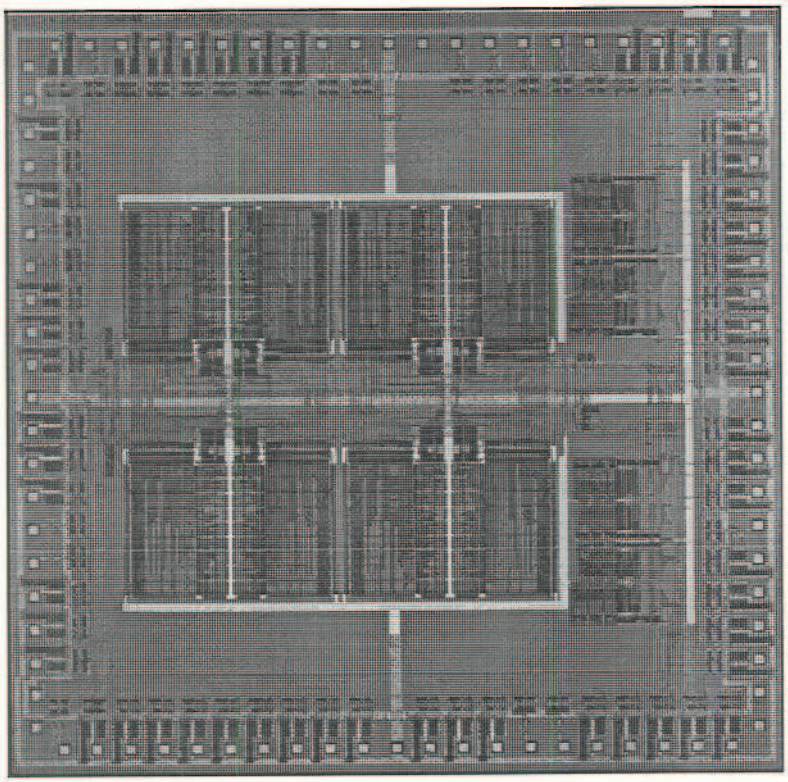
TECHNICAL REPORT NO. 274

Using the Digital Design Derivation System: Case study of a VLSI garbage collector implementation

by

C. David Boyer and Steven D. Johnson

April, 1989



COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

TECHNICAL REPORT NO. 274

Using the Digital Design Derivation System: Case study of a VLSI garbage collector implementation

by

C. David Boyer and Steven D. Johnson

April, 1989

COMPUTER SCIENCE DEPARTMENT

INDIANA UNIVERSITY

Bloomington, Indiana 47405–4101

Using the Digital Design Derivation System:
Case study of a VLSI garbage collector implementation*

C. David Boyer and Steven D. Johnson
Computer Science Department
Indiana University
Bloomington, Indiana

The *DDD* transformation system, under development at Indiana University, reflects an approach to digital design synthesis based on a purely functional algebra. The system has been used to derive several working designs realized in PLD (programmable logic device) technologies. This paper reports on the reimplementation of one of these designs, a garbage collector, in VLSI. The design example provides a context for discussing the method of synthesis and its mechanization. A generic system of control transformations was developed to aid in meeting the architectural constraints of the VLSI target. The exercise also illustrates the role of the algebra in managing the translation from conceptual architecture to physical organization.

1. Introduction

A stop-and-copy garbage collector is implemented in VLSI using a Digital Design Derivation (DDD) system currently under development at Indiana University. DDD reflects a formal approach to digital design synthesis based on the algebraic manipulation of purely functional expressions. It is the mechanized part of a design method whose benefits include the support of abstraction, provisions for animation, and a unified foundation for design automation. The project described here is part of a continuing effort to exercise the approach at practical levels of engineering.

DDD implements basic algebra for the interactive synthesis of synchronous systems. At this stage of its development, it is essentially an editor for making correctness preserving transformations on certain applicative specifications. These transformations may be applied at various levels to synthesize hardware system descriptions. A *design derivation* is that sequence of transformations used to produce an implementation. DDD is integrated with existing logic synthesis tools, currently PLD programmers and PLA layout assemblers. Thus, the derivation goal in DDD is to produce a realizable boolean system description. Section 2

---

*Research reported herein was supported, in part, by the National Science Foundation under grants numbered MIP 87-07067 and DCR 85-21497.
Paper to appear in: *IFIP WG 10.2 Ninth International Symposium on Computer Hardware Description Languages*, June, 1989.

is a brief review of formal research leading to the implementation of DDD and previous experimentation with the system.

Earlier, a PLD prototype garbage collector was derived with these tools [6]. This paper describes adaptations of DDD to reimplement the same design in VLSI. One of the requirements of the project was to use the same initial description for both implementations. Section 3 is a discussion of this behavioral specification which is shown in Appendix A. The original specification held implications for architecture that could not be attained in the VLSI target. For instance, the PLD implementation drove two parallel memories and included three adder circuits. Due to pin and area limitations, memory accesses and arithmetic had to be serialized. New transformations were developed for this kind of refinement. These are discussed in Section 4.

The underlying purpose of this project was to expose issues for formal and practical development. An emerging thesis of this research is that a flexible low level algebra is needed to manage the physical reorganization of digital descriptions. The data path of the VLSI collector is realized in three chips. The latter stages of derivation are concerned with sustaining correctness through the massive restructurings involved. Section 5 is a discussion of this aspect.

DDD transforms executable expressions. Section 6 explains how animation of the design notation was used in testing and simulation. The conclusion surveys directions for future research and experimentation.

2. The Approach to Synthesis

The DDD system embodies a general approach to system synthesis as applied to digital design. A functional algebra is employed to manipulate formulas modeling the structure and behavior of hardware systems. An implementation is derived by applying a sequence of correctness preserving transformations to a higher level description. The syntactic medium is a dialect of lambda calculus augmented with forms for recursive definition. This language enjoys a simple algebra. A goal of formal research is to study the algebra of synthesis by distilling it to a collection of basic laws. The same strategy is followed by Sheeran, who applies her work to a different class of design problems [15].

Johnson established a formal connection between the use of recursive function definitions to model control and the use of recursive sequence (or 'stream') definitions to model synchronous systems [9]. The theoretical foundation provides a framework for dealing with abstraction in design. DDD implements algebra that is independent of the *basis*—or ground vocabulary of constants, operations, and tests—to which it is applied. Although there is some back-end specialization for boolean terms, knowledge of particular bases (e.g. binary arithmetic) must be developed orthogonally.

In segregating the algebra of the metalanguage, the aim is to assure that the methodology and its design tools are adaptable to higher levels of description and disparate implementation targets. This quality distinguishes DDD from silicon compilers (as found in [5]), which