Design as Refinement Plus Constraint Propagation: The VEXED Experience

Louis I. Steinberg
AI/VLSI Project
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

Underlying any system that does design is a model of the design process and a division of labor between the system and the user. We are just beginning to understand what the main alternative models are, what their strengths and weaknesses are, and for which domains and tasks each is appropriate. The research reported here is an attempt to further that understanding by studying a particular model, the model of design as top down refinement plus constraint propagation, with the user making control decisions and the system carrying them out. We have studied this model by embodying it in VEXED, a design aid for NMOS digital circuits, and by experimenting with this system. Our primary conclusion is that this model needs further elaboration, but seems like a good basic model on which to build such systems.

I. Introduction

The task of designing something, e.g. a circuit, a program, or a mechanical device, is both intellectually challenging and economically important. It also requires large amounts of knowledge of a number of different kinds. Thus it is an important domain for AI, both in terms of building useful systems and in terms of understanding basic principles.

A number of researchers have focussed on developing useful systems to aid in some specific task in some specific domain. These include [Parker and Knapp, 1986, Bushnell and Director, 1986, Brewer and Gajski, 1986, Kowalski, 1985, Joobani and Siewioriek, 1985, Kim and McDermott, 1983]. However, underlying any such system there is either implicitly or explicitly a model of the design process, i.e. of the stages a design goes through between initial givens and final product, and of the operations that move it from stage to stage. We are just beginning to understand what the main alternative models are, what their strengths and weaknesses are, and for which domains and tasks each is appropriate.

The work reported here, like that of [Brown *et al.*, 1983, Tong, 1987], is an attempt to extend this understanding by explicitly studying a particular model of the design process. This model can be summarized by the equation,

```
DESIGN = TOP-DOWN REFINEMENT
         + CONSTRAINT PROPAGATION
```

Ideally, in designing a complex structure, one would like to use top-down refinement: first decompose the structure into a few main pieces and completely define the interfaces between the pieces, so that the design of each piece becomes a totally independent sub-problem. Each can be designed separately, and the pieces simply plugged together to solve the original problem. Unfortunately, until we explore the space of possible designs for the pieces, it is often impossible to know exactly what the interfaces should be.

One solution to this is common practice among human designers, and has also been used by Stefik in the Molgen system [Stefik, 1981]: leave the interfaces only partially specified. As you proceed with the design, decisions you make while working on one piece will further constrain what the interfaces of that piece must be, and thus constrain the alternatives for designing other pieces. We refer to this process of inferring how decisions at one place put constraints on options elsewhere as "constraint propagation".

In addition to a model of the design process, any design aid involves a *division of labor* between the system and the user. In systems that are to be fully automatic the division is simple: the system does it all. We, however, have been focussing on interactive systems. In particular, our approach has been to leave control decisions in the hands of the user, but leave all other processing to the system. That is, the user chooses which piece to refine next, out of all those still needing further refinement, and also chooses which way to refine it, out of all the alternatives that the system knows about. The system keeps track of which pieces need refining and what the alternative refinements are for a given module, and also does constraint propagation. This division of labor seems to build on the strengths of each party, making the computer responsible for completeness and consistency and the human responsible for strategy.

This model and division of labor are quite appealing, but also quite simple. Indeed, it soon became clear that they are too simple, and would have to be augmented to

[1]This work is being supported by NSF under Grant Number DMC-8610507, and by the Rutgers Center for Computer Aids to Industrial Productivity as well as by DARPA under Contract Numbers N00014-81-K-0394 and N00014-85-K-0116. The opinions expressed in this paper are those of the author, and do not reflect any policies, either expressed or implied, of any granting agency.

handle realistic tasks. However, our research strategy has been to stay with this model as much as possible, to see how far we can push it, to see where it fails, and whether the failures can be fixed by further elaboration of the model or whether they require starting over with an entirely different model.

We first tested the model by using it as the basis for a specific design aid, VEXED[2], in a specific domain, digital circuit design. More recently, we have extended the test by using the same model (and indeed almost entirely the same code, but with different knowledge bases) to build a design aid in another domain, mechanical design[Langrana *et al.*, 1986]. This paper discusses what we have learned from implementing and testing VEXED. The next section describes VEXED further, and the final section discusses our results and conclusions.

II. VEXED

First we will describe the way VEXED embodies this model of design: how it represents the circuit being designed, how it does refinement and how it does constraint propagation. Then we will show an example of VEXED's use. Finally we will discuss the implementation status of VEXED and describe the experiments we have done.

A. How VEXED Embodies the Model of Design

To embody our model of design, VEXED must represent both the structure and operation of the partially-designed circuit, and must be able carry out refinement and constraint propagation. We will deal with these issues in that order.

VEXED represents the structure of a circuit in a fairly standard way. A *module* represents either a single component or a group of components being viewed as a functional block. A *data-path* similarly represents either a single wire or a group of wires. The operation of a circuit is represented in a somewhat less standard way. The signal on a given data-path is called a "data-stream", and is thought of as a sequence of "data elements", e.g., a sequence of bits or characters. An individual element is referred to by its "subscript", i.e. its position in the sequence. Elements have a number of "features", including Type (e.g. Boolean), Data-Value (e.g. FALSE), Encoding (how the abstract data-type is encoded as voltages), and various timing-related features. For a further discussion of these representations, see [Kelly, 1985].

VEXED's knowledge of refinement methods is embodied in a set of "refinement rules", e.g., INCLUDE-MEMORY:

> IF the output at time t2 depends on an input at time t1, THEN one way to refine the module is into a memory, which holds the value from t1 to t2, and another module, which uses this stored value at time t2 to compute the output.[3]

The IF part of the rule describes the class of modules that this refinement method applies to. The THEN part describes how to do the refinement: the submodules, their initial specifications[4], and how they are connected. It is important to note that these refinement rules describe *legal, correct* implementations, but not necessarily optimal or even preferred implementations. They define the "legal moves" in the search for possible circuit implementations, but not a strategy for choosing among alternatives.

It is also worth noting that in VEXED refinement involves structural decomposition, breaking a module into its pieces, while in Molgen[Stefik, 1981] refinement involves going from a more abstract operation to a more specific one.

Constraint propagation in VEXED is done by the CRITTER system[Kelly, 1985]. Critter does two kinds of propagation.

- Firstly, CRITTER does a form of goal regression. Given a specification on the data-stream output by a module, and given the behavior of this module, CRITTER can determine what must be true of the inputs to the module to ensure that the output specification will be met.
- Secondly, CRITTER does a form of symbolic evaluation. Given a (possibly partial) description of the behavior of a module's inputs, and given the module's behavior, CRITTER can infer a description of the module's outputs.

Because of our representations, constraint propagation is simply a matter of symbol substitution (see [Kelly, 1983]). However, this process results in very large, complex expressions. Therefore, CRITTER also has an expression simplifier that uses a set of rewrite rules to simplify the resulting expressions as much as possible. Finally, CRITTER is capable of verifying that the specifications on a data-stream are satisfied by that data-stream's behavior. Again, this is done by a process of symbol substitution and simplification.

B. Example

Figure 1 shows the user interface[5] for VEXED, at the beginning of a typical design session. The circuit being designed is one bit of a content-addressable memory, and is referred to as the CAM-CELL. The screen is divided into several regions, or windows. The largest window is the region in which the circuit will be designed, and initially contains a large rectangle representing the CAM-CELL to be designed. The user has already entered the specifications for this circuit. These specifications include a description

[2] VEXED stands for Vlsi EXpert EDitor.

[3] This, of course, is an English paraphrase of the formal notation.

[4] To be augmented later by constraint propagation.

[5] VEXED is implemented for Xerox Interlisp-D machines using the Strobe object-oriented programming system from Schulmberger-Doll Research.

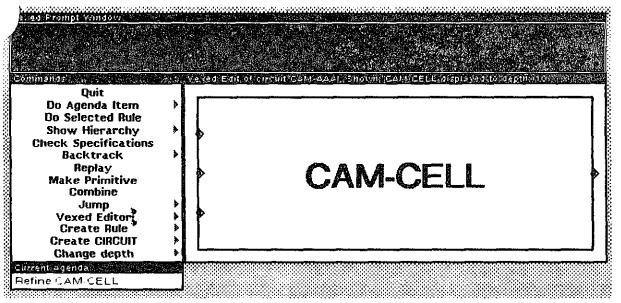

Figure 1: The VEXED Interface

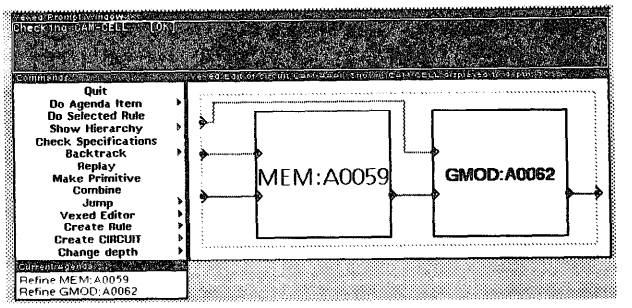

Figure 3: Result of Executing the Memory-Rule

of the inputs and outputs of the CAM-CELL, as well as a description of the function to be implemented. Figure 2 gives part of these specifications: the value of the output OUT at each time must equal some expression based on the values of the inputs at that and previous times, and for this output the boolean values TRUE and FALSE are represented by low (0 volts) and tristate (high impedance), respectively.

Attached to the main window is a list of commands and a list of pending tasks. As shown in the figure, the only pending task at this point is to refine the CAM-CELL. This list of pending tasks will be updated as the design proceeds, and new circuit submodules are introduced. In general, the user controls which portion of the design to focus on next by selecting one of the pending tasks from this list.

In this case, the user selects the (REFINE CAM-CELL) task, and the system then considers its collection of rules to determine which ones apply to this module. In this case, the advice offered by the system is that there are eight rules which suggest alternative methods for refining the CAM-CELL. The user may select one of these rules to be executed or, alternatively, may elect to ignore the system's advice, and manually edit the circuit.

Figure 3 shows the result of the user selecting

```
((I (ALL I))
 (EQUAL
  (DATA-VALUE OUT I)
  (EQUAL (DATA-VALUE MATCH I)
         (DATA-VALUE DATA-IN
           (PREVIOUS 1 J
            (EQUAL (DATA-VALUE LOAD J)
                   (QUOTE HIGH))
            I))))
 (EQUAL
  (ENCODING OUT I)
  (NMOS-BOOLEAN (FALSE TRISTATE) (TRUE LOW))))
```

Figure 2: Part of the Specifications for CAM-CELL

INCLUDE-MEMORY for the system to carry out. (This is the rule paraphrased above.) Execution of this rule has lead to a refinement of the CAM-CELL, which includes a memory module (called MEM:A0059), as well as second module (GMOD:A0062). Both modules have specifications given in the same representation as for the original CAM-CELL specifications. The MEM:A0059 specifications require that it store the value of the DATA-IN signal, whereas the specifications of GMOD:A0062 require that it produce an output depending upon a comparison between the output of MEM:A0059, and some of the inputs to the CAM-CELL. The list of pending tasks has also been updated so that the new tasks include refining MEM:A0059 and GMOD:A0062.

Refinement of the circuit continues in this fashion. The user directs the focus of attention by selecting which

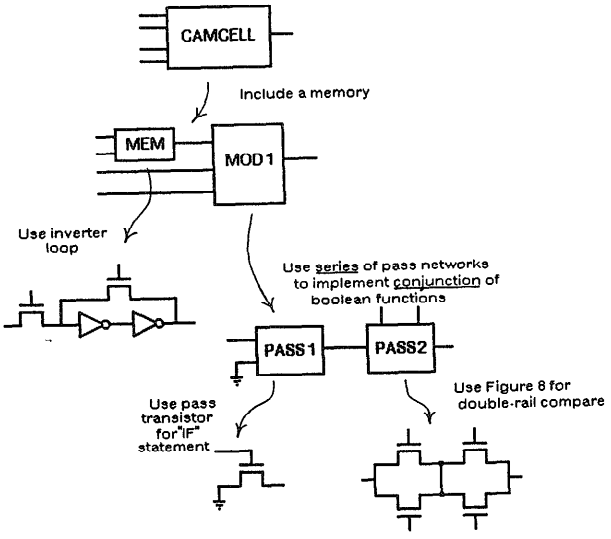

Figure 4: The Design Hierarchy

module is to be refined next. The system examines its rule base to determine applicable rules, and presents these to the user. The user may then select one of these, or may ignore this advice and elect instead to refine the module by editing it manually. Figure 4 shows the hierarchy of refinement steps which lead to a final circuit-level implementation.

C. Status of VEXED

There are three points to make about the status of VEXED.

First of all, VEXED has been fully implemented, and has about 50 refinement rules. These cover most of the standard NMOS design techniques for boolean functions, and also a for few latches. Work has recently started on a set of rules for CMOS circuits.

Secondly, VEXED has been used by students in our VLSI design class to do a homework assignment. The assignment was done by about ten teams of students, mostly two students per team. Each team designed one of three small circuits; one circuit was a full adder, and the others were of about the same size.

Thirdly, VEXED has had a number of capabilities added to it beyond refinement and constraint propagation.

- One facility any real system needs is a backtrack or "undo" facility that allows the user to retract decisions that turn out not to have the desired effect. VEXED has a chronological backtracking facility that allows the user to return the circuit to the state it was in at any previous time.
- It turns out that when a module is refined into sub-modules, a sub-module may occasionally need a signal as input that was not originally among the inputs of the parent module. Typically this happens with signals like clocks, ground, etc. To handle this situation, VEXED has "Get Signal" tasks, which are automatically entered on the task agenda when needed, and are handled by the user manually specifying where the needed signal should come from.
- A facility has been added for "Module Combining Rules". These specify how two modules can be combined into one simpler one, and provide for a kind of peephole optimization. For instance, two inverters in series can be combined into just a simple wire (as long as this change does not violate some timing constraint). Since it is always appropriate to try to combine modules, and since the circuit can be considered complete even if no combinations are done, these tasks do not go on the agenda. Rather, the user can point to a module and request that an attempt be made to combine it with each of its neighbors. There are currently only a few such rules, and this facility was not used by the VLSI students.
- Finally, there is now a "replay" facility for VEXED. This takes the sequence of refinements applied previously to some other circuit, or even to other parts of the current circuit, and applies them to the current module. To the extent that the refinement operations used previously are general, and apply in somewhat new circumstances, this is a way to reuse the *ideas* of a previous design even when the specific circuit is not applicable. See [Mostow and Barley, 1986] for a further discussion of this facility.

III. Results and Conclusions

As discussed above, we began with a model of the design process and of the division of labor between he user and the system, and we implemented VEXED to test these models. Our results can be seen as answering two broad questions:

- First, can a design aid embodying these models be implemented? Is it possible for a system to have a sufficient body of refinement methods, to find those applicable to a given module, to carry out the one selected by the user, and to do the constraint propagation?
- Secondly, if such a system were implemented could designers, especially those with no AI or even computer science background, use it to produce designs? The concern here was both whether the users could understand and use this design process, and also whether they could learn our specification language, which is quite different from standard hardware specification languages in its LISP-like syntax, in its data-flow style semantics, and in its representation of a data-stream as a sequence of values.

As the next two sections will describe, the answer to both of these is, "Yes, but."

A. Can VEXED be Implemented?

The fact that VEXED has been brought to the point where students in our regular VLSI class could successfully use it is evidence that it has indeed been implemented. Two issues remain: the size and coverage of the set of refinement rules, and the cost of constraint propagation.

As noted above, the current refinement rules cover most boolean combinational circuits for the NMOS circuit technology, and some latches. A truly useful system would require more complete coverage of combinational circuits and latches, as well as rules for a number of other kinds of circuits, e.g. multiplexers, and rules for higher level data-types such as integers and characters. However, in principle there seems no reason why these rules could not be added to VEXED. Based on the number of current rules and the coverage they give, we estimate that a version of VEXED that would be useful for real designers would need less than 1000 rules, and so would be within the scope of current technology for building and maintaining rule-based systems.

Remember also that user can step in and do a refinement manually whenever the system does not have a rule for the desired refinement method. This helps in two ways.

First of all, it means that there need not be as many rules before the system is useful; it probably takes far fewer rules to cover 90% of the refinement steps in each of a range of designs that it would take to cover 100% of the steps. Secondly, since the rules do not have to contain any control information, i.e. any information on which of the locally plausible refinements to actually do in a given design, it turns out that it is relatively easy to observe the user doing such manual refinements, and infer general rules. We are building a system called LEAP[Mitchell *et al.*, 1985] which will do just this. The first version of LEAP is almost completed.

Finally, VEXED uses an indexing structure to find relevant rules for refining a given module without testing the left hand sides of every rule, so the time to find relevant rules should grow less than linearly with the number of rules, and the time to find relevant rules is currently fairly short. Thus we do not expect the time to find relevant rules to be a major problem even with many more rules.

While the size of the rule set does not seem to be a problem, the cost, both in terms of memory space and in terms of time, to do constraint propagation does seem to be a major issue. In a circuit such as a full adder described at the transistor level, with about 20 modules, it takes five to ten minutes on a Xerox 1109 (DandeTiger) to do the constraint propagation after each refinement. The cost of constraint propagation seems to grow slightly less than linearly with circuit size, based on some initial impressions, but the delay for a full adder is barely tolerable and so to design anything much larger it will be necessary to reduce this cost.

One simple answer, of course, is to optimize our code, which is currently not very optimal, or to get a faster machine. In particular, the task of constraint propagation seems inherently parallel, since each constraint can be propagated along each path more or less independently; thus it would seem a natural application for a parallel machine.

Another answer is to find a way to do less propagation. At the moment, VEXED propagates every constraint everywhere it can as soon as it can. Perhaps limiting or delaying some of this propagation can reduce the cost. We are currently looking in to this possibility.

B. Can VEXED be Used?

Given that VEXED can be implemented, can it be used? Can non-AI types learn our specification language, and can they successfully do design with such a design aid as VEXED? Again, the answer is, "Yes, but."

About half of the class were students from the Electrical Engineering Department with no AI background and indeed relatively little Computer Science background, and even the Computer Science students included some who had not had any AI courses. The students were given no more documentation and other help (lecture, hands on help, etc.) than they are typically given for any other design aid used in the course. Never the less, they did succeed in specifying and designing their circuits. The few who did not finish were those who were halted by one or another of the minor[6] bugs left in VEXED.

On the other hand, the circuits some students designed were wildly sub-optimal. They took many more transistors than were necessary. That is, when they chose which refinement rule to use, they did not choose wisely. Partly this may be due to their inexperience as VLSI designers in general. Partly it may be due to their difficulty in understanding what each rule did. Each rule had a canned English description that said what its effect was, and another that tried to give advice on when to use it, but a major complaint from the students was that it was hard to understand this documentation and to figure out what the rules did. We are beginning to look into the whole area of how a system like VEXED could explain the rules and the state of the design to the user.

Finally, the difficulty in choosing rules may be inherent in the structure of a system like VEXED. I am a better designer than the students, and I understand the rules quite well, and thus I can get much better designs out of VEXED. However, I have to think very hard to do so. The problem is that VEXED's constraint propagation tells you the effects of *previous* refinement decisions in limiting the choices for the current decision, but it does not show you how each current alternative will limit the choices you will have on later decisions. To get a good circuit out of VEXED, the user has to have a clear global strategy in mind, and has to weigh each decision in the light of how it will contribute to that strategy.

Perhaps VEXED could try the constraint propagation that would result from each alternative, and inform the user what the effects of each would be on the remaining alternatives elsewhere. However, given the cost of constraint propagation, this may not be practical. The basic problem seems to be that since VEXED leaves the control issues entirely up to the user, it has no internal representation of the goals and plans that go into a strategy for designing the circuit, and thus cannot offer the user any support in deciding which module to work on next or which refinement to make. The DONTE system being developed in our research group by Chris Tong[Tong, 1987] is an attempt to study some of the issues of how a system based on top down refinement and constraint propagation might also make these control decisions.

In addition to the problems with choosing the right rule that the students actually had, there are two problems that did not come up but might have had they been designing larger circuits. One is that certain kinds of circuit are quite difficult to specify in our language. These are the circuits whose output at a given time depend on the entire past history of their inputs, or at least on an unbounded set of past inputs. These are not easy to express in a data flow oriented form. The solution here is either to find a more algorithmic specification language that can be translated into the data flow form, or to find a way to do

[6] Minor in the sense that we were able to quickly fix them.

constraint propagation directly with the more algorithmic language.

The second potential problem with larger circuits is that design really does involve more kinds of operations than just refinement and constraint propagation, and even more than get-signal, backtrack, replay, and combining modules. Examples include inserting "sub-goal" modules to fix conflicts, e.g. when one module produces output in serial and the next needs parallel input, you can put in a serial to parallel converter. Also, some operations are best viewed not as a refinement of a module into sub-modules, but rather as a recasting of the specification into a semantically equivalent but structurally different form, e.g. turning an complex boolean expression into sum-of-products form. And there are a few other such examples. However, all of them seem to be the kind of thing that can be added on top of the basic VEXED model, much as the module-combining rules have been added. Of course, further work is needed to be sure that they really can be added.

In summary, then, if ways can be found to help the user choose the right rules, and if the cost of constraint propagation can be controlled, and if the additional kinds of operations can indeed be added to the system, the VEXED model of design will indeed prove to be a good one on which to base interactive, knowledge-based design aids.

Acknowledgements

Both the programs and the ideas presented here are the work of many people in the Rutgers AI/Design group. I particularly want to thank Tom Mitchell, Jack Mostow, Chris Tong, Jeff Shulman, Tim Weinrich, Mike Barley, Atul Agarwal, and Sunil Mohan. Finally, I want to thank Chun Liew for help with text formatting.

References

[Brewer and Gajski, 1986] F. Brewer and D. Gajski. An expert-system paradigm for design. In *Proceedings of the 23rd Annual Design Automation Conference*, June 1986.

[Brown *et al.*, 1983] H. Brown, C. Tong, and G. Foyster. Palladio: an exploratory environment for circuit design. In *IEEE Computer Magazine*, December 1983.

[Bushnell and Director, 1986] M. Bushnell and S. Director. Vlsi cad tool integration using the ulysses environment. In *Proceedings of the 23rd Annual Design Automation Conference*, June 1986.

[Joobani and Siewioriek, 1985] R. Joobani and D. Siewioriek. Weaver: a knowledge based routing expert. In *Proceedings of the 22rd Annual Design Automation Conference*, June 1985.

[Kelly, 1983] V. Kelly. *The CRITTER System: Automated Critiquing of Digital Hardware Designs.* Technical Report WP-13, Rutgers AI/VLSI Project, November 1983. also appearing in the Proceedings of the Design Automation Conference, 1984.

[Kelly, 1985] V. Kelly. *The CRITTER System - An Artificial Intelligence Approach To Digital Circuit Design Critiquing.* PhD thesis, Rutgers University, New Brunswick, New Jersey, January 1985.

[Kim and McDermott, 1983] J. Kim and J. McDermott. Talib: an ic layout design assistant. In *Proceedings of AAAI-83*, pages 197–201, 1983.

[Kowalski, 1985] T Kowalski. *An artificial intelligence approach to VLSI design.* Kluwer Academic Publishers, Boston, 1985.

[Langrana *et al.*, 1986] N. Langrana, T. Mitchell, and N. Ramachandran. *Progress Toward A Knowledge-Based Aid for Mechanical Design.* Technical Memo CAIP-TM-002, Center for Computer Aids for Industrial Productivity, Rutgers University, January 1986.

[Mitchell *et al.*, 1985] T. M. Mitchell, S. Mahadevan, and L. Steinberg. Leap: a learning apprentice for vlsi design. In *Proceedings of IJCAI-85*, Los Angeles, CA., August 1985.

[Mostow and Barley, 1986] J. Mostow and M. Barley. Reuse of design plans. In *International Conference on Engineering Design*, Boston, MA., September 1986. Abstract accepted for ICED87.

[Parker and Knapp, 1986] A. Parker and D. Knapp. A design utility manager: the adam planning engine. In *Proceedings of the 23rd Annual Design Automation Conference*, June 1986.

[Stefik, 1981] M. Stefik. Planning with constraints (molgen: part 1). In *Artificial Intelligence 16:2*, pages 111–140, May 1981.

[Tong, 1987] C. Tong. Goal-directed planning of the design process. In *The 3rd IEEE Conference on AI Applications*, February 1987. also appears as Rutgers AI/VLSI Project Working Paper No. 41.