THE CASE FOR A NEW CONTROL STORE ARCHITECTURE FOR THE NEXT GENERATION OF VLSI COMPUTERS

David A. Patterson

2-14-80

ABSTRACT

Most computer manufacturers have questioned the usefulness of providing users with a writeable control store. For reasons important to computer manufacturers, alterable control store should be used in the next generation of VLSI computers.

Introduction

Very Large Scale Integrated circuit technology (VLSI) is both a blessing and a curse for computer manufacturers. Blessed with geometric growth in circuit complexity, manufacturers can put more and more functions into a single package. The curse is realized when the manufacturer is faced with designing, testing, and debugging timely products of expanding complexity. One of the main problems of VLSI computers is control. It is thus not surprising that the simplified design and debugging of microprogrammed control has made microprogramming the popular choice for VLSI computers [Stritter 78] [Patterson 79, 80] [Nash 79]. For example, two of the current VLSI computers, the 8086 and the MC68000, have microprogrammed control implemented in read only memory (ROM). Many of these ideas were presented before [Patterson 79, 80]; this paper generalizes these ideas beyond a single ~~idea~~ design.

The VLSI growth curve forces a manufacturer to begin the design of the

next computer at least as soon as the current computer is announced. We expect that successors to the 8086, Z8000, and MC68000 are under design or finished. We also expect that minicomputer and mainframe manufacturers will introduce VLSI computers. The goal of this paper is to persuade the designers of these future computers that control store should not be implemented entirely by ROM. For the purposes of this paper, we define the next generation VLSI chip to be at least 250,000 devices.* A VLSI computer is then a computer built from custom VLSI chips. This definition includes single-chip and multi-chip computers.

We now describe a new control store architecture and then explain four reasons for this design.

A New Control Store Architecture

It is easier to understand the reasons for a new control store architecture if we present the design first. There are three main concepts of the proposed new control store architecture:

(A) *Kernel ROM:* A ROM contains the simple, frequently used instructions plus bootstrap microcode.

(B) *Shared data and address pins:* As pins are precious in VLSI, separate pins for microinstructions and microinstruction addresses cannot be justified. This control store architecture shares the micro-architecture address and data lines with the macro-architecture address and data lines.

(C) *Microcode cache:* Rather than simply use RAM control store, we use an on-chip cache to contain the most frequently used microinstructions. A complete copy of the microcode is in main memory thereby avoiding the complexity of microcode page faults.**

---

* As a reference point, the MC68000 has about 70,000 transistors positions of which 40,000 are used (Nash 79).

** If we allow memory faults to occur for microcode fetches, then the computer must be able

The feasibility of each concept is now examined.

There is no problem implementing ROM and RAM on the same chip (e.g., 8048) and there is ample precedent for having control stores implemented with both ROM and RAM (e.g., HP 1000, VAX 11/780, IBM 360/85).

The sharing of address lines between architecture levels leads to conflicts. The first conflict is the form of the address and the data. One level uses macro-architecture addresses and data types while the other uses those of the micro-architecture. A cache solves this problem. As shown in Figure 1, the cache controller turns micro-addresses into macro-addresses and macrodata into microinstructions.

Another conflict occurs when both levels need to use the resource at the same time. Since one level must wait, this conflict will reduce performance of the VLSI computer. Again a cache provides the solution. The traditional cache fetches instructions over a bus and so must the microcache. The microcache, however, suffers the disadvantage that it must use a bus that is designed to perform adequately for the macro-architecture. Thus the time to fetch data over the bus will take several microinstructions. The question that must be answered is whether a microcode cache can effectively use a macro-architecture bus without degrading the performance of the CPU?

Certainly the microcache can use the bus some portion of the time without degrading performance; if this were not the case the CPU would be bus limited. This bottleneck would be then removed by adding a cache for instructions and data. Assume that the microcache can use the bus as much as 10 percent without degrading performance. Using the notation of I microcycles to fetch a block for a microcache miss and the hit ratio for the microcache is H, then the average fraction of bus cycle per microinstruction is (1-H)*I. Then varying H and

---

to save the state of the micro-architecture. This is certainly difficult to justify.

I over reasonable values, we get Table 1. It appears that the hit ratio must exceed .95 to achieve acceptable bus performance. This seems high, but studies dealing only with instruction hit ratios are in this range. For example, depending upon the size of the cache, Smith reports that the hit ratios of a COBOL program and a FORTRAN program vary from .95 to .995 [Smith 80].

High microinstruction hit ratios are also predicted by instruction set studies. Dynamic analysis of instruction sets find that only a few instructions are responsible for a large percentage of the executed instructions. For example, the IBM 360 has about 130 instructions, but a measurement of a compiler found that 10 instructions were responsible for 80% of all instructions executed, 16 use 90%, 21 use 95%, and 30 use 99% [Alexander 75].

The final question is where the microinstructions are stored when they are not in the microcache. The answer is that the microprogram is stored in whatever memory medium is used to store the program. If the VLSI computer is part of a traditional computer system that includes secondary storage, then the microcode will eventually be placed there. During system initialization, a copy of the microcode would be read into the main memory for accesses by the microcache. If the VLSI computer does not have secondary storage, it still must have a place to store its program and thus a place for its microprogram. For example, if the program is a ROM, then the microprogram would also be in ROM.

We conclude that the microcache is technically feasible for VLSI computers. We now proceed with the reasons why it is necessary.

H I	1	2	4	8	16	32	64
.80	.2	.4	.8	1.6	3.2	6.4	12.8
.90	.1	.2	.4	.8	1.6	3.2	6.4
.95	.05	.1	.2	.4	.8	1.6	3.2
.99	.01	.02	.04	.08	.16	.32	.64
.999	.001	.002	.004	.008	.016	.032	.064

TABLE 1

Values of the fraction of a bus per microinstruction versus the hit ratio H and the number of microcycles per microcache miss. (Everything below the heavy line uses the bus 10 percent or less.)

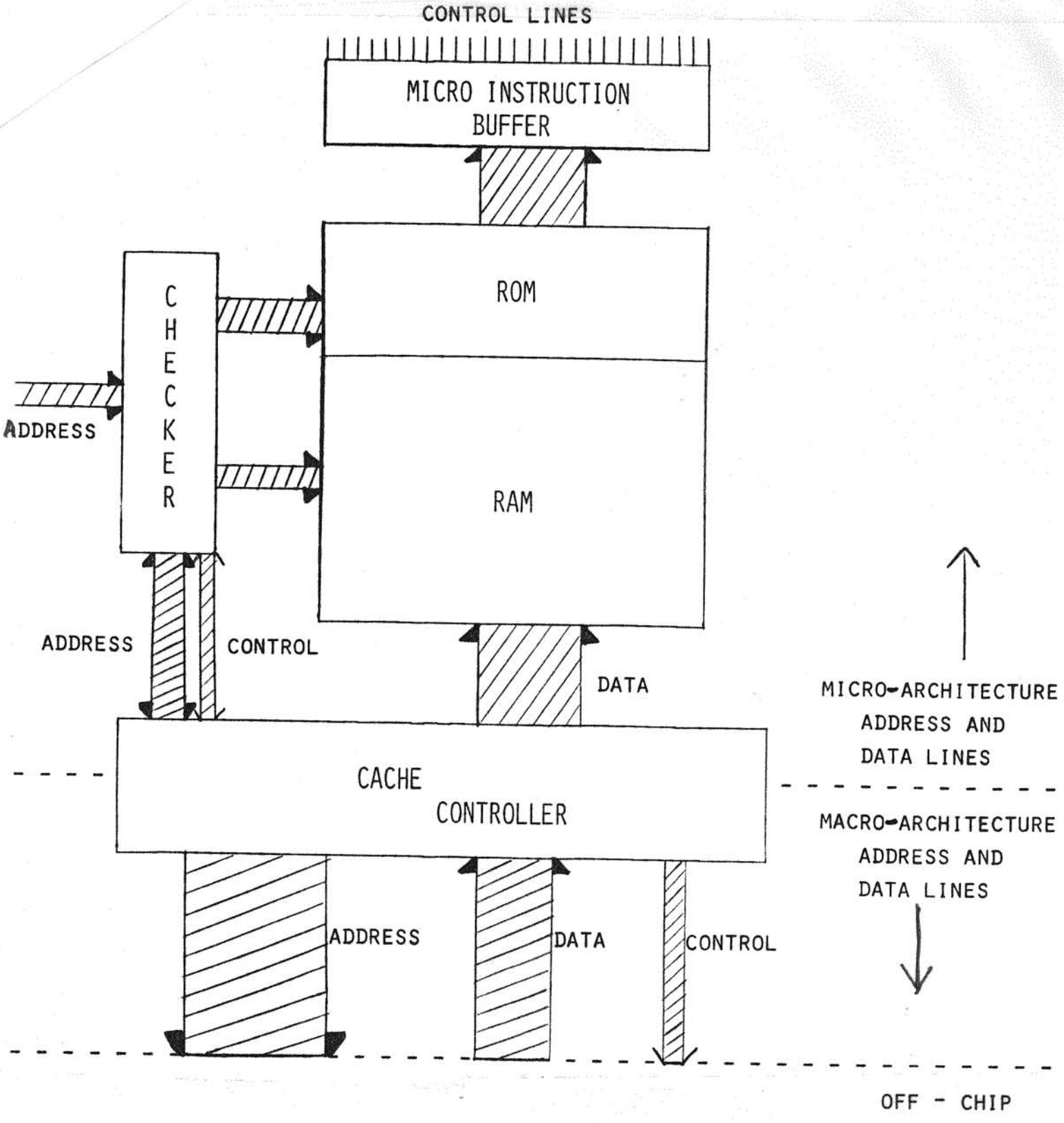

FIGURE 1. Microcode Cache Block Diagram.

Reason #1: Microdiagnostics

Testing the current LSI chips is a difficult problem: VLSI chips require that testing be considered during the design of the chip. Techniques such as signature analysis [Nadig 77] and level-sensitive scan design (LSSD) [Eichelberger 78] are being used to simplify testing. Williams and Parker point out three important testing problems:

(1) *Initialization:* Hardware testing requires the ability to set all memory elements to a fixed value before the test is run. Quoting Williams and Parker [Williams 79]:

> "Years of practical experience in testing have identified the lack of a synchronizing sequence as the single most destructive network (hardware) characteristic with respect to its test."

(2) *Observability:* This refers to the ease with which internal states of the hardware can be examined.

(3) *Controllability:* This deals with the ease of "steering" the hardware through its various functions.

These problems are difficult to solve when limited to the macro-architecture; the natural place is the micro-architecture. Test microprograms, called *microdiagnostics*, have been shown to be more accurate, faster, and less expensive to develop than software diagnostics [Husson 70]. Microdiagnostics also need a smaller portion of the hardware to be working to be able to correctly test the rest of the hardware. The combination of testable design methodologies and microdiagnostics will greatly simplify the task of efficient and thorough testing of VLSI chips.

Another important problem, which occurs before production, is the micro-architcture acceptance test. During development the micro-architecture is fre-

quently implemented before the instruction set microprogram is completed. Thus microdiagnostics are also used to see if the registers and data paths work as the microprogrammer thinks they should.

As microdiagnostics can be implemented in ROM, why do microdiagnostics imply new control store architectures? There are two reasons:

(1) *Unnecessary Function:* Microdiagnostics to perform the functions of initial design validation and manufacturing checkout are generally not the same microprograms. The goal of the former is to verify that the micro-architecture is really what the microprogrammer thinks it is. The goal of the latter is to test the logic used in the implementation of a debugged design. The ROM approach would require both functions to be provided even though the former test is obviously unnecessary after the chip is manufactured.

(2) *Wasted Silicon:* Microdiagnostics are clearly important yet they are rarely used. A ROM implementation means the chip must dedicate area solely for this function. This drawback will obviously drastically limit the size and therefore the effectiveness of microdiagnostics.

Another less serious problem is that changes to the IC layout will likely require changes to the microdiagnostics.

Our conclusion is that the appropriate memory for diagnostics is not ROM, and that the importance of good diagnostics (and potential ROM savings) justifies new control store architectures.

Reason #2: Microprogramming Errors

The trend towards more complex instruction sets is illustrated by the following pairs of computers built by the same companies (the left hand column contains the predecessors of computers in right hand column):

Intel	8080	-	8086
Zilog	Z80	-	Z8000
Motorola	6800	-	MC68000
DEC	PDP11	-	VAX11

Readers familiar with these eight computers will appreciate the increased complexity. One indication of the complexity of the instruction set is the size of the control store. For example, the size has grown from 256 x 80 in the PDP 11/40 to 5120 x 96 in the VAX 11/780.

The VAX 11/780 design team realized the potential for microcode errors in the ROM. Their solution was to use an FPLA and 1K words of Writeable Control Store (WCS) to patch microcode errors. To date 50 patches have been made.* Although difficult to document, it is likely that the same experience was shared by IBM, as the 360 line was implemented almost entirely with ROM and the 370 line uses alterable control store exclusively.

Complex instruction sets virtually guarantee the impossibility of getting all errors out the first version (or perhaps any version) of the microcode. The 8086 and MC68000 are only about the complexity of the PDP 11. The only 8086 microcode error was discovered before the masks were made, but the wrong version of the microcode was used. The error was corrected by patching the mask that controlled the contents of control store. The size of control store is 512 x 21 with less than 500 locations used [Stoll 79]. The first version of silicon of the MC68000 had microcode and circuit mistakes. The second version, which is a preproduction version called the XC 68000, comes with a short list of exceptions to the reference manual. The control store is effectively 550 68-bit microin-

---

* Since there was a good chance of errors in the complex of VAX instructions, some of these were implemented only in WCS. Each patch means several microinstructions must be put into WCS, so the 50 patches plus existing microroutines use a substantial portion of the 1024 words.

structions** [Nash 79]. The 10000 bit control memory of the 8086 contained almost no errors, while the virtual 40000 bit control memory of the 68000 had errors in two versions of microcode. If the instruction sets of the 8086 and 68000 evolve to require a 500,000 bit control memory like the VAX, microcode errors will be a serious design problem.

With VLSI chips, the response to a microcode error depends on the severity of the error and when it is discovered in the development-production cycle. The manufacturer has two solutions to the microcode error problem:

(1) *Require error free microcode* - The manufacturer must allocate enough time to exhaustively test each version or simply announce the computer whenever the microcode is believed to be error free. In the case of the NCR Criterion, microcode testing and debugging took one year [Ueijo 79]. The architecture proposed in this paper will work well with this approach as the VLSI computer need not change; only the contents of the memory that supplies microinstructions must change. Decoupling of the permanent microcode store from VLSI chip reduces the time penalty for this approach.

(2) *Note the error and correct it in a later version* - Given the economics of silicon, it is obvious that chips cannot be "recalled". With a microcache, the manufacturer can release revised versions of microcode as the product is shipped. Customers that want the latest version can get it from the manufacturer. Depending upon the environment, this could be done with a floppy disc or by a new ROM.

Our conclusion is this proposal can reduce development delays due to microprogramming errors and provide a simple mechanism for correcting errors.

** The 68000 uses a two-level control store with about 550 17-bit microinstructions and about 175 68-bit nanoinstructions.

Reason #3: Saving Silicon

As mentioned above, instruction sets as complicated as the VAX require as much as 500,000 bits. Even given VLSI estimates of 1985, this is a substantial portion of one or several chips. A microcode cache may help this problem if it uses less area than the complete microprogram implemented in ROM. Assume that most of the logic of a cache is the data memory. For memory chips, a static RAM cell requires about four times the number of transistors as a ROM cell [Patterson 79, 80].* Since caches are generally 1/10 to 1/100 the size of the backing memory, it is likely that less area is needed on chip for a microcache.

Clearly the use of the cache cannot actually save transistors; it can only move them off the CPU chip and into less expensive, denser, and rapidly designed memory chips. Our conclusion is that a microcache for complex instruction sets can actually require less area than ROM on VLSI CPU chips.

Reason #4: Expanding an Architecture

In the case of minicomputer and mainframe computer manufacturers, it is popular to provide a family of computers of varying cost and performance (e.g., PDP 11 and IBM 370). It is not unusual for a family's architecture to be enhanced after it has been introduced. For example, IBM added a set of operating system instructions to the 370, and DEC has extended PDP 11 architecture by adding commercial-oriented instructions in the PDP 11/44. Given the goals of the semiconductor manufacturers, a family may not to occur. It is likely, however, that minicomputer and mainframe computer manufacturers will introduce VLSI computers as a member of a family. If only some members of the family are enhanced, the manufacturer must have a different set of system programs for that model. An alternative that avoids this very troublesome and

---

* When the ROM is on the same chip as the CPU, the ratio may be even higher, but the argument holds for higher ratios.

expensive problem is for VLSI computers to use a microcache.

Summary

This paper began with a new control store architecture that relies upon a microcode cache plus kernel ROM to implement control memory. Four separate reasons were presented that require this architecture:

(1) Testing VLSI chips with microdiagnostics implemented in an alterable control memory allow much more elaborate tests and more effective use of control memory.

(2) The rising complexity of instruction sets makes microprogramming errors a serious problem in the development of a new computer. By moving the complexity off the CPU chips, the proposed architecture significantly reduces the cost and delay normally implied by these errors.

(3) Since caches are a small fraction of the total program size, and RAM cells require only about four times the area of ROM, this architecture reduces the area required by control memory on the CPU chips.

(4) Enhancements to an architecture family are most easily made when all implementations use an alterable control store. This proposal allows that feature to be effectively provided in even a VLSI member of that family.

Any of the four reasons can be used to justify using WCS. Collectively, the issues of chip testing, microcode errors, saving silicon "nanoacres", and architectural expansions present powerful arguments that the designers of the next generation of VLSI computers should use the proposed design.

Acknowledgment

This paper was considerably improved due to discussions with Dileep Bhandarkar, Peter Stoll, and Jim Goodman. Al Despain, Dave Ditzel, and Carlo Séquin gave suggestions on the presentation of this paper. The X-Tree group in general and Al Despain and Carlo Séquin in particular are responsible for the fertile environment which suggest, encourage, and examine new ideas.

This research was sponsored by the Defense Advance Research Projects Agency (DoD), ARPA Order No. 3803, and monitored by Naval Electronic System Command under Contract No. N00039-78-G-0013-0004.

References

[Alexander75] W.C. Alexander and D.B. Wortman, "Static and Dynamic characteristics of XPL Programs," *Computer*, pp. 41-46, November 1975, Vol. 8, No. 11.

[Anderson79] K.R. Anderson and H.A. Perkins, "Hardware Test Technology," *Computer*, pp. 7-8, October 1979, Vol. 12, No. 10.

[Bell78] C.G. Bell, J.C. Mudge, and J.B. McNamara, *Computer Engineering: A DEC View of Hardware* System Design, Digital Press, 1978.

[Despain78] A.M. Despain and D.A. Patterson, "X-TREE: A Tree Structured Multi-Processor Computer Architecture," *Conference Proc., Fifth Annual Symposium on Computer Architecture*, pp. 144-151, April 3-5, 1978.

[Eichelberger78] E.B. Eichelberger and T.W. Williams, "A Logic Design Structure for LSI Testability," *Journal of Design Automation & Fault-Tolerant Computing*, Vol. 2, No. 2, May 1978, pp. 165-178.

[Husson70] S.S. Husson, *Microprogramming: Principles and Practices*, Prentice-Hall, Engelwood, N.J., pp. 109-112, 1970.

[Nadig77] H.J. Nadig, "Signature Analysis -- Concepts, Examples and Guidelines," *Hewlett Packard Journal*, May 1977, pp. 15-21.

[Nash79] J. Nash and M. Spak, "Hardware and Software Tools for the Development of a Micro-programed Microprocessor," *12th Annual Workshop on Microprogramming*, pp. 73-83, November 18-21, Hershey, PA.

[Patterson79] D.A. Patterson, E.S. Fehr, and C.H. Séquin, "Design Considerations for the VLSI Processor of X-TREE," *Conference Proc.,*

Sixth Annual Symposium on Computer Architecture, pp. 90-101, April 23-25, 1979.

[Patterson80] D.A. Patterson and C.H. Séquin, "Design Considerations for Single-Chip Computers of the Future," Accepted for publication, *IEEE Journal of Solid- State Circuits, IEEE Transactions on Computers*, Joint Special Issue on Microprocessors and Microcomputers, February 1980.

[Robbi72] A.D. Robbi, "Microcache: A Buffer Memory for Microprograms," *COMPCON 72 Digest of Papers*, IEEE, September 1972, pp. 123-125.

[Smith80] A.J. Smith, "Cache Memories," *in preparation*, 1980.

[Stoll79] D. Stoll, *private communication*, 1979.

[Stritter78] E. Stritter and N. Tredennick, "Microprogrammed Implementation of a Single Chip Microprocessor," *Proc. 11th Workshop on Microprogramming*, pp. 8-16, November 19-22, 1978.

[Veijo79] W. Veijo, *private communication*, 1979.

[Williams79] T.W. Williams and K.P. Parker, "Testing Logic Networks and Designing for Testability," *Computer*, pp. 9-22, October 1979, Vol. 12, No. 10.