A KNOWLEDGE BASED APPROACH TO VLSI CAD THE REDESIGN SYSTEM

Louis I. Steinberg and Tom M. Mitchell

AI/VLSI Project
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

Abstract

Artificial Intelligence (AI) techniques offer one possible avenue toward new CAD tools to handle the complexities of VLSI. This paper summarizes the experience of the Rutgers AI/VLSI group in exploring applications of AI to VLSI design over the past three years. In particular, it summarizes our experience in developing REDESIGN, a knowledge-based system for providing interactive aid in the functional redesign of digital circuits. Given a desired change to the function of a circuit, REDESIGN combines rule-based knowledge of design tactics with its ability to analyze signal propagation through circuits, in order to (1) help the user focus on an appropriate portion of the circuit to redesign, (2) suggest local redesign alternatives, and (3) determine side effects of possible redesigns. We also summarize our more recent research toward constructing a knowledge-based system for VLSI design and a system for chip debugging, both based on extending the techniques developed for the REDESIGN system.

1 Introduction

Artificial Intelligence (AI) techniques offer one possible avenue toward new CAD tools to handle the complexities of VLSI. This paper summarizes the experience of the Rutgers AI/VLSI group in exploring applications of AI to VLSI design over the past three years. In particular, it summarizes our experience in developing REDESIGN, a knowledge-based system for providing interactive aid in the functional redesign of digital circuits. We also summarize our more recent research toward constructing a knowledge-based system for VLSI design and a system for chip debugging, both based on extending the techniques used by the REDESIGN system.

A. A Knowledge Based Approach to Software Organization

One of the techniques which has arisen from research in AI is the *knowledge based* approach to designing a system which is to achieve some task. The essence of this approach is to ask what knowledge (i.e. what facts and reasoning abilities) is used by a human expert in solving this task, and to develop data-structures and code which represent this knowledge explicitly, rather than to have the knowledge present in the system only implicitly.

Often the facts which are most useful are a collection of rules of thumb, derived from the expert's experience, which can be represented in a natural way as IF-THEN rules and which can be used in a fairly simple reasoning process. As will be discussed below, we have found it useful to represent design tactics as IF-THEN rules, but to represent other facts about circuits in other ways.

Researchers using the Knowledge Based approach in a number of areas have found it to have several interrelated advantages over more traditional techniques for organizing software:

- *It is easier to make incremental improvements.* Since the knowledge is represented explicitly it is easier to add additional pieces of knowledge and thereby make incremental improvements in the system's capability.
- *It is easier for the system to explain what it is doing and why.* Since the facts and reasoning processes used parallel those used by a human expert, it is often feasible for a knowledge based system to automatically generate an explanation of *how* it reached its conclusions which is understandable by a human domain expert who is not a computer scientist. For example, the program may indicate the chain of IF-THEN rules which was used to make a particular decision about the design of some circuit submodule.
- *It is easier for a human expert to determine what is incorrect or incomplete about the system's knowledge, and explain how to fix it.* Since the system can indicate what knowledge was used and how it was used, it is easier for an expert to determine what is wrong. Since the system is already structured around the kinds of knowledge the expert uses, it is easier to translate the expert's description of how to fix things into an actual change to the program.
- *It is easier to interactively use a human expert's abilities.* Long before a system is capable of handling a task completely automatically, it may be possible to construct a useful interactive system, which aids the user as much as it can given its limited knowledge, and in which the user can take over and do things when he is dissatisfied with the system's recommendations. Since the system's

*This material is based on work supported by the Defense Advanced Research Projects Agency under Research Contract N00014-81-K-0394. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

0738-100X/84/0000/0412$1.00 © 1984 IEEE

way of doing things is analogous to that of the user, it can be easier to coordinate the system and the user. This ability to interleave user input with the system's processing represents a major difference between knowledge-based approaches and other kinds of approaches, for instance, to silicon compilers.

B. Applying the Knowledge Based Approach to VLSI CAD

A number of research groups are currently exploring knowledge based approaches to various aspects of VLSI CAD (e.g., [1, 2, 3, 4]). Here we describe a knowledge based system called REDESIGN which addresses the following problem of functional *re*design: Given an existing circuit, its functional specifications, and a desired change to these specifications, help the user to determine a change in the circuit that will allow it to meet the altered functional specifications without introducing undesirable side-effects.** During this work, it became apparent that providing intelligent assistance for redesign depends on two quite different types of reasoning about the given circuit.

The first essential type of reasoning about circuits is *causal reasoning* about the interrelations among signals within the circuit. This is a generalization of the notions of circuit simulation and symbolic simulation. It involves tasks such as, given a description of the streams of data being input to a component, deriving a description of the output data streams. Or, given a specification of the required characteristics of the outputs of a component, determine what characteristics must be satisfied by the inputs to the component. The subsystem of REDESIGN which has been developed to solve these kinds of problems is called CRITTER.[5]

A second type of reasoning essential to redesign involves reasoning about purposes of components. For example, given a circuit, and its specifications, explain the role played by a particular component in implementing the overall specifications. Or, determine the range of components that can be substituted for this component without violating the overall specifications.

The next section describes our work on the redesign task, and the use of causal reasoning and reasoning about purpose in REDESIGN. Subsequent sections summarize our more recent attempts to extend these ideas and our current research on developing knowledge based systems for VLSI design and chip debugging.

II The Redesign Task

In the functional redesign problem the system is given the schematic of a working digital circuit (e.g., the display controller for a computer terminal), and its functional specifications (e.g. the fact that it displays 80 characters per line, 25 lines per screen, displays the cursor at a programmable address, etc.). The system is also given a data structure called a *design plan*, which relates the circuit schematic to its specifications. Given a desired change to the functional specifications (e.g., require that the terminal display 72 characters per line), the task is then to redesign the circuit so that it will meet these altered specifications***.

The formulation of the redesign problem presented here is very similar to planning problems in the AI literature, and the issues addressed in this work are related to those addressed by others working in the areas of planning and design, such as [6, 7, 8, 9, 10, 11, 12]. Our work is also related to that of [13], which deals with recognizing circuits rather than designing them, and which addresses the relations among circuit function, structure, and purpose.

The next subsection discusses the representation of circuits, and the notions of circuit behavior and specifications. The following subsection describes the two modes of reasoning about circuits employed by REDESIGN: causal reasoning and reasoning about purpose. We then illustrate the use of these modes of reasoning by REDESIGN, by tracing its use for a specific redesign problem.

A. Representing Circuits, Behaviors and Specifications

The structure of a circuit is represented by a network of *modules* and *data-paths*. A module represents either a single component or a cluster of components being viewed as a single functional block. Similarly, a data-path represents either a wire or a group of wires. The data flowing on a data-path is represented by a *data-stream*, and the operation performed by a module is represented by a *module function*. These representations are described in [14, 5]. One aspect of this circuit representation that has been important in REDESIGN is that data-streams represent the entire time history of data values on a data-path, rather than a single value at a single time, as in many circuit simulators. This has proven to allow considerable flexibility in reasoning about circuit behavior over time.

In reasoning about redesign, REDESIGN must distinguish between what happens to be true of the circuit (we refer to this as the circuit *behavior*), and what must be true for that circuit to work correctly (we refer to this as the circuit *specifications*). Therefore, for each module function and data-stream, both behavior and specifications are recorded. For example, the behavior of a particular module may state that its output will be the sum of its inputs, delayed by 100 nanoseconds, while the specifications for that module may simply require that the output be delayed by less that 500 nanoseconds.

B. Two Modes of Reasoning about Circuits

A variety of types of questions arise when redesigning a circuit. REDESIGN uses two separate modes of reasoning to answer these questions -- one to analyze circuit operation based on a causal model of the circuit, and one to reason about the purposes of circuit submodules (i.e. their roles in implementing the global circuit specifications). These two modes of reasoning are combined to provide assistance at various stages of the redesign process.

1. Causal Reasoning

Causal reasoning answers questions such as "If input X is supplied to the circuit module, what will the output be?"

**We chose the redesign problem over the design problem as our first task primarily because it raised a number of important issues about representing and reasoning about circuits in a more tractable context than design from scratch.

***It should be noted that the example circuits used in the work on REDESIGN were not actually VLSI. Rather they were board-level circuits built from standard TTL MSI parts. However, we believe that the same techniques apply directly to VLSI circuits designed with the standard cell approach.

and "If output Y is desired, what must be provided as inputs to the module?" X and Y here may be either complete descriptions or partial descriptions giving, e.g., just the start time or just the value; of course if the question gives only a partial description the answer may also be a partial description.

The question where a completely described input is given is the type of question answered by standard circuit simulators. However, redesigning a circuit requires answering the other kinds of questions as well. For example, if the circuit specifications call for the circuit output to have a certain duration, it is important to be able to determine which properties of the upstream signals will assure this duration. The CRITTER system answers these kinds of questions by a process of propagating full and partial descriptions of data-streams through the circuit, and can test whether a given data-stream's specifications are satisfied by its behavior. CRITTER also maintains a *Dependency Network* that records, for each specification, both its source and the path in the circuit through which it was propagated.

2. Reasoning about Purpose

A second kind of reasoning important in redesign concerns the roles, or purposes, of various circuit modules in implementing the overall circuit specifications. Questions of this sort that arise during redesign include "What is the purpose of circuit module M?" and "How are the circuit specifications decomposed into subspecifications to be implemented by separate sections of the hardware?". Questions of this sort can be answered by REDESIGN, by examining the *Design Plan* of the circuit.

The Design Plan is a data structure that shows how circuit specifications are decomposed and implemented in the circuit, as well as the conflicts and subgoals that arise during design. It contains enough information to allow "replaying" the original design, and is characterized in terms of a set of *implementation rules* that embody in executable form general knowledge about circuit design tactics. This Design Plan must be provided as input to REDESIGN, as part of the characterization of the circuit which is to be redesigned.

In order to illustrate the form of the Design Plan, consider the simple Character Generator Module (CGM) circuit shown in figure II-1. This circuit is similar to a standard circuit used in most video computer terminals -- it is the part of the terminal that translates the ASCII character codes into the corresponding dot matrix to be displayed on the screen. This circuit accepts as input (1) a stream of ASCII encoded *Characters*, (2) a stream of binary encoded integers, called *Slice-Indices* that specify which horizontal slice of the character dot matrix is to be displayed, and (3) several clock signals used for synchronization. The circuit must produce a stream of *Character-Slices*, each of which is a bit string corresponding to the dots to be displayed on the terminal screen for the selected horizontal slice of the input Character.

The heart of the CGM design is a read-only memory, the ROM6574. This ROM6574 stores the definition of the character font (the dot matrix to be displayed for each character), one Character-Slice per byte of memory. To retrieve the Character-Slice corresponding to a given Character and Slice-Index, the ASCII code for the character is concatenated with the binary representation of the Slice-Index, and used to address the ROM6574. The other components in this circuit are used to interface the ROM6574 to the desired input and output formats. For example, the CGM specifications require serial output while ROMs produce parallel output. Therefore, a shift register

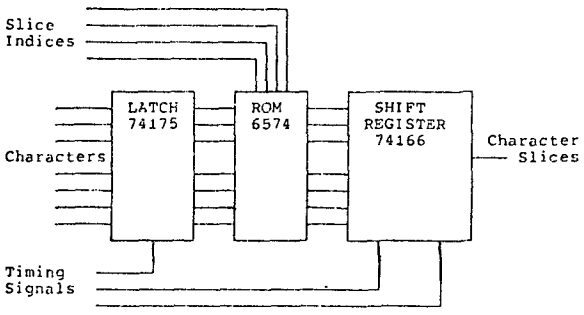

Figure II-1: The Character Generator Module

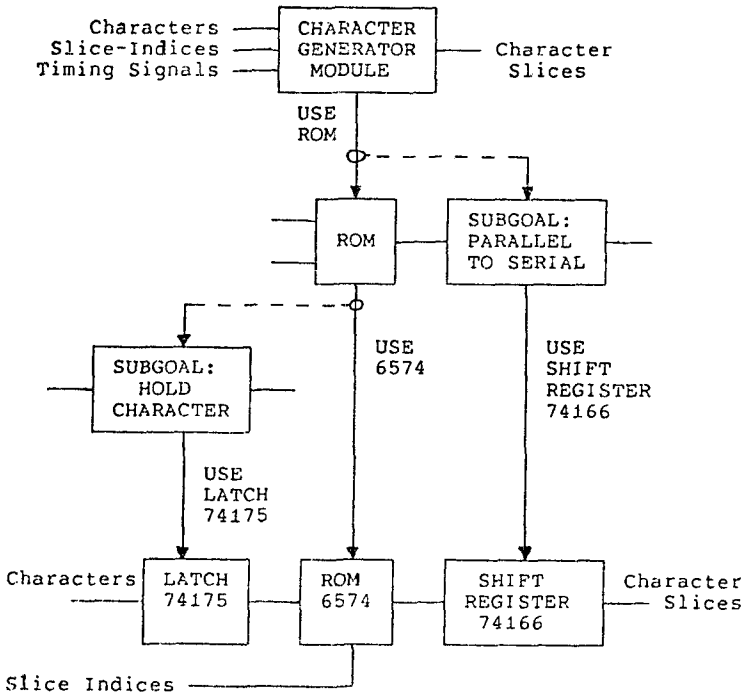

Figure II-2: Design Plan for the CGM

(SHIFT-REGISTER-74166) is used to convert the output data to serial. Also, because the address inputs to the ROM6475 must be stable for at least 500 nsec. while the input Characters are stable for only 300 nsec., a latch (LATCH74175) is used to capture the input Characters, and hold these data values stable for an acceptable duration.

The above paragraph summarizes the purpose of each circuit component and the conflicts and subgoals that appear during design. *This is precisely the kind of summary that must be captured in the Design Plan, in order to allow the REDESIGN program to reason effectively about the design and about the purposes of individual circuit components.*

Figure II-2 illustrates the Design Plan used to describe the CGM circuit to REDESIGN. Each node in the Design Plan corresponds to some abstracted circuit module whose implementation is described by the hierarchy below it. The topmost node in this Design Plan represents the entire CGM, and its functional specifications. The bottom most nodes in the Design Plan represent individual components in the circuit. Each solid vertical link between modules in the Design Plan corresponds to some implementation choice in the design, and is associated with some general implementation rule which, when executed, could recreate this implementation step. For example, the vertical link leading down from the topmost module in the figure represents the decision to use a Read-Only Memory (ROM) to implement the CGM. This implementation choice is associated with the implementation rule which states "IF the goal is to implement some finite mapping between input and output data values, then use a ROM whose contents store the desired mapping" (note this leaves open the choice of the exact type of ROM.)

Each dashed link in the Design Plan represents a conflict arising from some implementation choice or choices, and leads to a design subgoal, represented by a new circuit module with appropriate specifications. For example, a conflict follows from the implementation choice to use a ROM, and leads to the subgoal module labelled "Parallel-to-Serial-Subgoal". The conflict in this case is the discrepancy between the known output signal format of ROMs (i.e., parallel) and the required output signal format of the CGM (i.e., serial). The specifications of the new subgoal module are therefore to convert the parallel signal to serial. In a similar fashion, the implementation choice to use the specific ROM6574 leads to another conflict, and to the resulting subgoal to extend the duration of the input data elements.

Not shown are the links between the Design Plan and the Dependency Network, giving the specifications for the various data-streams.

By examining the Design Plan of a circuit, REDESIGN is able to reason about purposes of various circuit modules, and about the way in which the circuit specifications are implemented. The general implementation rules used to summarize the design choices can be used to "replay" the Design Plan for the similar circuit specifications, and thus allow for a straightforward kind of design by analogy.

C. Redesigning a Circuit

This section illustrates the use of both causal reasoning and reasoning about purpose in redesigning a circuit. It traces the actions of the REDESIGN program as it took part in a particular redesign of the Video Output Circuit (VOC) of a computer terminal. The Video Output Circuit (which contains the Character Generator Module discussed earlier) is shown in figure II-3. It is the part of the computer terminal that produces the composite video information to be displayed on the terminal screen. It produces this output from its combined inputs, which include the characters to be displayed, the cursor position, synchronization information for blanking the perimeter of the terminal screen, and special display commands (e.g., tc blink a particular character).

In this example, we consider redesigning the VOC to display characters in an italics font rather than its current font. Given a redesign problem, REDESIGN guides the user through the following sequence of five subtasks: (1) focus on an appropriate portion of the circuit, (2) generate redesign options to the level of proposed specifications for individual modules, (3) rank the generated options, (4) implement the selected redesign option, and (5) detect and repair side effects resulting from the redesign.

Focus attention on appropriate section(s) of the circuit. In many cases, the most difficult step in functional redesign is determining which portions of the circuit should be ignored. Focusing on relevant details in one locality of the circuit while ignoring irrelevant details in other localities can greatly simplify the complexity of redesign. In order to determine an appropriate focus, REDESIGN "replays" the Design Plan by reinvoking the recorded implementation rules with the changed circuit specifications. During this replay process, whenever an abstract circuit module is produced by some implementation step, its purpose is compared with the purpose of the corresponding module in the original Design Plan. If the purpose is unchanged, then the original implementation of this module will be reused without change in the new design****. If the new module has a different purpose than the corresponding module in the old Design Plan, (e.g., the new CGM must implement a different character font), an attempt is still made to apply the same implementation rule as in the original design (e.g., still try to use a ROM). If this implementation rule is not useful in the new design (as with the rule that suggests using the specific ROM6574), then REDESIGN stops expanding this portion of the Design Plan, and marks the corresponding portion of the circuit as a portion to be focused on for further redesign. The use of the Design Plan as sketched above leads in the current example to a focus on redesigning the abstract ROM module within the CGM within the VOC circuit. This abstract ROM module is implemented in the current circuit by two components as shown in figure II-2 (the ROM6574 and LATCH74175). A second method of focusing is possible, by using the Dependency Network produced by CRITTER. This method involves isolating those points in the circuit that possess specifications derived from the changed specification on the output data-stream. The resulting focus is generally broader than that determined from the Design Plan, because out of the many places in the circuit that can impact any given output specification, only a small proportion of these involve circuitry whose main purpose is to implement that specification.

****One must still make certain that changes elsewhere in the design do not interact dangerously with the implementation of this module. In REDESIGN, this is accomplished without having to directly examining the implementation of the module. Instead, design changes elsewhere in the circuit are checked for consistency with the constraints recorded in the Dependency Network produced by CRITTER.

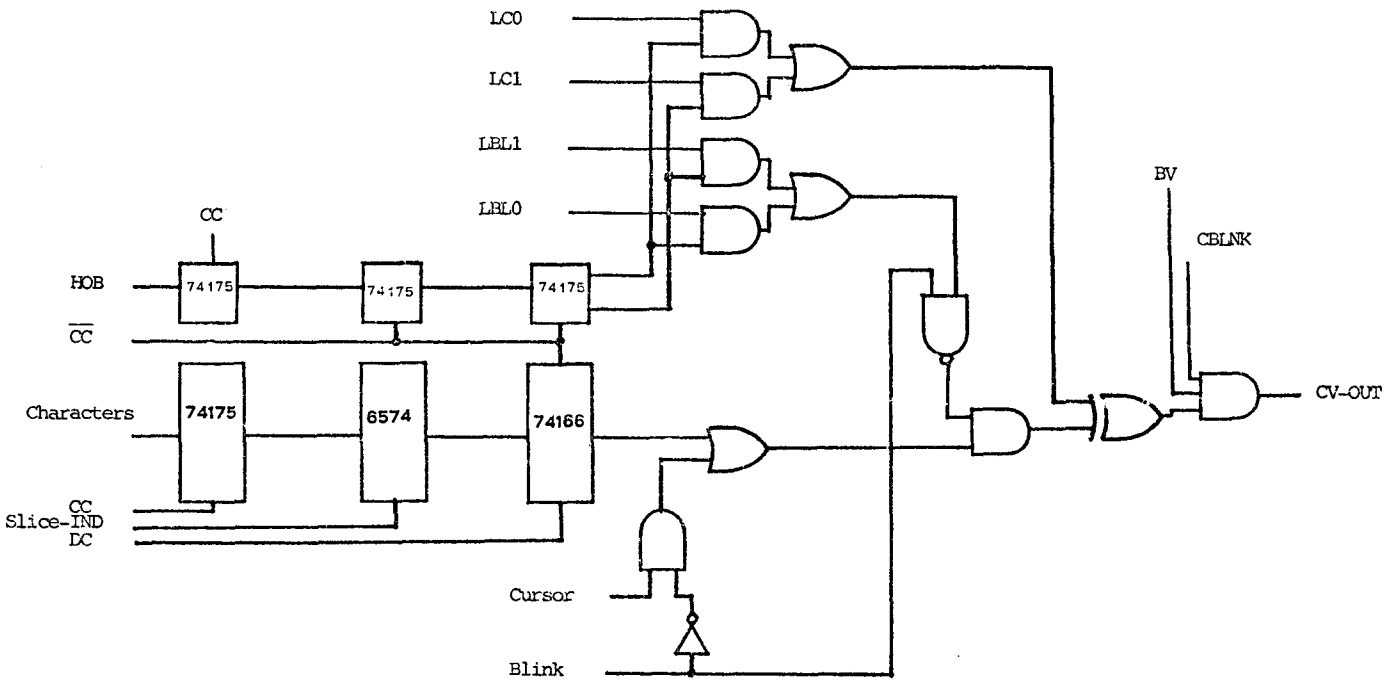

Figure II-3: Video Output Circuit

Generate redesign options to the level of proposed specifications for individual circuit modules. Once an initial focus for the redesign has been determined, redesign options are generated which recommend either altering the specifications of individual modules, or adding new modules with stated specifications. In both cases, only the new functional specifications are determined at this point -- the circuitry to implement these specifications is determined later. The constraint propagation capabilities of CRITTER provide the basis for generating these redesign options. In the current example, once REDESIGN has focused on the section of the VOC including the ROM6574 and LATCH74175, it considers the new output specification for this circuit segment, and propagates it back through this segment. Before each propagation step, REDESIGN considers the option of breaking the wire at that point and inserting a module to transform the values on that wire to values satisfying the required specification. In addition, it considers the option of altering the module immediately upstream, so that it will provide the required signal at that point. For each of the generated options, the new functional specifications are defined in terms of (1) the new specification to be achieved, and (2) a list of unchanged specifications found in the original Dependency Network, which are to be maintained. In the current example, the option generation process produces a list of five candidate redesign options. This list includes redesign options such as "replace the ROM6574 by a module which stores the new character font", and "introduce a new module at the output of the ROM6574, which will transform the output values into the desired font" (these options are described by the program in a formal notation, and the above are only English summaries).

Rank the generated redesign options. Heuristics for ranking redesign options can be based on a variety of concerns: (1) the estimated difficulty of implementing the redesign option (e.g., components with zero delay cannot be built), (2) the likely impact of the implemented redesign on global criteria such as power consumption and layout area, and (3) the likelihood and severity of side effects that might be associated with the redesign*****. In the current example, the heuristic that selects the appropriate redesign option suggests "Favor those redesign options that replace existing modules whose purpose has changed." In this case, since the purpose of the ROM6574 has changed, the option of replacing this component is recommended. The recorded Dependency Network and Design Plan also provide very useful information for estimating the relative severity of various changes to the circuit. Because the Design Plan shows the dependencies among implementation decisions (e.g., the purpose for the LATCH74175 is derived from the decision to use the specific ROM74175) it provides a basis for ordering the importance of components and associated constraints in the overall design (e.g., if the ROM6574 is removed, the LATCH75174 may no longer have a purpose for existing). This ordering of circuit modules, and of the data-stream constraints that they impose, provides an important basis for estimating the relative extent of side effects associated with their change.

Implement the selected redesign option. The above steps translate the original redesign request into some set of more local (and hopefully simpler) specification changes. While the implementation rules that REDESIGN possesses might be used for design******, the REDESIGN system does not make use of this potential. Thus, the user is left to implement the redesign option.

*****The current REDESIGN system has only a primitive set of heuristics for ranking redesign options.

******In fact they are used this way in the design consultant. See below.

Detect and Repair Side Effects Arising from the Redesign Once the redesigned circuit is produced, REDESIGN checks the new circuit segment to try to determine (a) that it does achieve the desired new purpose, and (b) that it does not lead to undesirable side effects. Undesirable side effects are detected as violations of the Dependency Network specifications at the inputs and outputs of the altered circuit segment. If a specification is violated, the new circuitry might be redesigned, or the specification might itself be modified or removed by redesigning a different portion of the circuit. The Dependency Network can be examined to determine the source of the violated specification, and to determine the locus of circuit points at which the specification could be altered.

D. Conclusions from the Work on REDESIGN

REDESIGN is a research prototype system that demonstrates the feasibility of providing intelligent aids for redesign and design of digital circuits. While the current REDESIGN system has many limitations (e.g., in the size of circuits it can handle, its inability to help with certain classes of redesigns, shortcomings of its causal reasoning methods, incompleteness of its knowledge base of implementation rules, etc.), it demonstrates clearly the importance of reasoning about causality and purpose in circuits when attempting to automate various subtasks involved in redesign and design.

Several features of REDESIGN have been important to its success. The most apparent of these are the means of combining reasoning about *causality* in the circuit, and reasoning about the *purposes* of parts of the circuit to assist in various subtasks of redesign. There are also some important aspects to *how* REDESIGN reasons about causality and purpose. In reasoning about causality, REDESIGN describes both the behavior and the specifications for a data stream, in a way that allows it to describe entire histories, rather than data stream values at single time instants. CRITTER can propagate these descriptions through the circuit, to build a Dependency Network showing how the specifications for each data stream are derived from the behaviors of the modules and the specifications for the circuit as a whole. In reasoning about purposes, we have viewed the original design process essentially as a planning problem, with subgoals derived both from the decomposition of parent goals and from conflicts between other subgoals. The Design Plan provides REDESIGN with an explicit summary of this planning process, with detail enough to replay the process, and to examine the particular relationships among design goals and subgoals.

III An Intelligent Aid for VLSI Design

To follow up on the work on REDESIGN, we are presently developing an interactive intelligent consultant, called VEXED,[15, 5, 16] to aid in designing cells and arrays of cells for VLSI circuits. VEXED begins with the functional specifications of the cells, and constraints on the placement of their interconnections, and is intended to produce a design at the sticks or perhaps layout levels. As an intelligent aid, VEXED is designed to offer advice about alternative methods of decomposing and implementing the desired function, about how to choose among such alternatives, and about detecting and handling interactions and conflicts among implementation choices. By running in background mode inside a graphics-oriented circuit editor, VEXED is intended to provide much the same kind of aid as that provided by a human expert watching over the shoulder of a designer during an editing session. The user has the ability to focus on a particular portion of the design, and to edit it as he pleases. However, the program may offer advice as it follows the tasks pursued by the user, provided its knowledge base contains expertise appropriate to the task at hand. In such cases, the user may elect to follow the consultant's advice, or to ignore it and implement the portion of the circuit as he wishes.

The design of the VEXED system builds upon our past experience with REDESIGN in several respects. Its design expertise is represented using the same type of If-Then rules used to characterize design steps in REDESIGN, and the two main modes of reasoning about circuits used by REDESIGN are also to be employed by VEXED. However, there are many new issues that must be addressed by VEXED, due to its focus on design rather than redesign, and due to our desire to develop it to the point of a practical tool for VLSI design. One of the major issues lies in building up and managing the knowledge base of design expertise. We expect that, as with many recent expert systems, in order to achieve high levels of performance VEXED may required several thousand If-Then rules. One interesting direction that we intend to pursue is to have VEXED acquire its own rules by observing the user's design steps, much as an apprentice assistant would learn from experience. In particular, in those instances in which the user disregards the advice of VEXED, the system should note the design step that the user takes, and attempt to form a general rule to characterize this step. For example, suppose that the current task is to implement a module that converts parallel to serial signals, and that based on its rule set VEXED suggests using a shift register from its component library. If the user ignores this advice, and instead uses the editor to construct his own circuit, then the system should note the circuit, verify that it accomplishes the desired function, and formulate a new rule that summarizes this new design tactic. Of course the task of formulating new rules in this way can be quite difficult, because such rules should be formulated with an appropriate degree of generality. We plan to base the method for generalizing rules on our previous work on learning heuristics and generalizing from examples[17], and believe that such a capability for acquiring knowledge from interactive problem solving is a crucial direction for research on knowledge based systems during the 1980s.

IV An Intelligent Aid for Chip Debugging

A second current thrust of the AI/VLSI group involves the development of an intelligent aid to assist in debugging VLSI circuits. In particular, we are concerned with the situation faced when the first samples of a newly designed circuit are tested. In the event that the circuit does not perform correctly, the task is to determine whether the failure is due to a design or manufacturing error, and to attempt to localize the cause of the failure. Our goal in this case is to provide an intelligent assistant that is able to generate and rank hypotheses regarding possible sources of the circuit failure, reasoning back from output failure symptoms to plausible internal faults. We find that the kinds of reasoning about the circuit that are essential for providing this kind of assistance in debugging overlap a great deal with the kinds of reasoning essential to design. In particular, the CRITTER system provides one mechanism for tracing output failure symptoms back through the circuit to generate candidate failure hypotheses, and the

hierarchical description of the circuit provided by the design plan is essential to controlling the combinatorics of the debugging process (i.e., the circuit is viewed hierarchically, so that the bug is first localized in terms of a small number of possible circuit modules, whose details are then examined in order to further localize the failure within the suspected module).

One thesis of this research is that debugging is best approached by considering design and debugging as interrelated problems. Not only is information from the design plan useful for constraining the debugging process, but the way in which the design is accomplished influences the difficulty of subsequent debugging. One straightforward example of this is the importance of designing VLSI circuits to allow internal signals to be observable at the output pads of the circuit. Furthermore, the result of the debugging process should certainly influence the redesign of the circuit. As our research on design and debugging progresses, we hope to develop ways of assuring closer coupling between these two processes.

References

[1] Kowalski, T.J. and Thomas, D.E. "The VLSI Design Automation Assistant: Prototype System." In *20th Design Automation Conference.* IEEE, August, 1983, 479-483.

[2] Stefik, Mark and Conway, Lynn "Towards the Principled Engineering of Knowledge." *The AI Magazine.* 3:3 (1982) 4-16.

[3] Zipple, R., "An Expert System for VLSI Design", MIT VLSI Memo 83-134

[4] Kim, J. and J. McDermott "TALIB: An IC Layout Design Assistant." In *Proceedings of the 1983 National Conference on Artificial Intelligence.* AAAI, 1983, 197-201.

[5] Kelly, V. "The CRITTER System: Automated Critiquing of Digital Hardware Designs", Technical report WP-13, Rutgers AI/VLSI Project, November 1983, to appear in Design Automation Conference, 1984.

[6] Green, C., et al. "Research on Knowledge-Based Programming and Algorithm Design", Research Report KES.U.81.2, Kestrel Institute, September 1982.

[7] J. McDermott "Domain Knowledge and the Design Process." In *Proceedings of the 18th Design Automation Conference.* IEEE, Nashville, 1981.

[8] Mostow, D.J., and Lam, M. "Transformational VLSI Design: A Progress Report", Technical report, USC-ISI, November 1982.

[9] Rich, Charles; Shrobe, Howard E.; Waters, Richard C. "Computer Aided Evolutionary Design For Software Engineering", AI Memo 506, Massachusetts Institute Technology, January 1979.

[10] Stefik, Mark Jeffrey, *Planning With Constraints,* PhD dissertation, Stanford University, January 1980.

[11] Sussman, Gerald Jay; Holloway, Jack; Knight, Jr., Thomas F. "Computer Aided Evolutionary Design For Digital Integrated Systems", AI Memo 526, Massachusetts Institute Technology, May 1979.

[12] Wile, David S. "Program Developments as Formal Objects", Technical report, Information Sciences Institute, July 1981.

[13] de Kleer, Johan, *Casual And Teleological Reasoning In Circuit Recognition,* PhD dissertation, Massachusetts Institute Technology, January 1979.

[14] Kelly, V., Steinberg, L. "The CRITTER System: Analyzing Digital Circuits by Propagating Behaviors and Specifications." In *Proceedings of the National Conference on Artificial Intelligence.* August, 1982, 284-289, Also Rutgers Computer Science Department Technical Report LCSR-TR-30, and Re-Design Project Working Paper #6

[15] Roach, J. "The Generalization of Symbolic Layout", Technical report WP-12, Rutgers AI/VLSI Project, November 1983, to appear in Design Automation Conference, 1984.

[16] Steinberg, L. and Mitchell, T. "Artificial Intelligence Aids for VLSI", Technical report WP-9, Rutgers AI/VLSI Project, June 1983.

[17] Mitchell, T. M., Utgoff, P. E. and Banerji, R. B., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in *Machine Learning,* Michalski, R. S., Carbonell, J. G. and Mitchell, T. M., eds., Tioga, 1983.