Behavioral Level Transformation in the CMU-DA System

Robert A. Walker
Donald E. Thomas

Electrical Engineering Department[1]
Carnegie-Mellon University

Abstract

The Carnegie-Mellon University Design Automation system (CMU-DA) [2] consists of a set of computer programs whose goal is to produce a complete design in a user-specified device technology, given as input a behavioral description of the piece of hardware to be designed and a set of constraints. This paper describes one of the tools in the CMU-DA environment - a software package to perform optimizing transformations at the behavioral level. Motivations for these transformations are given, and an example of their use is shown.

Keywords: *behavioral level, data-flow representation, optimizations, transformations, synthesis.*

Introduction

The Carnegie-Mellon University Design Automation system (CMU-DA) [2] consists of a set of computer programs whose goal is to produce a complete design in a user-specified device technology, given as input a behavioral description of the piece of hardware to be designed and a set of constraints. The programs themselves are organized in a hierarchical fashion, with individual programs or groups of programs working on the design at different levels of abstraction. The highest level, the *ISP Behavioral Level*, represents the behavior of the design in a formal, structured language called ISPS [1]. This level is completely independent of the target technology in which the design will be implemented. The ISP level is translated into the *VT Behavioral Level*, which represents the behavior of the design as a set of directed acyclic graphs called the *Value Trace (VT)* [5]. This level is also independent of the implementation technology, and represents the design in a form well suited for optimization and hardware design.

This paper describes a transformation package for optimizing a design at this level, and shows an example of its use. A more detailed report of this work may be found in [6].

The Value Trace

The original idea for the Value Trace was first proposed by Snow in his Ph.D. thesis [5] and was implemented in several steps [4], [3]. This Value Trace is a Directed Acyclic Graph (DAG) similar in nature to those used in optimizing compilers. The nodes of the graph, called *activities* or *operators*, represent an operation to be performed. The arcs of the graph, called *carriers*, represent the *flow* of data from one operator to another. The primary difference between the DAGs created by Snow and those used in optimizing compilers is the addition of control constructs to allow conditionals and subroutines, something not present when DAGs are used by optimizing compilers. Snow also identified a set of transformations that could be performed on the VT to optimize its representation while leaving its behavior unchanged. These transformations are the basis of those on which this work was based.

The major criteria of the Value Trace representation was that it be easy to manipulate by optimizing transformations, and that the feasibility and suitability of these transformations be easily evaluated. A problem involved with the production of optimal designs is the elimination (or, more realistically, the reduction) of the biases inherent in the ISP description. One major artifact of the designer's ISP coding style is the breaking up of the behavior into separate procedures. Although this allows for a structured approach to describing the behavior, the best implementation may not follow these procedure boundaries. Through transformation, it is possible to remove these boundaries, or to form others at will.

Another artifact of the designer's coding style is the use of temporary variables. Behavior level statements in which there are duplicated calculations in successive statements may be transformed to perform the calculations once and store the results in a temporary variable, using redundant operator elimination. The elimination of the duplicated calculations not only eliminates the operators and control steps involved with these calculation, but might eliminate a redundant piece of hardware. Thus, Value Trace transformations can be used as a design tool at the systems level of design.

Since there are tradeoffs involved in any design process, there must be some means for evaluating the results of any specific transformation. A set of *metrics* can be developed to aid in this evaluation of the results of the transformations. These metrics are measurements that can be performed on the Value Trace and that can be interpreted in such a manner as to give reasonable information on the changes effected by the transformations.

The Transformations

There are three major groups of transformations: operator transformations, SELECT transformations, and vtbody transformations. The operator transformations are transformations on individual operators or groups of operators. The SELECT transformations are transformations to act on the SELECT construct (used to implement the ISPS equivalents of conditional branching and

[1]This work was funded by Digital Equipment Corporation, and by the National Science Foundation under grant ENG 78-25755.

case statements). The vtbody transformations are transformations for working with whole vtbodies (groups of Value Trace operators).

Some of these transformations are listed below.

- *Constant folding* consists of replacing an operator which acts on one or more constants with a new constant.
- *Redundant operator elimination* may be used when two or more operators of the same type have the same inputs, and consists of eliminating one of the operators and replacing all references to its outputs with references to the outputs of the operator to be retained.
- *Dead Operator Elimination* consists of the removal of operators whose output is no longer used.
- *SELECT motion* consists of replacing similar activities within the branches of a SELECT with a single activity outside the SELECT, and vice-versa.
- *Vtbody inline expansion* is analogous to the inline expansion of a subroutine, and consists of replacing a call to a vtbody with a copy of that vtbody.
- *Vtbody formation* is the inverse of vtbody inline expansion, and consists of encapsulating a group of operators into a new vtbody and adding a CALL to it.
- *Loop unwinding* consists of inline expanding instantiations of a looping vtbody so that constant folding and other transformations might be applied. In this manner, a loop counter may be eliminated and replaced with a finite number of calls to a given subroutine.

Example - Intel 8080

The ISP description of the instruction decoding section of the Intel 8080 consists of a main loop for decoding the fields of the instruction, and several pages of subroutines, each of which contains the behavioral description for a particular instruction. Because each subroutine is only called once, it may be expanded inline and the old vtbody eliminated. Performing this inline expansion for the entire Intel 8080 gives the results shown in Table 1. This table also shows the results for two different types of controllers. The first type is that with explicit flow control (EFC) instructions, which requires a *non-zero* amount of time to execute flow control operators like the CALL and the SELECT. The second type of controller is that with implicit flow control (IFC) in *all* instructions, which requires *zero* time to execute these same operators.

Many ISP descriptions have some sort of decoding loop which decodes the instruction codes and calls separate subroutines to perform the appropriate operation, so the inline expansion of these subroutines is not uncommon. While it is simpler to write and understand a one-page instruction decoding loop which calls four pages of subroutines for the execution of the instructions than it is to write a five page loop which handles all of this, the latter would be represented much more efficiently at the VT level. The CMU-DA system is attempting to abstract away such biases imposed on the design by the designer's programming style, and inline expansion may be used to convert the internal representation of the first style into that of the second. In the full Intel 8080, this reduction of biases reduces the number of control steps, and thus the amount of microcode required for implementation, by 28 to 37 percent as shown in Table 1.

Cont.	Metric	Before	After	Change	
EFC	Control Steps	542	390	-152	-28.0%
	Operators	730	578	-152	-20.8%
IFC	Control Steps	361	226	-135	-37.4%
	Operators	730	578	-152	-20.8%

Table 1: Inline Expansion in the Intel 8080

Conclusion

The Value Trace and its transformations provide a basis on which a behavioral level design aid can be constructed. Biases originally introduced by designers in the ISP description can be removed, and changes in the structure of the ISP algorithm can be made. Design aids for VLSI systems must include such capabilities since they allow for transformation and optimization at the systems level of design.

References

[1] Mario R. Barbacci.
Instruction Set Processor Specifications (ISPS): The Notation and Its Applications.
IEEE Transactions on Computers C-30(1):24-40, January, 1981.

[2] Stephen W. Director, Alice C. Parker, Daniel P. Siewiorek, and Donald E. Thomas, Jr.
A Design Methodology and Computer Aids for Digital VLSI Systems.
IEEE Transactions on Circuits and Systems CAS-28(7), July, 1981.

[3] David A. Gatenby.
Digital Design from an Abstract Algorithmic Representation: Design and Implementation of a Framework for Interactive Design.
Master's thesis, Department of Electrical Engineering, Carnegie-Mellon University, October, 1981.

[4] Michael C. McFarland, S.J.
The Value Trace: A Data Base for Automated Digital Design.
Master's thesis, Department of Electrical Engineering, Carnegie-Mellon University, December, 1978.

[5] Edward A. Snow.
Automation of Module Set Independent Register-Transfer Level Design.
PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, April, 1978.

[6] Robert A. Walker.
A Transformation Package for the Behavioral Level of the CMU-DA System.
DRC Research Report DRC-01-14-82, Design Research Center, Carnegie-Mellon University, October, 1982.
Electrical Engineering Master's Thesis.