

Aspects of Full-Custom VLSI Microprocessor
Design and Implementation

By

Daebum Lee

B.S. (University of California) 1983

M.S. (University of California) 1986

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ENGINEERING

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Approved:

.....	<i>David A. Hodges</i>	4/18/89
.....	<i>Chair</i> <i>Randy H. Katz</i>	4/18/89
.....	<i>Fred Burdson</i>	4/22/89



ASPECTS OF FULL-CUSTOM VLSI MICROPROCESSOR DESIGN AND IMPLEMENTATION

Daebum Lee

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT


There is a broad spectrum of design styles that have proven successful for the construction of VLSI circuits and systems. Semi-custom to full-custom design styles offer a wide range of resulting performance, expected turn-around time, and required design effort. Implementation alternatives, such as replacing dynamic memory for static memory to implement a denser on-chip memory, also exist at all levels of design hierarchy. To make the best use of scarce resources on a single chip microprocessor and to make the emerging CAD tools truly useful, alternatives in the implementation of a microprocessor must be carefully evaluated. The research reported in this thesis focuses on issues concerning these alternatives, especially in the areas of on-chip memory design and automated control logic design.

The methodologies and techniques used to maximize the performance of a full-custom VLSI microprocessor, called the SPUR CPU, is initially presented to provide an overview of microprocessor design strategies. The rest of the research presented is transpired from new ideas and better alternatives which have become available since the SPUR CPU. These are based on lessons learned in the SPUR design and advanced computer-aided design tools such as multi-level logic synthesis system. A rigorous evaluation of these alternatives is attempted and results from the evaluation establish the effectiveness of the alternatives considered.

To increase the area efficiency of the on-chip memory, two memory design techniques are proposed and evaluated. Selective invalidation instead of refreshing, implemented using low overhead dynamic CMOS circuits, can effectively eliminate the refreshing requirement of

dynamic memory. With this scheme, the size of an on-chip local memory can be substantially increased without increasing the scarce silicon area. Trace-driven simulations show the effectiveness of this scheme over a simple invalidation scheme. The demand for high bandwidth local memory expedited by parallel execution of programs through multiple functional units requires a fast, stable, yet compact multi-port memory cell. A single-ended access, static memory cell operated at reduced voltage levels is proven to be useful for such applications.

A part of this research is devoted to investigating various layout styles for microprocessor's control. Recently, various VLSI CAD tools have emerged to facilitate the hard-wired control design. Behavioral synthesis and multi-level logic optimization systems provide particularly efficient and high-performance hard-wired logic implementation, even with semi-custom layout styles, such as standard cell-based design. All new design methods aim for simplicity and regularity. The standard cell based design style, when combined with multi-level logic optimization, can provide a resulting design as good as full-custom version but in much shorter design time.

A handwritten signature in cursive script that reads "David A. Hodges". The signature is written in black ink and is positioned above a horizontal line.

Professor David A. Hodges
Committee Chairman

*To my parents, my wife Joanne, and Lydia and Laura,
for their love and encouragements.*



ACKNOWLEDGEMENT

First and foremost I wish to thank Professor David A. Hodges and Professor Randy H. Katz for their invaluable guidance in my research. Professor A.R. Newton inspired my ideas on CAD-related research and suggested the evaluation on alternatives due to new waves in CAD tools. Professor Frederick E. Balderston was kind enough to serve both on my qualifying and dissertation committees.

I especially would like to thank Professor D.A. Patterson and other members of the SPUR project who made it an exciting project to participate in. I feel fortunate to have been a part of the SPUR project, as it provided me with the opportunity to learn about various aspects of the computer design. In particular, I thank D.K. Jeong with whom I have had enjoyed a great deal of technical discussions and a convivial working relationship. I also acknowledge the technical contributions of Richard Duncombe, Wook Koh, and Jacky Mak in building the SPUR CPU chip, and Ken Lutz for helping me with various aspects of chip testing.

The material presented here is based on research supported in part by the Defense Advanced Research Projects Agency under contract N00039-85-C-0269 and in part by Semiconductor Research Corporation.



TABLE OF CONTENTS

CHAPTER 1 : INTRODUCTION	1
1.1 Motivation	1
1.2 VLSI Microprocessor Design and Implementation	2
1.3 A VLSI Design Methodology for a Microprocessor	4
1.4 Related Work	5
1.5 Thesis Organization	6
1.7 References	8
CHAPTER 2 : VLSI IMPLEMENTATION OF THE SPUR CPU CHIP	12
2.1 Introduction	12
2.2 An Overview of the SPUR CPU Architecture	17
2.3 Hardware Implementation of the SPUR CPU	21
2.3.1 The Instruction Unit	21
2.3.2 The Execution Unit	23
2.4 Design, Verification, and Testing Methodology	30
2.4.1 Design Methodology	31
2.4.2 Verification Methodology	33
2.4.3 Testing Methodology	34
2.4.4 Design Metrics	37
2.4.5 Results	37
2.5 Summary	41

2.6	References	41
CHAPTER 3 : ON-CHIP MEMORY DESIGN		43
3.1	Introduction	43
3.2	On-chip Memories in Microprocessors	45
3.2.1	Local Memory for Instruction Store	45
3.2.2	Local Memory for Data Store	47
3.2.3	The Focus and Limitations of the Research	49
3.3	On-chip DRAM Caches	50
3.3.1	Why DRAMs?	50
3.3.2	Limitations of DRAMs for an On-chip Memory	51
3.3.3	Non-refreshing DRAMs for On-chip Caches	54
3.3.4	Evaluations	57
3.4	Multi-port Memory Design	67
3.4.1	Multiple Functional Units	67
3.4.2	Multiple Set of Register Files and Multi-port Cache Memory	68
3.4.3	Multi-port Memory Cells	71
3.4.5	Analysis and Comparison	74
3.5	Summary	77
3.6	References	78
CHAPTER 4 : CONTROL DESIGN ALTERNATIVES		82
4.1	Introduction	82
4.2	Microprocessor Control	83

4.2.1 Microprogrammed Control	85
4.2.2 Hardwired control	86
4.3 Alternative Implementations	88
4.3.1 Automatic Synthesis of Control Logic	89
4.3.2 PLA Based Control Design - A Traditional Approach	92
4.3.3 Standard Cell Based Control Design	93
4.3.4 Gate Matrix Based Control Design	94
4.4 Evaluation and Comparison	95
4.4.1 Method of Evaluation	95
4.4.2 Results of Evaluation and Comparison	98
4.5 Summary	105
4.6 References	107
CHAPTER 5: CONCLUSION	109
5.1 Summary	109
5.2 Future Research	110
5.3 Conclusion	111



1

Introduction

This thesis consists of three self-contained chapters that examine design and implementation of a VLSI microprocessor chip (the SPUR CPU), on-chip memory design techniques, and alternative implementations of microprocessor's control logic. Chapters 2, 3 and 4 are stand-alone presentations. In this introductory chapter, I provide an overview of the research and thesis organization.

1.1. Motivation

There is a broad spectrum of design styles (e.g. semi-custom or full-custom design styles, and static or dynamic memory for on-chip local memory design) that have proven successful for the construction of VLSI circuits and systems. For all these styles and for all the abstraction levels (behavior, logic, circuits, and layout) in the design hierarchy, good computer aided design (CAD) tools are indispensable. To make the emerging CAD tools truly useful and to take advantage of advanced VLSI technology, all of the many different design

styles must be carefully examined with choices made judiciously for a particular application.

The research reported in this thesis is originally motivated from the design and implementation of the SPUR CPU chip. After presenting the details of its design and implementations, I will examine several alternatives for a full-custom VLSI microprocessor design. Consequently, a detailed evaluation of alternatives will be available for future microprocessor development.

The objectives of the research are as follows:

- (1) Develop a better understanding of implementation alternatives in VLSI microprocessor design.
- (2) Examine alternatives rigorously, in order to be able to compare and evaluate them quantitatively.
- (3) Provide ideas or guidelines for future microprocessor design and for the development of computer-aided design tools.

1.2. VLSI Microprocessor Design and Implementation

Over the last decade, many commercial and research microprocessors have been successfully built on a single chip. A general overview of design steps in the development of the microprocessor is summarized in the following. Among various steps, this research focuses on the VLSI design issues, particularly on alternatives for a full-custom implementation of the VLSI microprocessor.

Architecture definition

High level architecture and instruction set is defined in this step. Global design issues such as language support, operating system support, memory management support and coprocessor support are considered and defined.

Technology selection

The performance of the microprocessor strongly depends on the technology in which the microprocessor is implemented. Emitter-coupled logic (ECL), CMOS, and NMOS technologies are among the most popular choices. The selection also strongly relies on the design environment supported by CAD tools, since some CAD tools may only work with a particular technology. The design style associated with each technology is also considered here. The selection of design style greatly influences design cost and turnaround time. Semicustom design, like the gate array design style, may require a few weeks to get the first working silicon, while highly optimized full custom design may take several years. Other semicustom design styles include sea of gates (channel-less gate array), standard cell, and macro cell design styles.

Microarchitecture design

An important task in this step is to specify a detailed behavioral description of the architecture. Since it represents a complete design of the microprocessor for a selected technology, it is used to verify architecture defined at a higher level as well as to provide diagnostic vectors for later stages of design verification and debugging.

VLSI design and implementation

The chip implementing the microarchitecture is designed in this step. First, the behavioral description is synthesized into several different modules, such as control logic, data path, and memory. Since implementation styles for each of these modules are very different, different implementation strategies are used. Design methodology for each of these modules is chosen so that the best overall performance can be achieved.

Design verification

Once schematics or a layout representation of the micro-architecture is made, this representation is verified against the behavioral description, to make sure that the two representations are totally equivalent. Usually a net list is generated from the schematic diagram or mask layout, and then logic simulations are performed on this extracted net

list and results are verified against the behavioral simulation. Design and electrical rules are also checked in this step.

Integrated circuits fabrication

Masks are made from the layout and the design is transformed into the silicon.

Testing

Testing verifies functional behavior, electrical performance and fabrication processes. Test vectors are generated so that they can cover as much area of the chip, and as many functionalities, as possible.

1.3. A VLSI Design Methodology for a Microprocessor

Optimizing performance in VLSI digital systems involves several design choices, including the choice of the best implementation methodology. Alternatives exist at all levels of design and each must be carefully examined to obtain an optimal implementation for a given architecture and technology. Because some of these processes are very time-consuming, designers rely on structured methods aided by a computer.

Many design strategies are used to deal with complexities in VLSI design [MeC80][WeA85]. One of the more frequently used strategies is to divide the design into several parts so that each can be implemented using the most efficient method. This would result in an optimal implementation in part by part. A balanced optimization is also important in such strategy, since overall performance is usually determined by the most critical part. Various tradeoffs among different parts, such as area versus timing, can be made to achieve the best overall performance.

In general, microprocessor implementation can be divided into three different activities: data path design, control logic design, and on-chip local memory design. All these parts are different and require different methodologies as well as different design styles. Alternatives and optimization techniques for the data path part of the microprocessor have been studied

extensively in many past research projects. On-chip memory design has become an important issue since i/o communication bottleneck of a single-chip processor can be significantly improved by having an on-chip local memory. As more chip area is devoted to the local memory (some microprocessors have more memory than other logic, e.g. TI Lisp chip [Bos87]) and many different memory organizations emerges, a separate (from data path design) consideration is required for on-chip memory designs. The research presented in this thesis addresses two separate issues regarding on-chip memory design and control logic design parts of the full-custom VLSI microprocessor.

1.4. Related Work

The research presented in Chapter 2 on the SPUR CPU chip is not a one-person project. Several graduate students have worked on various aspects of the research. The instruction set architecture of the SPUR CPU was defined by George S. Taylor [THL86], and the microarchitecture design was refined by Shing I. Kong [Kon89]. Mark D. Hill contributed in the design of on-chip instruction cache [Hil87]. Most of the work presented in Chapter 2 of this thesis is in the area I have participated in most, VLSI chip design and implementation.

Many papers have discussed on-chip memory design at both architecture and implementation aspects; these include [Goo83], [His84], [ACH87], [EiP88], [GoH86], and [Kad82]. Most concentrate on architectural design issues such as register versus cache and organization of on-chip caches. Agarwal et al. present the importance of the tradeoffs between on-chip cache architecture and implementation [ACH87]. They show that for on-chip caches other considerations besides hit rate are important. These include the total usable area, the timing of cache accesses, the physical organization of the cache, and the aspect ratio of the resulting design.

Two recent papers discuss using dynamic memory for on-chip local memories [Tra85][Bos87]. Tran presents a successful integration of high density 1T DRAM in digital signal processor chip [Tra85]. Bosshart et al. also present a memory intensive

microprocessor chip built for LISP processing [Bos87]. Over eighty percent of this chip is used to implement memories, including RAM's made of 4T dynamic cells. The DRAM's refresh when they are not required for other operations. A master refresh timer is also provided to enforce extra refresh cycles in case that there is any entry not refreshed. Several multi-port memory cells for on-chip memories have been proposed in [She84], [Kad82], [O'C87], [DiS79], and [Nak88]. The analysis of these cells is further carried out in [SLL87], [O'C87], and [Nak88].

The efficiency of two general approaches (microprogrammed and hard-wired) for designing the control unit of a VLSI microprocessor is examined in [Anc83]. By re-implementing the control unit of MC6800, this research shows that the hard-wired approach always gives minimum area but its design cost increases too rapidly with increasing complexity. The aim of reducing the design cost may lead designers to choose design styles less optimal in terms of silicon area but which use more regular structure.

For alternative control design using hard-wired logic, Hoffman compares the multi-level logic implemented in array structured logic (a CMOS extension of Weinberger array using domino logic), and a two-level PLA implementation [Hof85]. The control logic of the CMOS SOAR (SmallTalk on A RISC [Pen87]) microprocessor chip is used as the basis of comparison. He shows that a multi-level logic implemented in array form is faster and smaller than a PLA version of the same logic. The impact of library size on the quality of automated logic synthesis is investigated in [Keu87]. It concludes that an incrementally larger library size can considerably reduce area while meeting comparable timing requirements.

1.5. Thesis Organization

The main body of the thesis consists of three main chapters (Chapters 2, 3 and 4) which are written as stand-alone discussions rather than as tightly integrated parts of a whole. For this reason, the chapters can be read separately, in any order.

Chapter 2, *VLSI Implementation of the SPUR CPU Chip*, presents the design of a VLSI microprocessor chip for a multiprocessor workstation, called SPUR. The central processing unit (CPU) of the SPUR processor supports a multilevel cache scheme that includes a pre-fetching on-chip instruction cache, a coprocessor interface, and a support for a fast execution of LISP through a tagged 40-bit architecture [Hil86]. In addition to describing the implementation details of each part, an overall methodology is also presented. In order to build a working computer system based on the SPUR CPU chip, a reliable and efficient methodology was indispensable.

The research presented in the next two chapters (Chapter 3 and 4) is developed from the implementation of the SPUR CPU. New ideas and better alternatives have become available since the SPUR CPU, from the lessons learned in the SPUR design and the newly developed CAD tools. These must be examined rigorously to be useful for improving the performance of the next generation microprocessors. New and better alternatives in VLSI microprocessor design are presented using examples from the SPUR CPU chip, and comparisons are made to determine the effectiveness of the alternative.

Chapter 3, *On-chip Memory Design*, presents new techniques for on-chip local memory designs. Simple circuit design techniques, when properly adapted to the architectural design, can provide a cost-effective performance improvement. Using the selective invalidation scheme implemented with low overhead circuits can eliminate the refreshing requirement of dynamic memory, if used as a read-only or write-through cache. Using this scheme, a static memory can be replaced with a high density dynamic memory without performance or reliability degradations. Parallel execution of programs using more than one functional unit is an effective approach to increasing the processor performance [PIS88]. The bandwidth required by multiple functional units demands a high bandwidth fast memory with multiple ports. A single-ended static memory cell operated at reduced voltage levels can be as safe and fast as other multiport memory while consuming much less area. Using circuit simulations and static noise margin analysis [SLL87], it is determined to be feasible to implement multi-port

memory based on this cell.

Chapter 4, *Control Design Alternatives*, discusses the alternatives available for designing the control portion of the microprocessor. A common approach to regularizing the design of random control logic employs a structured logic element, such as PLAs and microcode ROMs, to implement the microprocessor's control. Emerging CAD tools, especially in multi-level logic synthesis and optimization [NeS86][Bra87], now allow a combinatorial portion of control logic to be mapped into different design styles such as standard cell-based design, which have not been well utilized in full-custom VLSI designs. In this chapter, I examine these alternative design styles by re-implementing the control units from the SPUR chips, and contrasting them with the full-custom version (with only PLA synthesis tools) available also from the SPUR designs.

Finally, Chapter 5 concludes the thesis and provides a summary of the research and future work.

1.6. References

- [ACH87] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz and J. Hennessy, "On-chip Instruction Caches for High Performance Processors", *Proc. of Conf. on Advanced Research in VLSI*, Stanford, CA, March 1987.
- [Anc83] F. Anceau, "VLSI-Processor Architecture and Design", in *VLSI Architecture*, Prentice Hall International, Englewood Cliff, NJ, 1983.
- [Bos87] P. Bosshart et al., "A 533K-Transistor LISP Processor Chip", *IEEE Journal of Solid-State Circuits* SC-22, 5 (October 1987).
- [Bra87] R. K. Brayton et al., "MIS: A Multi-Level Logic Optimization System", *IEEE Transaction on CAD of Integrated Circuits and Systems* CAD-6, 6 (November 1987), 1062-1081.

- [DiS79] A. G. F. Dingwall and R. G. Stewart, "16K CMOS/SOS Asynchronous Static RAM", *IEEE Journal of Solid-State Circuits SC-14*, 5 (October 1979).
- [EiP88] R. J. Eickenmeyer and J. H. Patel, "Performance Evaluation of On-Chip Register and Cache Organizations", *Proc. 15th International Symposium on Computer Architecture*, Honolulu HI, June 1988, 64-72.
- [Goo83] J. R. Goodman, "Using Cache Memory to Reduce Processor Memory Traffic", *Proc. Tenth International Symposium on Computer Architecture*, Stockholm, Sweden, June 1983.
- [GoH86] J. R. Goodman and W. Hsu, "On the use of Registers vs. Cache to Minimize Memory Traffic", *Proc. 13th International Symposium on Computer Architecture*, Tokyo, Japan, June 1986, 375-384.
- [HiS84] M. D. Hill and A. J. Smith, "Experimental Evaluation of On-Chip Microprocessor Cache Memories", *Proc. Eleventh International Symposium on Computer Architecture*, Ann Arbor, MI, June 1984.
- [HiI87] M. D. Hill, "Aspects of Cache Memory and Instruction Buffer Performance", Ph.D. Thesis, CS Division EECS Dept., UC Berkeley, November 1987.
- [HiI86] M. D. Hill et al., "Design Decisions in SPUR", *IEEE Computer* 19, 10 (November, 1986), 8-24.
- [Hof85] M. E. Hoffman, "Automated Synthesis of Multi-Level Combinational Logic in CMOS Technology", Ph.D. Thesis, EECS Dept., University of California, Berkeley, July 1985.
- [Kad82] H. Kadota et al., "A New Register File Structure for the High-speed Microprocessor", *IEEE Journal of Solid State Circuits sc-17*, 5 (October 1982).
- [Keu87] K. Keutzer et al., "Impact of Library Size on the Quality of Automated Synthesis", *IEEE ICCAD Digest of Technical Papers*, November 1987, 120-123.
- [Kon89] S. I. Kong, "Performance, Resources, and Complexity: A Systematic Approach to Microarchitectural Design", Ph.D. Thesis, EECS Dept., University of California,

Berkeley, May 1989.

- [MeC80] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
- [Nak88] Y. Nakagome, "Multiport Memory Design Consideration for Parallel Execution CPU Architectures", Electronics Research Lab. Technical Report, Department of EECS, U.C. Berkeley, November 1988.
- [NeS86] A. R. Newton and A. L. Sangiovanni-Vincentelli, "Computer-Aided Design for VLSI Circuits", *IEEE Computer* 19, 4 (April 1986), 38-60.
- [O'C87] K. J. O'Connor, "The Twin-port Memory Cell", *IEEE Journal of Solid State Circuits* sc-22, 5 (October 1987), 712-720.
- [Pen87] J. M. Pendleton et al., "A 32-bit Microprocessor for Smalltalk", *IEEE J. Solid-State Circuits* SC-21, 5 (October, 1987), 741-749.
- [PIS88] A. R. Pleszkun and G. S. Sohi, "The performance Potential of Multiple Functional Unit Processors", *Proc. 15th International Symposium on Computer Architecture*, Honolulu, HI, June 1988.
- [SLL87] E. Seevinck, F. J. List and J. Lohstroh, "Static-Noise Margin Analysis of MOS SRAM cells", *IEEE Journal of Solid State Circuits* sc-22, 5 (October 1987), 748-754.
- [She84] R. Sherburne et al., "A 32-bit Microprocessor with a Large Register File", *IEEE Journal of Solid-State Circuits* SC-19, 5 (October, 1984).
- [THL86] G. S. Taylor, P. N. Hilfinger, J. R. Larus, D. A. Patterson and B. G. Zorn, "Evaluation of the SPUR Lisp Architecture", *Proc. Thirteenth International Symposium on Computer Architecture*, Tokyo, Japan, June 1986.
- [Tra85] L. V. Tran, "A 32K Bit High-speed On-chip DRAM for Digital Signal Processor", *IEEE Custom Integrated Circuits Conference*, 1985, 166-169.

[WeA85] N. Weste and K. Ashraghian, *Principles of CMOS VLSI Design: A System Perspective*, Addison-Wesley, Reading, MA, 1985.



2

VLSI Implementation of the SPUR CPU Chip

2.1. Introduction

SPUR¹ (Symbolic Processing Using RISCs) is a multiprocessor workstation developed at the University of California at Berkeley as a testbed for research on parallel processing, particularly in LISP [Hil86]. A SPUR workstation, shown in Figure 2-1, can have 6 to 12 identical processors, each of which consists of a 128K-byte cache, a CPU, a floating point coprocessor, and a cache control and memory management unit (CMU) that assures the cache coherency among multiple processors. The picture of a fully populated SPUR processor board is shown in Figure 2-2. This chapter describes the VLSI implementation of the CPU chip, a 32-bit RISC microprocessor.

The SPUR CPU supports a multilevel cache scheme that includes a prefetching on-chip instruction cache, a coprocessor interface, and support for fast execution of LISP through a

¹SPUR is sponsored by DARPA under contract order 482427-25840, California MICRO, Texas Instruments, National Semiconductor, Cypress Semiconductor, Tektronix, and HP.

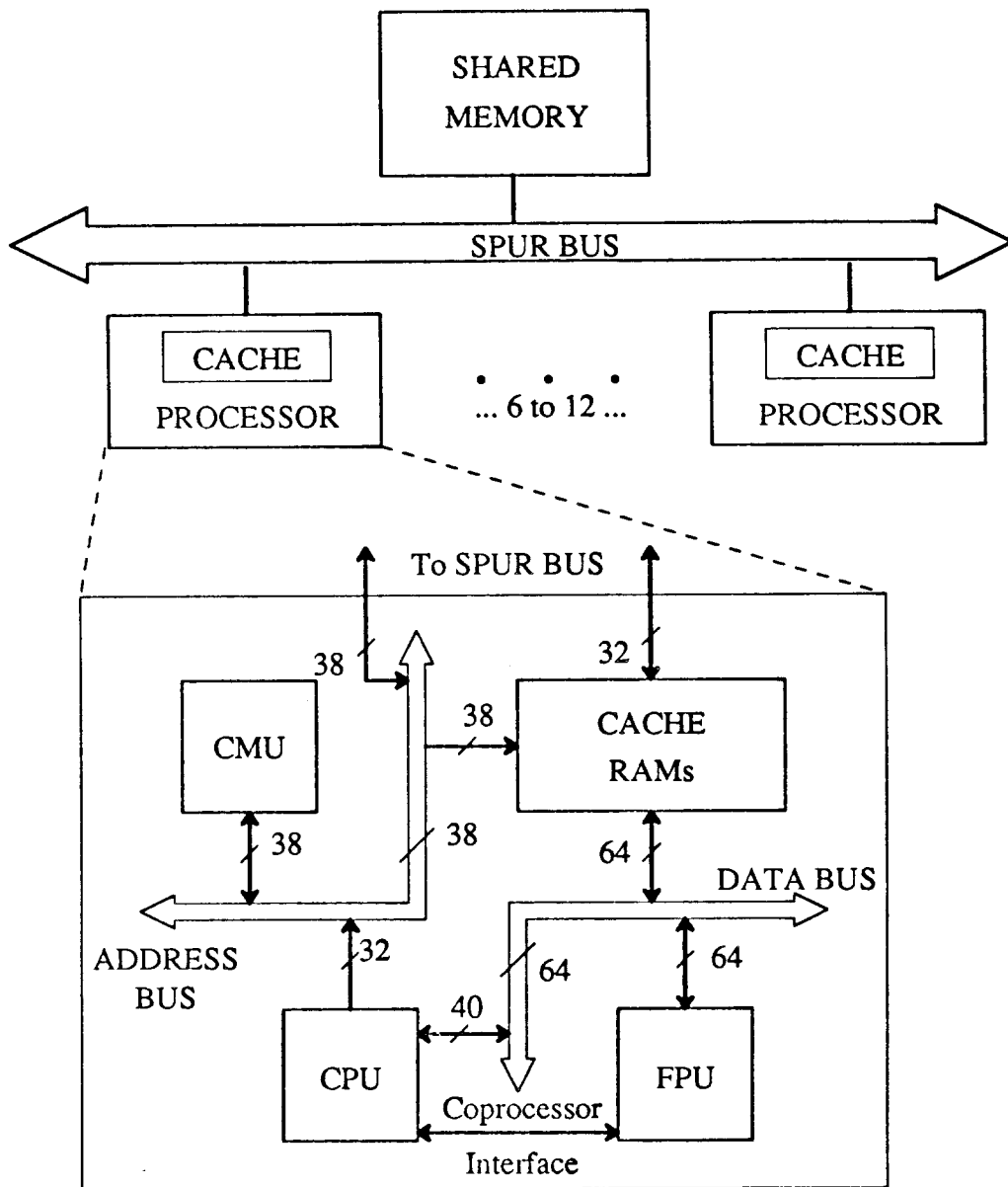


Figure 2-1. SPUR multiprocessor workstation

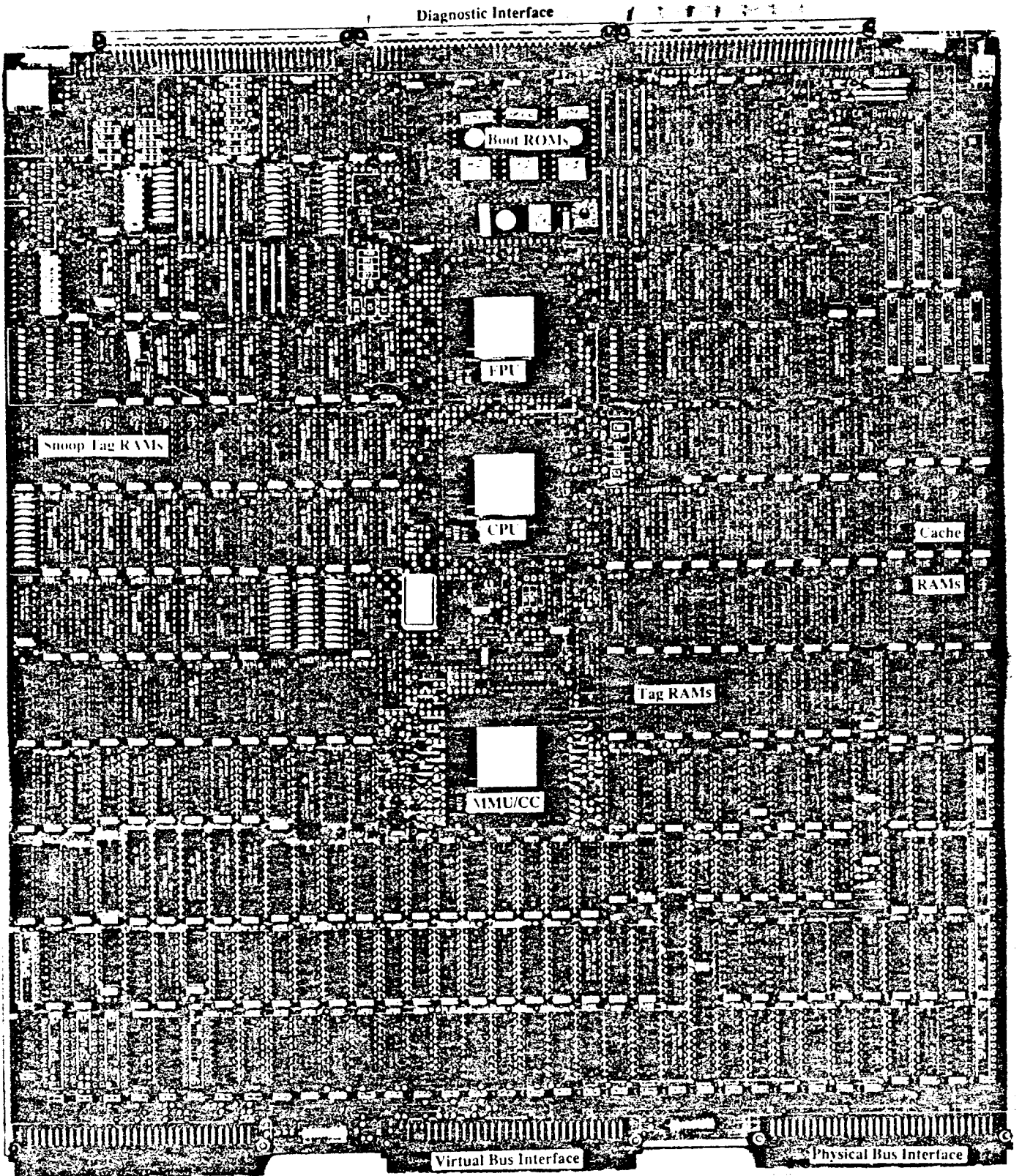


Figure 2-2. A SPUR processor board

tagged 40 bit architecture. The coprocessor interface supports concurrent CPU and FPU operations. It uses 27 pins to implement a low-overhead interface between the CPU and the FPU.

The chip, implemented in 1.6 μm , double metal CMOS technology, contains 115K transistors. The chip statistics are summarized in Table 2-1, and a chip photomicrograph is shown in Figure 2-3. An on-chip clock generator, based on a charge pump phase-locked loop with tapped delay line, provides accurate phase relationship with the board clock and also with clock phases of the other chips [Jeo87]. Nominal operating frequency with a 4-phase non-overlapping clock (18 nsec nominal per phase and 7 nsec non-overlap time between phases) is 10 MHz (12.5 MHz Max). A SPUR uniprocessor running LISP programs (Gabriel benchmarks) at 10 MHz can provide 2X performance improvement on the average over the Symbolics 3600 or VAX 8650, according to simulation [THL86]. A SPUR workstation with 6 to 12 processors is predicted to yield a sustained throughput of 40 to 70 MIPS, respectively.

Number of Transistors	115,214
Number of PLA's	13
Die Size	11.5mm x 11.5mm
Package	208-pin PGA (40 pins for power supply)
Process	Double-Metal 1.6 μm N-Well CMOS
Operating Frequency	10MHz
Power Dissipation	0.8W at 10MHz with 5V Supply

Table 2-1. Chip Statistics

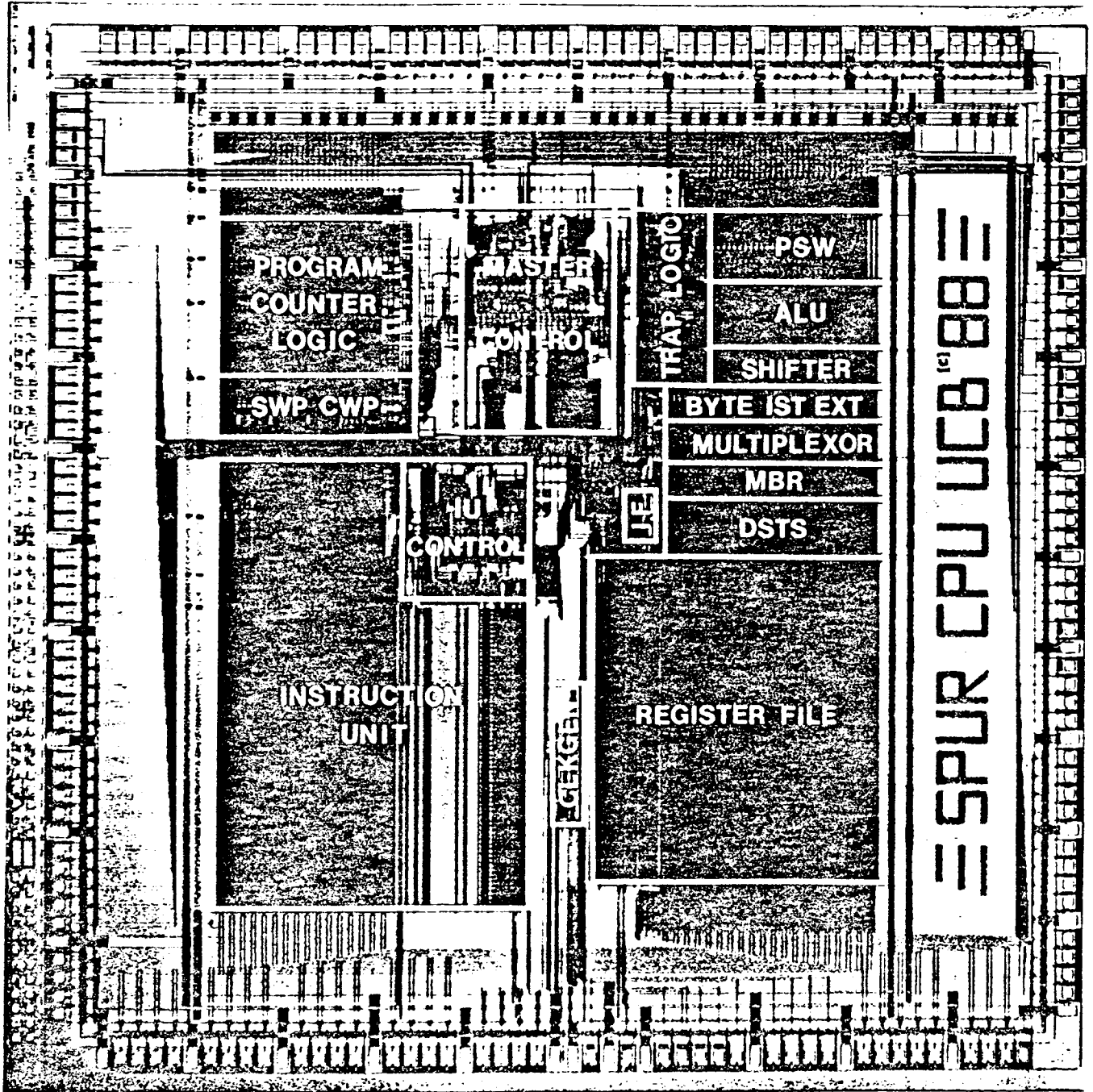


Figure 2-3. Chip microphotograph

The organization of the chapter is as follows: Section II gives an overview of the CPU architecture and execution pipeline. Section III focuses on the hardware required to implement various features of the SPUR CPU architecture. Section IV describes the design, verification, and testing methodologies of the full custom SPUR CPU chip. Finally, the summary and conclusion are given in section V.

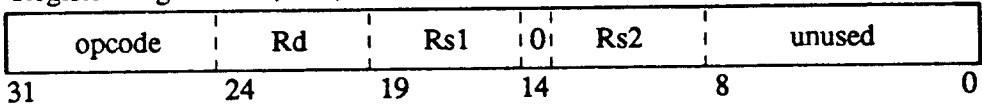
2.2. An Overview of the SPUR CPU Architecture

The SPUR CPU is a third-generation RISC microprocessor developed at the University of California at Berkeley. It is specifically designed to be used in the SPUR multiprocessor workstation. The architecture of the SPUR CPU is akin to those of previous RISC projects at U.C. Berkeley [Kat83], [Pen87]. Some new features, however, have been added: a coprocessor interface to support floating-point computation, an efficient interface to the cache-control and memory-management unit, and run-time hardware tag checking for fast execution of LISP programs. The instruction set of the SPUR CPU is carefully chosen such that an efficient implementation of the single-cycle execution of all instructions is possible.

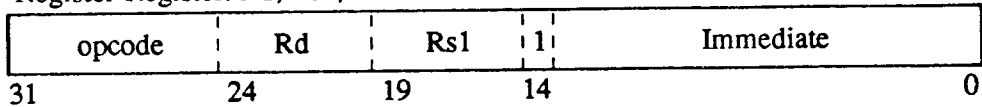
Like previous RISC processors, the SPUR CPU is a load-store machine. Memory is accessed only through load and store instructions. All other instructions are register-to-register or immediate-to-register oriented. There are four generic instruction types: register-to-register, store, compare-and-branch, and call-jump. Load and return instructions are special cases of register-to-register in which $(R_{s1} + R_{s2})$ or $(R_{s1} + Immediate)$ is used as an effective address. The R_d field specifies the register to be loaded for the load instruction type and is not used for the return instruction type. All instructions (40 integer and 20 floating point) are 32-bits wide and use fixed formats. The seven instruction formats are shown in Figure 2-4. The opcode and the register specifiers are in the same positions in all formats. The three-register format (RRR) is used for loads, register-to-register operations, special register operations and coprocessor operations. The two-register and one-immediate (RRI) is used for loads and register-to-register operations. Compare-and-branch instructions have three slightly dif-

ferent formats depending on the field specifying the condition.

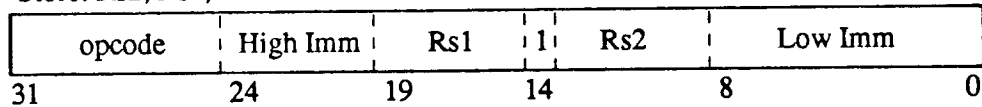
Register-Register: Rd, Rs1, Rs2



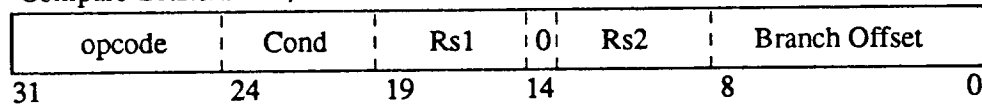
Register-Register: Rd, Rs1, Immediate



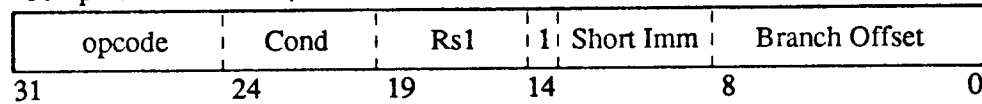
Store: Rs2, Rs1, Immediate



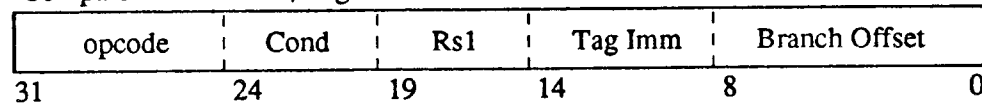
Compare-Branch: Rs1, Rs2



Compare-Branch: Rs1, Short Imm



Compare-Branch: Rs1, Tag Imm



Call, Jump: Word Address

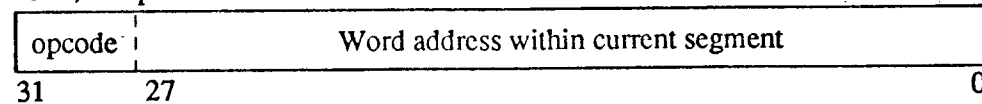


Figure 2-4. SPUR instruction formats

The CPU registers are organized in eight overlapped windows (128 registers) and 10 global registers accessible from any window (total 32 registers visible from one window). The overlapped window scheme considerably reduces the register save and restore overheads between procedure calls. The registers are 40-bit registers with 32 bits for data and an 8-bit tag used for runtime type checking and garbage collection. The 8-bit tag consists of a 6-bit object's type tag and a 2-bit generation numbers. LISP is supported with three types of hardware tag checking with traps to a software trap handler: data type checking for general computations, pointer type checking for list operations, and generation number checking for garbage collection based on the generation scavenging algorithm [Ung84].

The on-chip instruction cache provides the effect of an extra memory port, allowing simultaneous data memory reference and instruction fetch by the execution unit (EU). This leads to a four-stage pipeline (Figure 2-5) that eliminates the need for pipeline stalling whenever a load instruction is executed. Consequently, the CPU can issue and complete one instruction per cycle (peak performance rate of 10 MIPS per processor) as long as there are no instruction or external data cache misses. Branch conflict in the pipeline is resolved by a single cycle delayed branch with one instruction in the delayed slot. Data conflicts are resolved by hardware internal forwarding logic.

In order to facilitate the high-precision floating point computations and other possible coprocessing capabilities, the SPUR CPU incorporates a parallel interface to coprocessors. The floating-point coprocessor interface implemented in the current version of the CPU chip supports concurrent CPU and FPU operations. It uses 27 pins to implement a low-overhead interface between the CPU and the FPU. The FPU tracks CPU instructions issued by the instruction cache in the CPU via 22 pins carrying opcode and register specifiers. The CPU sends 2 control signals to the FPU, and the 3-bit FPU status is sent to the CPU. The CPU treats all FPU instructions as illegal instructions when the FPU is disabled. When the FPU is enabled, all FPU instructions except FPU load and store are treated by the CPU as NO_OP. For FPU load and store, the CPU computes the effective memory address and the FPU reads

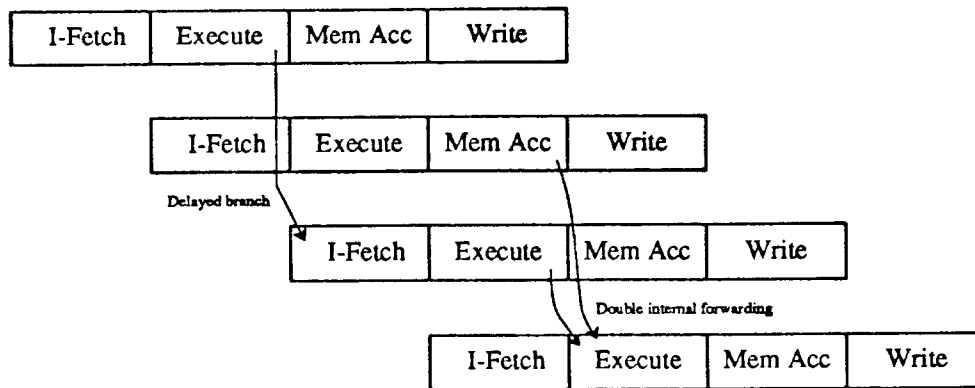


Figure 2-5. SPUR CPU pipeline

and writes the data directly from the external cache.

In the SPUR instruction set, a number of special load (7) and store (3) instructions are dedicated to cache control and virtual memory management. Although these instructions look almost identical to the CPU, appropriate cache operations are provided to the external CMU through the CMU interface. The interface consists of a 4-bit cache-opcode, two bits indicating the mode of operations (user vs. kernel and physical vs. virtual), and 9 other status bits of both the CPU and the CMU.

The unusual conditions that the CPU may face at runtime can be divided into four groups. Unusual conditions detected inside the CPU are called CPU exceptions: integer overflow, tag checking, window overflow and underflow, and so on. Unusual conditions caused by the FPU are called floating-point exceptions. All other unusual conditions occurring outside the CPU are called faults and interrupts. Faults occur in response to the execution of an instruction, while interrupts are asynchronous events that come from outside the processor (e.g. an i/o interrupt). The CPU responds to exceptions, faults, and interrupts by taking a vectored trap. The trap vector consists of a trap base address concatenated with the

trap type field. There is a priority ordering for cases when more than one unusual condition occurs at the same time. All traps are taken during an instruction's third pipeline stage, and hence only one instruction can cause a trap in any cycle. Traps can be disabled or enabled selectively by controlling the 8 bits in both kernel and user processor status words (KPSW and UPSW).

2.3. Hardware Implementation of the SPUR CPU

The major functional blocks are shown in Figure 2-6 and outlined in the chip photomicrograph (Figure 2-3). The major blocks are the execution unit (EU) and the instruction unit (IU). The EU is further divided into the upper data path, the lower data path, and the control. The 30-bit upper data path contains pipelined program counters and special registers. It is used for instruction address calculations and special register references. The 40-bit lower data path is for general computation on the tagged registers.

2.3.1. The instruction unit

The SPUR IU consists of a 512-byte (128 instructions) direct-mapped (16 blocks with 8 subblocks or 8 instructions per block) instruction cache. A novel feature of the SPUR instruction cache is a valid bit associated with each instruction word in the cache so that any subset of instructions within a block may be valid. The SPUR IU uses this flexibility to reduce demand miss time by loading only the fetched instruction rather than the entire block and to permit instruction prefetching to load the rest of a block in parallel with subsequent instruction fetches [Goo83], [HiS84]. If subsequent prefetches are successful, the miss penalty is just two cycles for the entire block containing the missed instruction.

The IU can operate in three different modes: (1) disabled, (2) enabled-without-prefetching, and (3) enabled-with-prefetching, controlled by two bits in the kernel processor status word (KPSW). In disabled mode, the IU fetches every instruction requested by the EU from the external cache. Disabled mode is useful for initial chip testing and for allowing

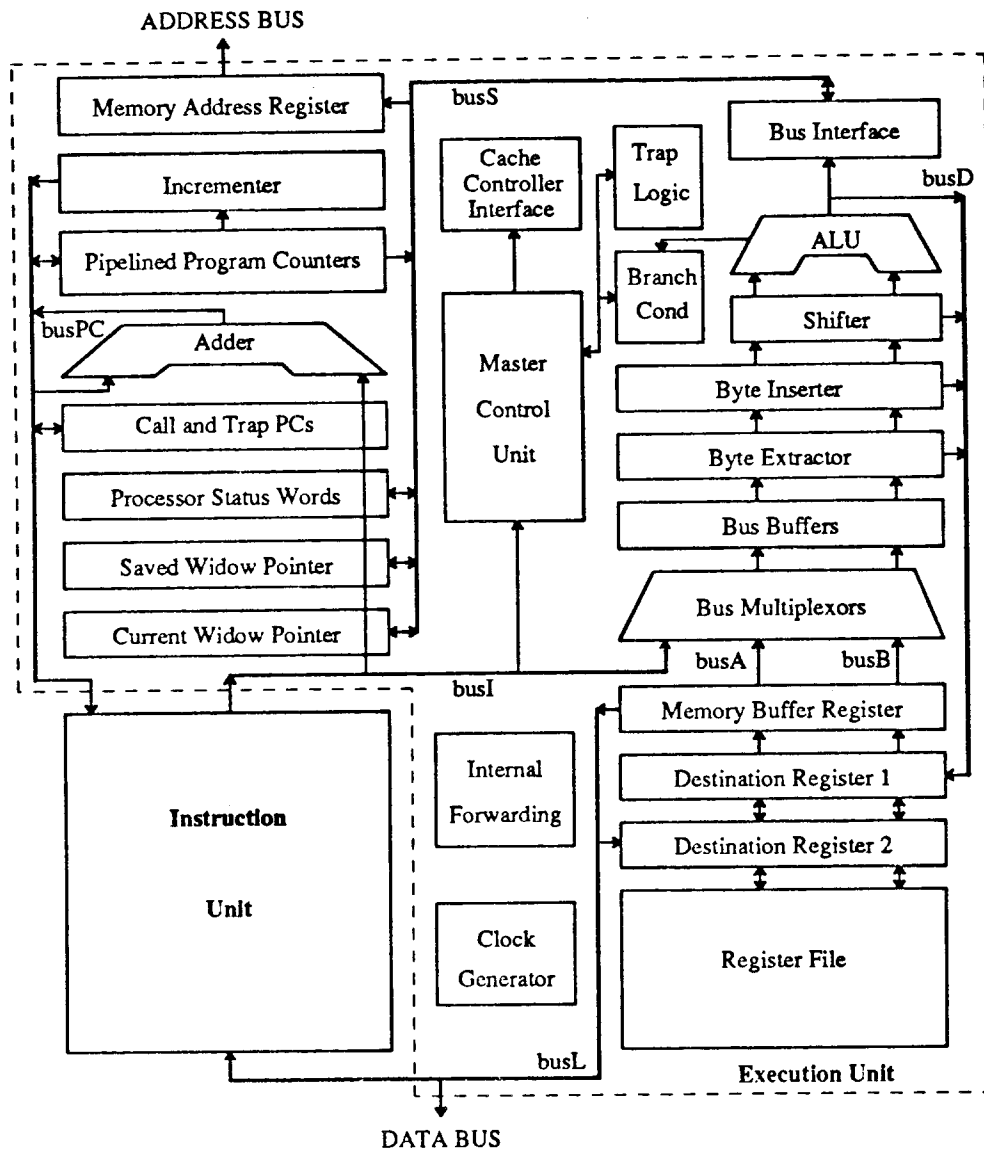


Figure 2-6. SPUR CPU block diagram

chips with stuck-at-type errors in the cache or tag array to function correctly, albeit more slowly. In enabled-without-prefetching mode, the IU will cache instructions upon demand misses but will not initiate any prefetches.

The normal mode is the enabled-with-prefetching. After the missed instruction is cached, prefetches are made to subsequent words within the block until another demand miss occurs or prefetch is blocked by the EU's external data access. These prefetches are "free", as they never interfere with external cache accesses, such as instruction fetch and external data reference, by the EU, because prefetch has the lowest priority. If prefetch causes an external cache miss, the cache controller simply ignores the request.

The instruction unit is controlled by two finite-state machines: one controls the fetching and the other controls the prefetching of instructions. Two finite state machines and other random control logic are partitioned into 6 PLAs, considering the timing constraints. Both IU and register file use the same 6T SRAM memory cell [She84]. The data portion of the cache is an array of 128 33-bit words. The tags are stored in a separate array (16 24-bit words) whose access time is significantly less than that of the data array. This allows the tag comparison to be done while the instruction is being read out from the data array. Bitwise comparison using an XOR gate is used for tag comparison and is followed by dynamic logic to determine a hit. The effective access time of the instruction cache including hit logic is under 12 nsec without using a sense amplifier.

2.3.2. The execution unit

Key features in the execution unit are a register file with eight overlapped windows, double internal forwarding for resolving register access conflicts, and run-time tag checking with traps to software on mismatch. The SPUR CPU has a 30-bit branch address adder in the upper data path, which together with the 32-bit ALU in the lower data path support one-cycle compare-and-branch type instructions. Rather than a complex barrel shifter, a combination of a byte extractor/insertor and a simple shifter is implemented.

A. The register file

The SPUR CPU has a total of 138 general-purpose registers organized in 8 overlapped windows and 10 global registers. Thirty-two registers are visible to the compiler at any one time: 10 globals, 10 locals, 6 overlapped with caller window, and 6 overlapped with callee window. Each register is 40 bits wide having a 6-bit tag, 2 bits for generation number and 32 bits for data. The same 6T SRAM cell used in the IU is used in the register file. The layout of SRAM cell is constrained by the pitches of the data path bit slice and the register decoders (two decoders per register). The result is a large but fast SRAM cell that does not require a sense amplifier.

The SPUR CPU architecture is register oriented and requires two reads and one write per cycle. The register access is time multiplexed for the separate reads and the write and is pipelined to minimize the critical path. Bit lines are decoded and precharged in the same phase, and the register array is accessed in the following phase by driving the wordline. The access time of the register file read is the critical path of the chip. It is measured to be under 14 nsec. For registers in the overlapped window, a special decoder shown in Figure 2-7 is used to map two different register addresses (one from the caller's window and the other from the callee's window) to one register [Kat83].

In the pipelined execution of the instruction stream, data interdependencies among instructions in the pipeline may arise. In the SPUR CPU, these interdependencies are detected and resolved by the hardware internal forwarding. That is, the results from preceding instructions are forwarded to the following instructions by the hardware before being written back to the register file, as indicated by the arrows in Figure 2-5. In the case of a 4-stage pipeline like the SPUR CPU, the data interdependencies may exist among 3 consecutive instructions since the write-back stage of the pipeline is delayed by two cycles after the execution stage. The result available from each instruction's execution stage, therefore, needs to be stored in temporary registers for two cycles and then forwarded to the following instructions. When both operands are registers, each register address is compared to the destination register address of the two preceding instructions. This may result in double internal forwarding, in that both

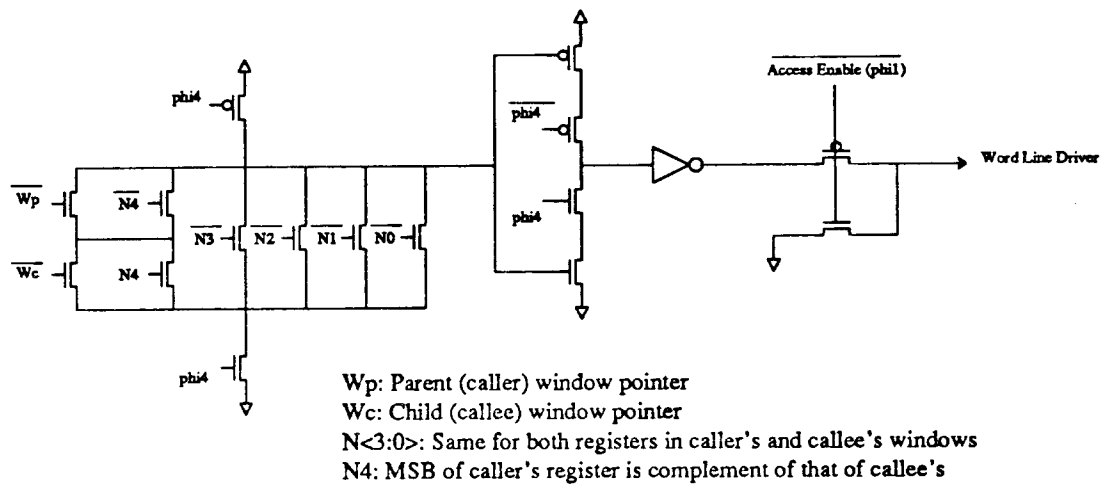


Figure 2-7. Overlapped window register decoder

operands are results of two preceding instructions and hence supplied from the temporary registers.

The hardware internal forwarding logic is in the critical path of the register file access, and it must be implemented without slowing down the cycle time. Like decoding and accessing the register array, it also is pipelined. Address comparisons are done in parallel with the decoding of the register file, and internal forwardings are made if necessary while the register file is accessed. Four address comparisons are necessary to detect all possible data dependencies. The address comparator must be fast to keep the cycle time short, and it must be compact to fit in the area between register decoders and temporary registers, as seen in Figure 2-3 (block IF). Bitwise comparison is done using a dynamic XOR, shown in Figure 2-8, and then the outputs are fed into the domino circuit for an address match. Since this XOR does not require complementary inputs, routing and area required are significantly reduced. A special multiplexor, shown in Figure 2-8, is used to minimize the signal delay through the internal forwarding logic that lies between the register file and the functional unit. If internal forward-

ing is necessary, the bus from the register file is disconnected by the transmission gate, and the bus to the functional unit is driven by the temporary register. The access time of register file reading (14 nsec) includes the delay through the internal forwarding logic.

B. The data path

The data path is divided into two parts: the upper data path for program counter logic and special registers, and the lower data path for general computations on tagged registers. Functional units in the lower data path include a byte-extractor, a byte-inserter, a simple

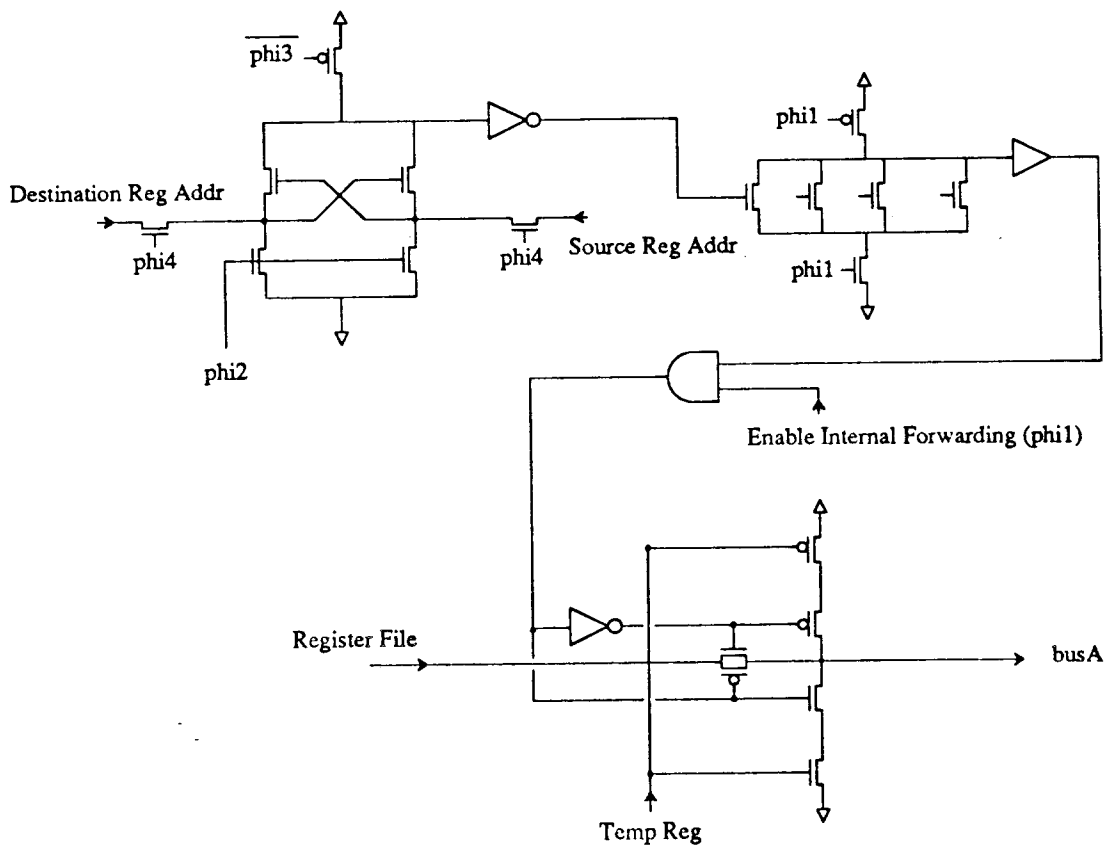


Figure 2-8. Internal forwarding logic

shifter that shifts up to three bits, and an ALU. The ALU provides XOR, OR, AND, ADD, and SUBTRACT operations and comparison for two 32-bit operands. The upper data path consists of a number of program counters to hold instruction addresses in the pipeline, an address incrementer and adder, and special registers such as window pointers and processor status words. All registers and counters are made of pseudo-static latches, such that each register is refreshed once per cycle. This is necessary because an indefinite pipeline stall is possible due to a long external cache miss.

In the SPUR CPU, compare-and-branch instructions are executed in only one cycle. A separate adder in the upper data path calculates the target addresses for all the compare-and-branch instructions while the ALU is in use for the comparison. Two different adder designs are employed. The 32-bit ALU uses four 8-bit carry lookahead adders implemented in domino logic, and evaluates the carry within 11 nsec. The 30-bit address adder is more compact because it uses a Manchester carry chain which has a carry propagation delay of 13.5 nsec.

The upper 8-bit slices of the lower data path are for tag-related operations. Operations on the tag and the data are logically independent, that is, no information moves between the two parts by carry propagation or any other implicit mechanism. For operations, the 6-bit tag type is checked in parallel with the data operation. If there is a tag mismatch and the tag trap enable bit is set in the user processor status word (UPSW), the CPU traps to the software. Generation tag checking (2 MSB) is done when a special store instruction ($ST_{40} R_{S1}, R_{S2}, Immediate$) is executed. Generation tag exception may occur if the object (R_{S2}) with a higher (younger) generation number is stored into the object (R_{S1}) with a lower generation number [Ung84]. The `read_tag` and `write_tag` instructions move a tag to and from the data portion of a register using the byte-extractor and the byte-inserter respectively, so that any arithmetic or logical operations may be performed on it.

To reduce the chip area and improve the circuit speed, the dynamic circuit technique called domino logic [KLL82] is heavily used in the design. Potential charge sharing problems are prevented either by the use of abundant clock phases or by careful layout of the critical

nodes. The SPUR CPU has 7 major busses to provide communications both externally and internally. Some of these busses have high capacitive loadings, and hence precharging is used to improve the speed of data flow through the highly capacitive busses. The high capacitance bus is precharged to high before being used and discharged conditionally through a strong NMOS pull down network when used. This not only reduces the signal delay through the bus but also minimizes the chip area required for a strong, large driver. Some logic function may be included in the pull-down network as well, further saving the chip area. Critical paths of the data path, register file, and instruction cache are summarized in Table 2-2.

C. The control

Four-phase clocking and a uniform four-stage pipeline for all SPUR integer instructions make the control section of the CPU relatively simple. The SPUR CPU uses internal instructions to handle pipeline interrupts, rather than requiring complex sequences for those exceptions. These internal instructions are *miss*, *trap_call*, and *read_pc* to handle instruction cache miss and all kinds of traps. These instructions are executed in the same way other instructions

phase	operation	critical path (nsec)
phi1	Register file - read	14.0
phi2	Instruction Cache - fetch	12.0
phi3	ALU - 32b carry propagation	11.0
phi4	Address adder - 30b carry propagation	13.5

Table 2-2. Critical path timing

are executed. The use of these instructions further simplifies the control design.

The control can be divided into three parts: master control, trap logic, and the interface to the cache control/memory management unit (CMU). The latter two are separated out from the master control to simplify the control design. Trap logic detects all unusual conditions during the pipelined execution of an instruction. All traps are taken during an instruction's third pipeline stage, and hence only one instruction can cause a trap in any cycle. The trap logic consists of pipelined modules, each of which operates at the corresponding stage of the instruction in the pipeline. The CMU interface logic generates cache opcodes according to the current instruction and the status of the CPU. It also buffers signals to and from the CMU.

The block diagram of the master control is shown in Figure 2-9. A centralized master control unit controls the processor sequencing and decodes the opcode into high level control signals. Local random logic blocks then decode the high level signals into low level signals using clocks. They also provide buffering of the low level signal according to the loading requirement. All signals controlling the data path are individually optimized so as to have equal delays relative to the clock edges. The separation of master control and local decoding/buffering significantly reduces the amount of routing between two sides, particularly in CMOS design where complementary signals are required in controlling the data path.

Most of the control logic in the SPUR CPU is implemented in static PLAs. The largest PLA is the one that decodes the opcode, which has 69 product terms with 40 outputs. The propagation delay through this PLA is about 15 nsec, well below the required timing of two phases or 50 nsec. All PLA outputs are evaluated once per cycle and need to be held in registers until the next cycle. The routing between the PLA and the registers may consume substantial chip area since the PLA output pitch is so small compared to the pitch of the registers. Thus, the registers (pseudo-static latches) are integrated into the output section of the PLA by widening the PLA output pitch (16 lambda to 20 lambda). This results in an unusually large PLA, but the chip area required is much less than if the PLA and the registers were separated, and the timing requirement is still satisfied.

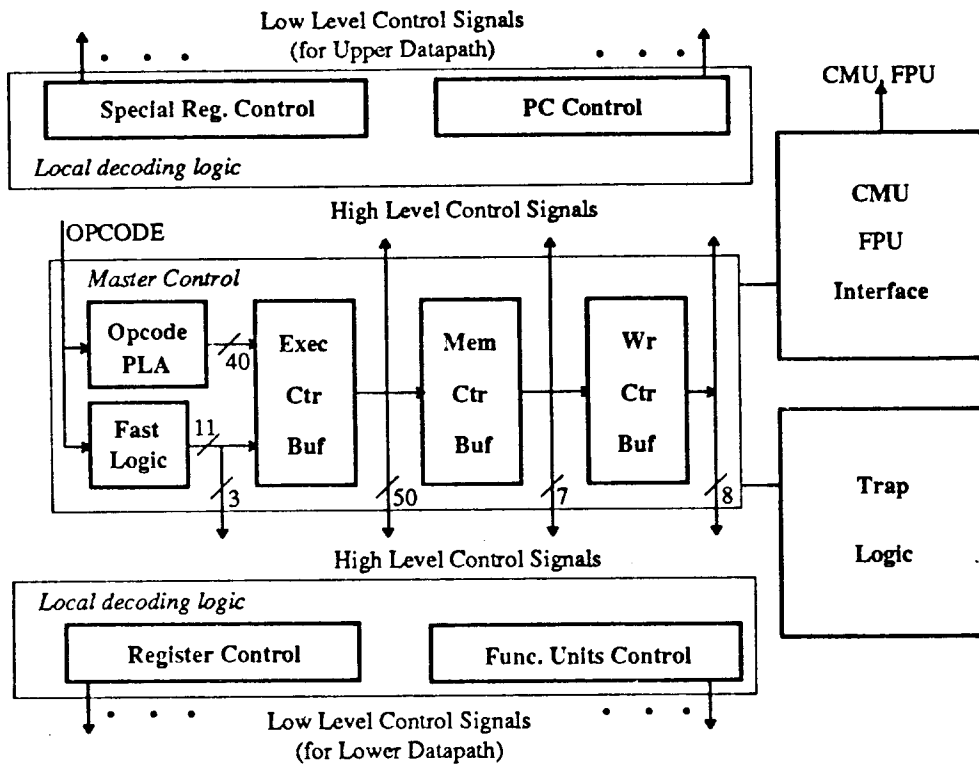


Figure 2-9. Block diagram of master control

2.4. Design, Verification, and Testing Methodology

Methodologies employed in the SPUR CPU design have been influenced by the following two themes of the SPUR project: (1) an overall system-wide rather than local optimization, and (2) designing a chip for a working system rather than an experimental prototype. Consequently, methodologies became very important since the chip being designed must meet all the functional requirements set for the system design as well as performance goals.

2.4.1. Design methodology

The design strategy incorporated both top-down and bottom-up approaches. The top-down flow was as follows: architecture definition, instruction set design, microarchitecture design, and a detailed functional/behavioral description of the hardware. The bottom-up flow was circuit design of basic components, layout of basic cells, assembly of major blocks using those cells, and global placement and interconnections. Both approaches were taken in parallel from the beginning, to achieve the highest performance at given technology and system design goals. For instance, many microarchitecture decisions were made after the feasibility of a certain hardware resource was carefully considered. Division of design tasks followed the same hierarchical boundaries of design abstractions: architecture and instruction set design, microarchitecture design, and VLSI implementation. One- or two-person groups were formed to take the responsibilities of each design level. Close interaction among different groups was necessary to make clean interfaces among themselves and design specifications.

Most of the CAD tools used in designing the SPUR CPU chip were developed at Berkeley, except those for the behavioral level design. The detailed design started with describing the behavior of the chip and its interactions with other components within the system. The functional behavior was written in ISP', a hardware description language, and simulated using the N.2 simulator [N.2 Simulator.]. The implementation of the hardware can be divided into two parts. Most parts of the control design were done using a set of CAD tools that automatically synthesizes the behavioral description of the combinational logic into the PLA [Seg87], [EEE88]. Other parts of the control logic (sequential) and data paths were designed manually but aided by another set of tools. These two paths are diagramed in Figure 2-10. For the automated synthesis path, only those parts of the hardware description containing combinational logic can be synthesized. For the manual part, logic and circuit design were done first for each block and followed by layout. Layout was done using an interactive layout editor, Magic [Ous85], with background design rule checking and hierarchical extraction. The extracted layout, which is a switch-level description of the chip, was simulated using

bdsim, a switch-level simulator [Seg87].

Timing analysis was done before the layout, to make early tradeoffs among many alternatives and after the layout, to perform an exact timing analysis with all parasitics correctly annotated. To estimate the critical paths of the chip more accurately, and thus to be able to determine the cycle time, a test chip containing a register file with internal forwarding was fabricated and tested [Lee86]. The measured critical path (register file read) was below 18 nsec, and this encouraged us to set the cycle time goal at 100 nsec.

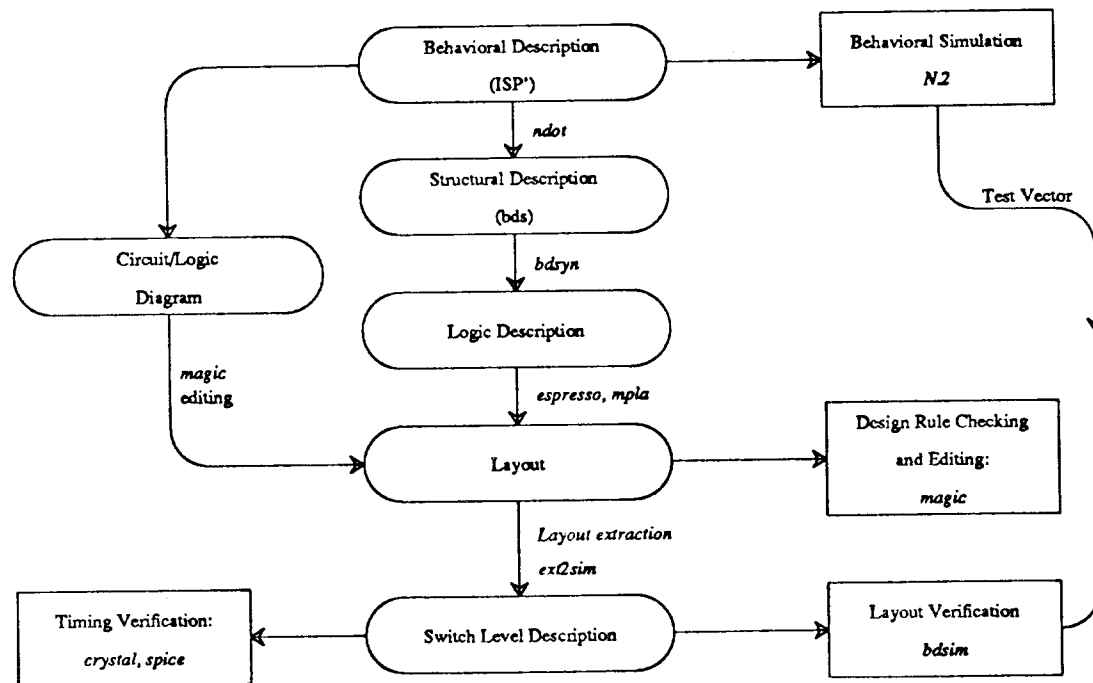


Figure 2-10. Design methodology

2.4.2. Verification methodology

The verification methodology was constructed following a bottom-up approach. As each individual module was designed, switch-level simulation was performed on the extracted layout to verify the design. A small set of hand-written test vectors was used for the simulation. Once individual modules were verified, they were connected and then simulated together until the integration reached the major blocks, the execution unit and the instruction unit. Test vectors up to this point were small and easy to generate by hand, since the test sequences required to verify operations on these units separately were relatively simple. After all major blocks were integrated, the verification effort was directed at both functional and switch levels.

Functional simulations are performed not only on each major component, to verify its internal functions but also on the external system level, to verify interactions among major chip sets. The diagnostics for the functional simulation were coded in SPUR instructions, and an instruction level simulator called *Barb* was used to debug the diagnostics. The diagnostics were intended to be stored in the start-up ROM on the processor board. The N.2 system provided simulated memories that could be used to model ROM or other types of memory. Therefore the diagnostics were assembled and loaded into the simulated memory. When the N.2 simulation was started, it was forced to go through a series of start-up sequences, making the CPU begin fetching instructions from the ROM containing the diagnostics. The diagnostics were then executed to completion or until failure. The same ROM image was used to program the EPROMs on the processor board, to be used for on-board testing of the chip.

Running extensive simulations on the hardware description verified many design ideas and functionalities, but it was still necessary to extract and simulate the layout of the entire chip. The extracted description is almost guaranteed to accurately model the real chip. However, developing the tests and examining the results for a complete switch-level simulation would be very difficult. To minimize the required work, the functional simulation should drive the switch-level simulation with automatically verifying that the two match at every

clock cycle. Fortunately, the N.2 simulator provides a "tracing" capability that logs all changes to a specified set of signals into a file. By tracing all inputs and outputs of an N.2 module, it is possible to obtain a set of switch-level test vectors automatically. These vectors along with expected results on output nodes are fed into the the switch-level simulation. The switch level simulator, *bdsim*, sets the input nodes according to the timing and vectors specified and verifies the output nodes with the expected results. Any unusual condition is recorded so as to be used in debugging.

A problem may arise because functional simulation and switch-level simulation may show different results under unusual states, such as unknown and initial states. For example, the functional simulator initializes all nodes to zero, while all nodes are set to unknowns initially in the switch-level simulation. When the chip is tested neither of these initial conditions is correct. To alleviate the problem, all internal states are initialized explicitly in the functional simulations. In the switch-level simulation, on the other hand, the detailed verifications are made after the initialization is done and all internal states are synchronized with those of functional simulation.

To have a working system rather than a prototype chip, all aspects of the design had to be verified, especially the interfaces to external chips. Table 2-3 summarizes the diagnostic vectors simulated in both functional and switch-level simulations.

2.4.3. Testing methodology

Several features were incorporated into the SPUR CPU chip to increase its testability. Passive scan registers are attached to all major busses to increase the observability. All signals put on these busses can be scanned out for examination. All major blocks are connected and communicate through these busses, so that the diagnostic capability is greatly improved. Many signals, like state bits of finite state machines in IU and the LSBs of the instruction address bus (busPC), are also routed out to pins to determine the exact status of the processor

diagnostics	test vector length (cycles)
CPU functions	13,113 (24%)
CMU interface	16,356 (29%)
FPU interface	1,543 (3%)
Lisp tags and traps	8,675 (16%)
Boot-up diagnostics	15,829 (28%)

Table 2-3. Diagnostics

at any time. The CPU sends out an instruction every cycle to the FPU (via busI), and it also provides the observability of the instruction being executed, including internal instructions. The IU and the EU can be physically separated by setting certain *diagnostic pins*. Furthermore, some of the lower order bits of instruction address bus were routed out to pins. Using these features, instructions can be delivered directly to the EU in case the instruction unit is not functional, by monitoring the instruction address (busPC<10:2>) available on pins.

The initial testing was done on a special board made for the SPUR CPU chip. The Tektronix DAS 9100 system is connected to the board and controlled from a SUN workstation. The test set-up is shown in Figure 2-11. The same vectors used in the switch-level simulations are converted into test vectors. For short-cycle testing, test vectors were downloaded to DAS and testing was performed. A special set-up was necessary for long-cycle testing, since the DAS can only hold up to 256 cycles of test vectors. Long vectors are divided into several parts to fit in the DAS capacity. The division was made at the instruction accessing memory (external cache), such that the CPU was deliberately made to stall on cache miss by control-

ling the CMU interface pin (cache busy), while the next portion of the vector is being downloaded. All signals acquired during the testing are transferred back to the SUN workstation for a cycle-by-cycle verification with the expected result. Most of the CPU functionalities are tested using the initial test set-up. After the debugging is done, the CPU chip is put on a SPUR processor board to test interactions with other components on the board, especially with the CMU.

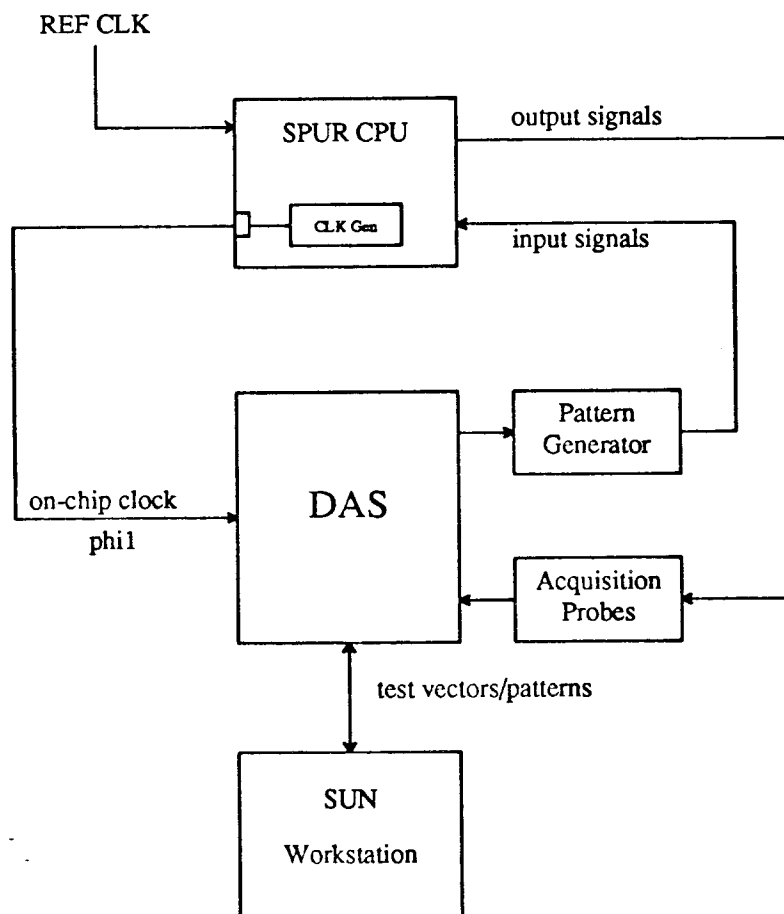


Figure 2-11. Chip test set-up

2.4.4. Design metrics

The design metrics for the SPUR CPU are presented in Table 2-4. It provides an approximate design time spent on both the circuit design and the layout, in terms of man-months. The total design time of the SPUR CPU is estimated at about 5 man years. This includes behavioral modeling, VLSI design, verification, and testing. Some of these activities were performed in parallel, and the times shown in Table 2-4 are for the VLSI design only. Approximately $\frac{1}{3}$ of the total development time (or $1\frac{1}{2}$ man years) was spent on verification and testing of the chip. There are total of 13 PLAs used in both the IU control and the master control. These PLAs are summarized in Table 2-5.

The transistor count of the chip reaches over 115,000. More than 50% of transistors or about 60,000 transistors are SRAMs used to implement the register file and the instruction cache, which occupy about $\frac{1}{2}$ of the total active chip area. Area estimates (percentages) shown do not include any routing region, so numbers may not add up to the totals. Regularity of each unit is computed by taking the ratio of total transistor to total drawn transistors of each unit. Comparison of design metrics to other microprocessors is presented in Table 2-6.

2.4.5. Results

The first-pass silicon had a few bugs, including circuit design, layout, and timing errors, but it worked enough to be used for initial debugging of the processor board. The layout errors discovered were misplaced well and substrate contacts onto signals rather than power supply lines. These effectively shorted the signal to either the ground or the power line, resulting in a stuck-at type fault. Some of these errors were corrected by isolating the misplaced contacts from the power supply using the laser restructuring technique provided by the Information Science Institute (ISI). Either the first level or the second level metal can be disconnected by using a laser shot through the passivation layer. The second (topmost) level metal lines with width of $3\ \mu\text{m}$ were cut successfully without affecting other structures

Block	Layout Area			# transistors	Regularity	Design Time	
	Height (lambda)	Width (lambda)	% Area			Circuits (Man month)	Layout (Man month)
Instruction Unit	4456	6540	24.3%	37622	17.8	1.0	2.0
IU_CTR	1508	1503	1.9%	1501	2.0	0.6	1.0
IB_Cache	4267	4899	17.4%	31583	500.0	0.3	0.7
IB_TAG	2423	1214	2.5%	4538	300.0	0.1	0.3
Register File	4412	5478	20.2%	42924	27.5	1.8	3.5
Registers	3006	4533	11.4%	33120	5520.0	0.5	1.5
Decoders	1089	4656	4.2%	6300	9.0	0.5	0.5
IF Logic	221	779	0.1%	210	2.1	0.5	0.5
DST1 & DST2	3100	930	2.4%	3294	11.0	0.3	0.5
Master Control	3522	3422	10.1%	3849	1.7	1.3	2.4
SEQUENCER	2424	3388	6.9%	2190	5.0	0.5	1.0
TRAP_LOGIC	645	586	0.3%	1070	1.2	0.5	1.0
CC_INT	877	534	0.4%	506	PLA	0.2	0.3
SPD_LOGIC	223	490	0.1%	83	PLA	0.1	0.1
Local Control			1.0%	721	1.0	0.4	0.5
RegFile_CTR	835	212	0.2%	137	1.0	0.1	0.2
Func_CTR	136	500	0.1%	46	1.0	0.1	0.1
PCLOGIC_CTR	290	2110	0.5%	372	1.0	0.1	0.1
SpecReg_CTR	279	811	0.2%	166	1.0	0.1	0.1
Special Registers	2530	1700	3.6%	3502	7.7	0.3	0.6
UPSW	2525	353	0.5%	905	10.0	0.1	0.2
KPSW	2536	504	1.1%	1020	10.0	0.1	0.2
CWP & SWP	2755	846	1.9%	1577	3.0	0.1	0.2
Functional Units	3175	1897	5.0%	5619	5.0	0.8	1.7
Byte-Extractor	3156	248	0.6%	209	7.0	0.1	0.2
Byte-Insertter	3156	262	0.7%	395	7.0	0.1	0.2
Shifter	3160	336	0.9%	768	6.0	0.1	0.3
ALU	3166	1078	2.8%	4247	4.0	0.5	1.0
PC Logic	2756	2098	4.8%	6370	4.3	2.0	4.0
Miscellaneous							
MBR	3093	482	1.2%	1619	11.0	0.1	0.2
MAL	1125	711	0.7%	748	30.0	0.1	0.2
Scan_Registers	3200	1600	4.3%	6028	36.0	0.1	0.3
bus_Interface	3154	584	1.5%	1581	17.8	0.2	0.5
CLK_GEN	455	2127	0.8%	392	1.0	1.0	1.0
PADS & Others				4239		0.5	0.5
Total	10140	11820	100.0%	115214	12.6	9.6	17.4

Table 2-4. Design metrics of the SPUR CPU

PLA	# product terms	# outputs	# inputs	Power (mW)
OPCODE	68	40	8	54.0
FAST_LOGIC	16	14	18	15.0
SPD_LOGIC	7	2	6	4.5
TRAP_ENABLE	15	13	24	14.0
TRAP_TYPE	11	9	11	10.0
CC_OPGEN	25	6	13	15.5
CC_INT	17	5	12	11.0
IU_CTR_P1	6	4	7	5.0
IU_CTR_P2	10	5	9	7.5
IU_CTR_P3	30	8	10	19.0
IU_CTR_P4	21	6	16	13.5
IU_FET_FSM	14	3	10	8.5
IU_PF_FSM	14	3	10	8.5
Total	254	118	157	186.0

Table 2-5. SPUR CPU PLAs

CPU	# transistors (1000s)	Regularity	Design time (man years)
SPUR CPU	115K	12.6	5.0
SOAR	36K	8.3	3.2
RISC II	41K	20.0	2.5
M68000	68K	12.0	14.2
80386	181K	NA	50.0

Table 2-6. Comparison of design metrics

nearby. Other problems found were timing errors and glitches on signals controlling the dynamic circuits. The glitch was caused by the excessive ringing on clock lines. The long running clock lines (10 mm) can have parasitic inductance and capacitance large enough to cause a substantial ringing, which may trigger any hazardous glitch.

Several electrical-rule checks were performed to avoid repeating the same errors for the second pass. However, there was still another layout error discovered after the fabrication. A portion of metal wire was missing, leading to a disconnected signal. A focused ion beam (FIB) IC development system, provided by the Seiko instrument company was used to fix the problem. Two holes were drilled on separated wires through the passivation layer to reach metal lines, using an ion beam, and connected using FIB-CVD (chemical vapor deposition) metal film deposition between the two points. The revised and repaired chip is fully functional and is used in a working SPUR processor board successfully executing its own operating system (Sprite) as well as many applications including LISP programs. The nominal operating frequency of the chip on the processor board is 10 MHz, while the maximum operating frequency is 12.5 MHz (80 nsec cycle time),

2.5. Summary

The SPUR CPU is a single-chip RISC microprocessor designed for a multiprocessor workstation. It supports a multilevel cache scheme including a prefetching on-chip instruction cache, a coprocessor interface, and support for the fast execution of LISP through a tagged 40-bit architecture. In order to build a working computer system based on the SPUR CPU chip, reliable and efficient methodologies were necessary throughout the design. The chip, fabricated in a 1.6 μm double metal CMOS process, works well in the multiprocessor system prototype, and it met both of the functional and performance goals set at the initial stage of the design. It runs at 10 MHz consistently for all programs and dissipates less than 0.8 W of power.

2.6. References

- [EEE88] *Oct. Tools Distribution 2.0 (Manual)*, Electronics Research Laboratory, UC Berkeley, 1988.
- [Goo83] J. R. Goodman, "Using Cache Memory to Reduce Processor Memory Traffic", *Proc. Tenth International Symposium on Computer Architecture*, Stockholm, Sweden, June 1983.
- [HiS84] M. D. Hill and A. J. Smith, "Experimental Evaluation of On-Chip Microprocessor Cache Memories", *Proc. Eleventh International Symposium on Computer Architecture*, Ann Arbor, MI, June 1984.
- [Hil86] M. D. Hill et al., "Design Decisions in SPUR", *IEEE Computer* 19, 10 (November, 1986), 8-24.
- [Jeo87] D. K. Jeong et al., "Design of PLL-Based Clock Generation Circuits", *IEEE J. Solid-State Circuits* sc-22, 2 (April 1987), 255-261.
- [Kat83] M. G. H. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI", Ph.D. Thesis, CS Division, EECS Dept., University of California, Berkeley, October 1983.

- [KLL82] R. H. Krambeck, C. M. Lee and H. S. Law, "High-Speed Compact Circuits with CMOS", *IEEE J. Solid-State Circuits* sc-17, 3 (June, 1982), 614-619.
- [Lee86] D. Lee, "Data Path Design Considerations for a High Performance VLSI Multiprocessor", Technical Report UCB/Computer Science Dpt. 87/318, University of California, Berkeley, November 1986.
- [Ous85] J. K. Ousterhout et al., "The Magic VLSI Layout System", *IEEE Design and Test of Computers* 2, 1 (February 1985), 19-30.
- [Pen87] J. M. Pendleton et al., "A 32-bit Microprocessor for Smalltalk", *IEEE J. Solid-State Circuits* SC-21, 5 (October, 1987), 741-749.
- [Seg87] R. B. Segal, "BDSYN: Logic Description Translator, BDSIM: Switch-level Simulator", Electronics Research Laboratory Memorandum, No. M87/33, UC Berkeley, May 1987.
- [She84] R. Sherburne et al., "A 32-bit Microprocessor with a Large Register File", *IEEE Journal of Solid-State Circuits* SC-19, 5 (October, 1984).
- [THL86] G. S. Taylor, P. N. Hilfinger, J. R. Larus, D. A. Patterson and B. G. Zorn, "Evaluation of the SPUR Lisp Architecture", *Proc. Thirteenth International Symposium on Computer Architecture*, Tokyo, Japan, June 1986.
- [Ung84] D. Ungar, "Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm", *ACM Software Engineering Notes/SIGPLAN Notices Notices Software Engineering Symposium on Practical Software Development Environments*, Pittsburg, PA, April, 1984.

3

On-chip Memory Design

3.1. Introduction

A fundamental limitation in microprocessor performance is set by the ratio of the amount of memory traffic to the available i/o pin bandwidth of the microprocessor chip. The microprocessor's memory traffic consists of instruction and data transfers in and out of the chip. As the cycle time of the microprocessor shortens with advances in the integrated circuit technology, the memory traffic required to balance the overall system throughput goes up rapidly [Kun86]. However, due to various limitations the i/o pin bandwidth remains relatively constant. The minimum pad size required for wire bonding has been unchanged for years. Moreover, the number of required power supply pins has risen to accommodate fast switching i/o pins, which in turn reduced the number of pins available for off-chip communication.

To obtain the highest possible performance in a single chip microprocessor architecture, off-chip communication must be minimized while integration of functionality is maximized.

One way of minimizing off-chip communication is to include local memory on a chip as a cache or a set of registers to hold frequently used instructions or data. Cache memories not only provide fast accesses to instructions and data but also reduce off-chip memory accesses by using the cached instruction and data repeatedly. Therefore, the on-chip memory design has a great impact on a microprocessor's performance and becomes increasingly important.

In this chapter, I will examine on-chip memory design issues and present new and efficient on-chip memory designs for microprocessors. The focus is on tradeoffs between the architectural design and the circuit design. Circuit design techniques, when properly adapted to the architectural design, can provide cost-effective performance improvement easily. Two key areas of interest in this research are using DRAMs as a cache on a microprocessor chip and multi-port memory design to facilitate the parallelism using multiple functional units. With high density dynamic memories, cache performance can be improved greatly since the storage capacity (size) of memory is one of the most critical parameters in the on-chip cache design. Multi-port memory can have a great impact on processor performance since it provides high bandwidth and also facilitates the concurrent operations. The results of the research can be useful as a design guide for alternatives in future microprocessor memory designs.

This chapter is organized as follows: Section 2 reviews the use of on-chip memories in existing microprocessors, as an instruction store and a data store separately. Section 3 presents a reliable and efficient way of using dynamic memory elements in a microprocessor chip. Section 4 presents the local on-chip memory design for multiple functional units. To satisfy the bandwidth requirement of multiple functional units, multiple memories or multi-port memory design are necessary. The implementation issues of these memories are considered in that section. Section 5 summarizes the on-chip memory designs.

3.2. On-chip Memories in Microprocessors

Microprocessor architecture is evolving as silicon integrated circuits increase in density. On-chip memories are becoming an established feature in single-chip microprocessor designs because they significantly improve performance. It is particularly important for single chip RISC microprocessors to include large, high-speed memories, because RISC chips must reduce off-chip memory delays to achieve the shortest possible cycle time. The organization of the on-chip memory is therefore very important in the design of high performance VLSI single-chip processors.

Memories are used in various forms on microprocessor chips. Fast storage for instructions and for data are two distinct needs for on-chip local memories. The separation of local memories for instructions and data is common, in part to increase effective memory bandwidth. A mixed instruction and data cache is not as effective as separate caches, unless dual-ported memory is used to resolve the memory contention between instruction and data memory references. When an on-chip memory is limited in its capacity, using it as an instruction cache or a data cache can be an interesting architectural tradeoff. This section begins with an examination of the use of local memories in existing microprocessors, then research focus and limitations are identified.

3.2.1. Local memory for instruction store

Microprocessor performance can be hampered by off-chip memory access delays. These delays are caused either by fundamental limitations in off-chip communication, or by i/o contention between instruction and data memory traffic through scarce i/o pins. On-chip instruction caches resolve these problems by caching instructions on the chip and supplying them directly to the execution unit. This allows i/o pins to be used primarily by the data memory accesses, and thus effectively provides a dual-ported access to the external memory.

Instruction caches are simpler to design than mixed caches because the cache is read-only (cache is written only to replace the missed block). Furthermore, since instructions show

a much higher degree of locality than data, even a small cache can improve processor performance significantly. Many existing microprocessors incorporate on-chip instruction caches in one form or another. Different instruction memory organizations are summarized in the following.

Prefetch buffer (PB) holds instructions sequentially forward from the current program counter in the instruction stream. PB's are usually organized in a FIFO of instruction words. Many computers such as IBM System/370 Model 158, and DEC VAX 11/780, have had PBs. Today's microprocessors have PBs implemented as a part of other types of instruction cache.

Instruction buffer (IB) uses caching and prefetching to reduce effective access delay as well as memory traffic. As a conventional cache, IB can be organized as a direct-mapped or a set-associative cache. For small IB's, however, it has been proved that a direct-mapped cache performs comparably to a fully associative cache with LRU replacement [SmG83]. Loading partial blocks upon IB misses (sub-block placement) is also effective in minimizing the memory traffic [Goo83], [Hil87b]. Important design parameters in designing an on-chip IB include cache hit and miss time, cache size, and aspect ratio of cache memory when it is actually laid out inside the chip [ACH87]. Microprocessors with on-chip IB are the Motorola 68020 [MMM84] and 68030 [MMM86], the National NS32532, the MIPS-x [Hor87] at Stanford, and the SPUR CPU [Hil86] at U.C. Berkeley.

Target instruction buffer (TIB) reduces effective instruction access time by caching instructions at branch targets or at the beginning of the instruction run. TIB's are usually implemented with PB's. Upon a non-sequential instruction fetch (i.e. branch), The TIB is accessed to provide (if hit in the TIB) the next instruction, which is the first instruction of the next instruction run. Subsequent sequential instruction fetches are handled by the PB. The AMD Am29000 [AAA87] RISC microprocessor uses a TIB with PB.

Branch Target Buffer (BTB) buffers the addresses of previous branches and their target addresses [LeS84]. BTB is used to reduce pipeline bubbles, resulting from waiting for the next instruction address to be determined. The instruction fetch address is compared with the

content of the BTB and if they match, the next instruction address is determined from the BTB. The performance of the BTB depends on the selection of a branch prediction algorithm, the size, and the organization (e.g. set-associativity).

Decoded instruction buffer holds the fully decoded instructions, so that instruction issued from instruction cache can be executed without any further decoding delay. The CRISP microprocessor [DMB87] uses this form of instruction cache with branch-folding. When a non-branching instruction is immediately followed by a branch, the two are folded together to form a single new decoded instruction.

Performance of on-chip instruction memory is characterized by the effective access delay of instructions over time. Cache access time and miss handling time are as important cache parameters as cache hit/miss rate, since together they determine the effective instruction access delay [Hil87b]. The physical size or aspect ratio of the on-chip instruction memory is also important because it must be fit within the area desired [ACH87]. For a given silicon area the fastest effective access delay can be achieved if the density of memory is maximized while the access time is at its minimum.

3.2.2. Local memory for data store

In general, there are two ways to organize the local memory for data, conventional cache and registers. Referencing behavior of the data memory is somewhat different from the instruction memory, and the memory for data can be controlled to some extent by the programmer [McD88]. Goodman [GoH86] showed that with a small size of on-chip memory, registers can be more effective than a cache in reducing access delays, if an optimal register allocation algorithm is used when compiling the program. Registers and data caches in microprocessors are organized in various ways to take advantage of different data access behaviors of programs.

Conventional cache, although it is usually invisible to the programmer, consistently works well and takes account of dynamic program behavior. An important parameter in

designing the small size on-chip data cache is a transfer size of data from memory to the cache, or line size (the line is also referred to as sub-block when transfer size is smaller than a cache block). Given cache size, a smaller line (sub-block) is proven more effective than a larger line for the data cache, due to the temporal locality [GoH86][Smi82]. A smaller line size also minimizes the off-chip memory bandwidth requirement.

Register file organizes registers in either single or multiple sets. Single set, general purpose registers have been widely used in microprocessors. Efficient use of on-chip registers depends on adapted register allocation scheme [Rad82][Hen81]. A multiple register set improves the processor's performance by reducing the off-chip memory traffic required to save and restore registers upon a call or context switch. A large register file of RISC II [Kat83] at U.C. Berkeley, organized in a stack of register sets, allocates new register sets dynamically on a per procedure basis.

Stack cache caches only memory references to the stack. It operates just like a conventional cache except that the stack pointer is used in managing the cache. When a miss occurs and the word to be replaced is dirty, it is written back to memory only if its location is below the top of the stack.

Top of stack cache is a set of high-speed registers which holds the top portion of the frequently used stack entries. It takes advantage of the fact that stack references will generally occur near the top of the stack, not scattered as in a data cache. The management of TOS registers is as important as register allocation in microprocessors with a general purpose register set. This type of cache has been used in the C machine [DiM82] at Bell Labs, and the Dragon [McC84] at XEROX PARC .

The silicon area used to hold a byte of data in cache differs from that used to hold a byte in a register. Cache requires tags, valid and dirty bits, and replacement information so that it can be managed dynamically by the hardware. Registers, on the other hand, must be managed efficiently by the software, and often require multiple access capability to provide high bandwidth between the execution unit and the register file. To use local memory most

efficiently, implementation tradeoffs, such as speed versus power or multi-port versus multiple sets of memories, must be carefully examined.

3.2.3. The focus and limitations of the research

In the previous two sections, we have briefly examined the use of on-chip local memories in many existing processors. Two key observations made from the above are summarized as the following. The research presented in this chapter is based on these two observations.

- (1) Since the on-chip memory is limited in its size, many different, complex, cache and register organizations are used for various optimizations. It is certain that the increase in memory size will not only improve the overall performance but also simplify the on-chip memory design.
- (2) The clock rate of a microprocessor's execution unit is increasing rapidly as IC technology advances, hence the bandwidth of the local memory must be sufficient enough to provide data at the rate of the execution unit's demand. Furthermore, to increase the system throughput by exploiting the parallelism in hardware, the use of multiple functional units becomes common. This, in turn, adds up the bandwidth requirement of the local memory. Consequently, a fast multi-port memory for multiple simultaneous read and write accesses may be necessary.

Silicon real estate is one of the scarce resources on a single chip microprocessor. Therefore, local memory must be used efficiently, and memory density must be maximized at a given silicon area. Traditionally, mainly due to reliability concerns, only static random access memories (SRAMs) have been used on most microprocessor chips. Dynamic random access memories (DRAMs) offer higher bit density at a given silicon area than SRAMs. However, due to fundamental limitations associated with DRAMs, such as refreshing requirement and complex self-timed controls, DRAMs rarely have been utilized as on-chip memories. In the following section, I propose techniques for a reliable and efficient use of DRAMs as on-chip

cache memories. The focus will be on the instruction cache. Trace driven cache simulations are used to analyze newly proposed schemes. The same techniques can be applied to the data cache under certain restrictions, which will be discussed also.

Increasing processor's performance by using multiple functional elements requires multiple local memories or multi-port memories, to match the bandwidth required by the multiple functional units. Multi-port memories are, however, much more expensive than single-ported memories in terms of silicon area required and operating speed. A micro-architect must examine all possible memory designs in order to build a high performance microprocessor. Within this research I will examine a few alternatives for multi-port memory designs, such as a dual-ported memory design based on 6T SRAM cells and extending the design into the multi-port memory with more than two read/write ports. The goal is to provide a guideline for making right tradeoffs for the multi-port memory design.

3.3. On-chip DRAM caches

3.3.1. Why DRAMs?

Static memories have been popular for on-chip memories because they do not require periodic refreshing or complex control circuitry. Dynamic memories need a periodic refreshing before the dynamic charge storage node loses its voltage level due to leakage current inevitable in silicon technology. The refreshing requirement of dynamic memories may interfere with the processor's normal operations, and thus they have been used rarely in a single chip microprocessor. However, the density that single-transistor (1T) or 3-transistor (3T) dynamic memory offers now stimulates designers to consider using dynamic memory as an on-chip memory.

The size of a SRAM cell used in microprocessors, typically 6T SRAM cell or its variants, is about four to eight times larger in area than that of some (3T or 1T) dynamic cells. For example, the 6T SRAM cell used in the SPUR CPU to implement the instruction cache

and a 3T dynamic memory cell are compared in Figure 3-1 (each layout contains four bits sharing power lines and bit line contacts). As discussed in the previous section, the size of the local memory is one of the most important design parameters. Therefore, it is quite appealing to use dynamic memories in place of static memory where increase in local memory size is crucial. With dynamic memories, such as 3T DRAMs, the size of the local memory at a given chip area can easily be quadrupled or increased even more. The 1T or 4T DRAM cells are not as useful as 3T cell for a single chip microprocessor since they may require special fabrication process (1T) or ratioed design (4T). Ratioing transistors may result in large cell area.

As we replace static memory with dynamic memory, more memory cells are integrated into the same area. In order to increase the overall performance, however, the speed or access time of the memory array must remain relatively constant over this change. It is the density (and hence the logical size of memory) that increases with dynamic memory, but not the physical size or the area of memory (parasitics are dominant factors in memory access delays). Therefore, with careful layout of the cell and good circuit design techniques, dynamic memory integrated in a given area can be as fast (especially for 3T dynamic cell) as static memory integrated in the same area, but with higher density.

3.3.2. Limitations of DRAMs for an on-chip memory

On-chip use of DRAM has serious drawbacks due to the difficulty of implementation using standard process technology, and reliability issues, such as refreshing requirement, and hard and soft errors of the DRAM (see below for a further explanation). The three most common types of DRAM cells are 1-transistor, 3-transistor, and 4-transistor DRAM cells, as shown in Figure 3-2. The high density, state of the art DRAMs use 1T DRAM cells. The process technology for such a high density DRAM is quite different from the process technology in which microprocessors are fabricated. Furthermore, the access time of 1T DRAM is usually much slower than 3T or 4T DRAMs due to slow sensing delay. The 3T or 4T dynamic

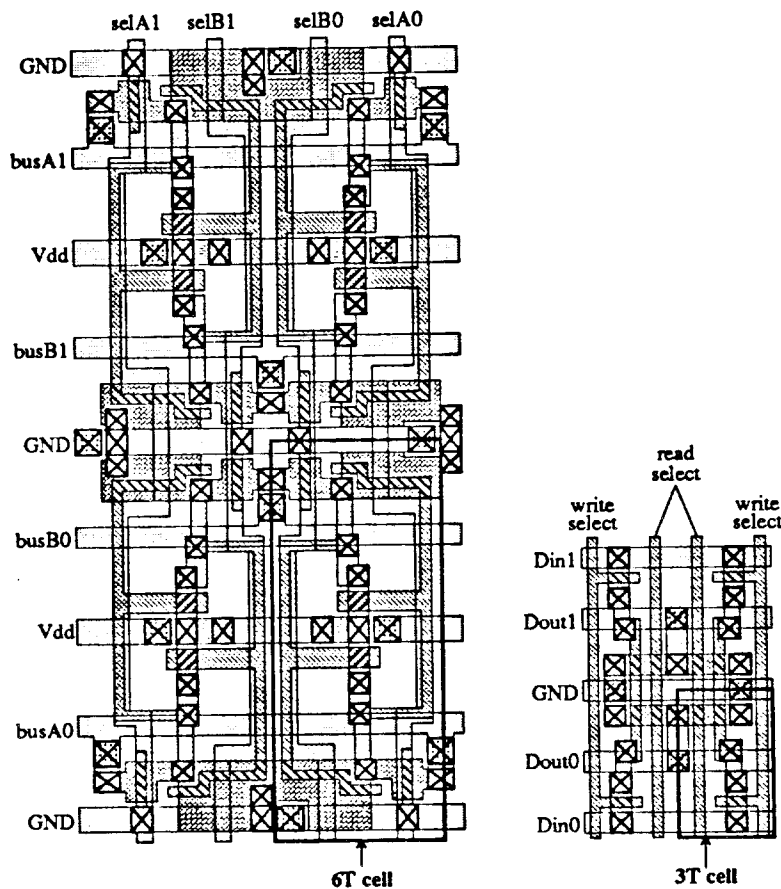
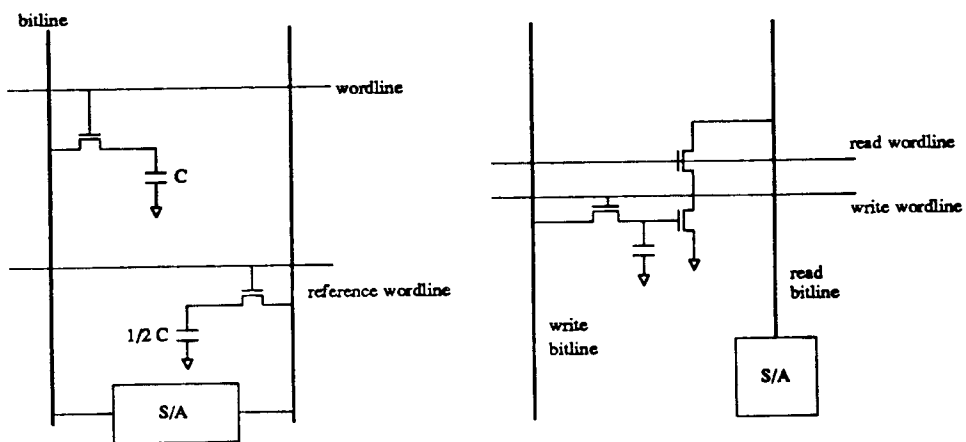


Figure 3-1. Comparison of CMOS 6T SRAM cell (of SPUR CPU) and 3T DRAM cell

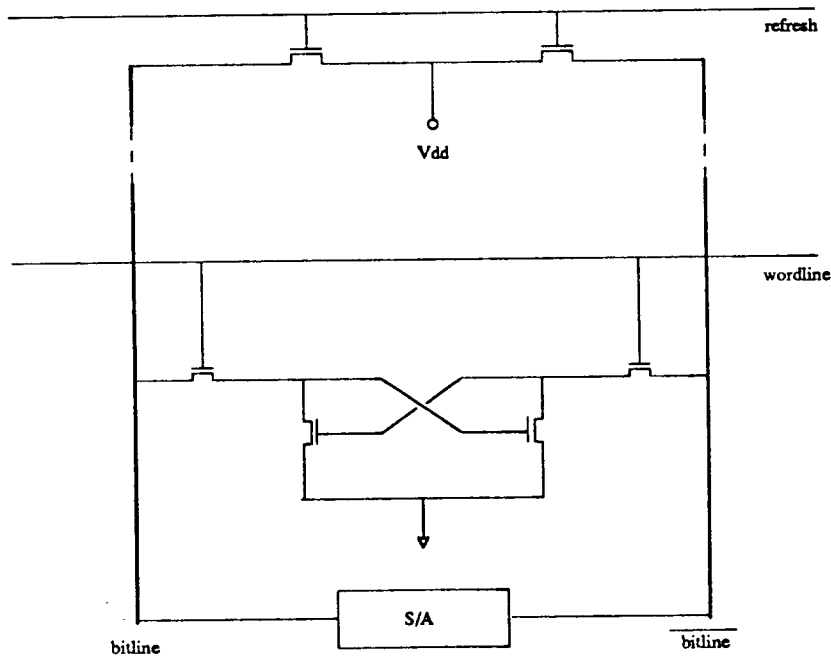
cells do not require special process technology and can be easily integrated into the single chip microprocessor. Moreover, they are more tolerant of process variations than 1T cells. The high density SRAM (4T and 2R loads) often uses special fabrication process to reduce the cell area by using load devices made of high resistance poly resistors).

Dynamic memory stores information as a charge on an isolated capacitive node. The charge on this node leaks away if left isolated for long time due to the leakage current associated with necessary silicon pn junctions. The information stored on a storage node may be lost if the charge leaks away too much. A refreshing operation reads the information before it



(a) 1T dynamic memory cell

(b) 3T dynamic memory cell



(c) 4T dynamic memory cell

Figure 3-2. Dynamic RAM cells

is degraded, and restores the charge to its original level. The refresh interval of dynamic memory can be determined by:

$$T_{ref} = \frac{f \times Q_{dynamic\ node}}{I_{Leakage}}$$

where f is a fraction allowed to be lost due to the leakage current.

With current technology, a dynamic storage node must be refreshed in as little as a one to two millisecond period. As capacitance on the storage node decreases, the refresh interval must be shortened. In the following section, I will present a method to overcome the refreshing overhead of dynamic memory.

Hard errors are usually originated from fabrication defects. It is therefore more probable to have hard errors when a larger chip area is devoted to the on-chip memory. Soft errors are induced by alpha particles or cosmic rays and are a well-known phenomena in the use of the DRAM. To make dynamic memory on a microprocessor chip safe and efficient, these problems must be overcome. Error detection and correction codes are extensively used to improve the reliability of dynamic memory systems, to handle hard and soft errors. Recently, some single chip DRAMs integrated these error detection and correction schemes on the chip [Yam84][Man87]. It may be desirable to have a simple form of these schemes in microprocessors with dynamic memory.

3.3.3. Non-refreshing DRAMs for on-chip caches

The integrity of the data stored in dynamic memory can only be assured by periodic refreshing. Ideally, refreshing should be done without affecting the processor's execution stream. Some microprocessors use software refreshes which are sometimes called refresh hiccups. When a timer interrupt occurs indicating a refresh interval, the microprocessor's control stops the on-going operation to refresh the dynamic memory. This scheme, however, is unacceptable because of the effort required to make sure all systems using this processor have the proper interrupt handler. It also degrades the processor's performance by interfering

with the processor's normal execution stream and by taking as many cycles as needed for refreshing the dynamic memory. Using simple circuit techniques and a few modifications in cache design can effectively alleviate these problems. Two schemes that eliminate the refreshing overhead of dynamic memory have been devised for implementation in hardware. These are: (1) invalidate on every refresh interval; and (2) selective invalidation on every refresh interval. These schemes are based on the following assumptions and restrictions [Hil87a]:

- (1) The cache contains copies of instructions or data, which also reside elsewhere such as external cache or main memory (e.g. instruction cache or write-through data cache).
- (2) The cache contains a number of blocks consisting of an address tag and one or more sub-blocks; associated with each sub-block is a VALID bit, so that any subset of block's sub-blocks may be valid.
- (3) The sub-block is the unit of transfer from off-chip into the on-chip cache.
- (4) All VALID bits associated with address tags or sub-blocks can be reset in parallel to invalidate the cache.
- (5) Any access to dynamic memory is considered as a refresh (read is always followed by write-back in DRAM's). In other words, the cache entry accessed during the last refresh interval need not be refreshed until the end of the next interval.

DRAMs, if used as a cache on a microprocessor chip under the above assumptions, need not be refreshed periodically. Instead, the cache may get invalidated once at each refresh interval. Most microprocessors with on-chip caches have a privileged instruction or other ways to invalidate their caches (see assumption 4 above). Thus, in the expense of a timer (frequency/clock counter) one can easily implement this scheme. This scheme, however, degrades the processor's performance by invalidating the active cache entirely every few milliseconds. This invalidation of the cache at the end of every refresh interval is referred to as the first scheme, invalidate on every refresh interval. The refresh or invalidation interval for

this scheme can be as long as that of the refresh period of dynamic memory.

The next scheme, the selective invalidation, is more elaborate than the first. An extra bit per each sub-block, in addition to the VALID bit, is used to hold a refreshing status. This bit is set at the beginning of each refresh interval, and reset selectively whenever the corresponding sub-block is accessed regardless of read or write access. At the end of the refresh period,

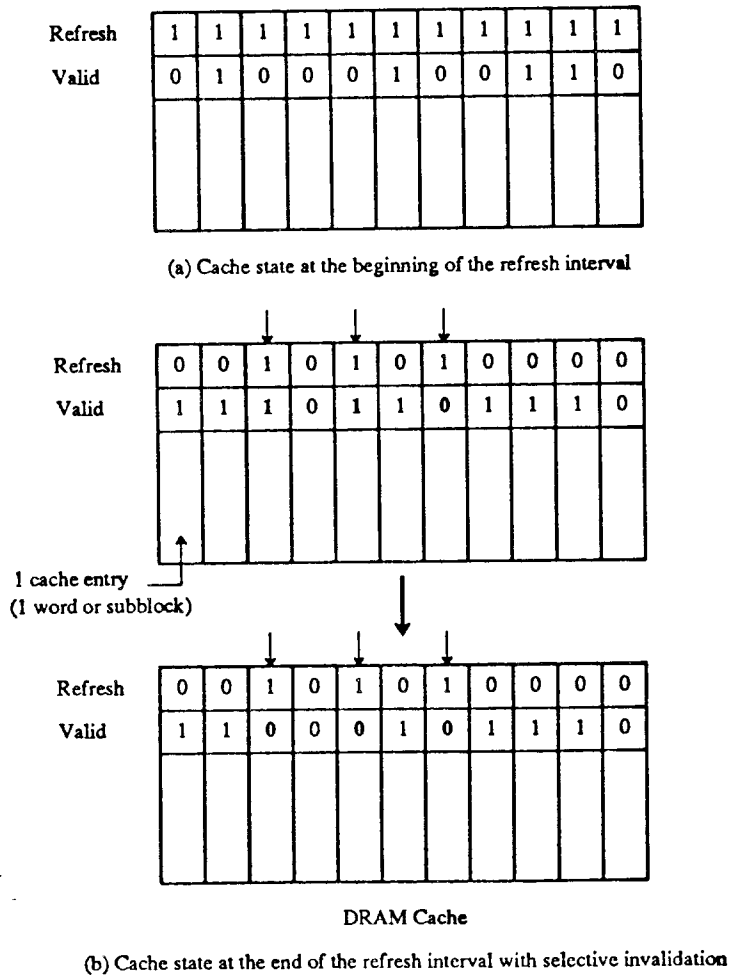


Figure 3-3. Dynamic RAM cache - selective invalidation

all of these bits and VALID bits are examined and only those sub-blocks not accessed during the last interval and still valid are invalidated selectively (see Figure 3-3). Invalidating the cache to preserve data integrity is no longer necessary. The fact that cache entries not accessed for a long time may not be needed in the future (cf. temporal locality) makes the DRAM cache performance very close to that of the SRAM cache. The performance of the DRAM cache with the selective invalidation is evaluated in the following section using trace driven cache simulations. This selective invalidation can be implemented with a small (six transistors per sub-block) circuit as shown in Figure 3-4. It can be easily extended to incorporate separate or multiple word lines if required, by adding one transistor per word line (see dotted transistor in the figure).

Instruction caches are usually read-only (written only when there is a miss to replace the missed block or sub-block), and one of the above methods can easily be employed if DRAM's are used to implement the cache. To expand the usage of these methods to the data cache, the cache must adapt the write-through policy for storing new value into its entry. Since all or any subset of cache entries may be invalidated at any time with one of these schemes, any newly-written cache entry must be stored in a safe place (main memory or external cache). With the write-through policy, subsequent accesses to the invalidated cache entries will miss and eventually retrieve the the correct data from the saved place. For a multi-level cache design, a write-through policy for an on-chip data cache (highest level in the hierarchy) is a reasonable choice since it is the simplest method to assure data consistency among caches in the hierarchy. A better write policy can still be applied to the external cache.

3.3.4. Evaluations

Two methods of using DRAM's without actual refreshing of memory cells are evaluated in this section. I use the instruction cache of the SPUR CPU (the SPUR IB) as an example for the evaluation. Although it is optimized for the given constraints of the SPUR CPU architecture, the SPUR IB is a good representative model among different instruction

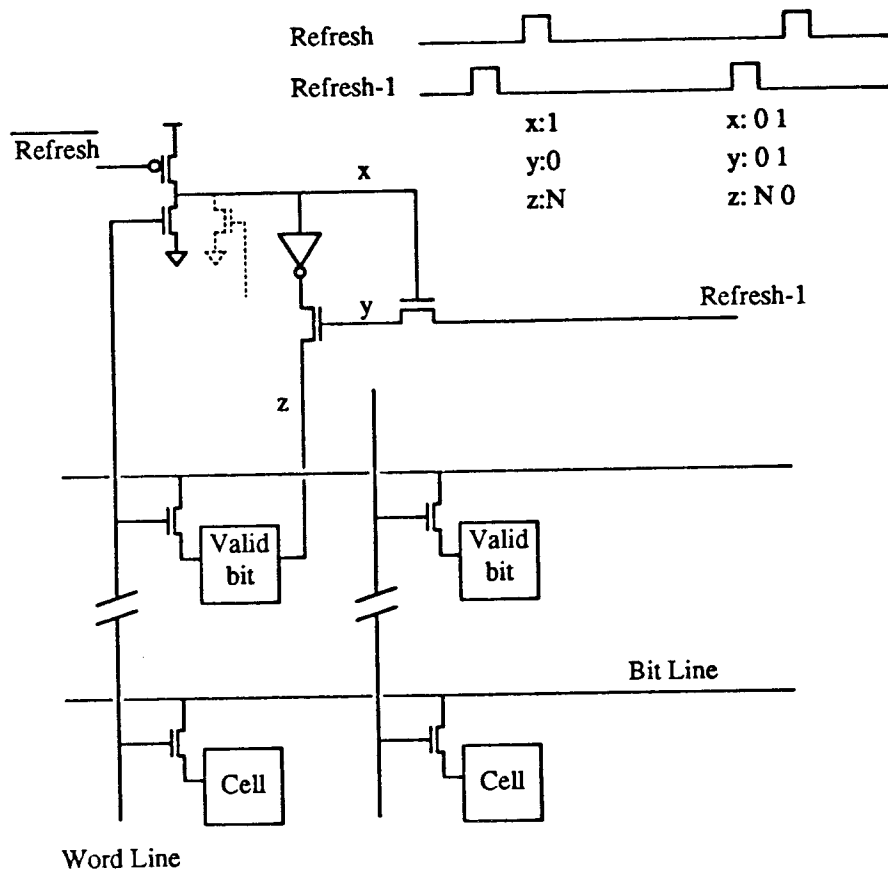


Figure 3-4. Circuits implementing the selective invalidation

cache organizations for single-chip microprocessors and hence is chosen for this evaluation. The SPUR IB was implemented using 6T CMOS SRAM cells. I will compare the performance of the SPUR IB to that of the SPUR IB implemented using DRAM cache with each of the above two methods, eliminating the refreshing overhead.

I use the miss ratio as a performance measure for different caches. Trace driven cache simulations that directly compute miss ratios of caches with different parameters are employed here. Other performance measures such as *effective access time* [Hil87b] may be easily computed from the miss ratio determined in this evaluation. The same traces used in

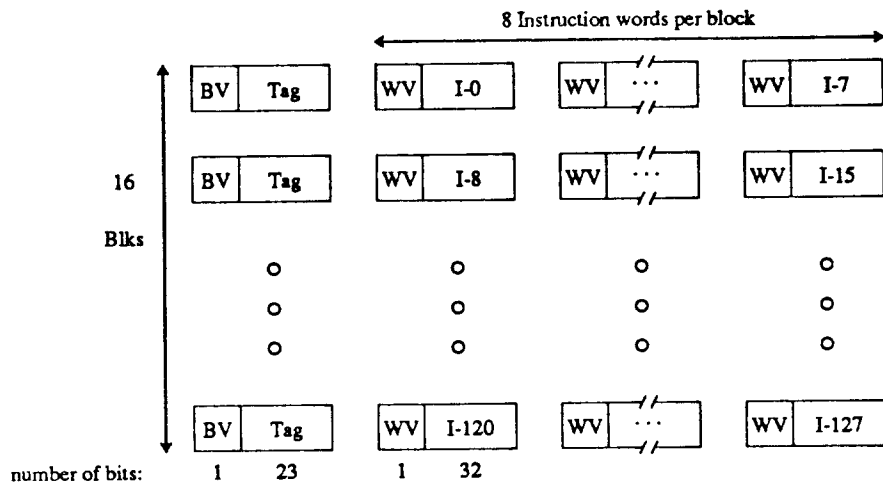


Figure 3-5. SPUR instruction cache organization

the SPUR IB design [Hil87b] are used here also. Those traces are:

- (1) Weaver, a production system written on top of OPS5 for VLSI chip routing [Joo85];
- (2) Rsim, a switch-level simulator simulating a counter [Ter83];
- (3) Slc, the SPUR Lisp compiler [ZHH87], based on the SPICE Lisp [THL86], compiling part of itself.

For each of these programs, two 500K-instruction dynamic trace sets showing different behaviors (medium and pessimistic) were collected (a total of six traces or three million instructions) [Hil87b]. Since miss ratio variation across the trace samples is small, subsequent results are based on miss ratios for a composite trace, formed by concatenating the six traces. The lengths of traces are the same, and so the miss ratio for the composite trace is equal to the arithmetic average of miss ratios from the individual traces.

The SPUR IB is an on-chip instruction cache organized in 16 blocks with eight sub-blocks in each block as shown in Figure 3-5. The size of a sub-block is 4-byte, or one 32-bit instruction, which is the off-chip data transfer size of the CPU chip. Associated with each

sub-block, or an instruction word, is a valid bit so that any subset of instructions within a block may be valid. The SPUR IB uses this flexible feature to reduce demand miss time by loading only the fetched instruction rather than the entire block, and to permit instruction pre-fetching to load the rest of a block in parallel with subsequent instruction fetches.

The architectural parameter that has the greatest impact on SPUR IB miss ratio is cache size. In the evaluation of DRAM caches, I keep all cache parameters of the SPUR IB unchanged except the cache size because it can vary with the DRAM implementation. Other parameters such as block (or sub-block) size, off-chip bandwidth (line size), and prefetch algorithm, will affect the cache performance, but their effects are the same for both SRAM and DRAM implementations. Table 3-1 shows the demand miss ratios of SPUR IB for six traces and Figure 3-6 plots the average miss ratio for different sizes of cache implemented

Cache Size	weaver		rsim		slc		Average
	medium	pessim	medium	notrap	medium	pessim	
512	23.49	21.26	22.98	24.04	21.20	23.68	22.78
2048	8.58	10.19	13.44	16.40	13.77	16.51	13.15
4096	7.18	8.31	8.06	11.39	10.93	12.61	9.75
8192	1.50	3.46	3.73	3.75	7.74	9.15	4.89
16384	0.96	2.03	2.42	1.46	4.81	6.37	3.01
32768	0.36	0.81	1.98	0.53	3.18	4.52	1.90

Table 3-1. Demand miss ratios (%) of SPUR IB for 6 traces

using SRAMs.

The evaluation of the first scheme, invalidating the cache at the end of every refresh interval, focuses on the effect of invalidation. The cache simulator, Dinero [Hil85], was slightly modified to simulate DRAM cache. A real cycle counter that counts not only references but also miss time and cycles lost to others, was used to invalidate the cache at accurate intervals. The maximum degradation in performance (increase in miss rate) due to the invalidation of DRAM cache can be determined by:

$$DMR = \frac{N_{sub-blocks}}{N_{ref}}$$

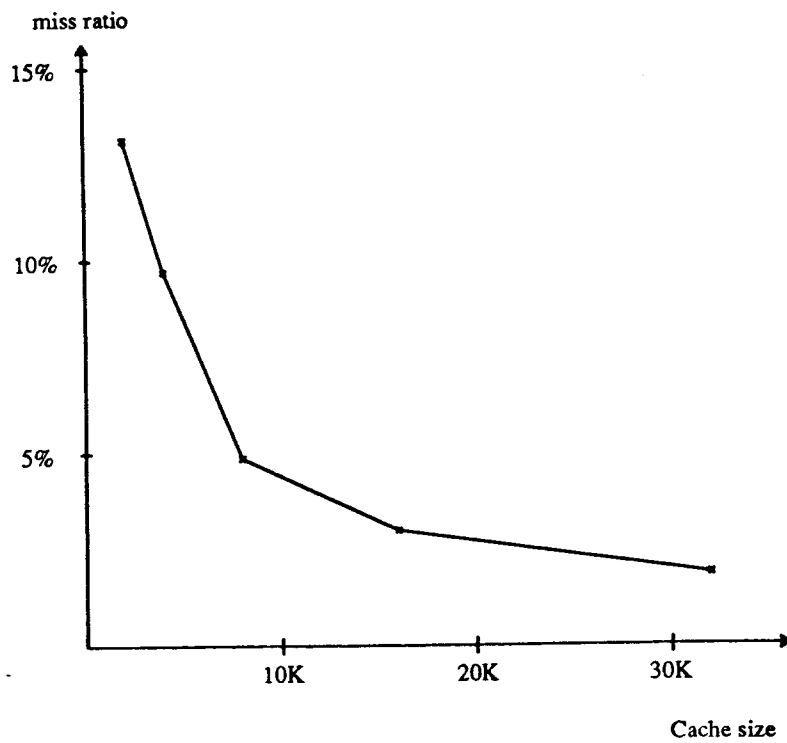


Figure 3-6. Demand miss ratio of SPUR IB with different sizes of cache

where DMR is a maximum miss ratio increase (degradation due to invalidations), $N_{sub-block}$ is a total number of sub-blocks in the cache, and N_{ref} is the average reference counts per refresh interval. The average reference counts can be estimated by:

$$N_{ref} = \frac{Q}{(1 + M \times miss_ratio + C)}$$

where Q is the refresh interval in number of cycles (e.g. 20,000 cycles), M is a miss time, and C is the cycles per instruction lost for other reasons (e.g., external cache misses). With a cycle of 100 nsec (the SPUR CPU's), the refresh interval is set to 20,000 cycles or 2 milliseconds for all simulations. Figure 3-7 plots the maximum possible increase of miss ratio for different sizes of caches. Although maximum bound set by the above equations is enormous for large caches, the actual difference in miss ratio between DRAM and SRAM caches is much less than the maximum. In fact many cache entries may not be used again later and will eventually be invalidated. Invalidating those cache entries may not degrade the cache performance, and hence the real difference in miss ratio is far less than the maximum predicted in Figure 3-7. The simulation run on the six traces mentioned above reveals this fact.

Figure 3-8 compares the miss ratio of a DRAM cache with the invalidation scheme to that of conventional SRAM cache as a function of cache sizes. Difference in miss ratios is negligible for small caches, but becomes substantial as cache size increases. More importantly, above a certain cache size the performance improvement by increasing the cache size diminishes (as indicated by an arrow in Figure 3-8), due to the frequent invalidations.

The selective invalidation scheme improves the cache performance by not invalidating the entire cache, instead invalidating only those entries not fresh and still valid. The cache simulator is also modified to correctly incorporate the selective invalidation. First, the refresh bit is added to each sub-block (or each access unit) structure, and new operation (selective invalidation) is added. With a timer (cycle counter) interrupt indicating refresh time, refresh and valid bits of each entry (or sub-block) are examined and invalidated, if necessary. The

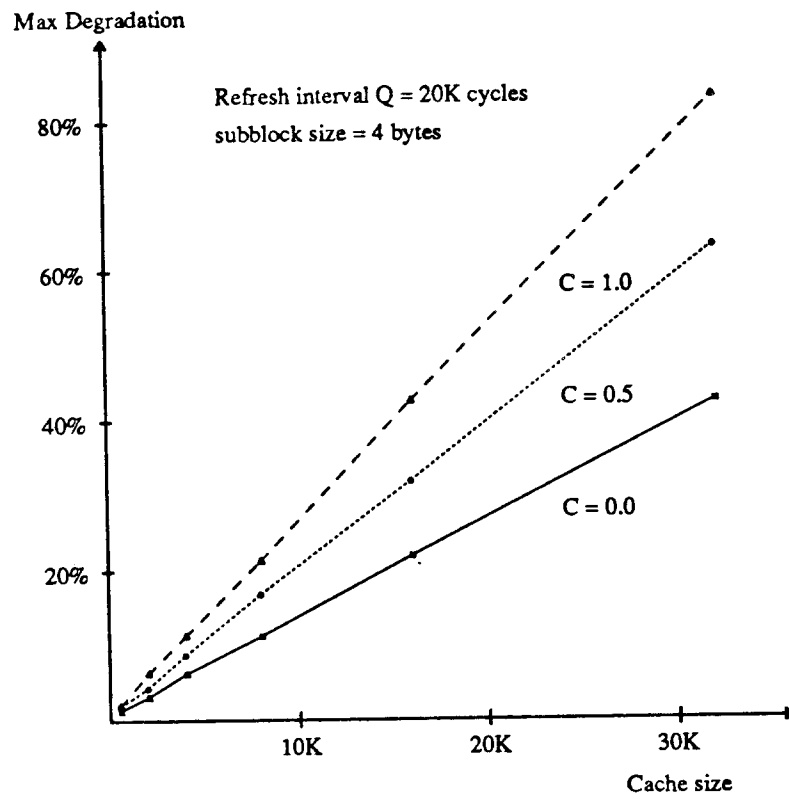


Figure 3-7. Maximum possible degradation due to invalidations

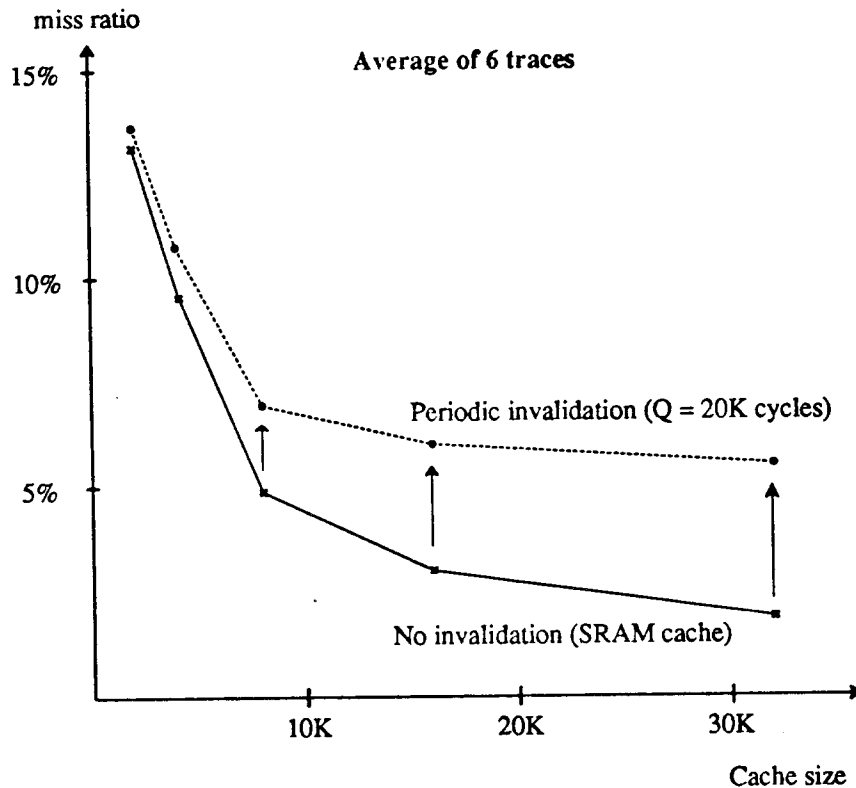


Figure 3-8. Effect of periodic invalidations on miss ratio

refresh interval must be a half of the required interval since some cells may hold valid data from the beginning of one interval through the end of the next.

The results of cache simulation run on the composite trace are shown in Table 3-2. The miss ratio versus size is plotted in Figure 3-9, and compared to the SRAM cache performance. The difference in performance is greatly reduced for even large caches by using selective invalidation. The miss ratio difference in large caches indicates that there are some cache entries with a very long lifetime but not accessed often, or there are some entries active at intervals greater than the refresh period. This may depend on the referencing behavior of program or data.

Cache Size (bytes)	Without Invalidation	With Invalidation	Selective Invalidation
512	22.78	22.88	22.78
2048	13.15	13.69	13.15
4096	9.75	10.82	9.77
8192	4.89	6.95	4.97
16384	3.01	6.02	3.25
32768	1.90	5.53	2.26

Table 3-2. Miss ratio (%) of DRAM cache (SPUR IB, sub-block = 4 bytes)

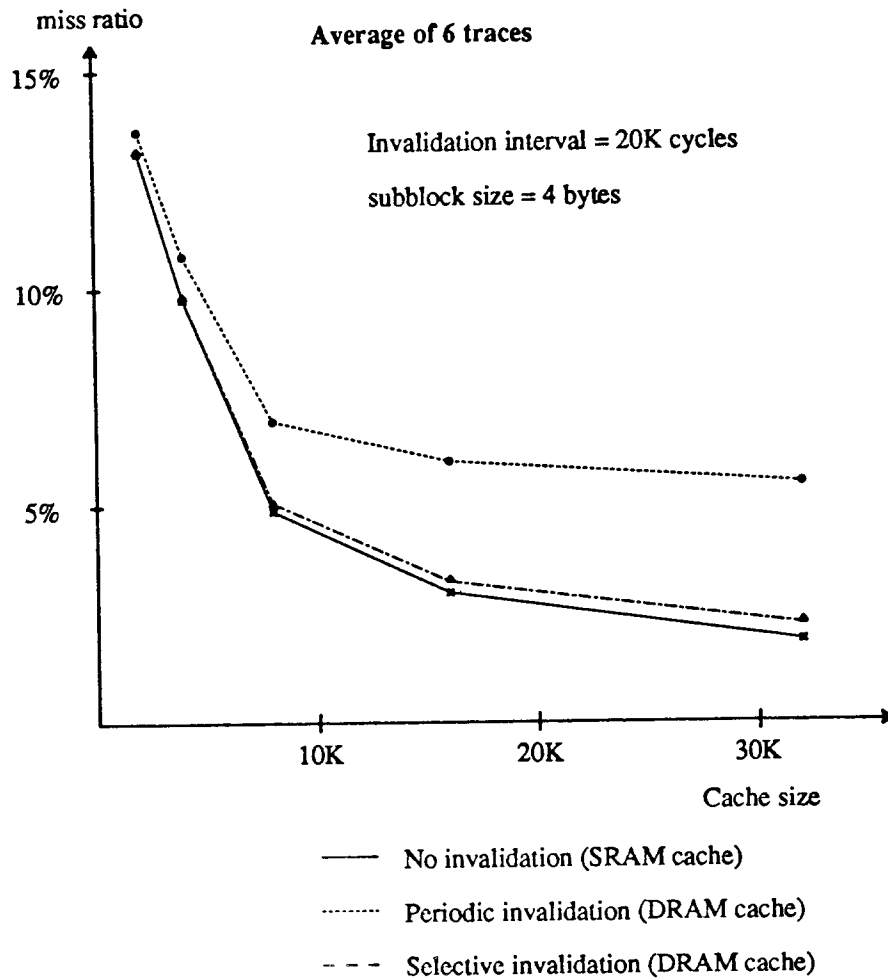


Figure 3-9. Effect of selective invalidations on miss ratio

The second most influential cache parameter on miss ratio of IB after size, is the size of the sub-block. Figure 3-10 shows a plot similar to Figure 3-9 for SPUR IB with twice the sub-block size. Such an improvement done on the SRAM cache will work equally on the DRAM cache.

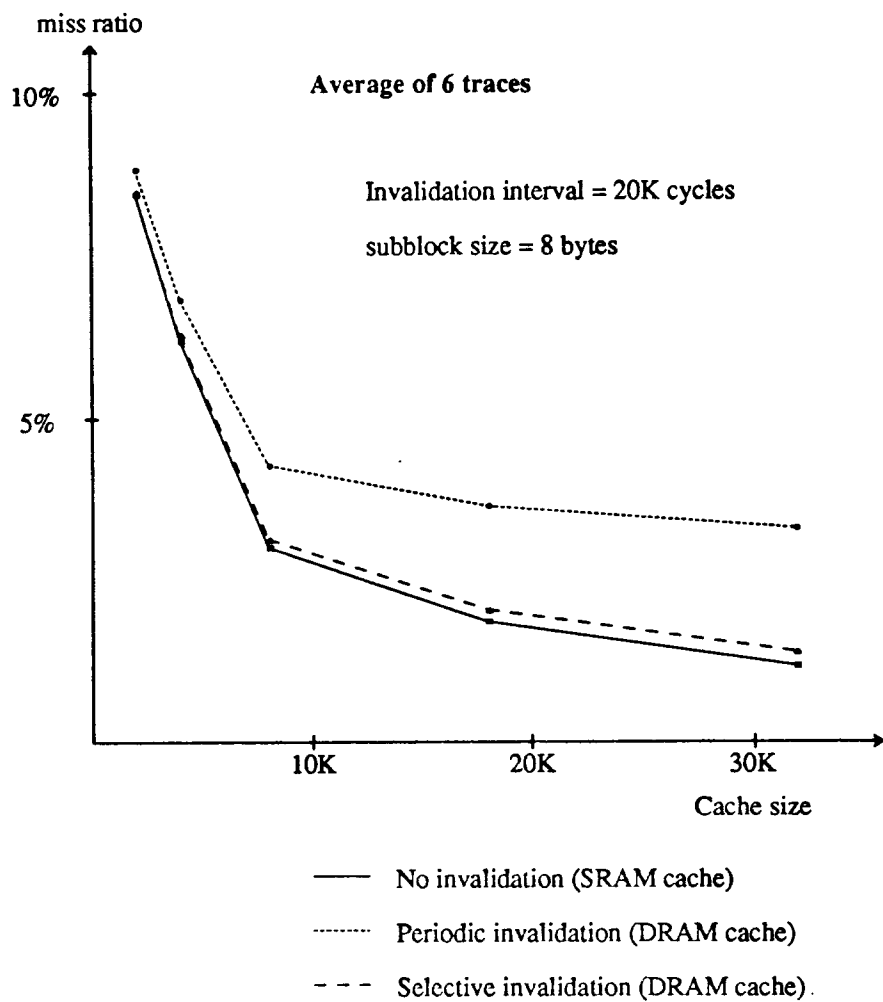


Figure 3-10. Effect of selective invalidations for different size of sub-block

3.4. Multi-port Memory Design

3.4.1. Multiple functional units

To improve the performance of a microprocessor, designers often look to approaches that permit parallelism, or overlap, in the instruction execution stream. Traditionally, pipelining has been one of the most popular of these approaches. Another technique that can be used

independently or to complement pipelining, is the use of multiple functional units. In either case, the application of such approaches can lead to a substantial improvement in a processor's maximum performance because the total computational resources that are simultaneously available to a running program is increased. However, a common resource such as memory (or on-chip local memory in the case of single chip processor), can become a performance bottleneck unless enough bandwidth between the memory elements and several functional units is provided.

It is well known that the size of the local memory must be large if the computational bandwidth of the processing elements is large, as represented by the "Amdahl's rule" [SBN82]. Furthermore, a well-designed microprocessor must provide "balanced" or "matched" bandwidth required by both local memory and functional units. This matching of the bandwidth is dependent upon an instruction set architecture (especially instruction format) as well as speed of circuits [Kuc78]. Together these two factors determine the required bandwidth from the memory hierarchy. Given that the single-chip microprocessor with one functional unit has a balanced bandwidth, if the number of functional units is increased by a factor of α , the local memory bandwidth also must be increased by the same factor, α (without any other optimization), to rebalance the processing capacity of multiple functional units.

3.4.2. Multiple sets of register files and multi-port cache memory

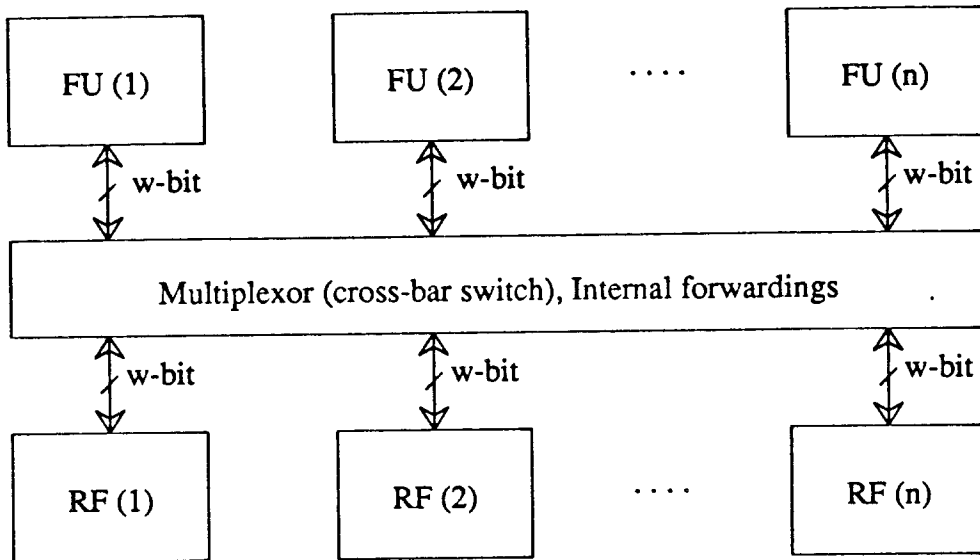
There are several ways to increase the bandwidth between the local memory and multiple functional units. Two prominent approaches are: (1) use multiple memories such that multiple functional units can access at least one of them simultaneously; and (2) use a multi-port memory such that multiple functional units can simultaneously access the common local memory. As previously mentioned in Section 2, many different on-chip memory organizations are possible with any of these two approaches. However, some memory organizations are particularly well suited to one of these approaches, while others are not. For instance,

having multiple caches can create cache consistency problems, even among local on-chip cache memories.

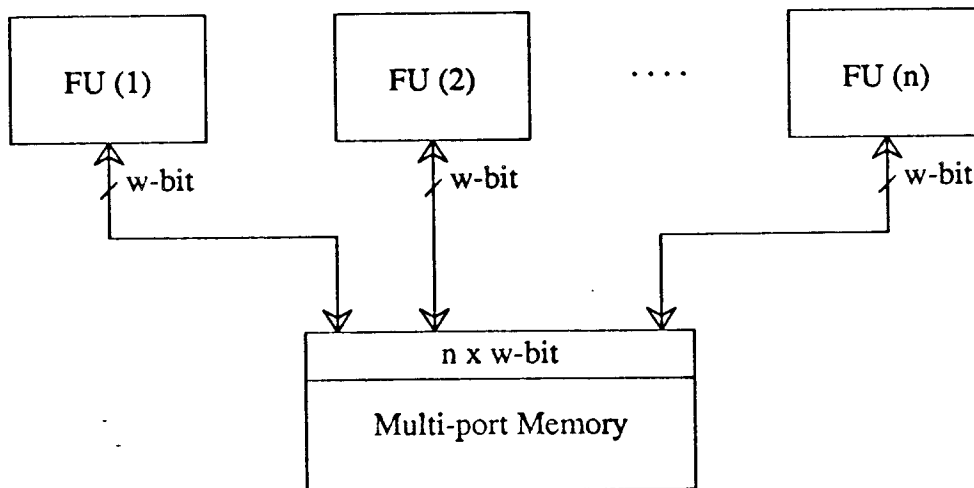
Multiple sets (or banks) of register files (Figure 3-11a) would be a better choice for the first approach. This is because the use of registers can be controlled to some extent by the programmer (or compiler), hence data consistency among register sets (banks) is not required or at least can be maintained by the software. With the multi-port memory approach (Figure 3-11b), using local memory either as a cache memory or registers would be acceptable, although a cache memory in one form or another would be a better choice since it does not require optimizations from a programmer or a compiler (also there is no need for cache consistency). The selection of cache memory type or register organization strongly depends on architectural constraints as well as area and speed requirements.

To make the best use of the local memory following the above two approaches, careful performance tradeoffs among different memory organizations should be made. The performance tradeoffs can span from the compiler design (optimal register allocation) to the actual implementation of the memory for multiple functional units. Within this research, however, only implementation tradeoffs, such as area required and access times difference among different memory designs, are considered.

Several different memory cells and analysis techniques to evaluate them have been proposed for a multi-port memory. The next section reviews some of those multi-port memory cells first, then yet another possible circuit design technique for a multi-port memory cell is proposed. Two local memory organizations, multiple sets of register files and multi-port cache implemented using the proposed memory cell, are chosen for each of the above two approaches, so that a direct comparison (multiple set versus multi-port) between two approaches can be made. The objective of this comparison is to determine a feasibility of N-port memory based on the cell proposed (where N can be greater than two), relative to the multiple set approach.



(a) Multiple register files



(b) Multi-port memory

Figure 3-11. On-chip local memory organizations for multiple functional units

3.4.3. Multi-port memory cells

Cross-coupled inverters have long been used as a static storage elements because these regenerative circuits are stable, compact, and more reliable than any other static cell. A conventional 6-transistor SRAM cell uses the cross-coupled inverters and two access transistors. The access transistor connecting the bit line and storage node is controlled by the word (select) line. A single-ported memory cell is accessed differentially from both bit lines for a read or a write per cycle. Several kinds of CMOS dual-ported memory cells (read-read, read-write, or write-write per cycle) based on this cross-coupled inverter cell have been used in microprocessor chips or other applications for many years. These are shown in Figure 3-12.

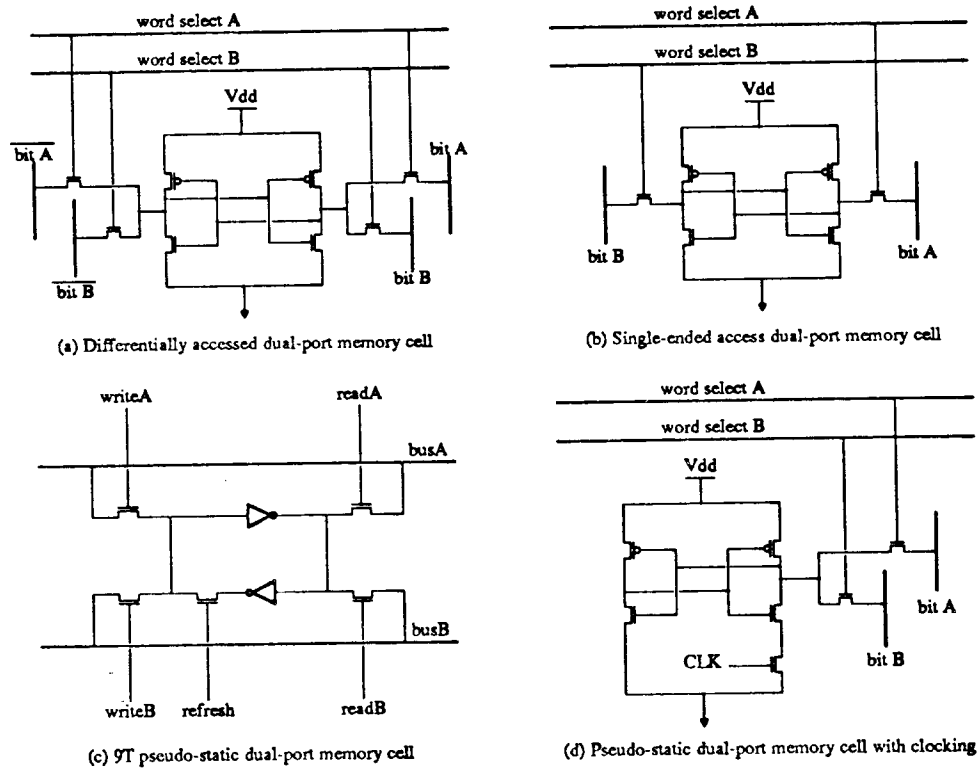


Figure 3-12. Multi-port memory cells

A single-ended access cell such as in Figure 3-12 (b) is more compact than the differentially accessed cell of Figure 3-12 (a), but requires a boosted word line to reliably write the storage node to a high state. Using precharged bit lines, each port of the single-ended access cell can perform independent read operations even with small transistors. A modified version of this single ended cell (single ended read accesses and differential write access per cycle) is used for the register files of both the SPUR CPU and the RISCII designs [She84]. The pseudo static cell design approach requires extra transistors to break the regenerative feedback action, which in turn makes the single-ended write operation performed simple and safe.

To extend a dual-ported memory cell to an n -port cell where n is greater than two, n or $2n$ (depending on the configuration) extra access devices and associated bit lines and word lines can simply be added in the same way the original access devices are connected. However, area increase due to this addition and a resulting slow access time as well as reduced safety of operation (noise margin) complicate the design of a multi-port memory cell. A number of methods have been proposed to characterize the cross-coupled memory cell in various aspects, such as simulation based analysis [O'C87] and static noise margin analysis of read/write operation with both single-ended and differential accesses [SLL87][Nak88]. Static noise margin (SNM) of a static memory cell is defined as the maximum value of static noise (dc disturbance such as offsets and mismatches due to processing and operating conditions) that can be tolerated by the memory cell (cross-coupled inverter flip-flop) itself before changing its states accidentally.

Next, I propose yet another circuit for a compact and efficient multi-port memory cell based on the 6T CMOS single-ended access cell approach mentioned above. One major drawback of the single-ended access cell is requiring a boosted word line (above V_{dd}) to perform a write operation safely. In CMOS design, bootstrap circuits of NMOS can be built, but may have some disadvantages which make them difficult to implement. First, junction breakdown is more probable because of a higher operating voltage when boosted. This becomes a more serious problem as minimum dimensions shrink with technological advances. Secondly,

the bootstrap capacitor may require a substantial amount of area because the bootstrap capacitance must be comparable to the capacitive loading of the word line.

To reduce the complexity in circuit design associated with the bootstrap driver, the voltage level on power supply line (Vdd) can be reduced to a lower level instead, such as 3 V. The word line operating voltage for a read can be at around 3 V while that for a write can be at 5 V, hence there is no need for a bootstrap driver. Reducing the voltage level on the supply line may affect the static noise margin or the stability of a cell. With a careful layout and proper ratioing of internal transistor sizes, the static noise margin can be improved.

The proposed multi-port cell using this technique is shown in Figure 3-13. This cell can provide $2n$ reads and writes (single-ended accesses for both reads and writes). For each access, separate word (row) select and bit lines are provided. Any combination of reads and

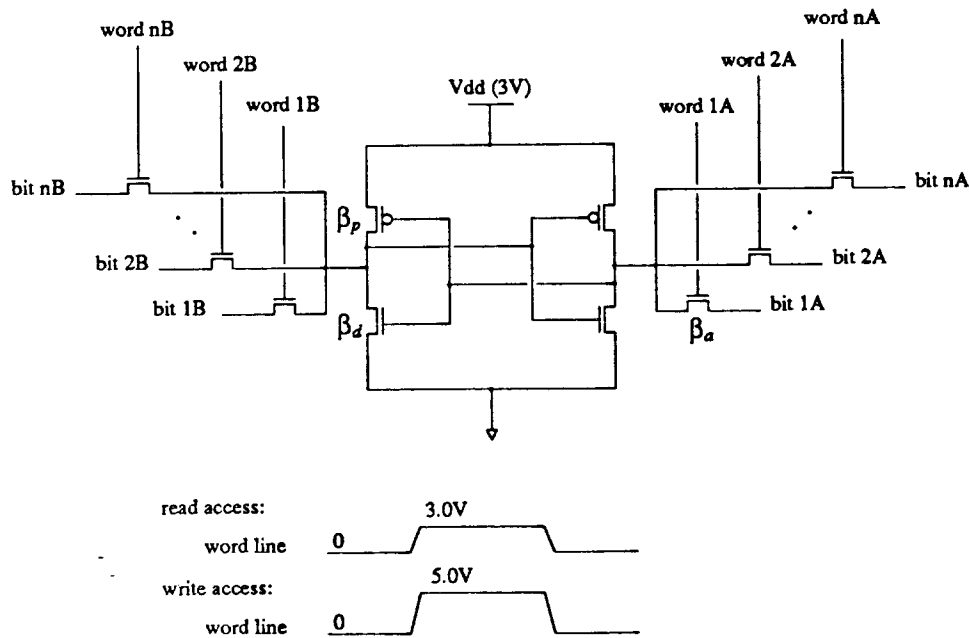


Figure 3-13. The single-ended access multi-port cell

writes (total of $2n$ operations) accesses to the array can be performed in each cycle, provided that writes at one or other ports are unambiguously resolved without corrupting the cell data. The internal forwarding scheme used in the SPUR CPU (see Chapter 2, Section 3), can be used to resolve read and write conflicts on a cell. As more ports are added, each register must sink more current to keep the access time constant with increasing capacitance on added bit lines. This, in turn, increases the cell area because all transistors must be scaled up accordingly. The maximum number of ports that can be attached to this cell with an acceptable access time and a reasonable cell area is about ten ($n = 5$).

3.4.4. Analysis and comparison

Three aspects are important for multi-port memory design: the cell area, the access time, and the stability of the cell. The cell area determines the size (density) of the local memory and often directly relates the access time of memory array. The stability of the memory cell determines the sensitivity of the memory to process tolerances and operating conditions. Considerable research has been performed in the past to analyze the stability of cross-coupled inverter cells [SLL87] [Lis86][JcF85][Nak88]. Recently, an analytical approach to modeling the stability of the flip-flop cell has been reported [SLL87]. The static noise margin, as defined in the previous section, is used as a stability measure in that approach. The analytical expression of the SNM has been further developed for various multi-port memory cells in [Nak88]. The SNM calculated in this analysis uses the expression from [Nak88] for the single-ended access memory cell.

Analysis of the proposed multi-port memory cell is done mainly by using circuit simulations (SPICE). The advantages of using simulation over just relying on static noise margin analysis are: (1) timing information is available; (2) parasitics can be taken into account; and (3) actual device design parameters are available to accurately estimate the area required. Including parasitics, such as bit line capacitance, in memory design is very important since parasitics have a dominant effect on the access time of memory. In the simulations performed

here, all parasitics are adjusted according to the configuration.

The feasibility of multi-port cache memory on a chip as compared to multiple sets of register files, depends on the memory cell used. If a multi-port memory cell is too large or too slow in comparison to the cell used for a common register file (for example, dual-ported read and single ported write), it may not be advantageous to have multi-port cache memory. When the total area of a multi-port cache memory array is less than the total area occupied by multiple register files having the same number of ports accessible from functional units, use of multi-port cache memory is justified. However, most multi-port memory cells result in much larger area than a simple, compact memory cell with a single or dual port. Using the single-ended access cell with reduced supply voltage as proposed for a multi-port memory cell can be more area-efficient than other multi-port cells, while maintaining reasonable access time and noise margin characteristics.

Table 3-3 shows several design parameters for a single-ended memory cell, when used for two or more ports to memory. Device parameters of the cell in each configuration are designed to have minimal area, fast access time, and ample noise margin. Access times are drawn from the circuit simulation (worst cases) with bit lines precharged at 3.0 V prior to a read and with bit line and word line capacitive loadings adjusted for the additional number of ports (1.0 pF initially with 2-ports). All operations are done without using a sense amplifier. Therefore, access time is directly related to the size of the pull-down transistor in the cell. Area estimates are derived from total gate area of transistors in the cell. As the number of ports increase, the cell area is dominated by pull-down transistors and access transistors. Static noise margin is calculated using transistor parameters obtained in the simulation, and plug them into the analytical expression mentioned above [Nak88]. Static noise margin is a function of operating voltages and ratios of transistors within the cell.

As the number of ports increase, the write delay also increases. This indicates that the write operation is getting more difficult as more ports are attached. Conversely, the read

Memory cell	# ports	β_p	β_d	β_a	area estimates (λ^2)	read delay (nsec)	write delay (nsec)	static noise margin (mV)
A. Register cell single-ended read differential write $V_{DD}=5.0$ $V_{read_select}=5.0$ $V_{write_select}=5.0$	2-port read or 1-port write	3/6	8/2	4/2	84 (1.0)	11.5	2.0	355
B. Multi-port cell single-ended read single-ended write $V_{DD}=3.0$ $V_{read_select}=3.0$ $V_{write_select}=5.0$	2-port	3/6	8/2	4/2	84 (1.0)	11.5	6.0	355
	4-port	3/3	16/2	6/2	136 (1.6)	9.0	8.5	524
	6-port	4/2	22/2	8/2	200 (2.4)	7.5	13.0	474
	8-port	8/2	30/2	11/2	328 (3.9)	7.2	18.0	491
	10-port	10/2	45/2	16/2	410 (4.9)	7.0	26.5	370

Table 3-3. Design parameters for multi-port memory cells

delay decreases as more ports are added. This is because the the size of the pull-down transistor needs to be increased with the number of ports, to improve the stability (static noise margin) of the cell. Reduction in the supply voltage and the word select line (when read) has little effect on the noise margin. It can be easily controlled by transconductance ratios (W/L) of transistors. Therefore, with careful transistor sizing a multi-port memory with a performance comparable to the single or dual port memory can be built using the single-ended access cell for both reads and writes with different operating voltage levels.

As mentioned in Section 2, the area used to hold a byte of data in cache differs from that used to hold a byte in a register. Cache requires tags and state bits so that it can be

managed dynamically by the hardware. To compare the multi-port memory and multiple set of register files fairly, this fact must be taken into account. However, since the purpose of this section is to examine the effectiveness of the proposed memory cell, I have only compared the the areas of cells with a different number of ports. The area (memory array only) required by multiple sets of register files can be estimated by simply multiplying the total number of ports required divided by the number of ports in the single set. As can be seen from the table, using the proposed single-ended cell we can integrate the multi-port memory in smaller area than required by the multiple set approach. To calculate the total area required for both approaches exactly, areas of other peripheral units such as decoders (approximately the same for both approaches) and multiplexors (for multiple sets of register files to route the register contents to proper functional units) or tags (also multi-port tag memory) also must be determined.

3.5. Summary

An important factor in VLSI system design is the large difference in available bandwidth between on-chip and off-chip communications. The communication bottleneck caused by the limited i/o pin bandwidth makes it desirable to pack as much functionality as possible into the restricted area of a single chip. Small local memories can improve the performance by significantly reducing the off-chip bandwidth requirement. In a single-chip microprocessor, silicon area is one of the scarcest resources, and designers must use it efficiently for given constraints to maximize the performance. Therefore, the organization of local memory must be effective and memory density must be maximized at a given silicon area.

In this chapter, two memory design techniques that can improve the performance without necessarily increasing the use of scarce silicon area, are presented. Traditionally, due to reliability concerns, only static memories have been used on a microprocessor chip. Dynamic memories offer more bits per unit area than static memories, but fundamental limi-

tations such as refreshing overhead, have prevented their use on a microprocessor chip. Using the selective invalidation technique proposed here can eliminate the refreshing overheads of dynamic memories, if used as a cache memory (read-only or write-through cache). This makes the replacement of static memory with high density dynamic memory possible, and results in better use of scarce silicon area. Trace-driven simulations show an effectiveness of this scheme over a simple invalidation scheme.

When multiple functional units are used to increase the performance by parallel execution, the demand for a higher bandwidth between functional units and local memory rises rapidly. Since a multi-port memory is prohibitively expensive, time-shared accesses to a single-port memory have been used when multiple accesses are necessary. A single-ended access memory cell operated at reduced voltage levels can be as safe and fast as differentially accessed cells. When this cell is used to implement n -port memory ($n > 2$) it can result in a total memory array area smaller than that of multiple register files with the same number of ports available to the functional units.

3.6. References

- [AAA87] *AM29000 User's Manual*, Advanced Micro Devices, 1987.
- [ACH87] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz and J. Hennessy, "On-chip Instruction Caches for High Performance Processors", *Proc. of Conf. on Advanced Research in VLSI*, Stanford, CA, March 1987.
- [DiM82] D. R. Ditzel and H. R. McLellan, "Register Allocation for Free: The C Machine Stack Cache", *Proc. of Symposium on Architectural Support for Programming Language and Operating Systems*, Palo Alto, CA, March 1982, 48-56.
- [DMB87] D. R. Ditzel, H. R. McLellan and A. D. Berenbaum, "The Hardware Architecture of the CRISP Microprocessor", *The 14th Annual International Symposium on Computer Architecture*, Pittsburgh, Pennsylvania, June 2-5, 1987, 309-319.

- [Goo83] J. R. Goodman, "Using Cache Memory to Reduce Processor Memory Traffic", *Proc. Tenth International Symposium on Computer Architecture*, Stockholm, Sweden, June 1983.
- [GoH86] J. R. Goodman and W. Hsu, "On the use of Registers vs. Cache to Minimize Memory Traffic", *Proc. 13th International Symposium on Computer Architecture*, Tokyo, Japan, June 1986, 375-384.
- [Hen81] J. Hennessy et al., "MIPS: A VLSI Processor Architecture", Technical Report No. 223, Computer Systems Laboratory, Department of Electrical Engineering & Computer Science, Stanford University, Stanford, November 1981.
- [Hil85] M. D. Hill, "", DineroIII Documentation, Unpublished Unix-style Man Page, U.C. Berkeley, October 1985.
- [Hil87a] M. D. Hill, *Private Communication*, U.C. Berkeley, July, 1987.
- [Hil87b] M. D. Hill, "Aspects of Cache Memory and Instruction Buffer Performance", Ph.D. Thesis, CS Division EECS Dept., UC Berkeley, November 1987.
- [Hil86] M. D. Hill et al., "Design Decisions in SPUR", *IEEE Computer* 19, 10 (November, 1986), 8-24.
- [Hor87] M. Horowitz et al., "MIPS-X: A 20-MIPS Peak, 32-bit Microprocessor with On-chip Cache", *IEEE Journal of Solid State Circuits* SC-22, 5 (October 1987), 790-799.
- [JeF85] R. C. Jeager and R. M. Fox, "Phase Plane Analysis of the Upset Characteristics of CMOS SRAM Cells", *Proc. of 6th Biennial Univ./Government/Industry Microelectronic Symp.*, Auburn, AL, June 1985.
- [Joo85] R. Joobbami, "Weaver: An Application of Knowledge-Based Expert Systems to Detailed Routing of VLSI Chips", Ph.D Dissertation, Dept. of EECS, Carnegie-Mellon University, July, 1985.
- [Kat83] M. G. H. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI", Ph.D. Thesis, CS Division, EECS Dept., University of California, Berkeley, October 1983.

- [Kuc78] D. J. Kuck, *The Structures of Computers and Computations*, John Wiley & Sons, New York, 1978.
- [Kun86] H. T. Kung, "Memory Requirements for Balanced Computer Architecture", *Proc. 15th International Symposium on Computer Architecture*, Tokyo, Japan, June 1986, 49-54.
- [LeS84] J. F. K. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer", *IEEE Computer* 17, 1 (January 1984), 6-22.
- [Lis86] F. J. List, "The Static Noise Margin of SRAM Cells", *Digest of Tech. Papers, ESSCIRC (Delft, The Netherlands)*, September 1986.
- [Man87] T. Mano et al., "Circuit Techniques for 16Mb DRAMs", *ISSCC Digest of Technical Papers*, New York, NY, February 1987.
- [McC84] E. McCreight, "The DRAGON Computer System: An Early Overview", *NATO Advanced Study Institute on Microarchitecture of VLSI Computers*, Urbino, Italy, July, 1984.
- [McD88] G. D. McNiven and E. S. Davidson, "Analysis of Memory Referencing Behavior for Design of Local Memories", *Proc. 15th International Symposium on Computer Architecture*, Honolulu HI, June 1988, 56-63.
- [MMM84] *MC68020 Technical Summary*, Motorola Semiconductors, 1984.
- [MMM86] *MC68030 Technical Summary*, Motorola Semiconductors, 1986.
- [Nak88] Y. Nakagome, "Multiport Memory Design Consideration for Parallel Execution CPU Architectures", Electronics Research Lab. Technical Report, Department of EECS, U.C. Berkeley, November 1988.
- [O'C87] K. J. O'Connor, "The Twin-port Memory Cell", *IEEE Journal of Solid State Circuits* sc-22, 5 (October 1987), 712-720.
- [Rad82] G. Radin, "The 801 Minicomputer", *Proc. SIGARCH/SIGPLAN Notices Symposium of Architectural Support for Programming Language and Operating Systems*, ACM, Palo

Alto, CA, March 1982, 39-47.

- [SLL87] E. Seevinck, F. J. List and J. Lohstroh, "Static-Noise Margin Analysis of MOS SRAM cells", *IEEE Journal of Solid State Circuits* sc-22, 5 (October 1987), 748-754.
- [She84] R. Sherburne et al., "A 32-bit Microprocessor with a Large Register File", *IEEE Journal of Solid-State Circuits* SC-19, 5 (October, 1984).
- [SBN82] D. P. Siewiorek, C. G. Bell and A. Newell, *Computer Structures: Principles and Examples*, McGraw Hill, New York, 1982.
- [Smi82] A. J. Smith, "Cache Memories", *Computing Surveys* 14, 3 (September 1982), 473-530.
- [SmG83] J. E. Smith and J. R. Goodman, "A Study of Instruction Cache Organizations and Replacement Policies", *Proc. 10th International Symposium on Computer Architecture*, June 1983, 132-137.
- [THL86] G. S. Taylor, P. N. Hilfinger, J. R. Larus, D. A. Patterson and B. G. Zorn, "Evaluation of the SPUR Lisp Architecture", *Proc. Thirteenth International Symposium on Computer Architecture*, Tokyo, Japan, June 1986.
- [Ter83] C. J. Terman, "Simulation Tools for Digital LSI Design", Tech. Report No.304, Laboratory for Computer Science, MIT, September 1983.
- [Yam84] J. Yamada et al., "A Submicron 1 Mbit Dynamic RAM with a 4-bit-at-a-time Built-in ECC Circuit", *IEEE Journal of Solid State Circuits* sc-19, 5 (October 1984).
- [ZHH87] B. Zorn, P. Hilfinger, K. Ho and J. Larus, "SPUR Lisp: Design and Implementation", Technical Report UCB/Computer Science Dpt. 87/373, U.C. Berkeley, September 1987.

4

Control Design Alternatives

4.1. Introduction

In a microprocessor design, many of the modules can be designed using regular and straightforward design styles (ROM, RAM, and bit-sliced data path). However, the control unit is often the 10% of the chip area that takes 90% of the time to design. Alternatively, if a fast but simplistic approach is used for design, a very efficient implementation will result. This chapter considers the automated synthesis of digital logic and especially the synthesis of the types of random logic seen in the control unit for full-custom VLSI microprocessors.

A common approach to regularizing the design of random control logic employs a structured logic element, such as PLAs, to implement the microprocessor's control. Automatic PLA synthesis tools have been widely used for many years. Recent developments in integrated circuits (IC) CAD offer VLSI designers a variety of implementation choices

which have not been available in full-custom VLSI design. In particular, multi-level logic synthesis and optimization techniques [Seg87][Bra87] allow combinational logic to be mapped into different design styles (in multi-level form) such as standard cell, gate-matrix [LoL80], and gate-array designs, in addition to the conventional implementation style based on PLAs. However, the relative merits of these alternatives for full-custom VLSI microprocessor design have not been well established. This chapter focuses on the evaluation of these alternatives for microprocessor control designs. The results should be useful as a guide for future microprocessor development.

This chapter will begin with a review of control design strategies. Section 2 presents two general approaches to implement control units in microprocessor, microprogrammed control and hard-wired logic implementation. Microprogrammed control design has been popular since there are many computer-aided design tools help implementing it automatically. Advances in CAD systems and recent developments in computer architecture, such as reduced instruction set computers (RISC), suggest that a fast, hard-wired implementation of control logic is now affordable and highly desirable for a high performance microprocessor. Section 3 discusses the automated synthesis of control functions and presents alternative implementations of the hard-wired control logic. Section 4 evaluates several prototypes implemented using the alternative methodologies presented in Section 3. Several examples from the SPUR design are used in this investigation. Section 5 summarizes the results obtained from the study.

4.2. Microprocessor control

Microprocessors generally consist of two parts: an execution unit and a control unit. The execution unit contains the resources needed to execute the microprocessor's instructions which include the general purpose registers; the arithmetic and logical unit (ALU); shifter, and instruction counters. The control unit "runs" the execution unit telling it what to do when.

Two general approaches to the control unit design are reviewed and compared in this section. The objective is to compare synthesis systems of two general approaches for the automatic generation of control logic. Microprogrammed control provides a flexible implementation using fast on-chip memory to store control instructions (microcode), but often requires several cycles to execute one instruction. Each instruction is implemented in several microinstructions that must be fetched from the storage (ROM) and decoded in each cycle. A hard-wired implementation can perform better than a microprogrammed control because each instruction is directly interpreted in hardware and can be executed in a single CPU cycle. However, the hard-wired design approach has been prohibitively expensive and inefficient, especially for a large and complex instruction set [Anc83]. The problems are largely due to increased complexity as an instruction set becomes richer and more features are required to implement it. This trend, however, is changing because of newly developed CAD tools for hard-wired logic synthesis.

Advanced CAD systems, such as those for multi-level logic synthesis and optimization, along with automatic layout generation systems, have made it possible for VLSI designers to re-consider the hard-wired implementation. In microprogrammed implementation, the control functions are described in special high-level programming language, then compiled down to microcode via various computer aids and computer-aided optimizations. More recently, a similar approach has become available for the hard-wired implementations (see Figure 4-1). A designer states the required behavior of the control functions in a hardware description language, such as ISP'. This description is then automatically *synthesized* into lower levels in design abstraction hierarchies (e.g. logic gates or layout). This automated design process for hard-wired implementation is analogous (Figure 4-1) to the design process used in the microprogrammed implementation and makes the hard-wired implementation as efficient and flexible as microprogrammed control.

4.2.1. Microprogrammed control

The function of the control unit in a microprocessor is to execute sequences of micro-operations for the successful completion of the processor's instructions. The control function that specifies a micro-operation is a binary variable. During any given time interval, certain micro-operations are to be active while all others remain idle. Thus the micro-operation steps within each time interval can be represented by a string of 1's and 0's called a "control word" or "microinstruction." For a control unit in which micro-operation sequences are stored in a memory such as this form is called *microprogrammed control*. Each microinstruction may contain as many bits as there are control points in the processor to control a variety of components operating in parallel (horizontal microinstructions). The number of control bits in a microinstruction word can be reduced by grouping mutually exclusive variables into fields and encoding the k bits in each field to provide 2^k micro-operations (vertical microinstructions). Each field then requires a hardware decoder to produce the corresponding control signals.

The complexity of the microprocessor control is due to the many different micro-operations performed in a given time sequence. Microprogrammed control is an elegant and systematic method for generating the micro-operation sequences, especially for a large and complex instruction set. In practice, the use of microprogrammed control has been tied to the architecture or instruction set to be implemented [Hop83]. It is generally easier to implement high level complex instruction sets in microcode, although it may result in slower implementation than hard-wired approach. Simple or reduced instruction sets do not normally require a microprogrammed control. Instead, a fast and simple hard-wired control is used to implement a single cycle execution of all instructions.

Microprogramming provides several advantages such as permitting structured approach to control unit design, which greatly improves debugging and tailorability. Moreover, it can be extended easily to include additional instructions beyond the original instruction set, or can emulate other instruction sets. It does so without modifying the existing hardware, other

than the control unit. With the continuing growth of semiconductor processing technology (especially ROM and RAM designs on a microprocessor chip), the microprogrammed control can be a cost-effective implementation for richer and more complicated instruction set microprocessors [Ber81]. Many microprocessors exemplify the microprogrammed control such as the Motorola MC680x0 [MMM84], the Intel 80386 [Gel87], and the National NS32532.

Although it is an elegant and flexible approach to designing a complicated control unit, writing the microprogram has remained a very difficult task. The problem is harder when there are many more potential microinstructions than there are regular processor instructions. Many computer aids have been developed, such as the compiler and debugger for writing the microprogram. These help the microprogrammed control design efficient and error-free. Microprogramming is still one of the efficient ways to design microprocessor control, but new alternatives available from the automated hard-wired control synthesis must be carefully evaluated and compared to the microprogramming.

4.2.2. Hard-wired control

The term "hard-wired control" refers to an implementation technique for a microprocessor control unit, in which conventional logic gates, such as NAND or NOR, steer the master clock phases to the control point. Each control point is driven by a gate with inputs that determine the conditions under which that control point is to be activated. Thus the process of control unit design consists of listing the control points to be activated as a function of the master clock phases and decoding the instruction's opcode. The logic optimization technique can be used to reduce the number of gates involved.

The control unit of a simple instruction set microprocessor can best be implemented in hard-wired logic. The hard-wired design will be faster than the microprogrammed design built from the same technology, since the former does not require the overhead of fetching and decoding microinstructions (micro-sequencing also complicates the design). Recent

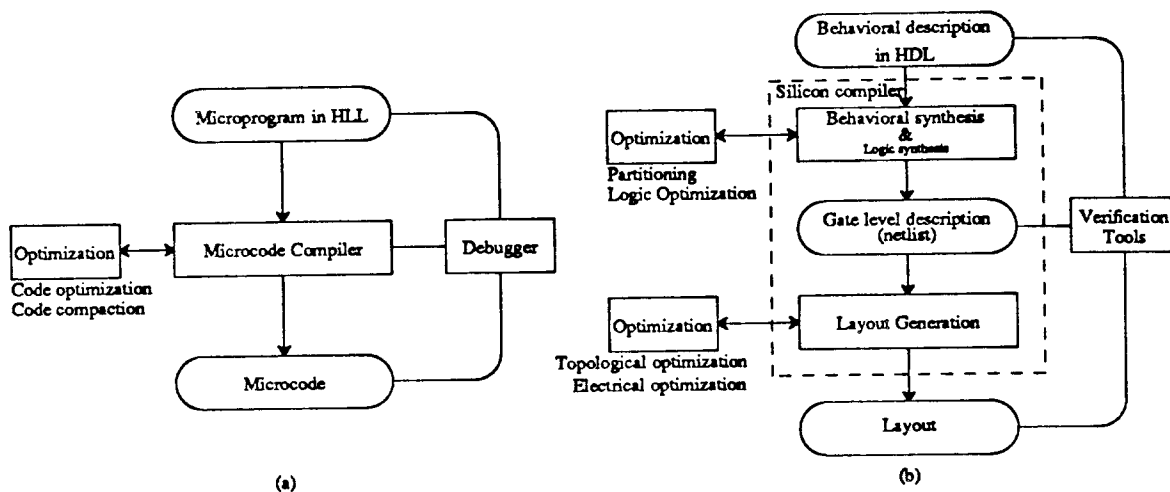


Figure 4-1. Design processes of microprocessor control for (a) microprogrammed control and (b) hard-wired control.

VLSI RISC microprocessors, such as SPARC [NaA88] and MIPS R3000 [Mou], employ hard-wired control design to achieve the fastest possible cycle time.

A version of hard-wired control well-suited to VLSI design has been the programmed logic array (PLA). A two-level representation of logic functions can be efficiently and automatically implemented with a PLA. Several optimization techniques, such as logic minimization [Bra87] and topological optimization [DeS83], further improve the quality of the design. However, certain multi-level logic functions do not map well into PLAs. In this case a multi-level representation may lead to a better implementation with a reduced gate count and a smaller area than the two-level PLA implementation. A comparable or shorter delay path is also possible via an optimum allocation of gates (technology mapping). In fact, a two-level logic representation can be seen as a special case of multi-level representations. Therefore, a general synthesis system for control logic design should offer multi-level synthesis tools which are able to select a two-level implementation whenever it is more effective in terms of area and/or speed. The multi-level logic can be mapped into many different

design styles such as the following: standard cells; gate matrix; Weinberger array [Wei67]; and gate arrays. Several CAD systems are being built for the design of random control logic especially in multi-level representation. In the next section new controller design strategies using these systems are presented and closely examined.

4.3. Alternative implementations

A large spectrum of design styles for the VLSI microprocessor has evolved, offering wide ranges of expected turn-around time, improved performance and reduced design effort. Most microprocessors use PLAs to implement the combinational part of the control unit in two-level logic representation. Gate-matrix and Weinberger arrays (usually for NMOS design) are array structured logic for standardized layout of multi-stage combinational logic networks. Semi-custom design styles such as gate arrays and standard cell based designs, aim at minimal design effort and faster turn-around time. The performance of semi-custom designs has not matched its forerunners, but has the potential to be greatly improved through recently-developed multi-level logic optimization techniques and automated layout generation systems associated with semi-custom designs.

In this section, I first discuss a generalized CAD methodology for a hard-wired implementation of a microprocessor control unit, then present strategies for three different implementation styles which can actually be used in implementing real world examples. Three styles, the PLA-based, the standard cell-based, and the gate-matrix based, are chosen for the following reasons: (1) These can be easily adapted and mixed with full-custom design styles; (2) designs generated using these styles are more silicon efficient and perform better than others; and (3) reliable CAD tools are readily available for these styles. The following section evaluates, in various aspects, different implementations of these styles as applied to examples from the SPUR design.

4.3.1. Automatic synthesis of control logic

A number of strategies are employed to deal with complexities in VLSI design. One most frequently used is to divide the design into parts such that each can be implemented using the most appropriate strategy (Figure 4-2). In general, designing the control part of the microprocessor is quite different from other parts such as data path or local memory. Due to its complexity, control unit design requires many iterations, thus portions of its design process need to be automated. Furthermore building a complex control logic, especially in hard-wired implementation, requires optimizations at each step of the design process, e.g. minimizing the area required while reducing the delay. A generalized design process for the control logic involves three steps [NeS86]: behavioral synthesis, logic synthesis and optimization, and layout generation (Figure 4-3).

Behavioral synthesis is a translation from a behavioral description of the control hardware to a detailed functional description such as register transfer level. The difficulty in this step stems from the many constraints, design objectives, and design configurations to consider. Logic synthesis generates a logic network from a functional description of combinational logic. One of the primary difficulties in this step is in discerning which sections of the description imply pure combinational logic, and which parts are intended to be sequential logic.

The logic synthesis step takes a functional description as an input and creates appropriate logic equations to implement the described logic. The output of the logic synthesis is minimized and mapped into logic structures or gates in the logic optimization process. The process of optimizing combinational logic is divided into two sections - a technology independent part in which logic optimization is performed, and a technology mapping phase in which the selection of the gates to implement the function is made. The only implementation style automatically synthesized from a high level description has been the PLA implementation with a traditional two-level logic optimization.

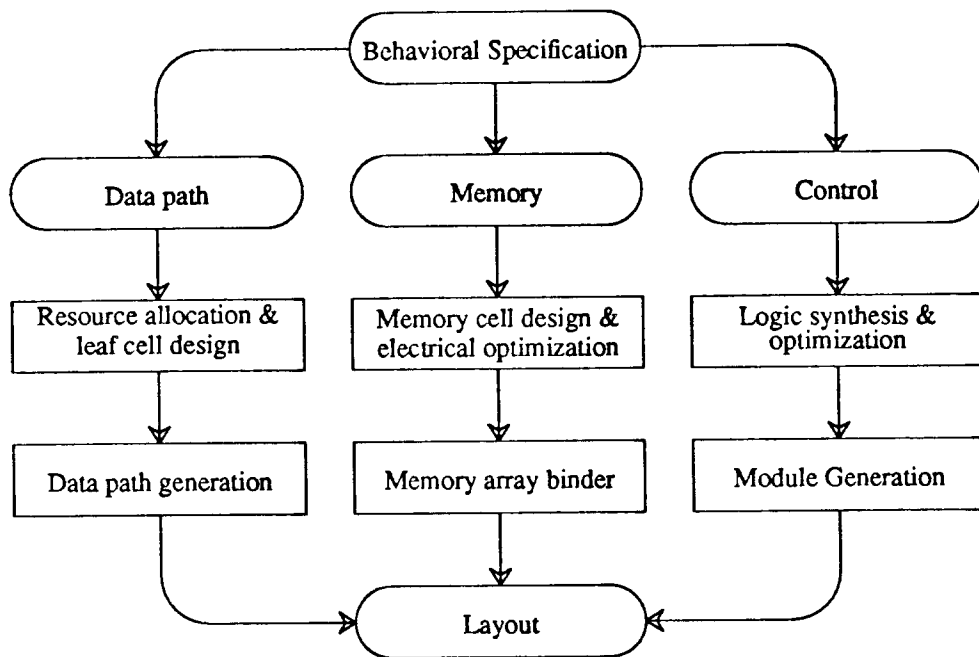


Figure 4-2. Separation of design methodologies

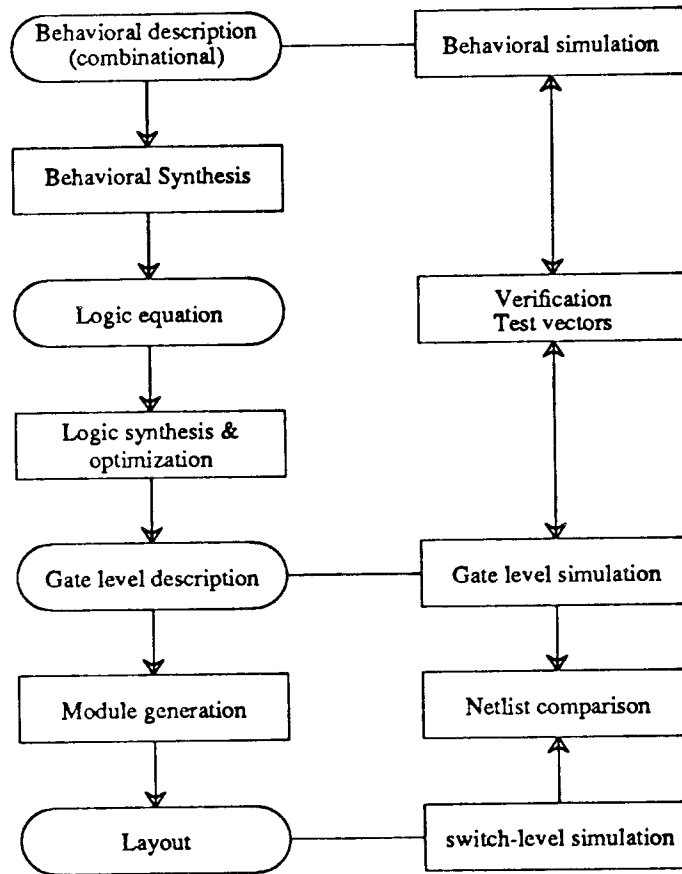


Figure 4-3. Design synthesis path of the microprocessor control

Implementing logic in multiple level form has several advantages. By manipulating multiple level logic, one can optimize the logic for minimum delay or minimum area [Bra87]. As the complexity of logic increases, PLA implementation suffers from declining performance *and* an increasing area requirement. This is not necessarily the case for multiple level implementation where tradeoffs between speed and area can easily be made. Multiple level logic can be implemented using a broad spectrum of design style such as standard cell, gate array, or other array structured logic elements, gate matrix and Weinberger array. Automatic layout generation tools for such technology are mature enough to generate a high

quality layout when optimization of logic is properly done.

4.3.2. PLA based control design - A traditional approach

PLAs are two dimensional array logic implementing a canonical sum of products two-level combinational logic function. The PLA consists of two planes, the AND plane and the OR plane. The AND plane maps the primary inputs to the product terms while the OR plane maps the product terms into the outputs. In practice both these planes are implemented as NOR structures with an arbitrary number of inputs.

PLAs are among the most popular structures for the implementation of two-level logic functions. Most of the recent microprocessors include PLAs in the control part. Because of their regular structures, PLAs can be laid out automatically. Many PLA layout generators have been built based on simple mapping of the Boolean equations into the layout of the PLA. To obtain an effective design, several optimization techniques are necessary. They include the following: logic minimization; topological optimization; and layout and circuit optimizations.

Logic minimization both reduces the area occupied by the PLA and improves its electrical performance by minimizing the number of product terms required. Once the logic minimization is completed, topological optimization can be performed to minimize the core array of the PLA. The topological optimization itself does not contribute directly to the implementation of the logic functions. The objective of the topological optimization is to "fold" rows and/or columns of the PLA planes such that multiple logical rows or columns can share a physical row or column [DeS83]. This reduces the total number of rows and columns required, hence minimizing the area of the PLA. Layout and electrical optimizations [Hed85] concentrate on the performance of the large PLA. The signal delay through a large PLA can be minimized by transistor sizing, laying out interconnects in metal layers, or using a sense amplifier between the planes and at the output.

A microprocessor control unit not only includes random combinational logic, but also requires sequential logic elements, such as latches, in order to implement a control block with the finite state machines. PLA-based finite state machines have been used in the design of several microprocessors. The finite state machine uses a PLA to implement the combinational part of the logic, and the outputs (state bits) of the PLA are fed back to the inputs of the PLA via clocked registers. Other logic blocks with both combinational and sequential parts can be implemented in a similar manner. However, the separation of combinational parts and sequential parts of the control logic is yet to be automated. As described in Chapter 2, the control unit of the SPUR CPU followed this methodology.

4.3.3. Standard cell-based control design

The standard cell approach to VLSI chip design provides the designer a quick, flexible design. It may be less dense than a full-custom designed chip, but it performs far better than gate array designs. A standard cell library includes simple cells such as INVERTER, NAND, and NOR gates as well as complex gates such as the AND-OR-INVERT gate and various flip-flops. The cell library greatly simplifies the automated synthesis path by isolating technology dependencies from the synthesis system. All cells in the standard cell library have identical height and variable width, depending upon the complexity and size of each cell. Each cell contains a completely interconnected function and can be abutted to other cells without any adjustment. The automated synthesis system treats the standard cell as an abstract object like a bounding box with terminals, and thus placement and routing of cells becomes technology independent.

The layout synthesis for standard cell-based design consists of three parts: (1) design of the standard cell library, (2) logic minimization and technology mapping (selection of cells), and (3) placement and routing of cells. The cell library is usually provided by the silicon foundries, but can be custom designed for specific needs. The size of the library is important to achieve an optimal implementation [Keu87]. Both logic minimization and optimal tech-

nology mapping [Det87] rely heavily on different types of standard cells available in the library. The impact of the cell library on the final design will be investigated in the following section.

4.3.4. Gate matrix based control design

The gate matrix layout style [LoL80] utilizes the configuration of a matrix composed of intersecting rows and columns to provide transistor placement and interconnections (Figure 4-4). The matrix format structure, which is orderly and regular, gives high device packing density and allows ease of checking for layout errors. The columns of this matrix, implemented in polysilicon, serve as a transistor gate and interconnection. The rows are implemented in diffusion and form transistors with a column at the intersection. The pitch of the columns and rows are determined by the minimum separation allowed between polysilicon lines with contacts and transistors (diffusion) or interconnections (metal), respectively.

The automatic synthesis path for the gate matrix layout style from a logical description is very similar to that for PLAs. Logic minimization is done first, and the optimized logic equation is mapped to the gate matrix. Like the folded PLA, topological optimization [DeN87] can be performed to reduce the total number of rows and columns required. However, unlike PLAs, gate matrix can be used to map multi-level logic representation and implement the mixed combinational and sequential circuits. Latches or registers can be laid out and mixed with combinational parts inside the gate matrix.

With both multi-level logic minimization and topological optimization, gate matrix provides very high packing density for multi-level logic functions, but resulting performance may not be as good as with other implementations. This is because the size of the transistor is fixed and several transistors in a series connection can be laid out inefficiently with very high interconnect parasitics. To obtain an optimal implementation with gate matrix layout, both an optimal partitioning of logic functions and electrical optimization (or transistor sizing) are necessary.

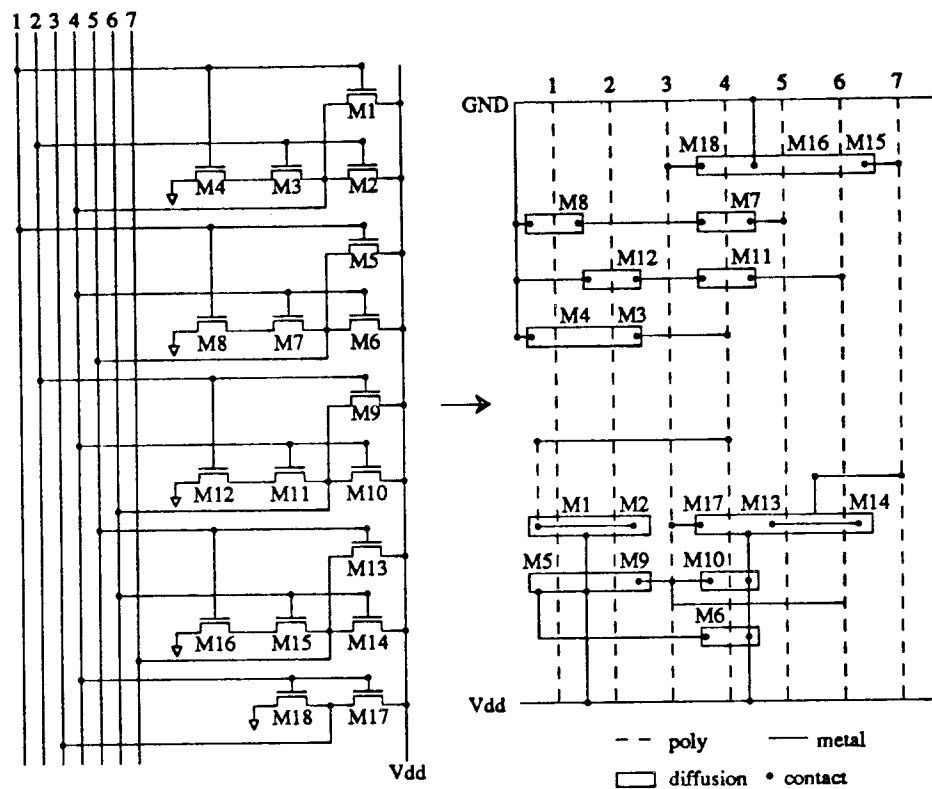


Figure 4-4. Gate matrix layout (from [LoL80])

4.4. Evaluation and comparison

4.4.1. Method of evaluation

To evaluate alternative implementations rigorously, correct performance measures must be used. The primary and more quantitative parameters are critical path timing, area, power consumption, and design time. The secondary and more qualitative parameters may be the flexibility and the testability. Flexibility measures the ease of changing design, depending on overhead associated with regenerating the layout from the altered description. Testability is hard to measure unless built-in test structures are incorporated. For purpose of evaluation, I

use the following parameters: critical path timing; area required; estimated power consumption; design time; and flexibility. Some parameters can be gathered directly from the resulting implementation, while others are based on qualitative judgement.

The examples used in this experiment are from the SPUR design:

- (1) *SPUR Instruction Unit Controller* (*iu_ctr*) controls the fetching and prefetching of the instruction cache as well as handles a miss.
- (2) *SPUR CPU Master Control* (*master_ctr*) controls the pipeline execution of CPU instructions, provides both cache controller and floating point unit interfaces, and handles traps and interrupts.
- (3) *Cache Controller Sequencer* (*cc_seq*) implements the processor cache control functions including access requests from the CPU, read and writes on cache memories, and translates a virtual address to a physical address.

The SPUR designs were full-custom with the PLA-based control implementations, thus only other styles needed to be re-implemented. The design processes for different styles is depicted in Figure 4-5. All designs begin at the same abstraction level, the functional description written in BDS. A set of synthesis tools built around the OCT design database [Har86] at UCB are used to create the layout. Logic synthesis and optimization steps are identical for all implementations. The hardware description of the control function is translated into logic equations and optimized using CAD tools called *bdsyn* and *mis*, respectively. The optimized logic is then mapped to different layout styles within *mis*, and the final layout is generated using appropriate layout generation tools. For array-structured logic, topological optimization is performed to further improve the design.

The layout generation tools used include *Wolfe*, a standard cell place and route system which uses *Timberwolfe-SC* for placement and *YACR* for routing, *GEM* (gate-matrix layout), and *MPLA*. The final layout is transformed into another database called *magic* [Ous85], to be extracted and evaluated. Converting the database assures fair comparisons with already exist-

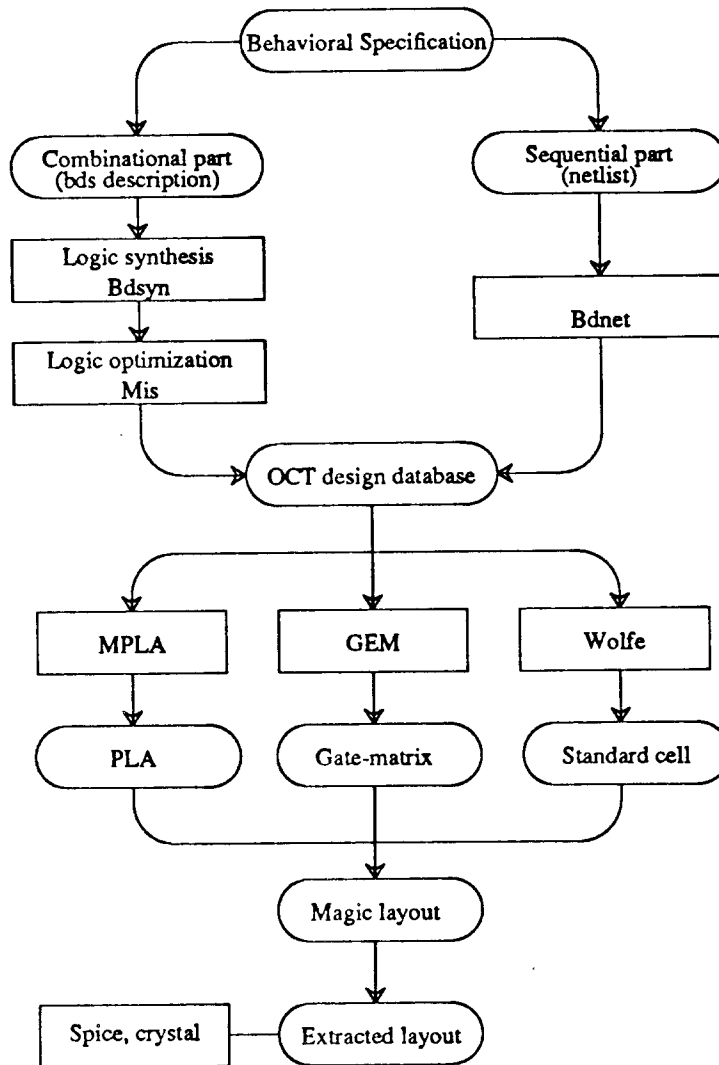


Figure 4-5. VLSI design environment with OCT

ing full-custom PLA-based implementations in magic format. Electrical performance is measured by running timing analysis tools, *crystal* and *spice*, on the extracted layout. Power consumption is estimated using extracted capacitances of all switching nodes.

4.4.2. Results of evaluation and comparison

Results from the evaluations are summarized in Tables 4-1, 4-2, and 4-3. Both the SPUR IU control (*iu_ctr*) and the master control (*master_ctr*) of the SPUR CPU have four different versions: One full-custom implementation with PLAs, two versions of standard cell-based design with different cell libraries, and one gate-matrix implementation. The SPUR CC sequencer (*cc_seq*) has five different versions. One extra version is implemented using electrically optimized PLA. Sense amplifiers are used in the middle of the PLA planes and also at the outputs of the PLA. The sense amplifier reduces the voltage swing on the product term and output lines where parasitic loadings may slow down the propagation delay. Two standard cell libraries were built for this experiment and cells in the libraries are listed in Table 4-4. LIB1 consists of only seven simple gates, while LIB2 includes more complex gates like XOR and AND-OR-INVERT gates.

A. Critical path timing

The most important performance parameter of the control design is the critical path timing, as it often determine the cycle time of the microprocessor. To obtain an accurate timing performance, all circuits are analyzed by extracting the layout with all parasitics taken into account. For the *iu_ctr*, standard cell-based designs perform very close to the full-custom design, while the gate-matrix version would still satisfy the required timing. Consistent results are also observed in different *master_ctr* implementations. The *cc_seq* consisting of combinational logic only, on the other hand, shows different results. PLA implementation of *cc_seq* without electrical optimization (using sense amps) performs much worse than standard cell-based implementations. This indicates that for a large combinational logic network, the multi-level implementation is better than the two level PLA implementation, where parasitics will have a dominant effect on the critical path. The gate-matrix version of *cc_seq* far exceeds the timing requirement. For reasonable performance with gate-matrix design, a proper partitioning of the logic or electrical optimization is crucial. This can be seen in the

gate-matrix versions of the iu_ctr and master_ctr, where proper partitionings were already made.

	IU_CTR1	IU_CTR2	IU_CTR3	IU_CTR4
Design Style	Full custom with PLA tools	Standard cell	Standard cell	Gate matrix
No. of cells designed	9	17	7	none
Total No. of gates	6 PLAs (95/30) 3 random logic	169 (13)	193	824 transistors
Area	1300x1400 (1.00)	1260x870 (.602)	1220x880 (.590)	1250x1470 (.99)
Total Capacitance (switching)	17.2 pF	31.7pF	32.8pF	13.3pF
Power Estimation	30 mW	Less than 10 mW	Less than 10 mW	Less than 5 mW
Timing	(nsec)	(nsec)	(nsec)	(nsec)
phi1	10.10	12.10	13.35	19.25
phi2	11.90	9.85	11.35	10.35
phi3	9.30	7.90	9.55	11.65
phi4	12.15	11.50	12.40	16.50
Design time	4 wks	2wks	2wks	2wks

Table 4-1. SPUR IU control

	MC1	MC2	MC3	MC4
Design Style	Full custom with PLA tools	Standard cell	Standard cell	Gate matrix
No. of cells designed	14	17	7	none
Total No. of gates	5 PLAs (133/67) 9 random logic	384 (38)	417	2310 transistors
Area	1920x3070 (1.00)	2570x1530 (.667)	2380x1540 (.622)	2680x2410 (1.10)
Total capacitance (switching)	44.45pF	82.90	83.97pF	45.32pF
Power Estimation	50 mW	Less than 30 mW	Less than 30 mW	Less than 15 mW
Critical path timing (nsec)	20.50	19.00	20.85	26.70
Design time	10 wks	4wks	4wks	4wks

Table 4-2. SPUR master control

The effect of library size on the timing performance of the standard cell implementation is very small compared to the difference in total gate counts. This is because using complex gates with the large library may reduce the total gate counts but not necessarily reduce the delay times (since complex gates are slower than simple gates). In fact, the conversion ratio of simple gates to a complex gate is about two to three for all implementations, and complex gates are about two to three times slower than simple gates. This proves that a large set of

library cells does not necessarily optimize the performance. Therefore, without spending a great deal of time to design and optimize a comprehensive cell library, one can obtain a high performance implementation with a small number of library cells. It is also possible to further improve the performance of the standard cell version by optimizing the cells for a particular design. If the library is small, optimizing and maintaining the cell library can be greatly simplified. In turn, this reduces overall design time. Logic minimization criteria may also affect the timing performance. In this experiment, the same logic minimization steps were used in all different implementations.

	SEQ1A	SEQ1B	SEQ2	SEQ3	SEQ4
Design Style	PLA with S/A	PLA	Standard Cell	Standard Cell	Gate matrix
No. of cells designed	1	1	17	7	none
Total No. of gates	207 p-terms 36 outputs	207 p-terms 36 outputs	491 (38)	526	2526 transistors
Area	2060x3610 (1.00)	1240x2130 (.355)	1730x3160 (.735)	1750x3180 (.748)	1940x5360 (1.40)
Total capacitance (switching)	254.37pF	119.91pF	98.82pF	101.98pF	26.64pF
Power Estimation	60mW	60mW	Less than 20 mW	Less than 20 mW	Less than 5 mW
Critical path timing (nsec)	18.00	46.00	24.50	26.70	90.10

Table 4-3. SPUR CC Sequencer

B. Area

In a single-chip microprocessor, chip area is one of the scarce resources. Array-structured logic elements offer a very high integration of transistors at a given silicon area. However, if multiple units of such a structure are interconnected, the layout might not have optimum area efficiency. This can be seen in the area difference in the PLA-based design and the standard cell-based design. Optimizing placement and routing CAD tools produce better results with many smaller standard cells rather than a few large PLAs or gate matrix arrays. Even with topological optimization, the gate-matrix and PLA-based design still require larger areas.

A folded PLA can have a smaller core array area, while not necessarily minimizing the area occupied by the entire PLA layout. This is because the I/O buffers now must be attached to both sides of each array. As previously mentioned, to obtain a reasonable performance out of gate-matrix design, the logic being designed needs to be partitioned into multiple gate-matrices. Without partitioning, the area required by large logic network may increase unreasonably, as evidenced by the `cc_seq`. However, partitioning also imposes a similar problem to that encountered with multiple PLAs. Gate-matrix design with folded columns and rows minimizes the matrix area itself, but overhead associated with interconnects of multiple gate-matrices reduces the area efficiency.

As for the timing performance of the design, the effect of the library size on the area is negligible. The total areas required for `iu_ctr` and `master_ctr` are actually less with the small library than the large one. The same reason for the timing performance also applies here. Complex gates can replace two or three simple gates, but their sizes are again about two to three times larger than simple ones.

C. Power consumption

There are two components that establish the amount of power dissipated in CMOS VLSI circuits. These are:

LIB1	LIB2
INV (3)* 2,3,4-input NOR 2,3,4-input NAND 2-input AND 2-input OR 2-input XOR 2-input XNOR 21-AOI 22-AOI 21-OAI 22-OAI	INV 2,3,4-input NOR 2,3,4-input NAND
17 Gates	7 Gates

* 3 different strength inverters

Table 4-4. Standard cell library

- (1) Static power dissipation due to leakage current or pseudo-NMOS circuits such as static PLA with pull-up transistor (PMOS) always on.
- (2) Dynamic power dissipation due to switching transient current or charging and discharging of load capacitances.

In PLA-based design, static PLAs are the main source for power dissipation. Static PLA, although it consumes more power than other configurations, is usually fast and easy to design. With static PLAs, the designer can make straightforward tradeoffs between power and circuit speed. Dynamic PLAs are also fast and only dissipate dynamic switching power, but require multiple phases and careful design to avoid timing hazards. All implementations

in this experiment used the static PLA, hence the power consumption of the PLA-based design is much greater than other designs.

Both standard cell-based design and gate-matrix design use CMOS gates. The power dissipation of these circuits consists mostly of dynamic components. Any static dissipation is due to the reverse biased leakage current that flows across the junction between diffusion region and the substrate. Static power dissipation due to leakage current for a small circuits operating at five volts is usually a few nano-watts.

Dynamic power consumption for fully complementary MOS circuits can be estimated by the equation:

$$Power_{dynamic} = C_{total} V^2 f$$

where C_{total} is the sum of all capacitances on switching nodes, V an operating voltage and f a switching frequency of the circuit. Capacitances shown in Tables with power estimates are the sum of capacitances on all switching nodes. Switching nodes can be identified easily from the extracted layout. The total of switching capacitance consists of gate capacitances of transistors (input to a gate), interconnect wire capacitance, and junction capacitance on the output node of the gate. Gate-matrix designs show less power dissipation than others, but this results from the transistor sizes in the gate-matrix design being somewhat fixed, and lack of electrical optimization in the process of generating the layout.

D. Design time and flexibility

Given that the process of designing a microprocessor on silicon is complicated, the role of good VLSI design aids is to reduce the complexity and assure the designer of a working chip in a reasonably short period of time. The time spent to design the control unit of a microprocessor and the flexibility of the design are closely related. The complexity of the control unit often requires several iterations of the same design steps. It is, therefore, desirable to have most of the design steps automated. The automation of the design, in turn, reduces the overall design time, and increases the flexibility of the design to easily

incorporate frequent changes.

The three styles of implementation currently being evaluated have most of their design steps automated. Improvement on the design time using these styles is tremendous compared to the full-custom approach, and allows VLSI designers to use hard-wired control in microprocessor design. A full design cycle of control implementation using these styles from a behavioral specification can take as little as a few hours. This permits designers to revise the entire design as many times as necessary. Once the logic is partitioned into combinational and sequential parts and is optimized, the rest of the implementation is straightforward, relying on automatic layout synthesis tools. Both standard cell-based and gate-matrix-based designs have been produced by fully automated module generation tools, as well as global placement and routing tools. They are therefore much more flexible and take much less time than the full-custom approach with only automated PLA generation tools. The design times shown in the tables are based on actual estimates from the SPUR design and time spent on the re-implementations.

A separate and additional effort is required to build the cell library for the standard cell-based design. Building a comprehensive, fully characterized cell library can be a time-consuming process. However, it has been noted that with small library, multi-level logic optimization tools can produce hardware as good as that produced with a large library. This fact, along with other performance measures, makes the standard cell-based design more attractive than other designs. It is also easier to incorporate the sequential part of the design than others. With forthcoming sequential logic synthesis and optimization systems (or mixed combinational and sequential logic synthesis and optimization), the standard cell-based design will become a prime choice.

4.5. Summary

The key complexity of microprocessor design stems from designing the processor's control unit, which may take up a small portion of the chip area but can consume most of the

design time. For the past decade or so, microprogrammed control design has been a popular approach to designing this complex portion of the processor, due to its flexibility and computer-automated design processes. Recently, various VLSI CAD tools have emerged to facilitate hard-wired control design. Automatic synthesis and optimization techniques at different abstraction levels have made several alternatives available to full-custom VLSI design. Behavioral synthesis and multi-level logic optimization systems provide particularly efficient and high performance hard-wired logic implementation even with semi-custom layout styles, such as standard cell-based design.

In this chapter, I have examined alternatives in the hard-wired control design by re-implementing the control units from the SPUR chips using different design styles, and contrasting them with the full-custom version with only PLA synthesis tools. I found:

- (1) The hard-wired approach to the microprocessor control design has been greatly improved by advanced VLSI CAD tools, especially in design time and the quality of the design. With these design aids, the process of designing the hard-wired control has shared the efficiency and flexibility of the microprogrammed control.
- (2) With recent development in multi-level logic synthesis and optimization techniques, hard-wired logic can be mapped not only into a two-level PLA implementation, but also into various multi-level logic implementation styles which can provide performance comparable to or better than the traditional two-level PLA implementation.
- (3) Among many different implementation styles, standard cell-based design has a prime potential for use as microprocessor control. CAD tools built around the standard cell-based design are also sound and optimizations occur at all levels of abstraction, such as logic design, and placement and routing. Multi-level logic optimization for standard cell-based design is effective, and even with a small library it can generate the optimized logic that can perform as well as that generated with a large library. A small library further reduces the overall design time.

By no means is the evaluation in this experiment complete. A different result is possible for other designs. The benefit of using examples from the SPUR design is that there exists working hardware, which is full-custom designed and hence comparisons can be drawn.

4.6. References

- [Anc83] F. Anceau, "VLSI-Processor Architecture and Design", in *VLSI Architecture*, Prentice Hall International, Englewood Cliff, NJ, 1983.
- [Ber81] R. Bernhard, "More Hardware Means Less Software", *IEEE Spectrum* 18, 12 (December 1981), 30-37.
- [Bra87] R. K. Brayton et al., "MIS: A Multi-Level Logic Optimization System", *IEEE Transaction on CAD of Integrated Circuits and Systems CAD-6*, 6 (November 1987), 1062-1081.
- [DeS83] G. DeMichelli and A. L. Sangiovanni-Vincentelli, "Multiple Constrained Folding of Programmable Logic Array: Theory and Applications", *IEEE Transactions on Computer-aided Design CAD-2*, 3 (July 1983), 151-167.
- [Det87] E. Detjens et al., "Technology Mapping in MIS", *IEEE ICCAD Digest of Technical Papers*, November 1987, 116-119.
- [DeN87] S. Devadas and A. R. Newton, "Topological Optimization of Multiple-Level Array Logic", *IEEE Transaction on CAD of Integrated Circuits and Systems CAD-6*, 6 (November 1987), 915-941.
- [Gel87] P. Gelsinger, "Design and Test of the 80386", *IEEE Design and Test*, June, 1987, 42-50.
- [Har86] D. S. Harrison et al., "Data Management and Graphics Editing in the Berkeley Design Environment", *Proc. ICCAD*, November 1986, 24-27.
- [Hed85] K. S. Hedlund, "Electrical Optimization of PLAs", *Proceedings of IEEE 22nd Design Automation Conference*, 1985, 681-687.

- [Hop83] M. Hopkins, "A Perspective on Microcode", *Proceedings of the 21st Annual IEEE Computer Conference (Spring COMPCON 83)*, San Francisco, February 1983, 108-110.
- [Keu87] K. Keutzer et al., "Impact of Library Size on the Quality of Automated Synthesis", *IEEE ICCAD Digest of Technical Papers*, November 1987, 120-123.
- [LoL80] A. D. Lopez and H. S. Law, "A Dense Gate Matrix Layout Method for MOS VLSI", *IEEE Journal of Solid State Circuits SC-15*, 4 (August 1980), 736-740.
- [MMM84] *MC68020 Technical Summary*, Motorola Semiconductors, 1984.
- [Mou] J. Moussouris et al., "A CMOS RISC Processor with Integrated System Functions", *Proc. of 1986 COMPCON, IEEE*, , 126-131.
- [NaA88] M. Namjoo and A. Agrawal, "Implementing SPARC: A High-Performance 32-bit RISC Microprocessor", *Sun Technology*, Winter, 1988, 42-48.
- [NeS86] A. R. Newton and A. L. Sangiovanni-Vincentelli, "Computer-Aided Design for VLSI Circuits", *IEEE Computer* 19, 4 (April 1986), 38-60.
- [Ous85] J. K. Ousterhout et al., "The Magic VLSI Layout System", *IEEE Design and Test of Computers* 2, 1 (February 1985), 19-30.
- [Seg87] R. B. Segal, "BDSYN: Logic Description Translator, BDSIM: Switch-level Simulator", Electronics Research Laboratory Memorandum, No. M87/33, UC Berkeley, May 1987.
- [Wei67] A. Weinberger, "Large Scale Integration of MOS Complex Logic: A Layout Method", *IEEE Journal of Solid State Circuits SC-2*, 4 (December 1967), 182-190.



5

Conclusion

5.1. Summary

In Chapter 2, details of designing the SPUR CPU chip were described. The methodologies and techniques used to maximize the performance of a full-custom VLSI microprocessor provides an overview of microprocessor design strategies. The rest of the research presented in this thesis is developed from new ideas and better alternatives which have become apparent since the development of the SPUR CPU chip.

In Chapter 3, two alternative memory design techniques were presented: dynamic memory for an on-chip cache memory, and a compact high bandwidth memory with multiple ports. Selective invalidation instead of refreshing, implemented using low overhead dynamic CMOS circuits, can effectively eliminate the need for a periodic refreshing of dynamic memory. With this scheme, the size of an on-chip local memory can be substantially increased within a given allocation of scarce silicon area. Trace-driven simulations show the

effectiveness of this scheme over a simple invalidation scheme.

When multiple functional units are used to increase the performance by parallel execution, the demand for a higher bandwidth between functional units and local memory increases rapidly. Using multi-port memory to balance the bandwidth required by multiple functional units previously has been very expensive due to the large cell area requirement. When a single-ended access memory cell is operated at reduced voltage levels, it can result in a fast, stable memory while the area required is relatively small. Several multi-port configurations are designed and analyzed to demonstrate the feasibility of multi-port memories based on this cell.

In Chapter 4, alternative implementation styles for microprocessor's control logic were investigated. Recently, various VLSI CAD tools have emerged to facilitate hard-wired control design. Automatic synthesis and optimization techniques at different abstraction levels have made several alternatives available to full-custom VLSI design. Behavioral synthesis and multi-level logic optimization systems provide particularly efficient and high performance hard-wired logic implementation, even with semi-custom layout styles, such as standard cell-based design. I have examined alternatives in the hard-wired control design by re-implementing the control units from the SPUR chips using different design styles, and contrasting them with the full-custom version also available from SPUR designs. I found that the standard cell-based design can result in the best implementation style among others for various aspects of resulting design.

5.2. Future Research

As more chip area is devoted to on-chip memories, several different types of local memories are being integrated in a single-chip microprocessor. It is interesting to see how much memory needs to be allocated for instruction versus data, register versus cache, or for some other purposes such as for memory management functions (e.g. TLBs).

As to multi-port memory design, I have examined implementation issues only. However, to investigate the overall performance tradeoffs other issues must be carefully considered. These include instruction format for multiple operations, and using hardware or software to balance the bandwidth required by multi-port memory and functional unit, and/or designing an optimizing compiler that extend the register allocation scheme to multiple register files.

For automated synthesis of control logic, the impact of library size for standard cell based design needs to be explored further to determine an ideal library size, in conjunction with optimal gate (technology) mapping. It requires making generalizations about the cost of creating and maintaining a library as well as assumptions about application domain. Since routing area is an important component of standard cell layout, impact of library size on routing region also need to be investigated.

Array structured logic such as gate matrix offers a dense layout but is often marred by poor electrical performance. For further improvement, an optimal partitioning of logic and electrical optimization such as transistor sizing are necessary.

5.3. Conclusion

Optimizing performance in full-custom VLSI microprocessor involves several choices, including the choice of the best design methodology and the best implementation styles. There is a broad spectrum of implementation styles that have proven successful for the construction of various modules in microprocessor chip. In general, the implementation of microprocessor can be divided into three activities: data path design, control logic design, and on-chip memory design. The latter has become important as more and more chip area is devoted to local memories, to minimize off-chip communication traffic that uses the scarcest resources of the microprocessor chip, the i/o pins. The research presented in this thesis focuses on implementation issues of on-chip memory and control logic of a full-custom VLSI microprocessor.

Since the on-chip memory is limited in its size, an optimal implementation of local memory becomes increasingly complex. Many different cache and register organizations are proposed for various optimizations. When dynamic memory is substituted for static memory, the size of on-chip memory can be increased without increasing the chip area. A provision must be made so that the operation of DRAM may not affect the processor's normal execution hence hampering the performance. Using simple circuit design techniques and a small modification of the cache, periodic refreshing requirement of DRAM can be effectively eliminated. A multi-port memory facilitates the parallel processing using multiple functional units. Similarly with DRAM cache above, a simple circuit design technique can lead to a compact yet fast and stable multi-port memory cell.

A portion of research is devoted to investigating various layout styles. All new design methods aim for simplicity and regularity. Full-custom design aiming for high performance but taking long design time can be adopted when area or timing considerations are critical, such as in high frequency data path design. Using automated synthesis of control logic with semi-custom design styles, particularly in multi-level representation, makes the design process efficient and easy, and the resulting design is comparable to that produced by full-custom design. The standard cell based design style, when combined with multi-level logic optimization, can provide a resulting design as good as full-custom version but in much shorter design time. It is also shown that even with a small size library, the resulting layout is better in both delay timing and area than other semi-custom styles.