INDIANA UNIVERSITY
COMPUTER SCIENCE DEPARTMENT

TECHNICAL REPORT NO. 386

System Factorization in Codesign

A Case Study of the Use of Formal Techniques to Achieve Hardware-Software Decomposition

Bhaskar Bose, M. Esen Tuna, and Steven D. Johnson

JUNE 1993

To appear in the proceedings of the *1993 IEEE International Conference on Computer Design* (*ICCD '93*), Cambridge, Massachusetts, October, 1993.

System Factorization in Codesign *

A Case Study of the Use of Formal Techniques to Achieve Hardware-Software Decomposition

Bhaskar Bose†, M. Esen Tuna, and Steven D. Johnson

Indiana University
Computer Science Department
Bloomington, Indiana, U.S.A

Abstract

A major element of codesign is the task of decomposing a design in order to target some of its components to hardware and some to software while maintaining the integrity of the execution model. We illustrate how a previously developed algebraic technique we call system factorization adapts to this notion of decomposition. As an example, we describe how the mechanization of system factorization was used in the formal derivation of an implementation of Hunt's FM9001 microprocessor description using the DDD design derivation system. This case study demonstrates the benefits to system-level design in combining an executable modeling language, its associated formal-reasoning systems, hardware synthesis tools, and a hardware development platform in an integrated prototyping environment.

1 Introduction

With the increasing complexity and diversity of applications employing VLSI technology, design environments providing a unified framework for specification, design, and simulation/modeling are needed to relate hardware and software domains [1]. In this article we report on some experiences with *codesign* in a system that is unified by an underlying formal framework, based in functional model theory. The model theory is known to be highly expressive; its specialization to hardware system design is a task of our research. Such an approach should begin to yield advantages over more pragmatic tool development at a level when the need to support conceptual abstraction balances with the desire for a high degree of automation. We think this balance point may be in the area of codesign.

A design methodology based on the algebraic manipulation of purely functional forms provides a basis for our *digital design derivation* (DDD) framework [2, 3]. DDD provides a unified environment for specification, hardware derivation, and modeling. Specifications are written in a functional language. Equivalence preserving transformations are applied to these specifications for hardware derivation. The initial specification, as well as transformed expressions are executable in the functional modeling language.

This environment has been integrated with the Logic Engine Hardware Development Platform [4] providing a seamless integration of hardware and software. We use this capability to incrementally map our specifications to hardware while maintaining the global integrity of the system description with the formal system.

Issues in codesign relate to decomposing a system into related hardware and software components. Our methodology provides transformations for *system factorization* to decompose and restructure the design [5]. These transformations point to a general mechanism to isolate components of a specification for implementation in either the hardware or software domain.

In this paper we present an exercise in codesign in the context of developing a hardware prototype for the DDD-FM9001 [6]. The DDD-FM9001 is a 32-bit general purpose microprocessor realized in FPGAs derived from Hunt's Nqthm FM9001 specification [7] us-

*Research reported herein was supported, in part, by NSF: The National Science Foundation, under grants numbered DCR 85-21497, MIP 87-07067 and MIP 89-21842, and by NASA: The National Aeronautics and Space Administration under grant number NGT-50861.

To appear in the proceedings of the *1993 IEEE International Conference on Computer Design* (*ICCD '93*), Cambridge, Massachusetts, October, 1993.

†Email: bose@cs.indiana.edu

ing the DDD system. Details of the derivation of the DDD-FM9001 are reported in [6]. The DDD-FM9001 experiment provided a context in which we could apply our methodology and illustrate our framework.

In the following section we present our methodology. Section 3 details how the DDD-FM9001 was realized in hardware. We discuss how system factorization was used to isolate components so that they could be incrementally mapped onto hardware. Finally, in section 4 we close with some retrospective remarks.

2 The Design Framework

In the DDD design framework, specification, design derivation, and modeling are integrated in a unified framework. This framework consists of an executable modeling language, a formal reasoning system for circuit derivation, and a hardware platform for prototype development. We use the Scheme [8] programming language as our specification and executable modeling language, as well as the implementation language of our formal system. The language is based on functional model theory and is well suited for symbolic manipulation.

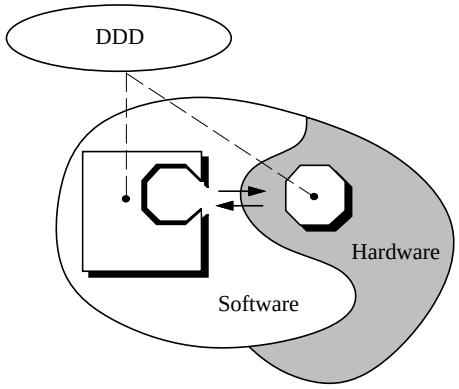


Figure 1: DDD Design Framework

Figure 1 illustrates the codesign aspect in our design methodology. The DDD system by means of system factorization decomposes the executable model of the circuit into subcomponents realized in hardware and software. The decomposition provides a mechanism for maintaining the integrity of the specification by deriving communication parameters, denoted by the arrows, between the software specification and hardware realization, as well as deriving a behavioral specification for the hardware component, denoted by the hexagon.

2.1 The DDD System

The DDD system was developed to help explore and demonstrate a formalization of digital design based on a functional algebra. The system implements a set of equivalence preserving transformations for deriving circuit descriptions from high-level functional specifications. DDD mechanizes the transformations needed for circuit derivation, but requires substantial guidance to perform derivation. The system is a collection of transformations operating on Scheme expressions. The system is used interactively or through command scripts to transform behavioral specifications to boolean sub-systems. The system has been integrated with various logic synthesis tools to generate realizations in PALs, FPGAs, and custom VLSI.

2.2 Logic Engine Hardware Platform

The Logic Engine [4] is a hardware development environment consisting of a Logic Engine Board and a PC host. The Logic Engine Board is a large printed circuit board designed for the testing and prototyping of hardware systems. The board has a general purpose prototyping area, a variable rate system clock, and input/output ports which interface to the PC host. The PC host provides a library of software routines written in Scheme to interface with hardware on the Logic Engine Board. The interface provides a transparent cosimulation environment between the hardware components and the executing software model.

3 A Case Study: The DDD-FM9001

The DDD-FM9001 was a design exercise to construct a hardware implementation of the FM9001 [7] microprocessor description using the DDD system. The FM9001 is a 32-bit general purpose microprocessor mechanically verified by Hunt in the Nqthm theorem prover [9] and realized in LSI Logics gate array technology. Details of the proof are reported in [7].

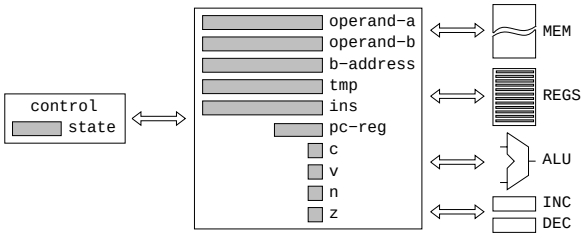


Figure 2: DDD-FM9001 Logical Organization

The processor incorporates 32-bit addressing, 16 32-bit general purpose registers, 5 addressing modes, a 16 function ALU, and a conditional assignment statement.

cmd	M	A	B
0	M	A	B
1	M	A	B
2	M	A	B
3	M	rd(T, M)	B
4	M	A	B
5	M	A	B
6	M	s-ext(...)	B
7	M	T	B
8	M	rd(dcr(T), M)	B
9	M	rd(T, M)	B
10	M	A	B
⋮	⋮	⋮	⋮
14	M	A	T
15	M	A	rd(dcr(T), M)
16	M	A	rd(T, M)
17	M	A	B
⋮	⋮	⋮	⋮
25	wr(BA, M, bv(...))	A	B
26	M	A	B

Figure 3: RTT Fragment Before Factorization of M

As a codesign exercise in the context of building a prototype, the DDD-FM9001 was incrementally mapped to hardware. We began with a complete executable CPU specification in software. Transformations were applied interactively to the specification in order to restructure and decompose the design for hardware implementation. As components were factored, signals were generated within the CPU specification in order to communicate with the factored components. The factored components were then implemented in hardware. As components were mapped to hardware, the entire system remained executable, and the formalism maintained the integrity of the design. Ultimately, the entire specification was mapped into hardware.

3.1 Software to Hardware Migration

The initial specification was decomposed into a control component and a structural component. Transformations were applied to the structural component in order to isolate the memory from the CPU specification. At this stage in the derivation, the memory component was implemented in hardware. The migration of the memory into hardware was transparent from the perspective of the CPU model. Further transformations were applied to factor the register file and arithmetic components from the datapath. Figure 2 illustrates this decomposition. As with the memory, the register file was implemented in hardware while the execution model remained consistent. Additional transformations were applied to the remaining components in order to restructure the design for the intended target technology. Finally, the remaining components were mapped onto hardware. In the next section we look at the key elements involved in factorization in the context of the factorization of the memory component.

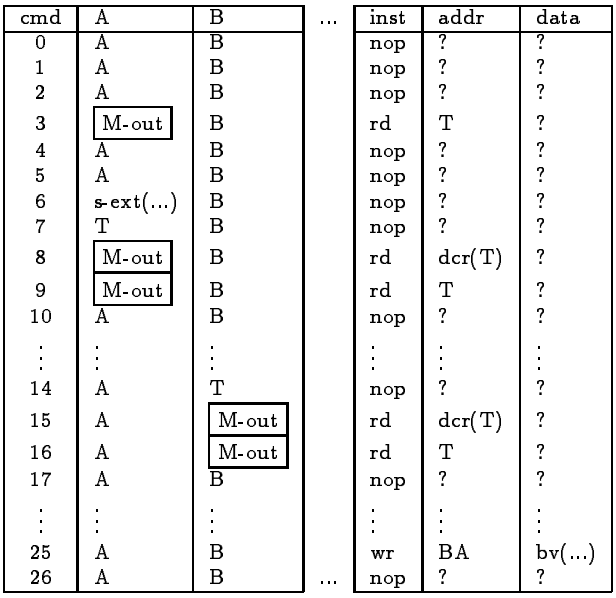
```
M-out = absM(M0, inst, addr, data)
where
    absM(M0, inst, addr, data) = rd(addr, M)
    where
        M = M0 ! construct(inst, M, addr, data)
        and
            construct(inst, M, addr, data) =
                case inst
                    [nop → M]
                    [rd → M]
                    [wr → wr(addr, M, data)]
```

Figure 4: Abstract Memory Specification

3.2 Factorization of Memory

Figure 3 shows a fragment of the register transfer table (RTT) of the DDD-FM9001. The RTT defines a set of register transfers that occur in parallel given the corresponding command code. For example, in figure 3, `cmd = 14` denotes that the registers `M`, `A`, and `B` will be updated with the values `M`, `A`, and `T` respectively. Highlighted in the boxes are expressions relating to the memory signal, `M`. In the process of factorization, the DDD system isolates these signals and derives a new RTT (figure 5) and a behavioral specification of the factored component (figure 4). The goal of factorization is to transform the memory from a functional abstraction to a process abstraction.

Applying system factorization to the original RTT returns the transformed RTT in figure 5. DDD derives a set of signals, `inst`, `addr`, and `data`, necessary to communicate with the factored component, from the operations on memory. In addition, a signal de-



cmd	A	B	...	inst	addr	data
0	A	B		nop	?	?
1	A	B		nop	?	?
2	A	B		nop	?	?
3	M-out	B		rd	T	?
4	A	B		nop	?	?
5	A	B		nop	?	?
6	s-ext(...)	B		nop	?	?
7	T	B		nop	?	?
8	M-out	B		rd	dcr(T)	?
9	M-out	B		rd	T	?
10	A	B		nop	?	?
⋮	⋮	⋮		⋮	⋮	⋮
14	A	T		nop	?	?
15	A	M-out		rd	dcr(T)	?
16	A	M-out		rd	T	?
17	A	B		nop	?	?
⋮	⋮	⋮		⋮	⋮	⋮
25	A	B		wr	BA	bv(...)
26	A	B	...	nop	?	?

Figure 5: RTT Fragment After Factorization of M

noting the value from memory, `M-out`, is derived and substituted for each occurrence of the read operation to memory. DDD also generates a behavioral specification (figure 4) for the memory component encapsulating the memory signal and the operations on the memory. The resulting table composed with the abstract memory specification computes the same function as the original table. For example, in row `cmd = 3` of the original RTT, the expression `rd(T, M)` in `A`, is replaced with `M-out` in the transformed table, where `M-out` is the value of $\texttt{absM}(M_0, \texttt{inst}, \texttt{addr}, \texttt{data})$. The values for `inst`, `addr`, and `data`, correspond to the original expression `rd(T, M)`. The result is a description of a communicating system composed of a CPU and memory process.

4 Conclusions

In this paper we have shown how an algebraic approach can be used to decompose a design in order to map components to hardware. We showed how this methodology was applied to a design example, the DDD-FM9001, in constructing a realization. In this example, the benefits of combining an executable modeling language, a formal reasoning system, logic synthesis tools, and a hardware prototyping platform were exploited.

The realization of the DDD-FM9001 was constructed incrementally as the derivation progressed towards a complete hardware implementation. At points in the derivation, parts of the specification were in hardware, and parts were in software. The gradual migration of software into hardware provided a means of testing hardware components individually while maintaining their relationship with the rest of the system.

In [10], the notion of system factorization is extended to include protocol specification to formalize a more general process decomposition. Future work will involve the integration of more sophisticated factorization techniques in our framework for design decomposition.

References

[1] P. Subrahmanyam, G. D. Micheli, and K. Buchenrieder, "Hardware-software codesign," *Computer IEEE*, pp. 84–87, Jan. 1993.

[2] S. D. Johnson, *Synthesis of Digital Designs from Recursion Equations*. Cambridge: The MIT Press, 1984.

[3] S. D. Johnson, B. Bose, and C. D. Boyer, "A tactical framework for digital design," in *VLSI Specification, Verification and Synthesis* (G. Birtwistle and P. Subrahmanyam, eds.), pp. 349–383, Kluwer, 1988.

[4] D. Winkel, F. Prosser, R. Wehrmeister, W. C. Hunt, and C. Hess, "A student VLSI hardware tester," in *Proceedings of the Microelectronic Systems Education Conference and Exposition*, pp. 15–24, 1990.

[5] S. D. Johnson, "Manipulating logical organization with system factorizations," in *Hardware Specification, Verification and Synthesis: Mathematical Aspects* (M. Leeser and G. Brown, eds.), pp. 260–281, Springer-Verlag, July 1989.

[6] B. Bose and S. D. Johnson, "DDD-FM9001: Derivation of a verified microprocessor. An exercise in integrating verification with formal derivation," in *Correct Hardware Design and Verification Methods* (G. J. Milne and L. Pierre, eds.), vol. 683, pp. 191–202, Springer-Verlag, 1993.

[7] W. A. Hunt, "A formal HDL and its use in the FM9001 verification," in *Mechanized Reasoning in Hardware Design* (C. Hoare and M. Gordon, eds.), Prentice-Hall, 1992.

[8] W. Clinger and J. Rees, "The revised^4 report on the algorithmic language Scheme," *Lisp Pointers*, vol. 4, no. 3, pp. pages 1–55, 1991.

[9] R. S. Boyer and J. S. Moore, *A Computational Logic Handbook*. New York: Academic Press, 1988.

[10] K. Rath and S. D. Johnson, "Toward a basis for protocol specification and process decomposition," in *Proceedings of the IFIP Conference on Hardware Description Languages and their Applications*, 1993.