchy of instances of design domain knowledge templates and associated design objective knowledge templates generated during the design process.

*Sub-problem selection* involves the choice of a design domain instance, i.e., a node in the decision tree, the choice of a design objective associated with that instance and, possibly, the choice of a set of design specifications to be satisfied by the solution. The sub-problem so selected by the designer is solved during the next *sub-problem solution step*. Depending on the design objective and the current step of this objective, the problem solving process involves one of five basic activities: conceptual design; building sub-plans; validating generated sub-plans; requesting execution of sub-plans or validating the prior execution of sub-plans.

Upon conclusion of a sub-problem solving step, a new sub-problem solving cycle is started. Notice that this basic "sub-problem selection - sub-problem solution" cycle is performed in a consistent and homogeneous fashion throughout the entire design hierarchy, independent of the abstraction level at which the design domain instances are defined (i.e. system, algorithmic, register-transfer level, logic, etc.). Also notice that our model of the design process does not enforce any specific patterns for decision tree generation and traversal.[1] Thus designers have the freedom to determine the critical components or paths of the design, and can focus on those elements first.

Whenever design domain decomposition occurs, the top-level objective associated with the parent domain must, in general, be propagated to the sub-domains. This is the way domain decomposition impacts the design objective hierarchy that is simultaneously being developed. Also, a design objective will typically be generated (at the level of the parent domain) whose goal is to rebuild the original element from the cells or components into which it was decomposed.

## 4 Minerva - A Prototype Design Process Manager

A prototype design process manager has been implemented, called Minerva. [2]In order to illustrate the Design Process Manager capabilities we briefly, by example, illustrate how a designer interacts with Minerva during a "problem solving cycle". When a designer selects a sub-problem, a window showing the hierarchy of design domain instances for the current decision tree is created, as shown in Figure 2. The designer is allowed to browse the decision tree, inspecting the *status* and *current design decisions* for the existing design domain instances.

Each design domain instance will have an identifiable status such as: "*non-available selectable objectives*", meaning that there are no sub-problems currently ready to be solved within this design domain instance; "*busy*", meaning that another designer is concurrently working within this design domain instance (this allows multiple designers to be working on different aspects of the same design problem simultaneously); and "*ready for selection*", meaning that there is at least one sub-problem within the design domain instance that is ready to be addressed, and that no other designer is addressing it simultaneously.

In Figure 2, design domain instances that are available for selection are indicated by having the button "Select Domain" active. A designer who wants to know why a particular domain is not available for selection can click on the button "Show Status". Also, in order to have a better insight on the design domain instance, the design can click on the button "Inspect Decisions", and the design decisions that had been made for the instance will be displayed, although they cannot be modified at this point.

After selecting a given design domain instance, a window showing the associated hierarchy of objective instances is created, as shown in Figure 2. The designer is also allowed to browse the objectives hierarchy, inspecting the *status* of the existing objective instances.

The status of a design objective instance may be: "*accomplished*", meaning that the associated sub-problem

1. The decision tree represents at each moment the sub-problems that can be addressed concurrently.

2. Minerva has been implemented in C++.

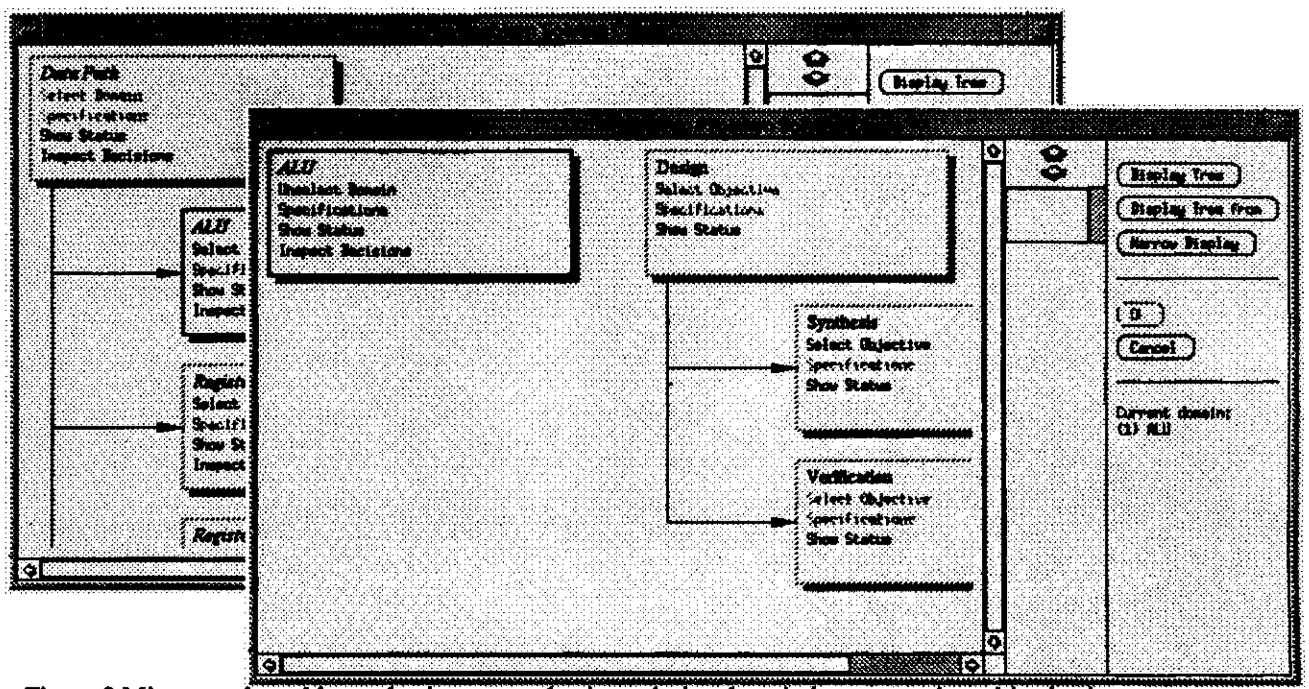

Figure 2 Minerva sub-problem selection step: selecting a design domain instance and an objective instance

was successfully solved; "*waiting for sub-objective accomplishment*", meaning that the associated sub-problem depends on the results of other sub-problems that have not yet been addressed; "*waiting for design space event*", meaning that the results of executin a plan are still not available; and "*ready for selection*", meaning that the sub-problem associated with the sub-objective has not yet been addressed, and that there is no active sequencing constraints for it.

In Figure 2, design objective instances that are available for selection are indicated by having the button "Select Objective" active. The designer can determine why a particular objective instance is not available for selection by clicking on the button "Show Status".

Upon completion of "sub-problem selection", the designer enters one the possible "problem solving" steps, conceptual design, building sub-plans, validating generated sub-plans, requesting execution of sub-plans or validating the prior execution of sub-plans. For each step an appropriate window opens directing the designer towards solution of the selected problem (space limitations preclude us from showing these windows here). Once this sub-problem has been solved, the sub-problem selection process resumes until all sub-problems have been addressed and the target problem is solved.

## 5 Conclusions

In this paper we have extended the CAD framework paradigm to support Design Process Management allowing designers to concentrate exclusively on those issues that concern the creative and exploratory phases of design. A prototype Design Process Manager, Minerva, has been realized. Currently the Minerva knowledge base supports the design of datapaths, and to perform conceptual design on DSP Filters. The framework executive component interacting with Minerva currently has available the Berkeley Synthesis Tools for logic synthesis [6]. Expansion of the knowledge base, in order to cover the analog circuit domain is underway.

## 6 Bibliography

[1] J. Brockman, T. Cobourn, M. Jacome and S. Director. "Odyssey CAD Framework". To appear in *IEEE DACT Newsletter on Design Automation.*

[2] J. B. Brockman and S. W. Director, "The Hercules CAD Task Management System". In *Proceedings of the IEEE International Conference on Computer-Aided Design*, IEEE, 1991.

[3] D.W. Knapp and A.C. Parker. A Design Utility Manager: The ADAM Planning Engine. In *Proceedings of the 23th ACM/IEEE Design Automation Conference*, ACM Press, 1989, pages 48-54.

[4] Ramesh Harjani. "OASYS-A Framework for Analog Circuit Synthesis". *Ph.D. thesis, Dept. of ECE, Carnegie Mellon University, CMUCAD-89-24.* March 1989.

[5] A. M. Dewey and S. W. Director. "YODA-A Framework for the Conceptual Design of VLSI Design Systems." In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, ACM Press, 1989, pages 380-383.

[6] G. DeMicheli. "Computer-Aided Synthesis of PLA-Based Systems". *UCB/ERL M84/31.* 1984.