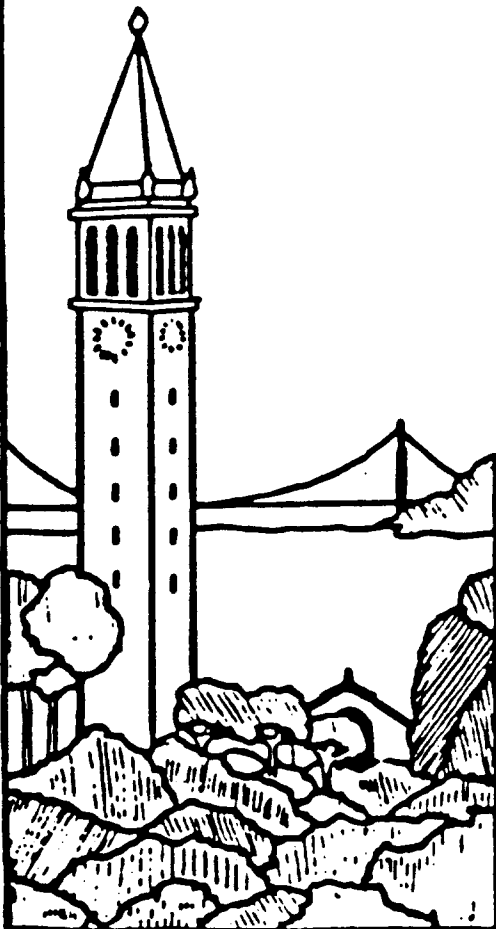


**Data Path Design Considerations
for a High Performance VLSI
Multiprocessor**

Daebum Lee



Report No. UCB/CSD 87/318

November 1986

**Computer Science Division (EECS)
University of California
Berkeley, California 94720**

Data Path Design Considerations for a High Performance VLSI Multiprocessor

Daebum Lee

Department of Electrical Engineering
and Computer Science
University of California
Berkeley, California 94720

ABSTRACT

A VLSI data path implementation for the SPUR (Symbolic Processing Using RISC's) processor is presented. There are many tradeoffs to be considered in the design of a microprocessor data path. Often, these tradeoffs are interrelated and thus increase the complexity of the design. This report focuses on the design of the CMOS data path with the tradeoffs considered throughout the implementation of the data path for the SPUR CPU.

November 17, 1986

Author Daebum Lee
Title: Data Path Design Considerations for a High Performance VLSI
Multiprocessor

RESEARCH PROJECT

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, in partial satisfaction of the requirements for the degree of Master of Science, Plan II.

Approval for the Report and Comprehensive Examination:

Committee: David G. Hodge, Research Adviser

Nov. 19, 1986

Date

Randy H. Katz

November 17, 1986

Date

TABLE OF CONTENTS

1. INTRODUCTION

2. ARCHITECTURAL ASPECTS AND CONSTRAINTS OF THE SPUR CPU

- 2.1 The 4-stage Pipeline
- 2.2 Clocking Specification
- 2.3 Functional Units
- 2.4 The SPUR CPU chip
- 2.5 Execution of Instructions by the Lower Data Path

3. IMPLEMENTATION OF THE SPUR DATA PATH

- 3.1 The Design styles and the Scalable CMOS Technology
- 3.2 The Register File
- 3.3 Internal Forwarding
- 3.4 Functional Units
- 3.5 Design Metrics of the Lower Data Path
- 3.6 Experimental Results - A Testchip for the Data Path

4. ALTERNATIVE IMPLEMENTATIONS

- 4.1 Dual-port Read and Write Memory Cell
- 4.2 An On-chip Dynamic Memory

5. CONCLUSION AND SUMMARY

REFERENCES

APPENDIX

Data Path Design Considerations for a High Performance VLSI Multiprocessor

Daebum Lee

Department of Electrical Engineering
and Computer Science
University of California
Berkeley, California 94720

1. INTRODUCTION

Evolving integrated circuit technology permits high performance CPU's to be built on a single chip. As a result, the data path design for such a high performance microprocessor becomes increasingly complex. It often involves numerous performance and cost tradeoffs. Furthermore, optimization of the design must be done interactively with different levels of hierarchy to achieve global optimization rather than the local optimization. Consequently, the data path design depends heavily on the microarchitecture of the processor. The design objective can differ very much for one architecture to another. This report presents the design of a data path optimized for the specific microprocessor architecture, SPUR (Symbolic Processing Using RISCs).

SPUR is a multiprocessor workstation being developed at UC Berkeley to support LISP programming environments through multiprocessing [Hill85]. It includes three custom VLSI chips, the CPU, the floating point coprocessor (FPU), and the memory management unit (CC-cache controller). The SPUR CPU chip is the third generation RISC processor designed and implemented at U.C. Berkeley. This report describes the details of the data path implementation for the SPUR CPU chip.

In Section 2, a brief overview of the SPUR CPU microarchitecture is presented. This will help illustrate the implementation details of the data path presented in the following section. The pipeline organization is discussed first since it has a great impact on the data path timing. It also imposes hardware requirements, such as multiple read and write capability of the register file. Clocking specifications and requirements of functional units are then presented. After looking at a CPU block diagram and global floor plan, the three most typical instruction executions are presented to illustrate

how the subsystems in the data path are utilized to implement the instructions.

Section 3 focuses on the implementation of the SPUR CPU's lower data path. The implementation strategy used through out the chip is presented first, along with the process technologies in which the SPUR chips will be fabricated. The implementation details of the register file, internal forwarding logic, and functional units are further scrutinized in the following sections. A part of SPUR CPU chip was assembled into a testchip and fabricated. Testing results are presented at the end of the Section 3.

Finally Section 4 examines some alternatives in designing a high performance microprocessor data path. The register file with multiple read and write ports can affect processor's performance as well as it's pipeline organization. Thus, the design of double port read and write memory cells are considered. The limitations of implementing a dynamic memory on the processor chip are also examined, and finally an implementation of on-chip cache using dynamic memory is presented.

2. ARCHITECTURAL ASPECTS AND CONSTRAINTS OF THE SPUR CPU

The microarchitecture of the SPUR CPU has been influenced by previous RISC projects at UC Berkeley [Kate83] [Joan85]. Some features of previous Berkeley RISC architecture were kept in the SPUR CPU, while others were deleted and new features were substituted. For example, a large register file with an overlapping window scheme is implemented in the SPUR CPU, while the 32-bit barrel shifter included in RISC II is replaced by a simple shifter that shifts up to 3 bits. Advances in technology permit an instruction cache to be implemented on the CPU chip. This on-chip cache will improve the performance by reducing the contention between instruction and data memory traffic, in addition to reducing the effective instruction access time.

This section will provide a brief overview of the SPUR CPU architecture as well as some detail about key features. Implementation issues of those key features are further discussed in the following sections. A detailed description of the SPUR CPU microarchitecture can be found in [Kong86], and the instruction set is described in [Taylor85].

2.1 The 4-stage pipeline

The basic idea behind the SPUR pipeline organization came from the three-stage pipeline in the RISC-II processor [Kate83]. In an ideal three-stage pipeline, each register-to-register instruction takes 3 cycles to finish. However, instructions that reference memory, such as load, take four cycles to complete. After a 4-cycle instruction, the following 2 instructions must be suspended. Figure 2-1 illustrates this suspension sequence. The reasons for the suspension are listed below:

- (1) There is a register file write conflict between the 4-cycle memory access instruction and the following register-to-register instruction.
- (2) There is a memory access conflict between 4-cycle instruction (Mem Access stage) and the instruction entering the Ifetch stage of pipeline.

There are many ways to avoid this type of pipeline suspension. For example, if the register file is made of 2-port read-write memory elements, the first conflict can be resolved. The second conflict can also be resolved by including an on-chip instruction buffer (cache) or separating a cache

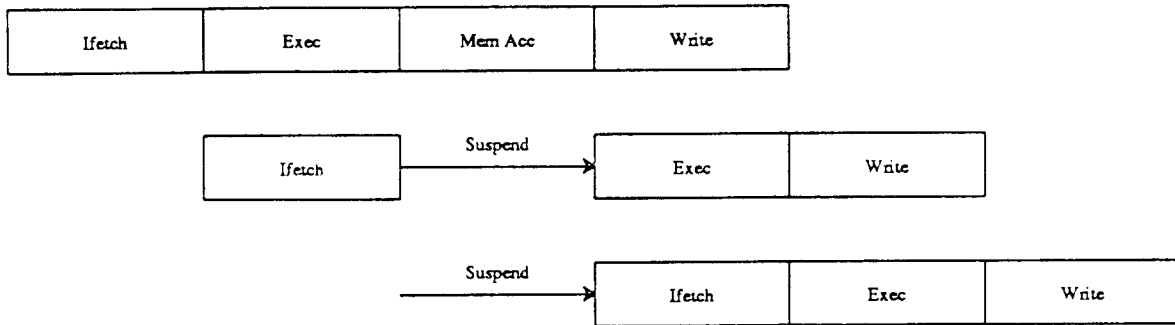


Figure 2-1. 3-stage pipeline

(memory) into data and instruction caches, which allows the CPU to reference both data and instruction simultaneously.

The solution employed in the SPUR CPU was to include an on-chip instruction buffer and to stretch all 3-cycle instructions to a uniform 4-cycle pipeline [Kong86]. The third stage can be a memory access cycle for those instructions that reference memory, and can be an idle stage for the register-to-register instructions, and is illustrated in Figure 2-2. The idle stage of the stretched pipeline eliminates the double port write requirement on the register file, while the on-chip instruction buffer eliminates the conflict between instruction fetch and memory access. Moreover, adding one extra cycle to delay the write stage does not degrade the performance of the CPU. The result of each ordinary instruction is still available to the next instructions through double internal forwarding (double internal forwarding is explained below).

The 4-stage pipeline organization now requires double internal forwarding, as shown by the arrows in Figure 2-2. This avoids data hazards caused by referencing the register whose new value is not yet written into the register file. Extra temporary latches must be provided to store the results that might be internally forwarded, along with the logic that detects the forwarding conditions. However, there is an instruction sequence where this internal forwarding logic cannot resolve the data interdependency: When an instruction following a load instruction references the register being loaded. The loaded value won't be available until late in the memory stage of the load instruction. By this time the 2nd instruction has almost finished its execution stage. The result of the instruction

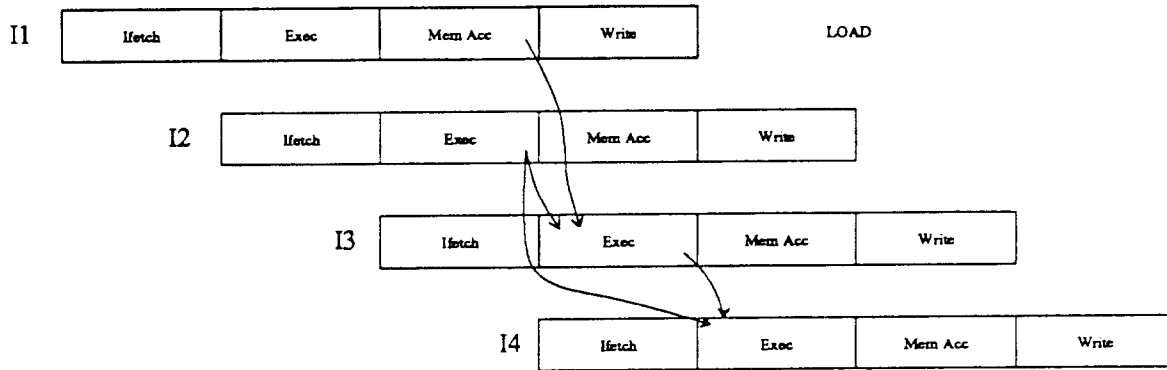


Figure 2-2. 4-stage pipeline of the SPUR CPU

in this sequence is thus undefined. The implementation details of this double internal forwarding circuits are presented in Section 3-2.

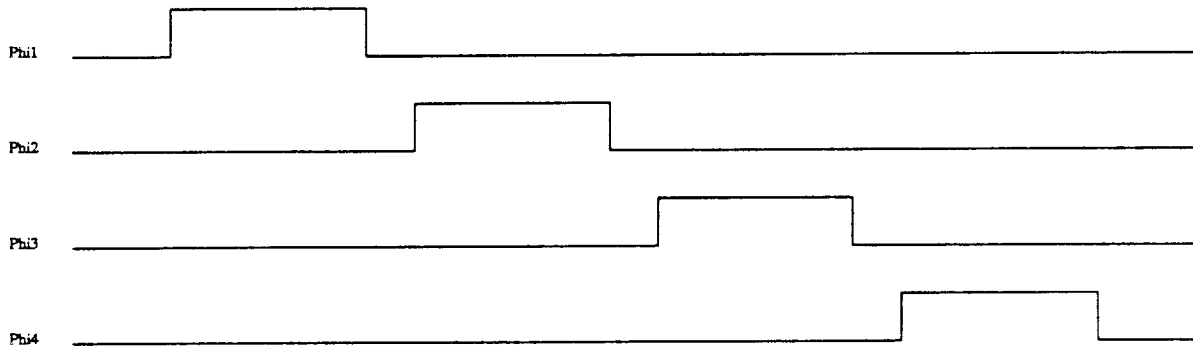
2.2 Clocking Specification

The clocking scheme of the SPUR CPU is a direct result of its register file and instruction buffer operations, which are in turn dictated by the 4-stage pipeline operations. At least 3 phases are needed to permit read and write within one cycle in both blocks (section 3-2 explains why 3 phases are needed). These phases are a precharge phase prior to the read, a phase for the read, and a phase for the write. However, the register file and the instruction buffer (IB) require different timing for their own operations. Therefore a 4-phase non-overlapping clocking scheme is chosen.

The operations of the register file and IB with a 4-phase non-overlapped clocking scheme are summarized in Figure 2-3. Note that a non-overlapped time is required to discharge the word-line before precharging the bit-line.

To eliminate clock skew caused by the delay of bringing the phases on a chip, an on-chip 4-phase clock generator has been designed and tested for use in SPUR CPU chip [DKJeong86].

The timing specification for this 4-phase clock depends on the critical paths within each phase. The initial timing analysis indicates that 25 nsec for each phase with 10 nsec non-overlapped time is required. This results in cycle time of 140 nsec.



	Phi1		Phi2		Phi3		Phi4	
nsec	25	10	25	10	25	10	25	10
Reg File	Read		-		Write		precharge bit line	
Decoder	Precharge		Decode for write		Precharge		Decode for read	
Bit line	Discharged (read)		-		Driven for write		precharged to 1	
Word line	Driven for read		discharged to 0		Driven for write	*	discharged to 0	

	Phi1		Phi2		Phi3		Phi4	
IB	Precharge bit line		Read		-		Write	
Decoder	Decode for read		Precharge		Decode for write		precharge	
Bit line	Precharge to 1		Discharged (read)		-		Driven for write	
Word line	Discharged to 0		Driven for read		Discharged to 0		Driven for write	*

* Word line must be discharged during the non-overlapped time, before the bit lines being precharged.

Figure 2-3. Operations of the register file and the instructor. buffer

2.3 Functional Units

Hardware support for complex functions often requires substantial silicon area on the chip as well as an increase in the basic cycle time. In RISC architectures, most complex functions are left out and only simple minimum functions are used sequentially to implement complex functions. In the SPUR CPU, the major functional units required are the arithmetic logic unit (ALU), the shifter, and the byte extractor/insertter.

The ALU is a simple 32-bit adder and includes some basic logical operations such as OR, AND, and XOR. These logical operations are readily available from the input section of the adder, and hence do not consume extra silicon area.

The shifter is used to implement arithmetic and logical shift operations. The SPUR CPU implements 1 to 3-bit logical left-shift, 1-bit logical right-shift, and 1-bit arithmetic right-shift operations.

Tag checking and operations on tags are necessary features because the SPUR CPU is designed to support LISP. The 40-bit register is made of an 8-bit tag and a 32-bit data word to support this. The byte extractor and insertter are also used to implement instructions that do operations on tags. The byte-extractor takes a byte out of the source register and puts it onto the destination bus, and the byte-insertter inserts a byte into a 40-bit word.

With 4-phase clocking, all functional units accept inputs at the leading edge of ϕ_2 and drive the output bus at the leading edge of ϕ_4 . The implementation of these functional units are presented in section 3-3.

2.4 The SPUR CPU chip

The SPUR CPU chip consists of two major blocks: The Instruction Unit and the Execution Unit. The execution unit is further divided into the upper data path, the lower data path, and the control unit. The 40-bit lower data path is for general computation on the 40-bit tagged registers, and the 30-bit upper data path is for instruction address calculations and special register references.

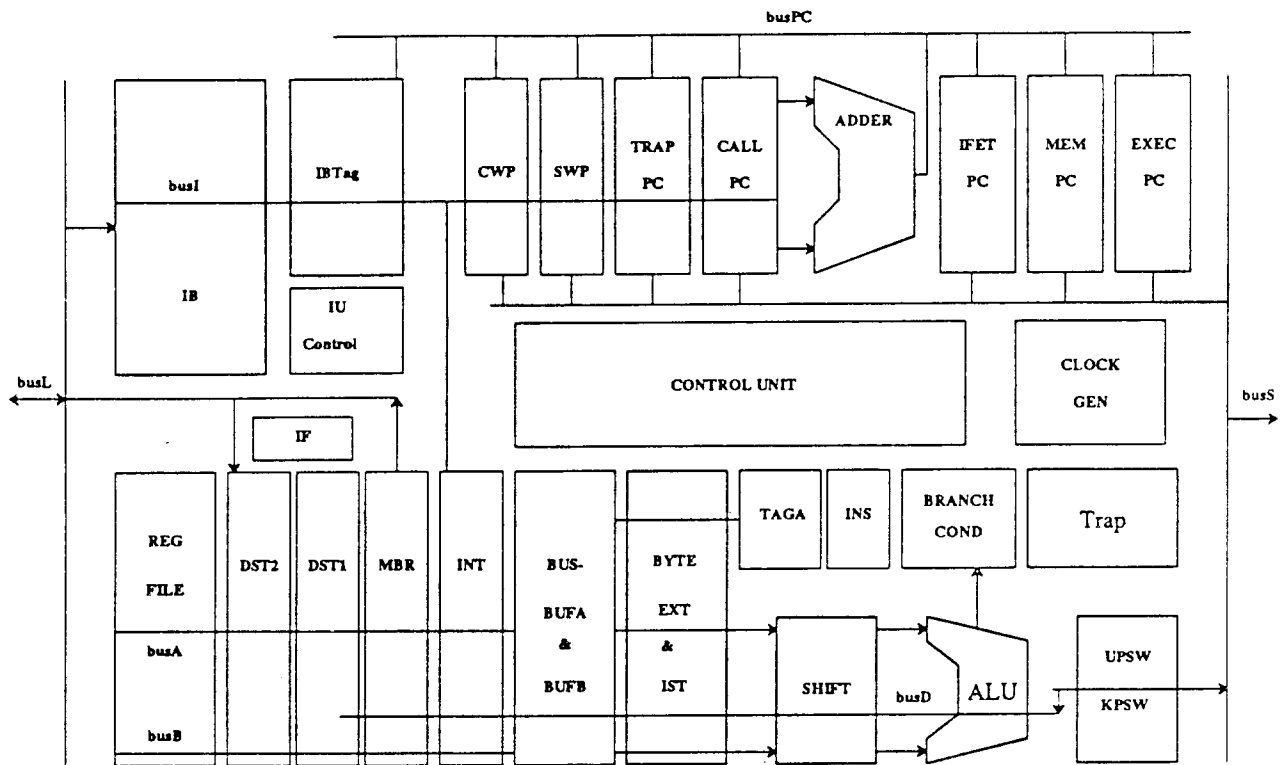


Figure 2-4. CPU Block Diagram

Figure 2-4 presents a block diagram of SPUR CPU chip. The basic elements and functional units within each major blocks are described:

Instruction Unit(IU)

Instruction Buffer(IB): 512 bytes or 128 32-bit words direct-mapped cache. It is organized into 16 sub-blocks each containing 8 words.

IB Tag: Tag storage for IB

IU controller: Instruction unit controller, consists of 2 finite state machines

Execution Unit(EU)

Lower data path

Register file: 138 40-bit registers organized into 8 multiply overlapped widows. Registers are dual-ported for read but single-ported for write.

DST1, DST2: These two temporary latches hold the results of two instructions, which have

finished their execution stage of pipeline but waiting for the register file write stage. If conditions are met, these results are forwarded to the next instructions which need them.

Internal forwarding logic: The logic that detects internal forwarding conditions.

MBR: Memory buffer register which stores the data to be written to the external memory.

Functional Units:

Byte Extractor/Inserter: Byte extractor and inserter

Shifter: A simple logical and arithmetic shifter

ALU: A simple adder with XOR, OR, and AND logical operations

Trap Logic: Generates trap requests for various conditions

Branch Cond: Uses the outputs of the ALU to evaluate the conditions for all compare and branch instructions.

KPSW, UPSW: UPSW and KPSW are special registers which hold User and Kernel Processor Words, respectively. Logically, they belong to the upper data path, but due to space limitation of the upper data path they are placed in the lower data path.

Upper data path

CWP: Current window pointer points to the register window in use

SWP: Saved window pointer points to the memory address that holds the saved window in memory overflowed from the register file.

TrapPC: Holds the target address for potential trap request

CallPC: Holds the target address for the jump or call instructions

Adder: Calculates the destination address for all compare and branch instructions while the ALU is performing the comparison.

IfetPC: Holds the address of the instruction currently in the instruction fetch stage of the pipeline.

ExecPC: Holds the address of the instruction currently in the execution stage of the pipeline.

MemPC: Holds the address of the instruction currently in the memory access stage of the

pipeline.

Control Unit

The control unit decodes the instruction OP codes into high-level control signals for the rest of the CPU. Since a 4-stage pipeline is used, it must deliver these high-level signals in time for the correct pipeline stages. These high-level signals are then routed to the local decoding blocks to form low-level control signals, which directly control the data paths.

Floor Plan: The IB and the Regfile are the largest (in size) single blocks in the chip. The size and aspect ratio of these two blocks lead to the floor plan of the chip as shown in Figure 2-5. The upper data path, which calculates the instruction addresses is placed by the IU in the upper half of the chip. The lower data path, which is for general computation on the 40-bit registers, is placed next to the Regfile in the lower half of the chip. Both the instruction buffer and the register file use the same memory element. Furthermore, both data paths are implemented in bit-sliced fashion along with this basic memory element. This not only saves the design time but also provides the possibility of sharing some useful layouts in both data paths. The lower data path gets all control signals from the control unit through the local decoding logic. All control signals are buffered by control line buffers adjacent to the data path. It also accepts the register address and the immediate fields of the instruction from the IU through busI. BusL is for the memory data, while busS is for the memory address and it may also be used to transfer data between the upper data path and the lower data path.

The implementation of the instruction unit (IU) is well described in [RichD86], and that of the upper data path of the execution unit (EU) is in [Wook86]. This report will focus on the implementation issues of the lower data path.

2.5 Execution of instructions by the lower data path

This section illustrates the execution of instructions by the lower data path with architectural features and clocking specifications presented thus far. These illustrations not only give an insight of how subsystems in the lower data path are utilized, but also provide a glimpse of the timing

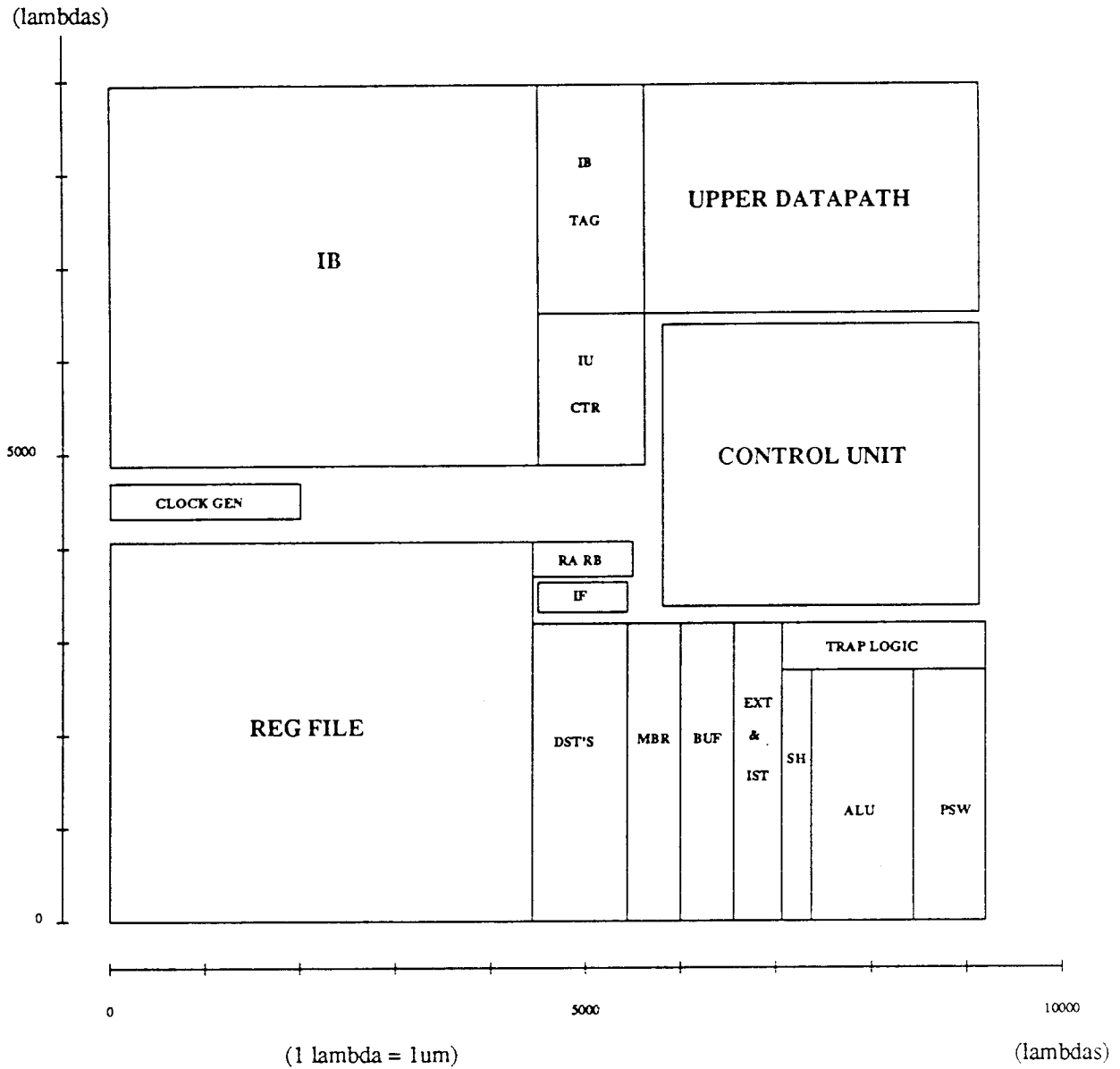


Figure 2-5. The floor plan of the SPUR CPU chip

constraints they have.

Among many instructions supported by the lower data path, the execution of these most common instructions are chosen in the illustrations. The execution of those three instructions involve almost all of the subsystems in the data path.

- (1) Register-to-register operation using one of the functional units.

$$RD \leftarrow RS1 \text{ (FU) } RC$$

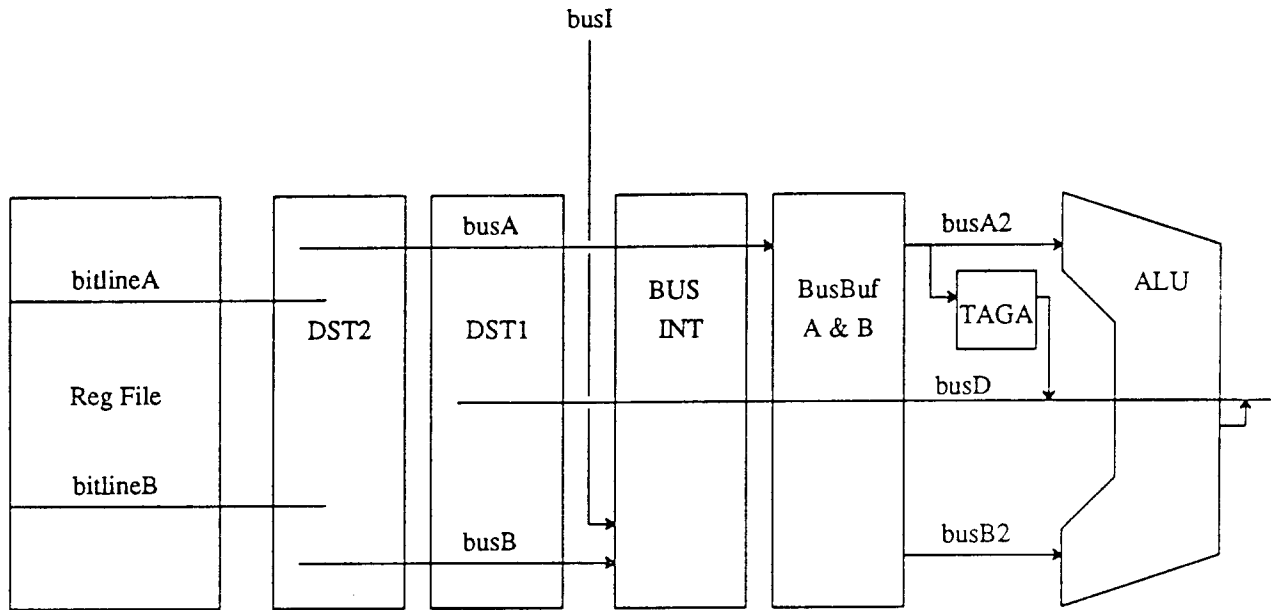
- (2) LOAD instruction

$$RD \leftarrow \text{MEM}[RS1 + RC]$$

- (3) STORE instruction

$$\text{MEM}[RS1 + RC] \leftarrow RS2$$

Figures 2-6, 7 and 8 illustrate the execution of the above 3 instructions by the lower data path.



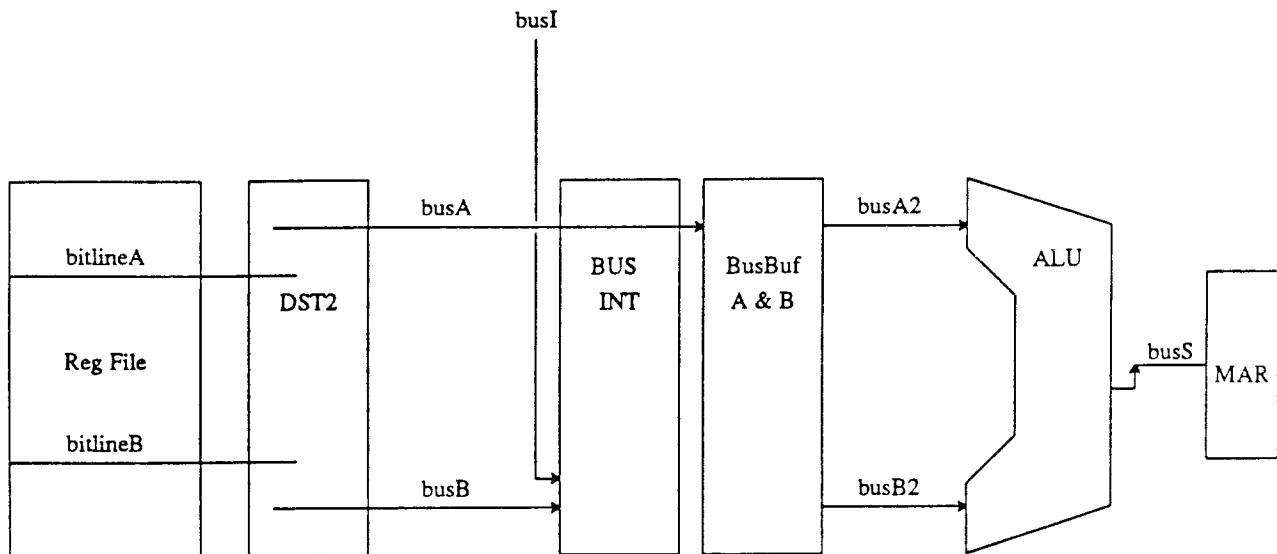
Cycle 1 Ifetch	Phi1	
	Phi2	
	Phi3	Instruction decoding in control unit
	Phi4	Register file decoding - RS1

Cycle 2 Exec	Phi1	BusBufA \leftarrow busA, BusBufB \leftarrow busI or busB
	Phi2	ALU \leftarrow busA2, busB2; TAGA \leftarrow busA2[tag]
	Phi3	ALU
	Phi4	DST1 \leftarrow busD \leftarrow ALU; DST1[tag] \leftarrow busD \leftarrow TAGA

Cycle 1 Mem Acc	Phi1	
	Phi2	
	Phi3	DST2 \leftarrow DST1
	Phi4	

Cycle 1 Write	Phi1	
	Phi2	Register file decoding for write - RD
	Phi3	RD \leftarrow DST2
	Phi4	

Figure 2-6. Illustration of register-to-register operation
RD \leftarrow RS1 (ALU) RC



Cycle 1 Ifetch	Phi1	
	Phi2	
	Phi3	Instruction decoding in control unit
	Phi4	Register file decoding - RS1

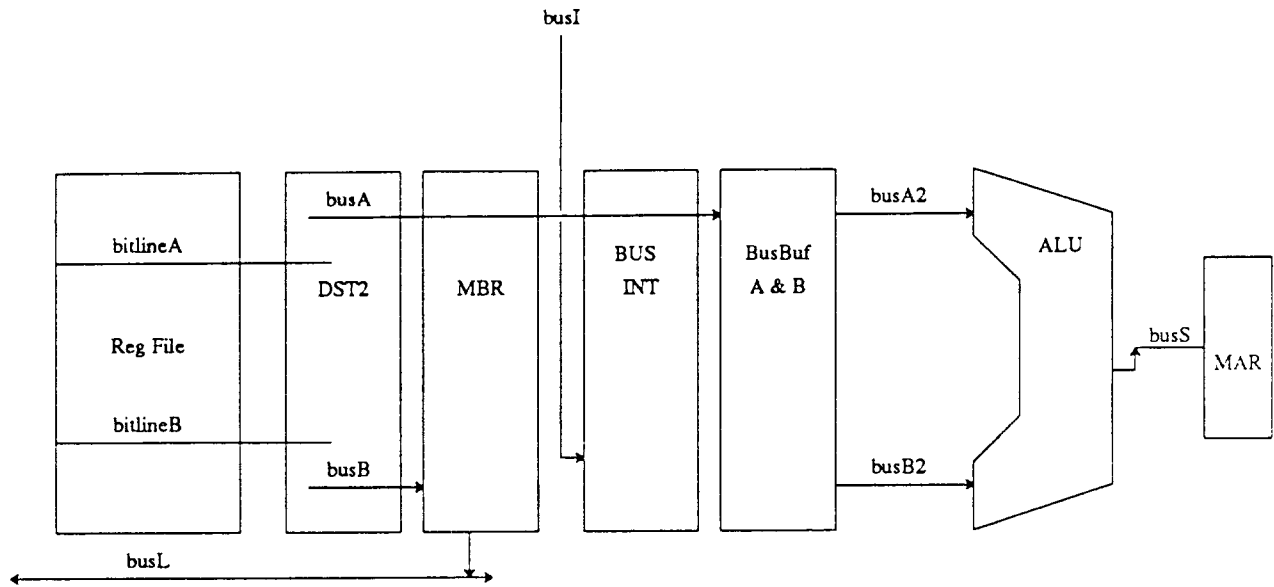
Cycle 2 Exec	Phi1	BusBufA <-- busA (RS1), BusBufB <-- busI or busB
	Phi2	ALU <-- busA2, busB2
	Phi3	ALU
	Phi4	MAR <-- busS <-- ALU

Cycle 1 Mem Acc	Phi1	
	Phi2	
	Phi3	DST2 <-- busL <-- MEM*
	Phi4	

Cycle 1 Write	Phi1	
	Phi2	Register file decoding for write - RD
	Phi3	RD <-- DST2
	Phi4	

* Assume no cache miss

Figure 2-7. Illustration of LOAD instruction : RD <-- MEM[RS1 + RC]



Cycle 1 Ifetch	Phi1	
	Phi2	
	Phi3	Instruction decoding in control unit
	Phi4	Register file decoding - RS1 & RS2

Cycle 2 Exec	Phi1	BusBufA <-- busA (RS1), BusBufB <-- busI (RC) MBR <-- busB
	Phi2	ALU <-- busA2, busB2
	Phi3	ALU
	Phi4	MAR <-- busS <-- ALU

Cycle 1 Mem Acc	Phi1	busL <-- MBR
	Phi2	
	Phi3	
	Phi4	

Cycle 1 Write	Phi1	
	Phi2	
	Phi3	
	Phi4	

Figure 2-8. Illustration of STORE instruction : MEM[RS1 + RC] <-- RS2

3. IMPLEMENTATION OF THE SPUR DATA PATH

This section presents the implementation details of the SPUR CPU's data path. Before examining the low-level details, it may be advantageous to look at an overall picture of implementation strategies. For example, the circuit design styles used throughout the CPU chip, and the process technologies in which the chip will be fabricated.

3.1 The Design Style and the Scalable CMOS technology

The general design strategy used in data path implementation follows the theme of global optimization rather than local optimization. Due to the limited resources on a single chip, a compromise should be made on the local optimization of any subsystem, such that the ultimate goal of global optimization can be achieved. Furthermore, the overall performance of the data path strongly depends on how closely subsystems are connected to the data path and how efficiently they interact with each other. Most subsystems in the SPUR data path are designed and optimized by carefully considering these facts.

The design style used throughout most of the chip is dynamic CMOS design. Within such a design style, charge redistribution and clock skew can be severe problems. The general guidelines of avoiding these problems are well presented in [Nora83] and [Kong85]. The following rules are followed in all dynamic circuits designed in the chip:

- (1) In the case of domino logic, all inputs must be either settled before the evaluation or they can make only one transition from 0 to 1 during the evaluation.
- (2) Any input that is more likely than others to make a 0 to 1 transition during the evaluation phase, is placed further away from the precharged node (closer to the GND).
- (3) Clocked CMOS latches are used for latching inputs from precharged nodes, such as precharged bus.
- (4) Dynamic nodes are intentionally laid out to have larger capacitance (about 5X or more) than any node that can potentially share charge. Consequently, even if charge

redistribution takes place, the logic level of the dynamic node is not degraded to an incorrect logic level.

Dynamic circuit design style often speeds up the data flow. For example, a highly capacitive bus can be precharged to a high level and then pulled down to low or remain high according to the logic. Since NMOS transistors are usually 2-3 times stronger than their CMOS counter parts, pulling down a capacitive bus is fast with a modest amount of silicon area. Some highly capacitive busses in the SPUR CPU chip, such as busD and busS, are implemented this way.

The SPUR pipeline can be suspended for a long period of time due to memory or coprocessor operations. Therefore, provision should be made to prevent any loss of valuable information stored on capacitive nodes during those long pipeline suspensions. Since charge on capacitive nodes leaks away after a certain period time (approximately 2-4 msec), it is necessary to refresh such nodes periodically. Most latches and registers are, therefore, made to have a refreshing feature. Since 4 phases are available everywhere on the chip, they are used as refresh signals. This will ensure all dynamic latches and registers are refreshed every cycle even during the indefinite pipeline suspension as long as the power and clock are still on.

Initially, the SPUR CPU chip was planned to be fabricated in a 2 μm double metal CMOS process. Advances in technology make it possible to fabricate it in scalable CMOS processes. Several design rules have been merged to produce a general set of scalable CMOS design rules. Moreover, circuits used in the SPUR CPU chip are designed to be tolerant of process variations. As an example, even though the SPUR CPU chip uses large amount of silicon area for the memories, it does not use sense amplifiers in any of them, due to the sense amplifier's relatively heavy dependence on the process.

The processes available to fabricate SPUR chips are the 2 μm CMOS process at XEROX PARC and the CMOS40 process at HP's Northwest IC division (Corvallis, OR). The CMOS40 process is equivalent to the 1.6 μm lambda-based technology ($\lambda = 0.8 \mu m$). Both 2 μm CMOS and the CMOS40 processes are double metal processes. SPICE model parameters for both processes are

included in Appendix. All SPUR chips use a pad-frame containing 208 pads, although the CPU chip only needs approximately 180 pads. The remaining pads will be used to probe various signals in the chip for diagnostics.

3.2 The Register File

The significance of having a large register file in a RISC type data path has been well discussed in [Kate83]. As in the RISC II processor, the SPUR CPU includes 138 registers organized into 8 overlapping windows. Unlike RICS II, these registers are 40-bit long in which an 8-bit tag is augmented to the 32-bit data to support tag checking. In this subsection, a CMOS implementation of the register file is presented.

3.2.1 Memory cell design

The register-oriented RISC instruction format usually has addresses of two source registers and a destination register. With the 4-stage pipeline in the SPUR CPU, it leads to double reading and single writing of register file, all within one cycle. Among the different memory cells considered, the popular 6T static RAM cell based on CMOS technology, as shown in Figure 3-1, was chosen. The NMOS version of the same cell was also used in both RISC II and SOAR [Sher84] [Joan85].

The conventional SRAM uses sense amplifiers to speed-up the read and write. The SPUR CPU chip, however, avoids the use of sense amplifiers due to the complexity of circuits. An alternative is to precharge the bit line prior to read and discharge it according to the data stored in the memory cell. This alternative has been adopted in SPUR CPU for its simplicity and its insensitivity to processes variations. Consequently, at least 3 phases are needed to complete read and write within a cycle - one for read, one for write and one for precharge.

While the sense amplifier uses both bit lines for reading, precharging scheme only requires one bit line for the reading. This makes it easy to design a two-port read register file with just two busses. The access transistors M3 and M4 in Figure 3-1, are driven by two different select lines, such that two accesses can occur simultaneously through two busses. Cell reading is done by merely

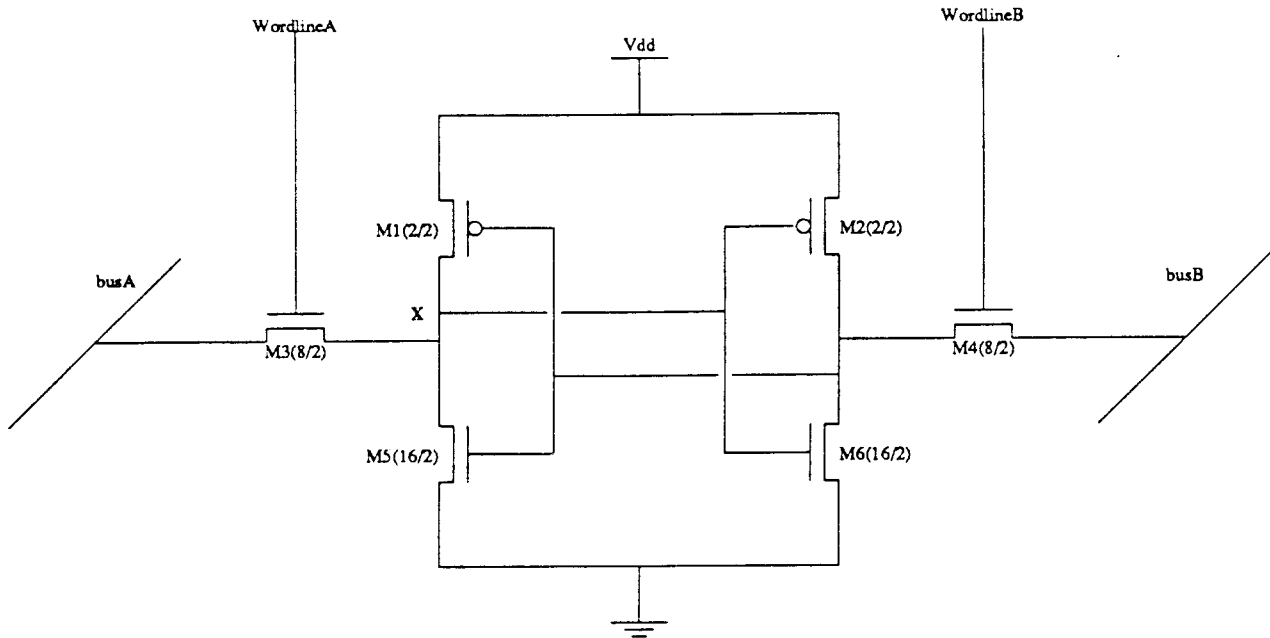


Figure 3-1. CMOS 6T SRAM cell

discharging a precharged bit line. In this scheme, however, the reading can be quite slow because the selected cell must discharge the heavily loaded bit lines by several volts.

The read delay can be reduced by widening the pulldown transistors of cell (M5, M6) as well as the access transistors (M3, M4). However, there are two drawbacks to widening the access transistor. First, when the cell is selected and "0" is stored, current flows from the bit line through the access transistor (M3) and the pulldown transistor (M5) to the ground. To avoid altering the state of the cell, the conductance (W/L) of access transistor must be several times smaller than that of pulldowns, so that the drain (node X in Figure 3-1) voltage of the pulldown transistor does not rise above the threshold of the coupled device. Secondly, as the width of access transistor increases, the drain area of that device also increases. This, in turn, increases the bit line capacitance, since the drains of all 138 access transistors contribute to the bit line capacitance. Optimum conductances of these two devices exist, and SPICE was used to determine the optimum sizes of these devices.

The simulations indicate that a conductance (W/L 's) ratio of 2 will ensure safe operation. It seems to be a small ratio, but right after the selection is made, the pulldown transistor is in its linear

region of operation while the access transistor is in saturation, hence the effective resistance of access transistor is much higher than that of pulldown and the ratio of 2 was satisfactory. With this ratio of 2, and bit-line precharged to 5V, the voltage at the drain of pulldown (node X in Figure 3-1) stayed below 0.6V until reading is done, according to SPICE simulation. Since the threshold voltage of the NMOS transistor is close to 0.9V, this design has a safety margin of about 0.3V. The safety margin to the logical threshold, on the other hand, is greater than 1.0V.

Precharging the bit-line to less than 5V can also reduce the reading delay. However, the layout of a cell (described below) results in transistor sizes wide enough to meet our timing goal even with bit lines precharged to 5.0V.

The major effort in layout was directed at minimizing the area of register file considering all the constraints, as well as reducing parasitics to achieve the fastest access time. In addition to the constraints mentioned above, the layout of register cell is further limited by the pitch-matching constraints imposed by the decoder and the size of the data path bit-slices. Each register needs two decoders to provide double reading. The width of the cell is limited by this requirement. On the other hand, since the data path is designed with a bit-slice for each subsystem, the height is determined by other subsystems, as shown in Figure 3-2. In the SPUR data path, the ALU requires a height of about 74 lambdas (1 lambda is $0.8 \mu m$ in the CMOS40 process and $1 \mu m$ for the XEROX process) to include all circuits needed within a bit-slice. These constraints results in a final cell size of 32 lambdas (width) by 74 lambdas (height). Given this cell size, the access and pull-down transistors were sized such that the access time can be minimized while maintaining the ratio between them for the safety margin. The final sizes of the transistors are also indicated in Figure 3-1.

The layout of 4 register cells is shown in Figure 3-2. They are laid out together to share power lines and bit-line contacts. This not only minimizes the area but also reduces parasitics, such as bit-line capacitance. Since a CMOS process with 2 layers of metal is available, the cell design uses metal-1 word lines and metal-2 bit lines. Therefore, the only significant parasitics are capacitances on these lines, and the resistance of these lines can be neglected. The extracted capacitances of word-line and bit-line are 1.2pF and 2.1pF respectively. The power lines are also run in metal-2

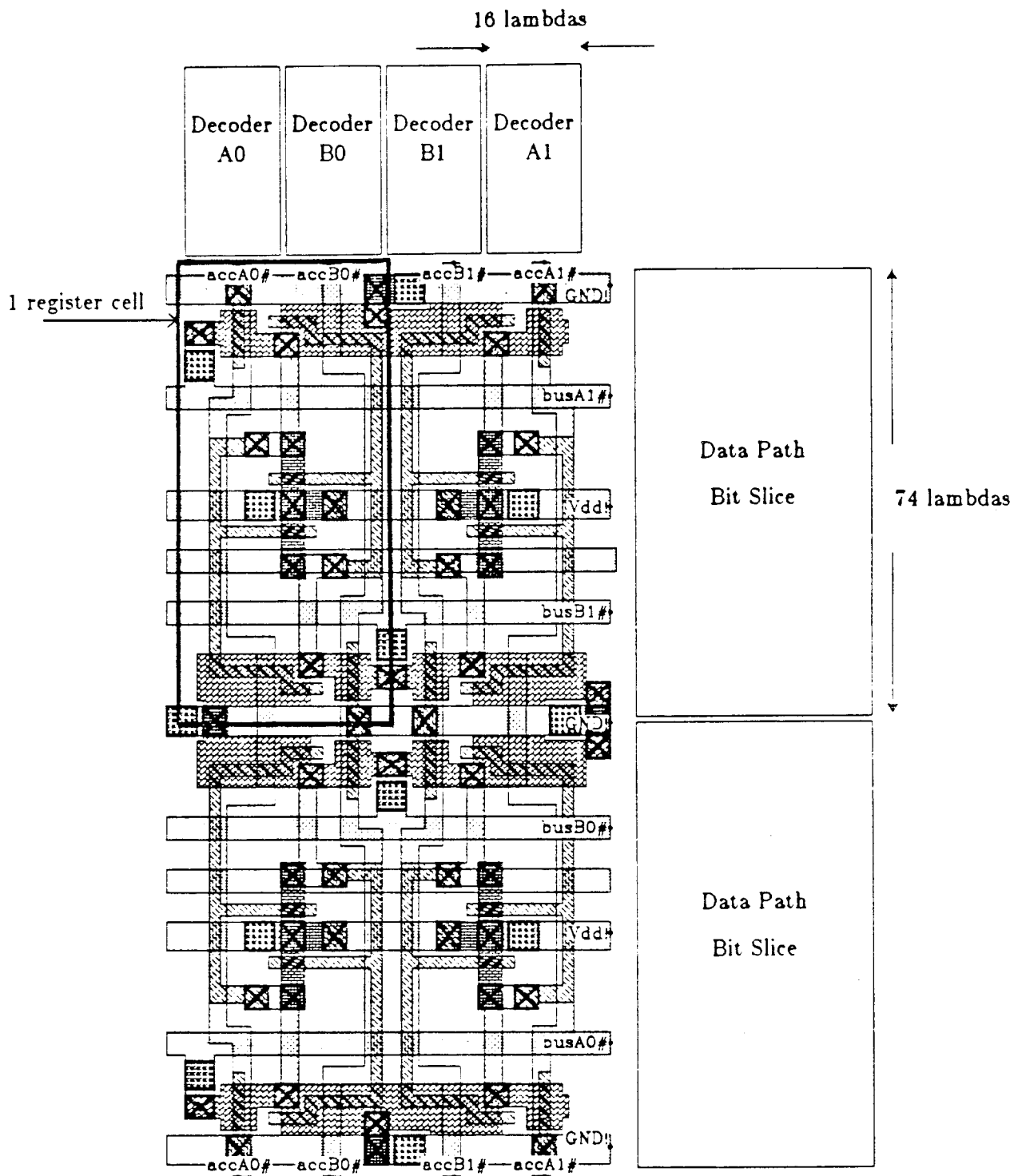


Figure 3-2. Layout of register cell

horizontally. The SPICE simulation results of register file using this memory cell is summarized at the end of Section 3.2.2.

3.2.2. Decoders and Address latches

In the SPUR CPU, the register addresses of an instruction being executed are available to the register file during ϕ_3 of the Ifetch (1st cycle) stage of its pipeline, and registers are read during ϕ_1 of the execution stage (2nd cycle). Therefore accessing the register file can take 3 phases, and it is also pipelined. In ϕ_3 , the register file latches the addresses from busI and busCWP, and drives the decoders. Decoders are then evaluated during ϕ_4 to select two of 138 registers, while bit-lines are precharged. Having selected two word-lines, the register file read proceeds by driving those two word line during ϕ_1 . As a result, the read time in ϕ_1 only consists of driving a word-line and then discharging a bit-line. Likewise, the write access time consists of driving the word-line and the bit-line until the cell is written to the desired value. Since writing uses both busses with large bit-line drivers, writing is done much faster than reading, but only one register can be written at any time.

The block diagram of the register file is shown in Figure 3-3, and the circuit diagrams of the address latches and decoders are shown in Figure 3-4. Clocked CMOS latches and domino circuits are used in the address latches and decoders. These circuits are excellent examples of the design style and charge redistribution avoidance rules discussed in section 3-1.

The RA and RB latches are essentially the same. When read, they latch in different addresses, and when write, they latch in the same address from the RD such that only one register is selected. Shift registers are used to implement RD to store the address of the destination register, which will be written back during ϕ_3 of write stage (4th cycle) of the pipeline. All these units are made of pseudo-static registers, such that each register is refreshed in every cycle. An indefinite pipeline stall mandates this refreshing, since charge on any capacitive node can leak away after certain period of time (~2msec).

The window address for a read is latched in during ϕ_3 , and that for a write is latched in during ϕ_1 . The CWP block in Figure 3-3 decodes the 3-bit window address using static logic gates

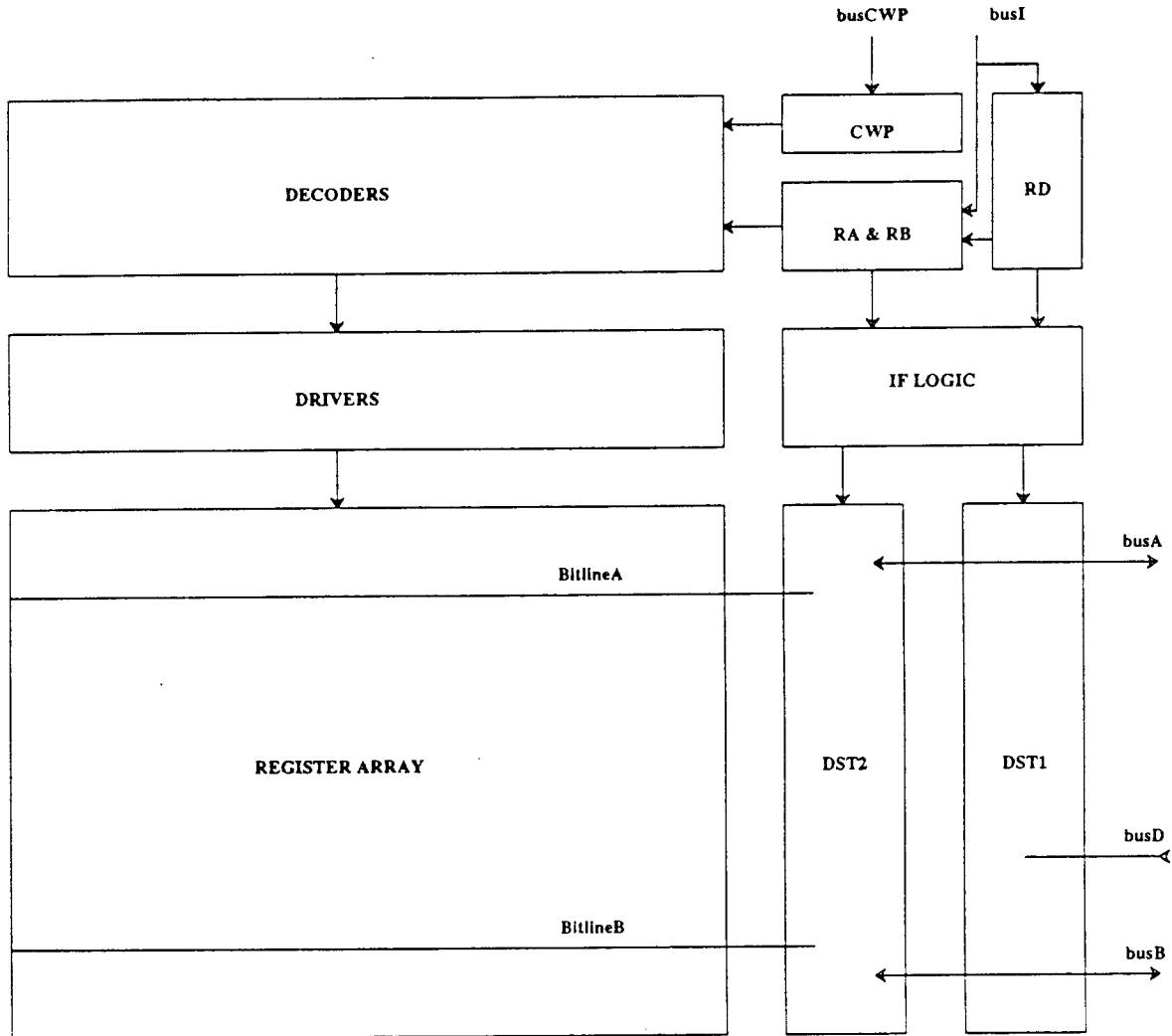


Figure 3-3. Block diagram of the register file

and drives the word-line decoder all within the same phase. Predecoding of the window address uses the special decoder of Figure 3-5, which was suggested in [Kate83]. This decoding circuit implements the overlapping window scheme by mapping two different logical addresses into one physical address. The register numbering for the SPUR CPU can be found in Appendix. The numbering is such that the overlap registers appear in their two windows with 5-bit address that differ in only 1-bit position. It was the numbering of registers that forced special decoders to be used in the register file.

SPICE simulations of the register file are summarized in the Table.3-1. The register file read is one of the most critical paths in the SPUR CPU chip. A testchip that contains the register file and the

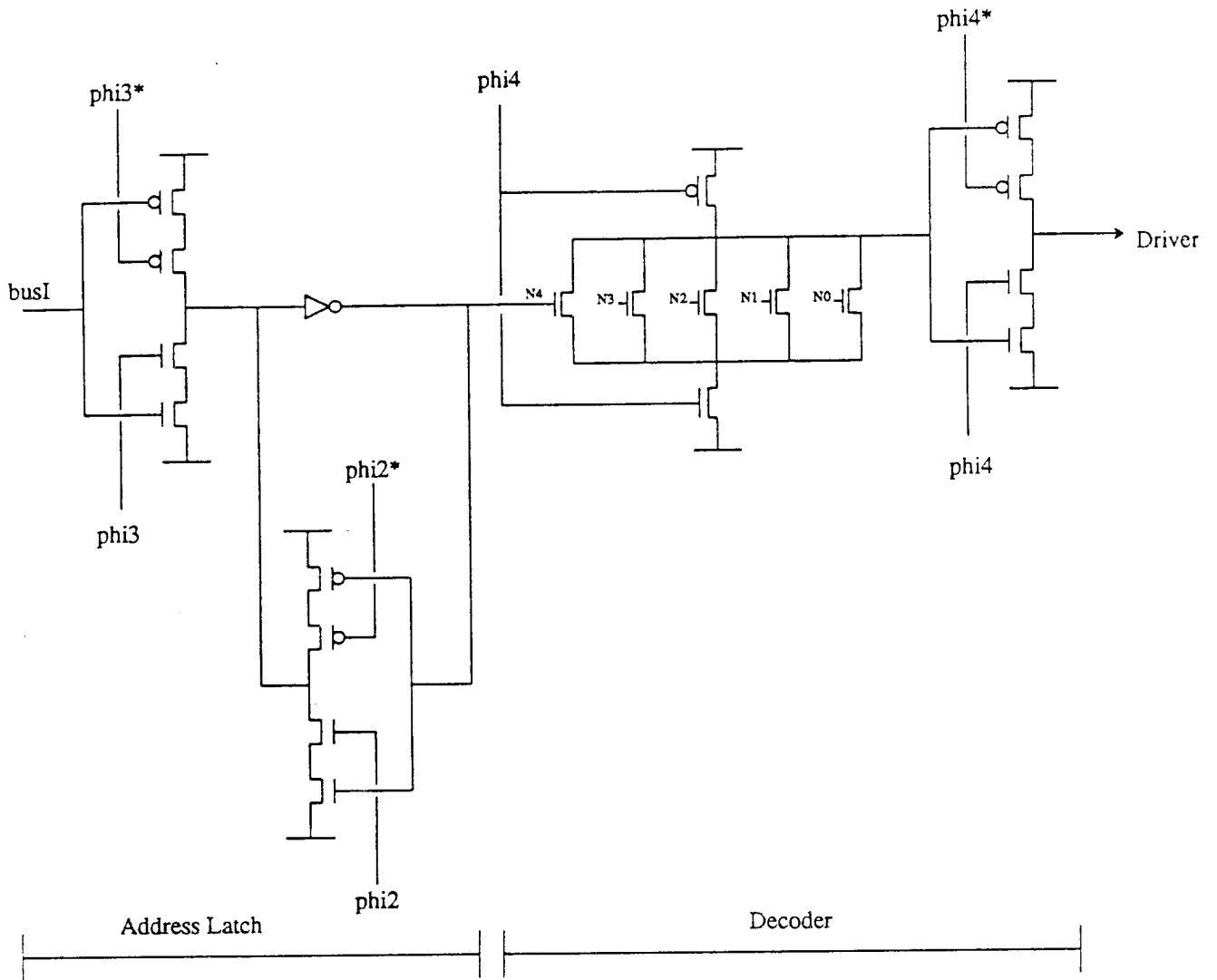


Figure 3-4. Circuit diagram of address latches and decoders

internal forwarding logic was assembled and fabricated in 2 μm CMOS process at XEROX PARC. The test results are presented in section 3-5. The experimental results are very close to the simulations.

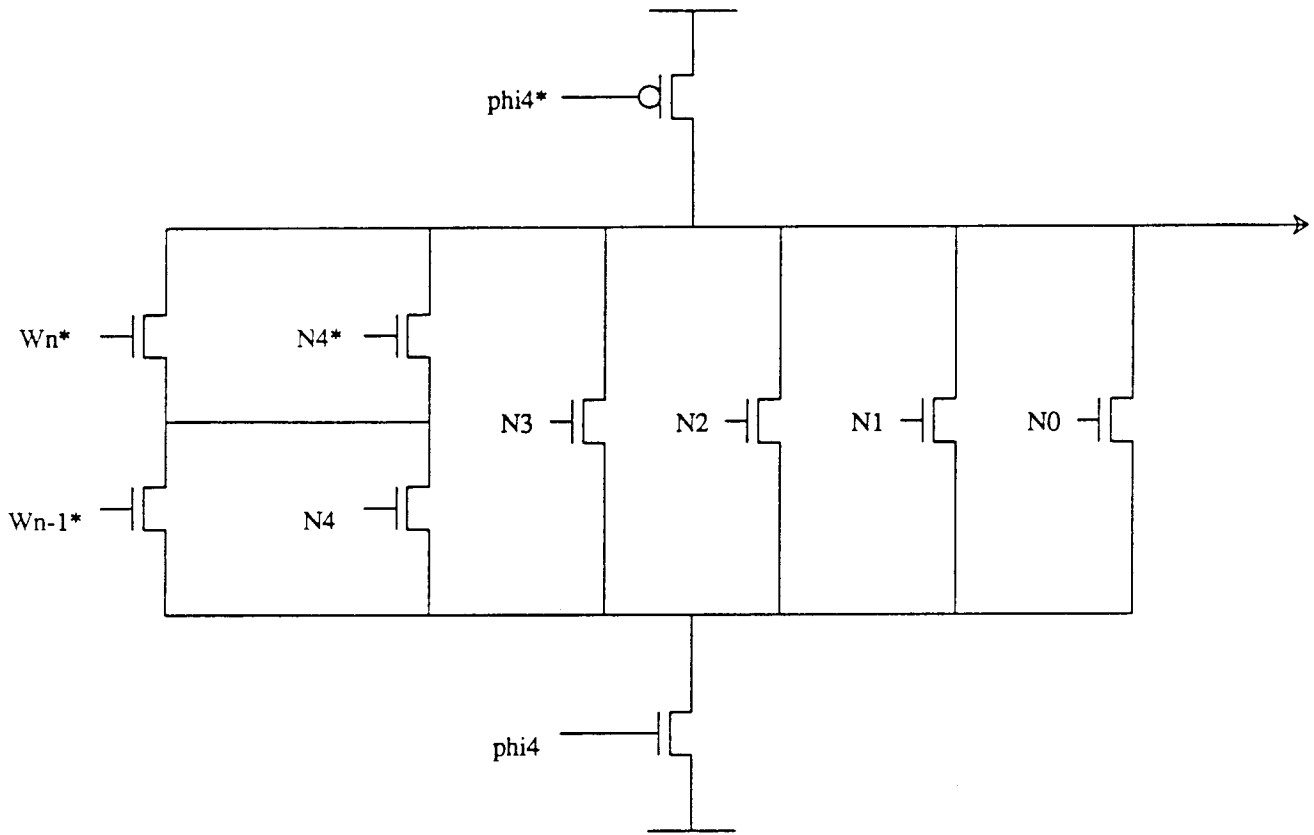


Figure 3-5. Decoder for the overlapping window registers

3.3 Internal Forwarding

In the pipelined execution of the instruction stream, data interdependencies among instructions may arise. In both RISC II and SOAR, where 3-stage pipelines were employed, the data interdependency exists only between two consecutive instructions [Kate83] [Joan85]. That is, the result of any instruction needs to be forwarded only to the next instruction, requiring a single internal forwarding.

In the case of a 4-stage pipeline, the data interdependencies may exist among 3 consecutive instructions since the write-back stage of the pipeline is delayed by 2 cycles after the execution stage. This data dependency among 3 consecutive instructions can only be resolved by the double internal forwarding scheme illustrated in Figure 2-2. The results available from the execution stage, therefore, need to be stored in a temporary registers for 2 cycles. Two temporary registers, called DST1 and DST2, are provided to store the results from two consecutive instructions. A third

Block/Operation	Phase	Delay(nsec)
CWP	phi3, phi1	8
RA & RB	phi1, phi3	6
Decoder	phi2, phi4	6.5
Read	phi1	16.5
Write	phi3	10

Table 3-1. Signal Delays in Register File : SPICE simulation

instruction in the pipeline might use either of these.

The DST1 receives the result of instruction at the end of the Exec stage when execution is completed. This result is passed onto DST2 during the Mem stage and is written into the register file from DST2 during the write stage. Since the register file writing is done during phi3 while reading is done during phi1, the forwarding from DST2 is still necessary to provide the temporary result that might be read in phi1. This occurs when the instruction with the Exec stage coincides with the write stage of the previous instruction.

The data dependencies can be detected by comparing the addresses of register references in the instruction currently in the Ifetch stage with addresses currently held in RD for later use in the write stage. Four equality comparators are needed to detect the following conditions (see figure 2-2):

$$DST2_to_busA : RS1,I3 = RD,I1$$

$$DST2_to_busB : RS2,I3 = RD,I1$$

$$DST1_to_busA : RS1,I3 = RD,I2$$

$$DST1_to_busB : RS2,I3 = RD,I2$$

The comparisons must be done in parallel with the decoding of register file, such that busA and busB are driven by the temporary results during phi1 if internal forwarding conditions are met.

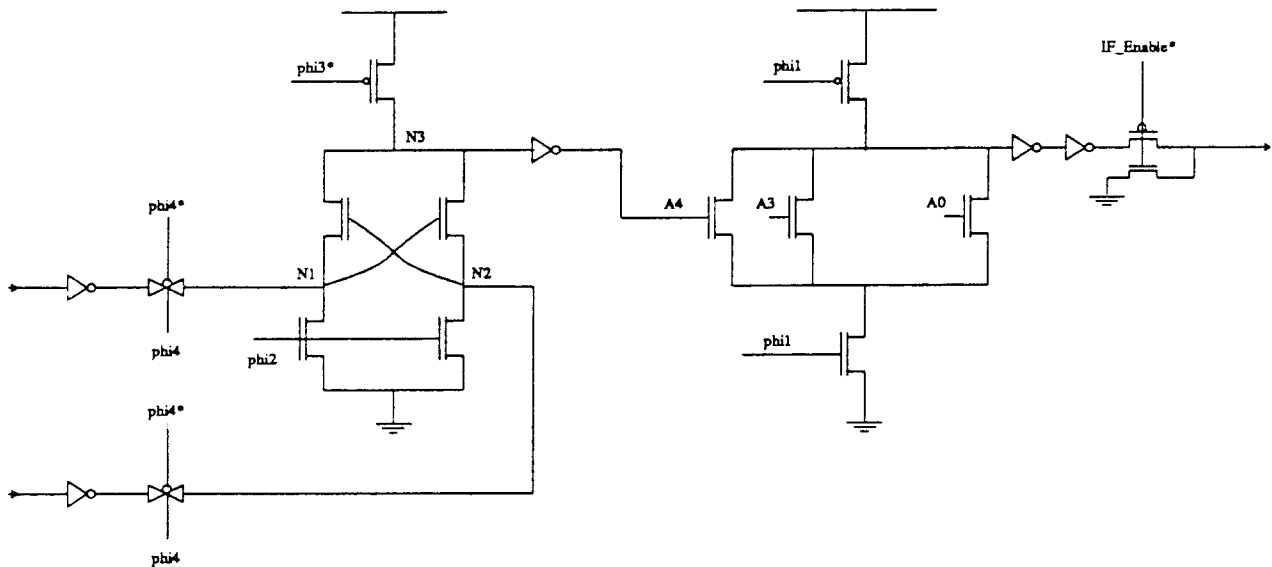


Figure 3-6. Comparator using dynamic CMOS XOR circuits

The comparators must be fast to keep the cycle time short, and must be laid out compactly to fit between the DST's and the register file decoder, as shown in Figure 3-1. Thus, a fast dynamic XOR circuit as shown in Figure 3-6 has been used. Bit-wise comparisons are done using this dynamic XOR circuit, and then the outputs of these XOR's are fed into the Domino circuit for an address match. Since this XOR circuit uses only true values of inputs, routing and area consumption are minimal. The comparator circuit operates as follows:

- (1) During phi2, node N1 and N2 are discharged to GND.
- (2) Node N3 is precharged to Vdd during phi3
- (3) The XOR is evaluated during phi4 by opening transmission gates connected to nodes N1 and N2. Notice that nodes N1 and N2 are actively driven through the transmission gates. If two inputs are different, the node N3 will be fully discharged to GND. If both inputs are low, it will remain high at precharged value of Vdd. However, if both inputs are high, the logic level at node N3 will be momentarily unstable depending on the arrival of

inputs, but will be restored to the value as low as $(V_{dd}-V_t)$ by the time inputs get stabilized. In any case, the final value of node N3 will be stabilized within ϕ_4 .

- (4) The Domino circuit (OR) is evaluated for an address match during ϕ_1 . At the same time the IF enabling signal is asserted, the output of Domino gets through to the output of the IF logic block. The outputs of the IF logic remains at GND until a match is made. In the case of a match, the register file busses are disconnected from busA/busB during ϕ_1 , as explained below.

After conditions for internal forwardings are met, busA and/or busB must be driven by the values stored in the DST's instead of the values from the register file. The circuits in Figure 3-7 are used to disconnect the register file busses from busA and busB only if internal forwarding needs to be done. In the case where two forwardings are destined for the same bus, e.g. DST2_to_busA and DST1_to_busA, the result from DST1 shall drive the busA because the result in DST1 is more recent than that in DST2. Since DST1 is placed close to busA, the DST1_to_busA will always precede the DST2_to_busA.

The circuit used to implement double internal forwardings must not lengthen the read time of register file during ϕ_1 . However, the signal delay through these circuits are added to the read time of register file. Thus, inverters and transmission gates used for the internal forwarding must be designed to minimize the signal delay through them. The inverters are designed such that their sizes increase by approximately 3 times each stage [Mohsen79]. Since register file busses are changing very slowly when reading, including a few inverters between register file bus and busA(busB) actually helps getting a fast transition on busA and busB. With the circuits designed as above, the extra delay that is added to the read delay was not more than 3 nsec. The SPICE result shown in section 3.2.2 includes this extra delay. The signal delays within IF block are not significant compared to the read delay of register file. SPICE simulation of IF logic indicates the delay in ϕ_1 is about 12 nsec.

One advantage of separating the busses is that capacitances on busA and busB that the DST's drivers must drive, are reduced by disconnecting the highly capacitive bit lines of the register file. This reduction in capacitance in turn eases the design and layout of internal forwarding circuits.

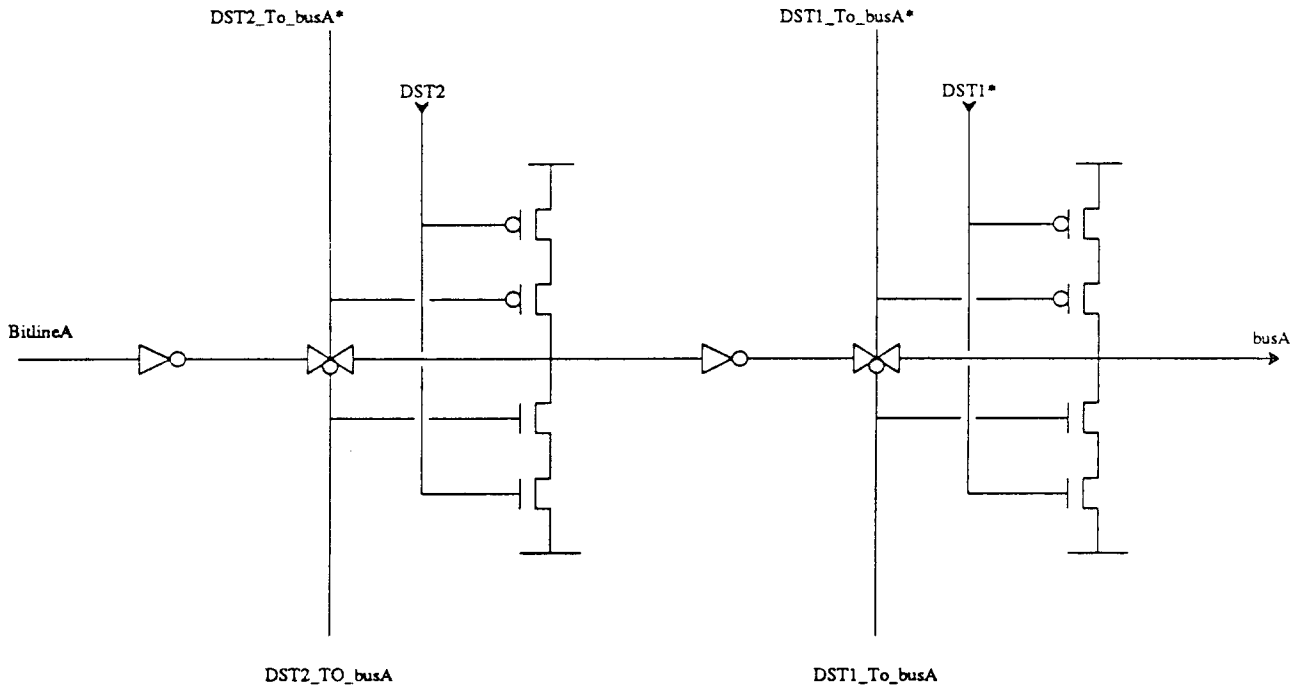


Figure 3-7. Circuit diagram of double internal forwarding logic

3.4 Functional Units

The three functional units, as introduced in section 2, are the ALU, the shifter, and the byte extractor and inserter. The shifter and the byte extractor/inserter are implemented simply using pass gate logic - a CMOS transmission gate followed by an inverter. These blocks latch in their inputs during ϕ_2 and drive the output bus (**busD**) in ϕ_4 . They have more than a phase to evaluate their outputs, and thus timing is insignificant compared to the ALU. The optimization policy of these blocks is to minimize the silicon area. The layouts of these two blocks can be found in Appendix.

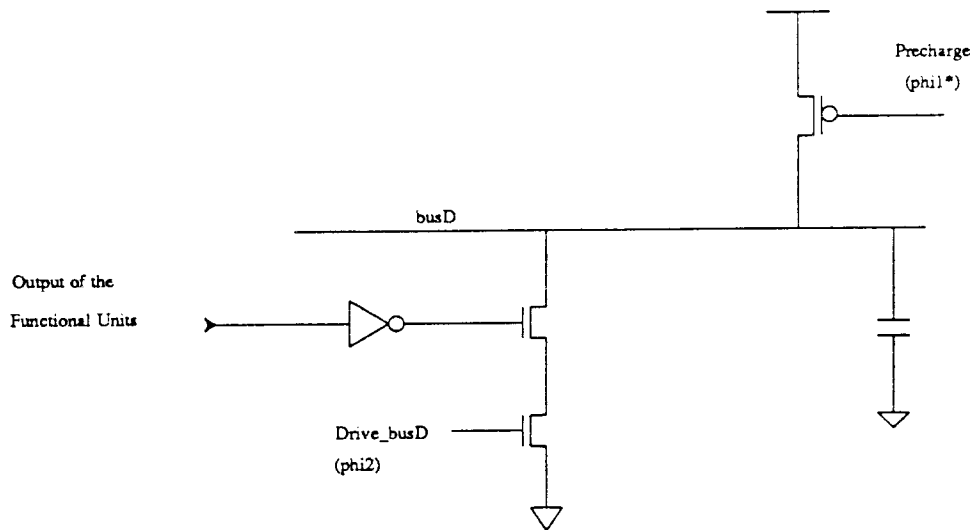


Figure 3-8. The driver used in functional units to drive busD

The output section of all functional units drive busD, making busD highly capacitive. To speed up the data flow through this bus, busD is precharged to a high level before it is driven during phi4. The circuit of Figure 3-8 is used for all output drivers of functional units. As discussed at the beginning of this section, precharging highly capacitive busses saves the area and speeds up the data flow.

The ALU performs Add, Subtract, XOR, OR, and AND. The ALU design is essentially an adder design since the latter 3 logical operations are readily available from the input section of most adders. Subtract can also be done by complementing one of the inputs, and setting the carry-in bit before performing the addition. The ALU is divided into 3 subblocks: the input section, the carry propagation block, and the sum block which is also an output of ALU. The ALU operates in 3 phases. The input section latches in inputs in phi2, and the carry propagates in phi3, and finally the sum is computed and the result of ALU is put onto busD during phi4.

The carry propagation delay of ALU is the most critical path within the functional units, which is done during phi3. The secondary critical path in phi3 is register file write. Thus, the ALU propagation block is designed to have about the same delay as the register file write. The 8-bit block carry

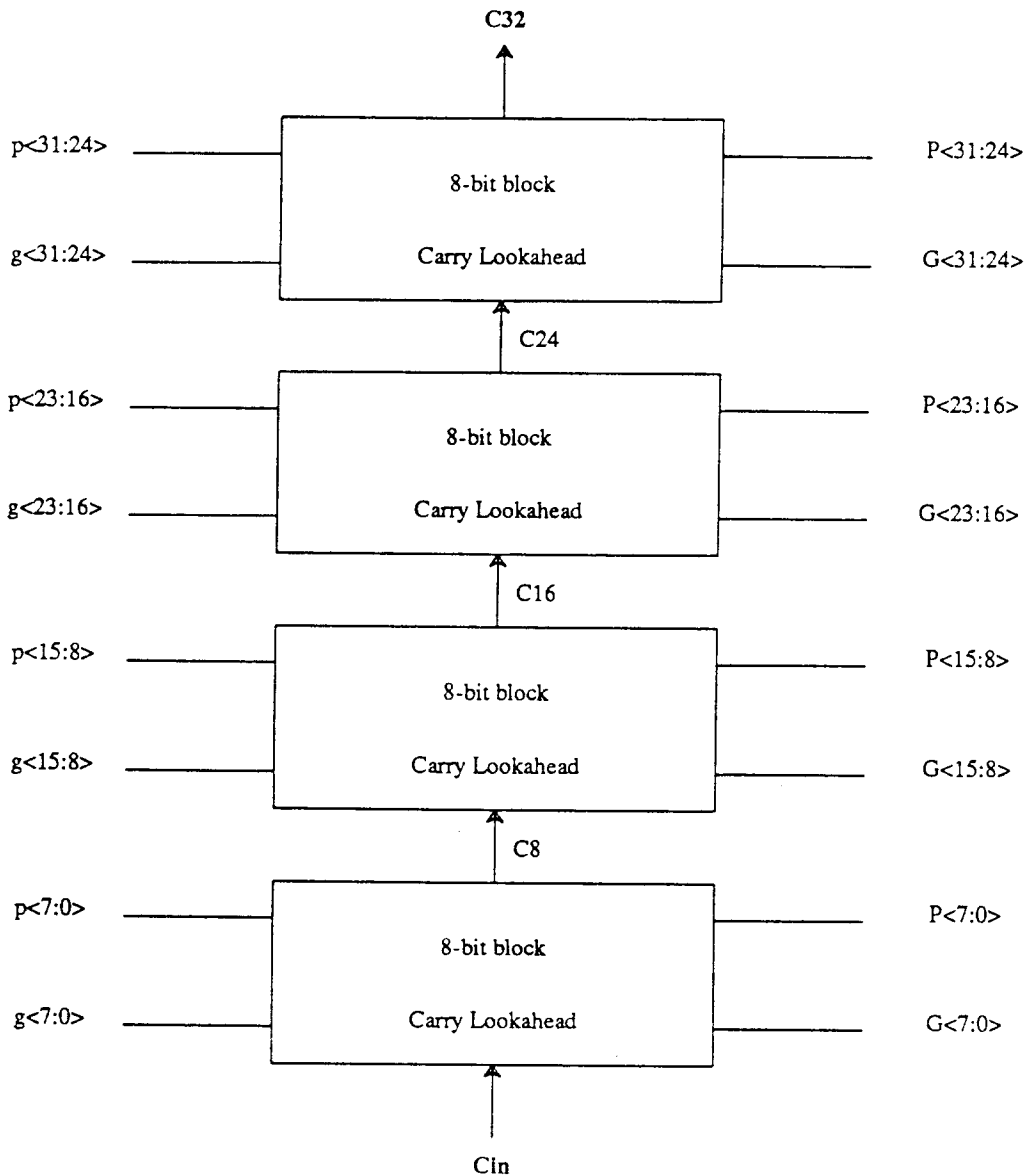


Figure 3-9. Block diagram of ALU carry propagation section

lookahead scheme is chosen among many different implementation styles available, for being reasonably fast while less area consuming than a full 32-bit carry lookahead adder [Kong85]. Four 8-bit block carry lookahead subblocks are cascaded to form a 32-bit carry propagation block as shown in Figure 3-9. The Domino circuit implementation of the above carry propagation scheme meets the timing requirement.

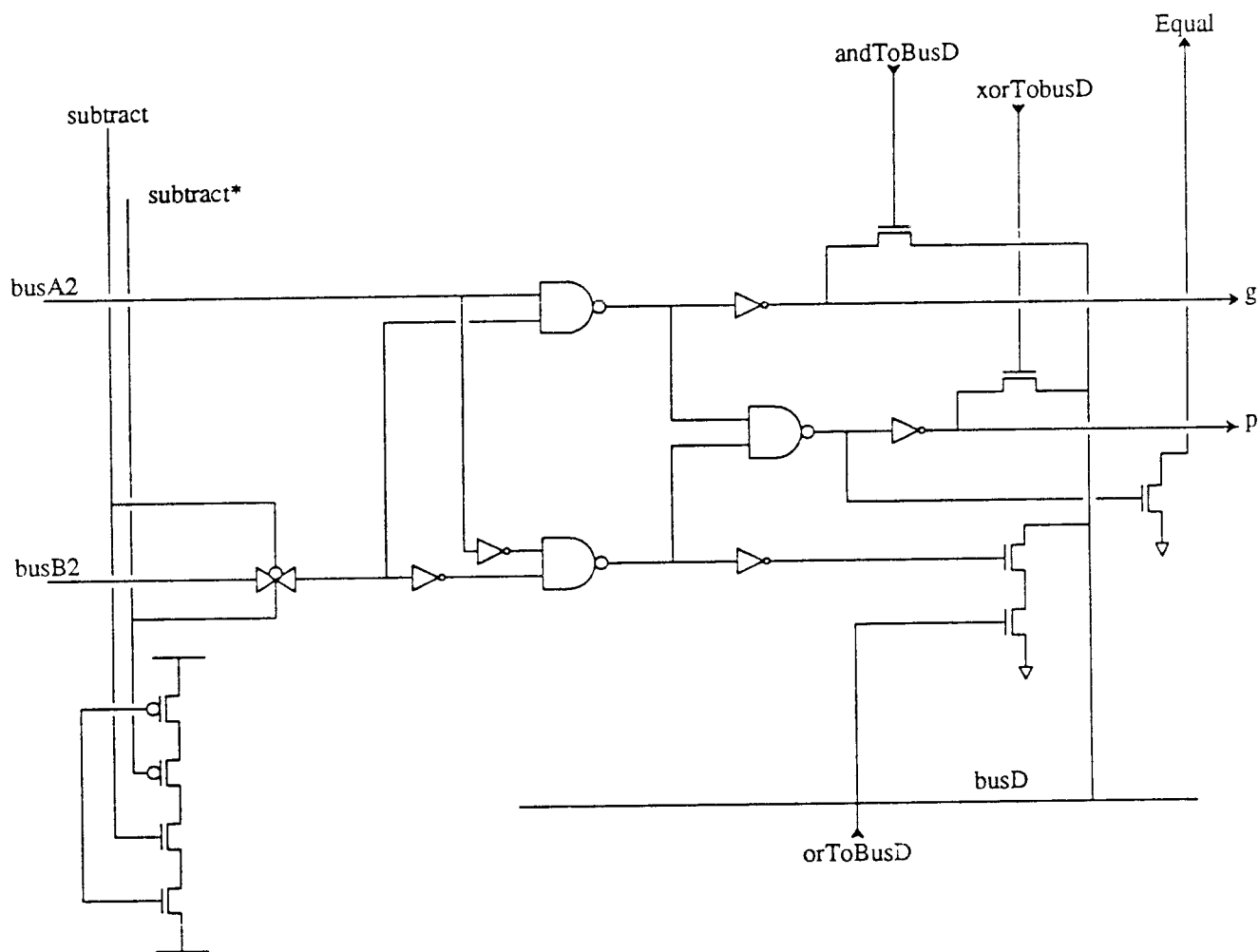


Figure 3-10. Input section of the ALU

The circuit diagram of the ALU input section is shown in Figure 3-10. Note that three logical operations are included in this section, as well as circuits complementing one input for subtract. The logic diagram and floor plan of the carry propagation block are presented in Figures 3-11 and 3-12, respectively. The critical path within an 8-bit block is marked with arrowed line in Figure 3-11. Finally the circuit diagram of the sum block is shown in Figure 3-13. As with other functional units, the output merely discharges the precharged busD. However, in this case the sum logic is included in the output driver. The layout of the ALU can be found in Appendix. The results of SPICE simulations for the ALU are summarized in Table 3-2.

Block/Operation	Phase	Delay(nsec)
ALU Input Section	phi2	8.5
Carry Propagation	phi3	14.0
Sum Block(output)	phi4	10.0

Table 3-2. Signal Delays in ALU : SPICE simulation

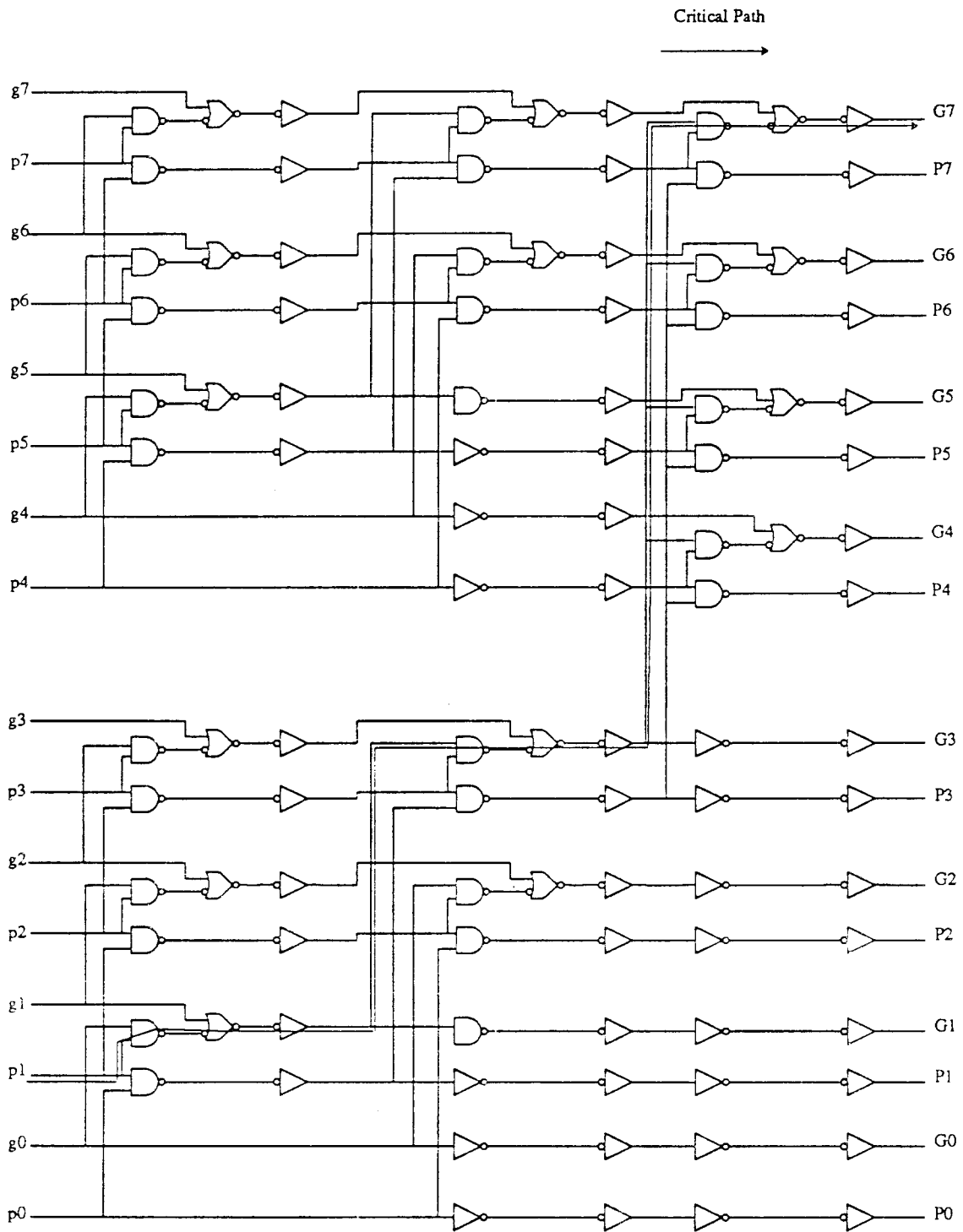
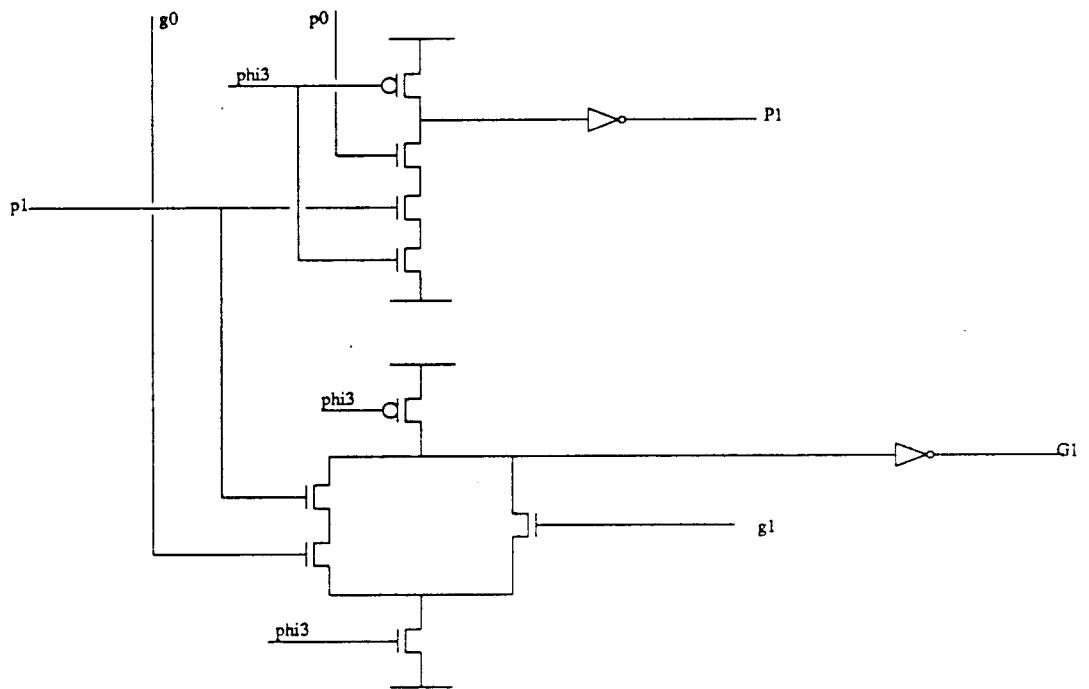


Figure 3-11. Logic diagram of 8-bit carry lookahead block


1	2	5	4
1	2	5	4
1	3	5	4
0	3	5	4
1	2	3	4
1	2	3	4
1	3	3	4
0	3	3	4

1: Domino logic 1



2 and 5 : Same as 1 but with different sizes of transistors

4: same as 1 but without inverters

3: buffers 

0: empty area (routing)

Figure 3-12. Floor plan of 8-bit carry lookahead block

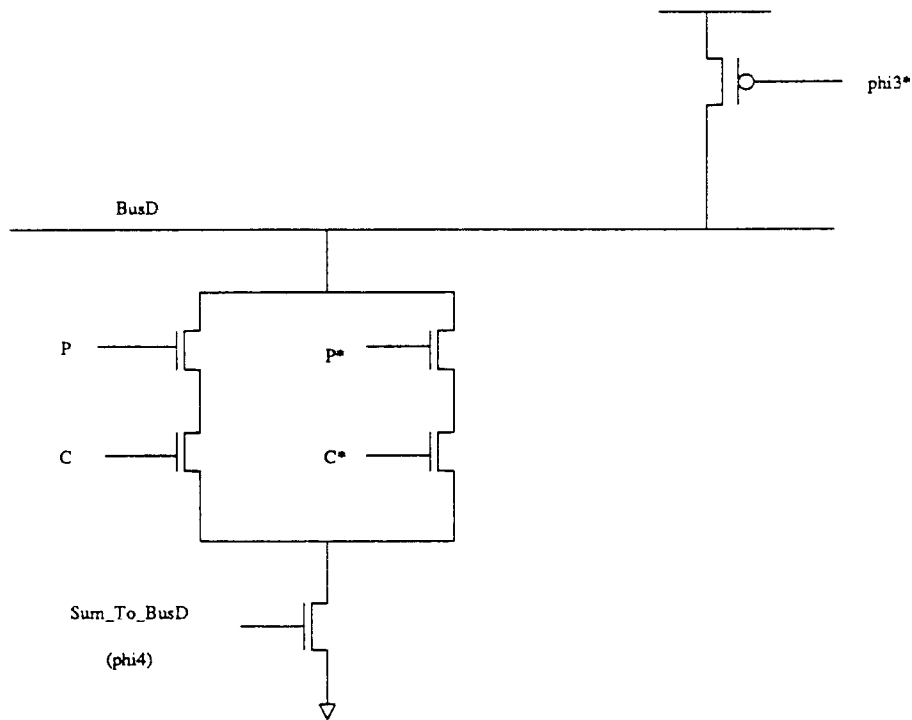


Figure 3-13. Sum block - the output section of the ALU

3.5 Design metrics of the lower data path

The design metrics for the SPUR CPU lower data path is presented in Table.3-3. It provides an approximate design time spent on both the circuit design and the layout, in terms of man-months. Extra time spent on the layout due to a change in design rules is not included in the design time calculation. The transistor count on the SPUR CPU chip reaches over 120,000. About 60,000 transistors are in the lower data path and master control, 40,000 in the instruction unit, and about 20,000 in the upper data path. Transistor counts for various blocks within the lower data path are presented in the same table, as well as the sizes of these blocks.

Block	Layout Area		# transistors	Regularity*	Design Time	
	Height (lambda)	Width (lambda)			Circuit (Man Month)	Layout (Man Month)
Register file	4090	4520	39800	86	1.0	2.0
IF Logic	220	780	210	2	0.5	0.5
DST1, DST2 & MBR	3150	1400	4480	32	0.25	1.0
Byte Ext/Ist	3150	510	580	7	0.1	0.25
Shifter	3150	335	680	15	0.1	0.25
ALU	3150	895	4200	21	0.5	0.75
KPSW & UPSW	3150	800	1770	32	0.1	0.25
TRAP LOGIC	633	1300	730	1.2	0.5	1.5
Bus Interface	3150	448	1090	8	0.2	0.5
Lower Data path	4090	9180	53540	29	3.25	7.0

* total number of transistor divided by number of transistors designed

Table 3-3. Design Metrics of SPUR CPU's Lower Data path

3.6 Experimental results - A test chip for the data path

A testchip containing the register file and internal forwarding logic was assembled and fabricated in 2 μm N-well CMOS process, at XEROX PARC. Figure 3-14 shows the checkplot of the testchip, and microphotographies of various parts in the testchip are shown in Figure 3-15. The size of the chip is 1.04mm by 1.04mm including pads, and it was packaged in 144 PGA chip carrier.

The following functions were tested using Tektronix DAS 9100 system, and found to work.

- (1) Register file read and write
- (2) Register window overlapping
- (3) Double internal forwarding
- (4) Repetition of previous operations in case of a pipeline stall

Due to the limitation on the DAS system, a clock frequency of 0.5MHz was used to operate the testchip. With this clock frequency, the reading delay in register file was measured and shown in Figure 3-16. The testchip was assembled such that we can observe the word line, the bit line, and the address lines in the decoder, with externally supplied clocks. Also several pads are connected such that I/O pad delay can be easily measured. By observing these signals and delays we could estimate the read delay. It was observed that this delay was less than 20nsec. The SPICE simulation showed the same delay at about 16.5nsec.

Even though the testchip contains only a small part of the whole CPU chip, it does include circuits that need to be tested before the fabrication of the real CPU chip. These circuits reflect the design style and the design rules that we followed to avoid any charge redistribution problems in dynamic circuits, such as dynamic decoders of register file and the comparator circuits used to implement IF logic. The register file was implemented with 6T CMOS SRAM cells, and also needed to be tested to estimate the read delay. Since this delay is the most critical path in the entire chip, it is then used to determine the minimum phase and hence the cycle time. The testing results, therefore, gave us a confidence in designing circuits and achieving timing goals.

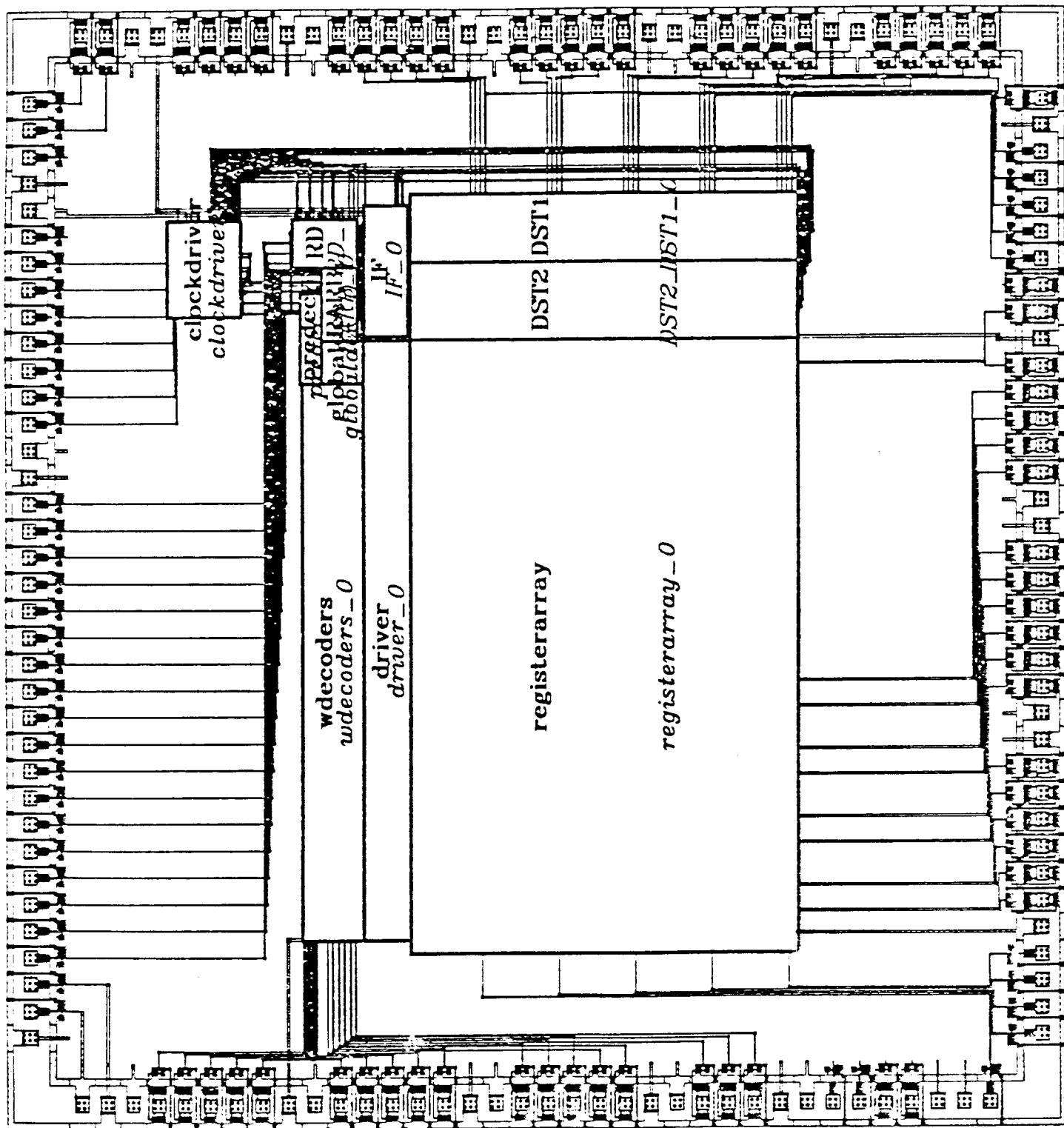
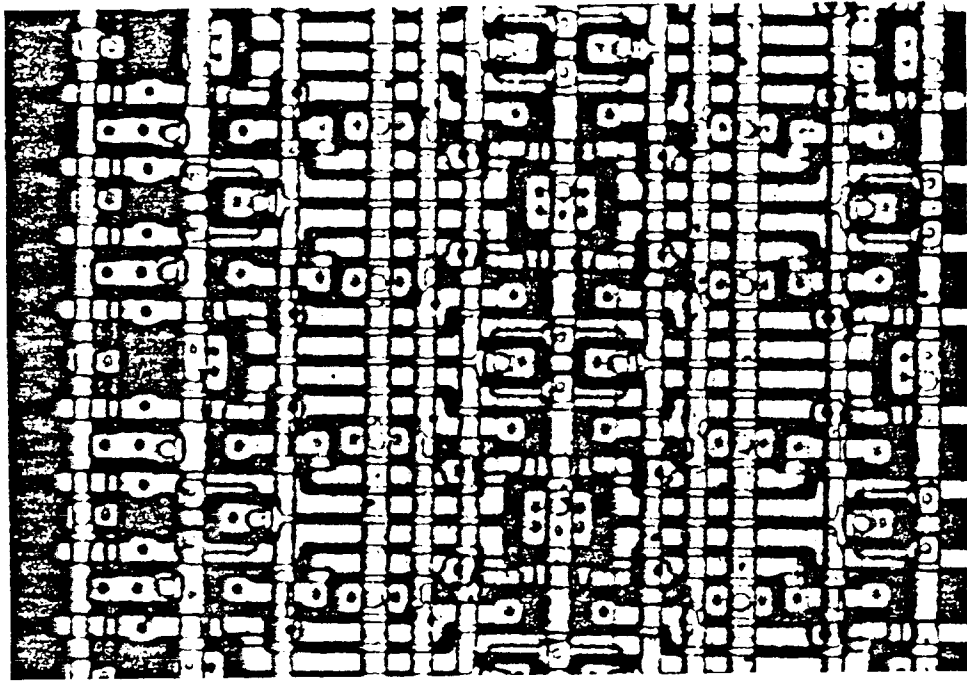


Figure 3-14. Testchip



Register Cell

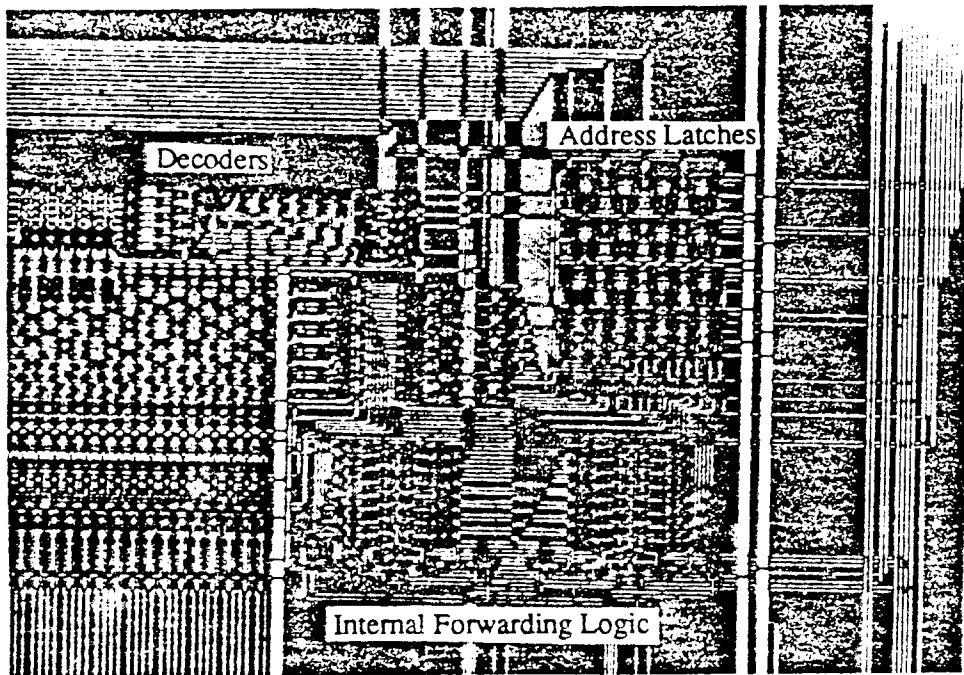
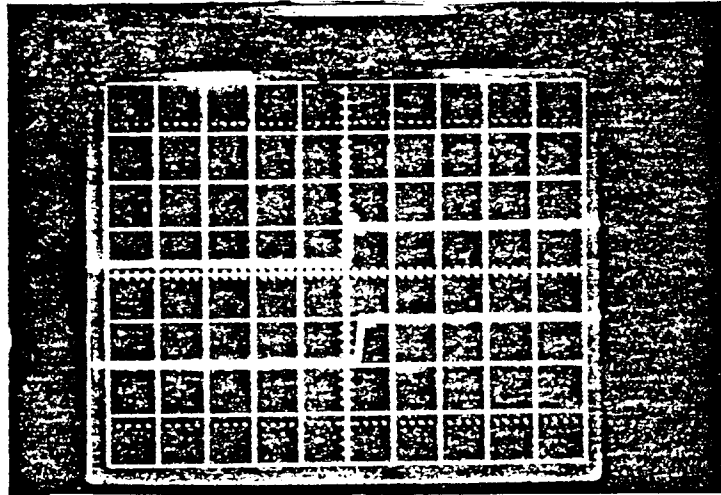


Figure 3-15. Microphotographies of the testchip

(a) I/O pad delay

(Scale: Horizontal 50nsec/div, Vertical 5V/div)

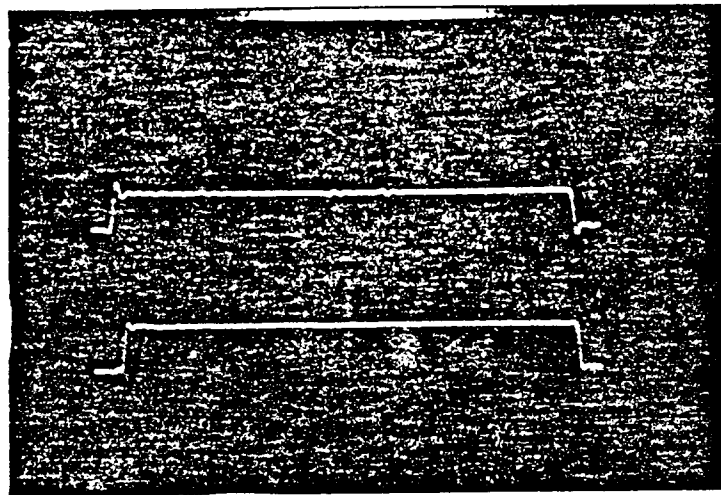


input

output

(b) Delay on word line

(Scale: Horizontal 100nsec/div, Vertical 5V/div)

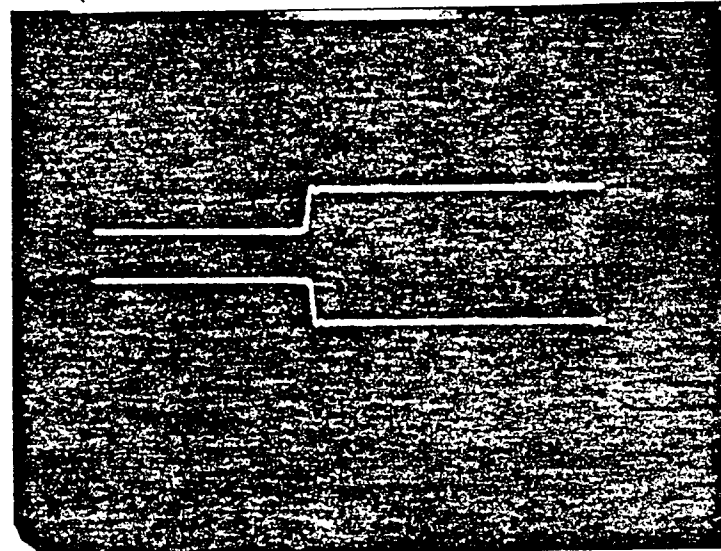


Phi1

Word line

(c) Delay between word line and bit line

(Scale: Horizontal 100nsec/div, Vertical 5V/div)



Word Line

Bit Line

Figure 3-16. Test Results

4. ALTERNATIVE IMPLEMENTATIONS

The design of microprocessors often requires interaction among different levels of design. For, example, a feedback from the low-level design can affect the high level design in various ways, and vice versa. In this section, I will look at some design alternatives in the data path implementation, and the effects these alternatives would have on microarchitecture and performance of the SPUR CPU. First, using a dual-port read and write memory cell in the register file will be examined. This alternative can affect the pipeline organization. Secondly, effects of having dynamic memories on the chip will be considered. The most difficult problems using dynamic memories are the process dependency and the complexity of the sensing and refreshing circuits.

4.1. Dual-port read and write memory cell

A critical issue in register-oriented architectures is the register file organization and its timing. A variety of memory configurations can yield a wide range of register file bandwidths. Therefore, one needs to understand tradeoffs involved in selecting one configuration over another.

As pointed out earlier in Section 2, some pipeline organizations require suspension of pipeline operations to avoid register write conflict. If the register file is capable of dual-port writing, this conflict can be resolved, and pipeline suspension due to the conflict of two simultaneous writes is no longer required. The SPUR CPU avoids this conflict by changing the organization of the pipeline, such that register write conflict never takes place.

The design of the register cell can vary by having different arrangements of word lines and bit lines. These lines can be shared for both reading and writing, or separate lines can be provided for reading and writing. Tradeoffs between these two approaches involve economy of layout area, the speed, and the concurrency of operations using them. The register cell used in the SPUR CPU follows the shared bit line approach. In this cell, dual-port reading was implemented relatively easily by separating the word lines, but writing must use both bit lines to ensure a safe writing. Figure 4-1 shows an alternative register cell that has dual-port writing capability. This 9T pseudostatic RAM cell was used in Caltech OM-2[MeCo80] and HP FOCUS chip[Beyers81].

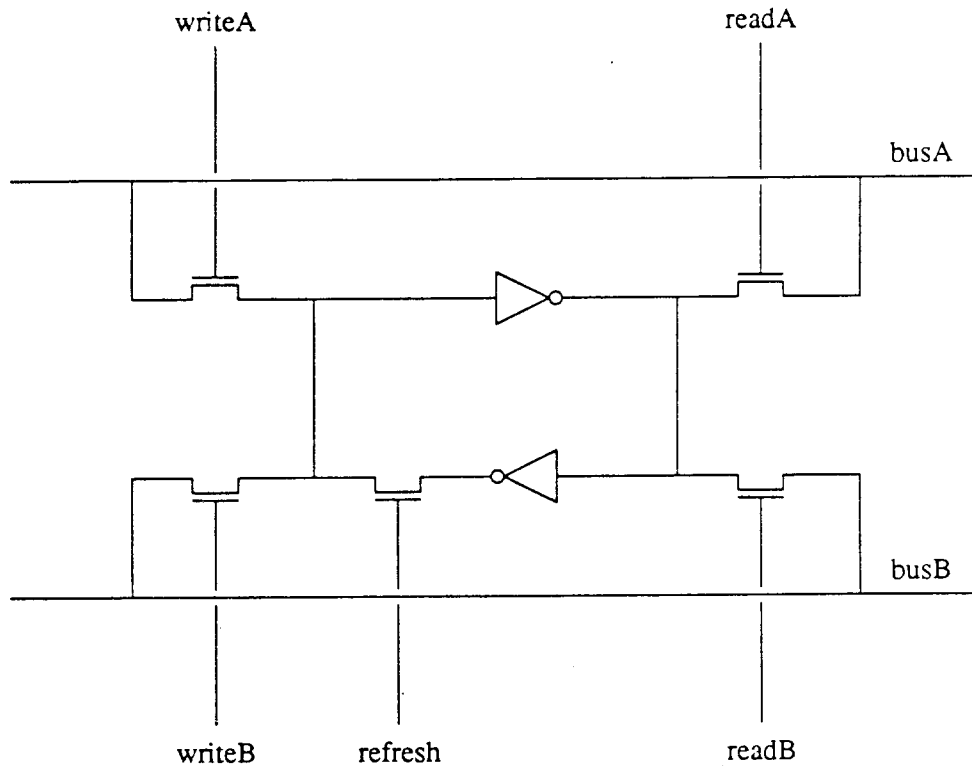


Figure 4-1. 9T dual-port R/W memory cell

The 9T cell provides both dual-port read and dual-port write capabilities of the register file. However, area consumed by having 9 transistors often discourages the use of this cell. Therefore, the design of a memory cell with dual-port capabilities but including less transistors has been considered. If the 6T SRAM cell can be written using only one bit line, it is an excellent candidate for the dual-port write memory cell. The reason both bit lines are needed for writing is as follows: When the cell is written, the flip-flop(cross-coupled inverters) is accessed through the NMOS access transistors. To change the state of the flip-flop if it has a different value, we must actively pull one side down while pulling the other side up to make sure that the state changes. Since both actions are done through the NMOS access transistors, pulling up is not as effective as pulling down because the high signal is degraded as it passes through the NMOS access transistor. Thus, writing is done mostly by pulling down one of the bit lines according to the value to be written. This observation leads to the conclusion that writing "0" can be done using only one bus, as for reading, but it is the

writing "1" that prevents us having dual-port write with the 6T cell.

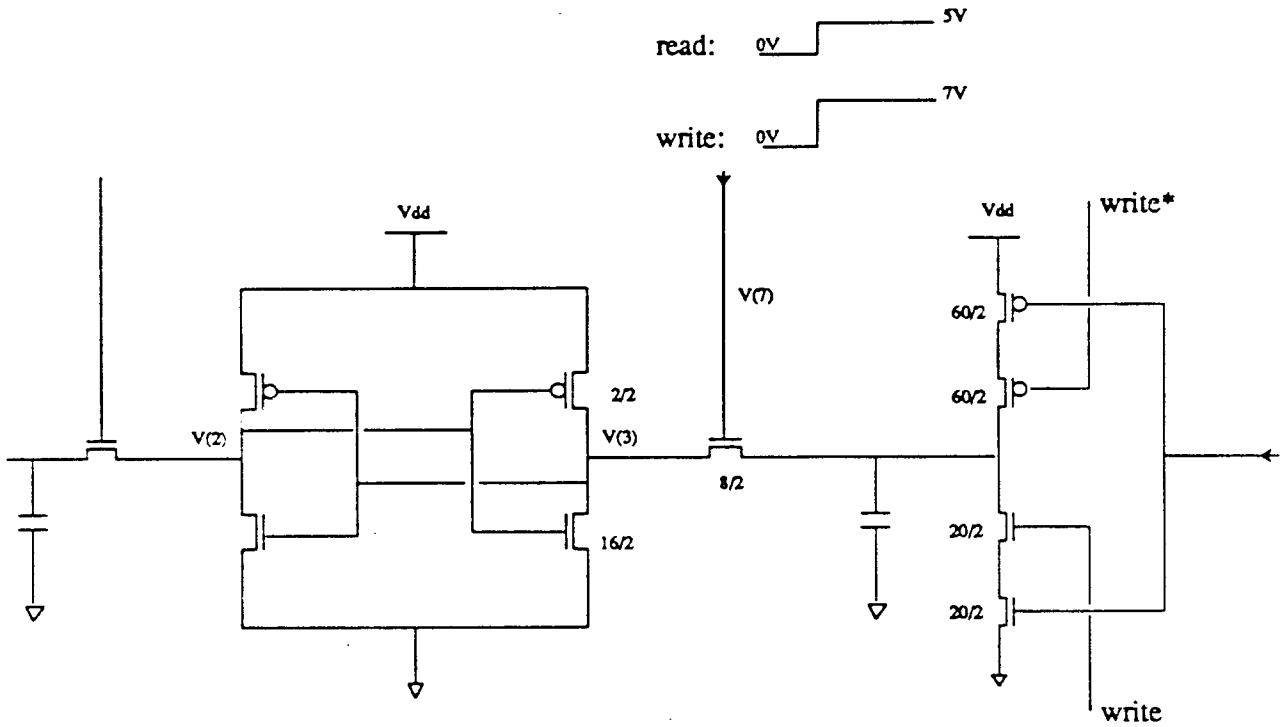


Figure 4-2. CMOS 6T dual-port R/W memory cell

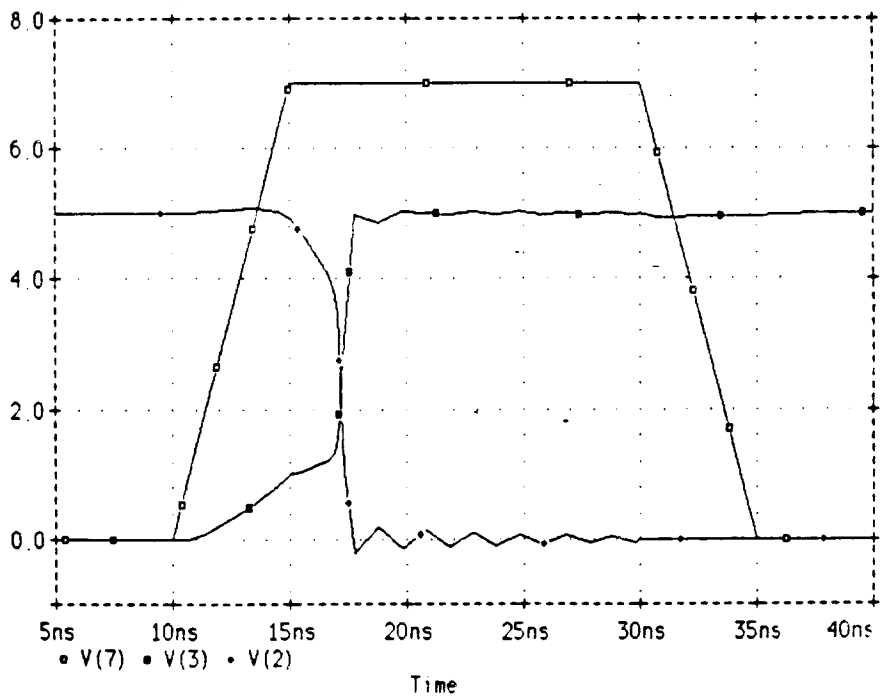


Figure 4-3. SPICE simulation on CMOS 6T dual-port R/W memory cell

If we use different operating voltages for controlling this 6T cell with appropriate device sizes, however, writing "1" can be done safely with only one bus. Figure 4-2 shows one possible implementation of this idea. The sizes of the transistors are kept the same as that of the cell used in the SPUR CPU, but voltage levels on word line are different for reading and writing. For reading, the word line is driven at 5.0V by a normal CMOS inverter. For writing, a special driver is used to drive the word line at 7.0V. The SPICE simulation of writing with only one bus is shown in Figure 4-3. A CMOS counterpart of bootstrap circuits in NMOS can be used for the special driver. However, such a CMOS driver has some disadvantages which make it difficult to implement. First, junction breakdown is more probable because of higher operating voltage. This becomes a more serious problem as minimum dimensions shrink with advances in technology. Secondly, the bootstrap capacitor can take up a substantial amount of silicon area, since bootstrap capacitance ought to be comparable to the capacitance it drives.

Still another possible use of 6T SRAM for dual port write memory, based on the above idea, is shown in Figure 4-4. NMOS pull-up transistors are used in place of CMOS pull-up transistors, with their gates tied to the V_{dd}. Note that the W/L ratio of transistors are different from the cell using CMOS pull-ups. The word line operating voltages are ~3.5V for the reading, and 5V for the writing. The bit lines are precharged to ~4V prior to the reading, and driven at full 5V when write. Results of SPICE simulations showing R/W of this cell are shown in Figure 4-4b. Note that both read and write are done safely, especially internal node does not rise more than 0.6V until reading ("0") is done. A driver with an NMOS pull-up can be used to drive the word line at ~3.5V for the reading, while a CMOS inverter can be used to drive the word line for the writing, eliminating the need for bootstrap circuits.

This NMOS 6T cell can be laid out in much less area than a CMOS cell since N-to-P or well-to-well spacing is no longer required. Thus, it can result in a denser memory array while maintaining the same speed of the CMOS SRAM array. The only disadvantage of this cell is the static power consumption due to the NMOS pull-ups. This static power can be reduced by lengthening the NMOS pull-ups, or the gates of these pull-ups can be pulsed with an extra control line. The area saved by

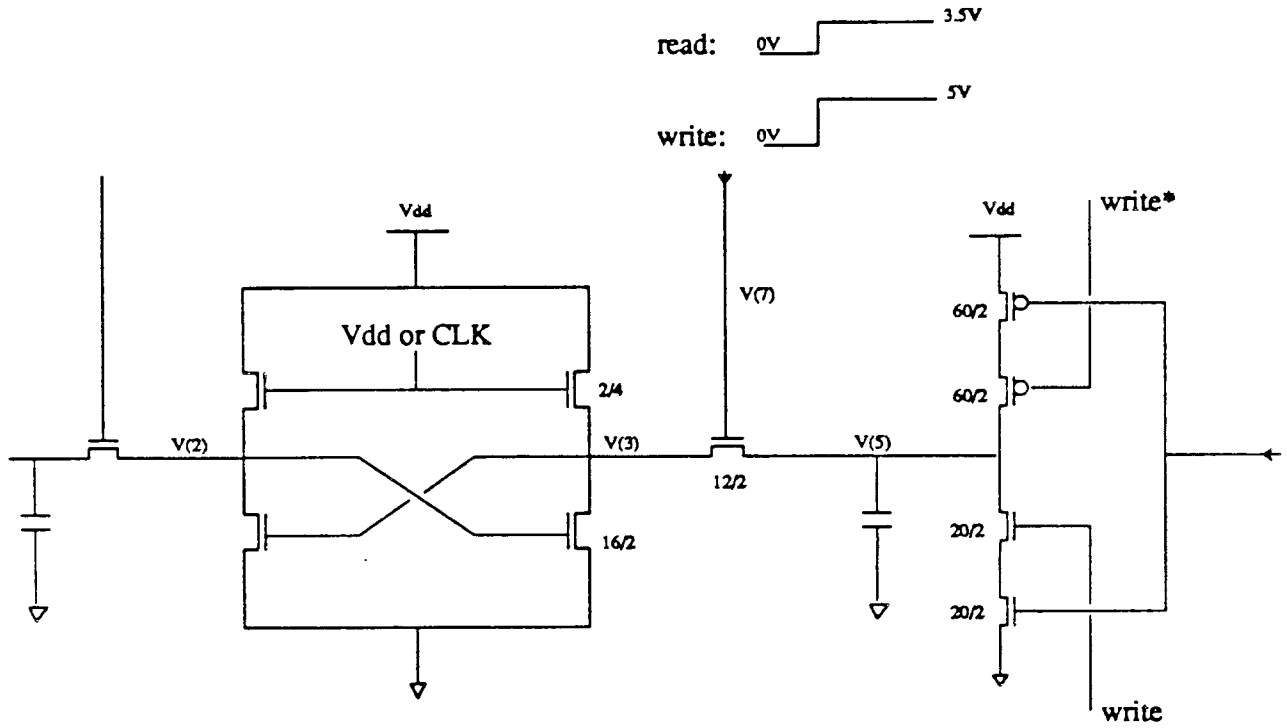


Figure 4-4a. NMOS 6T dual-port R/W memory cell

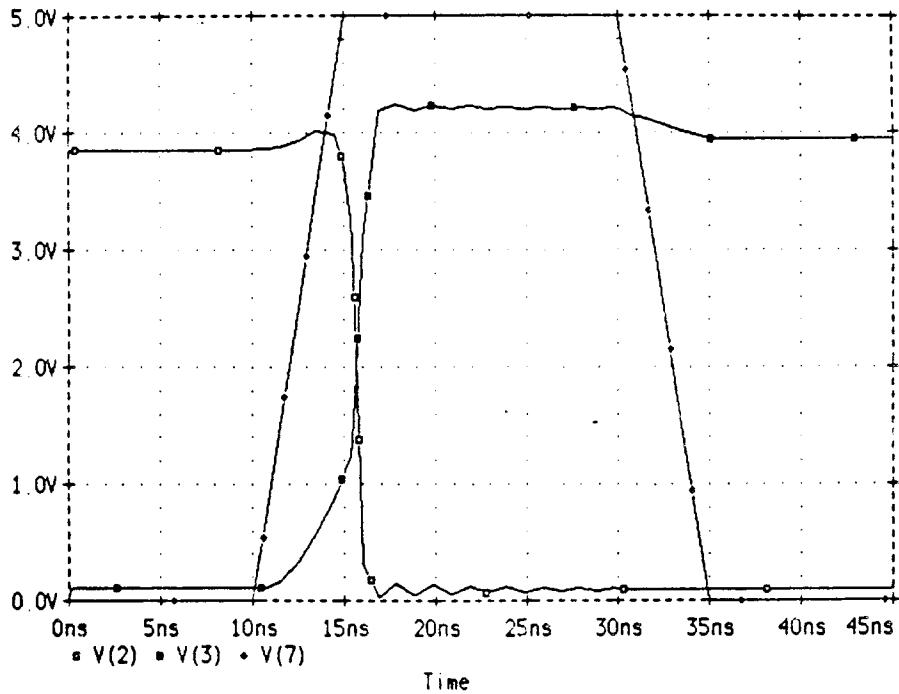


Figure 4-4b. SPICE simulation on NMOS 6T dual-port R/W memory cell - write

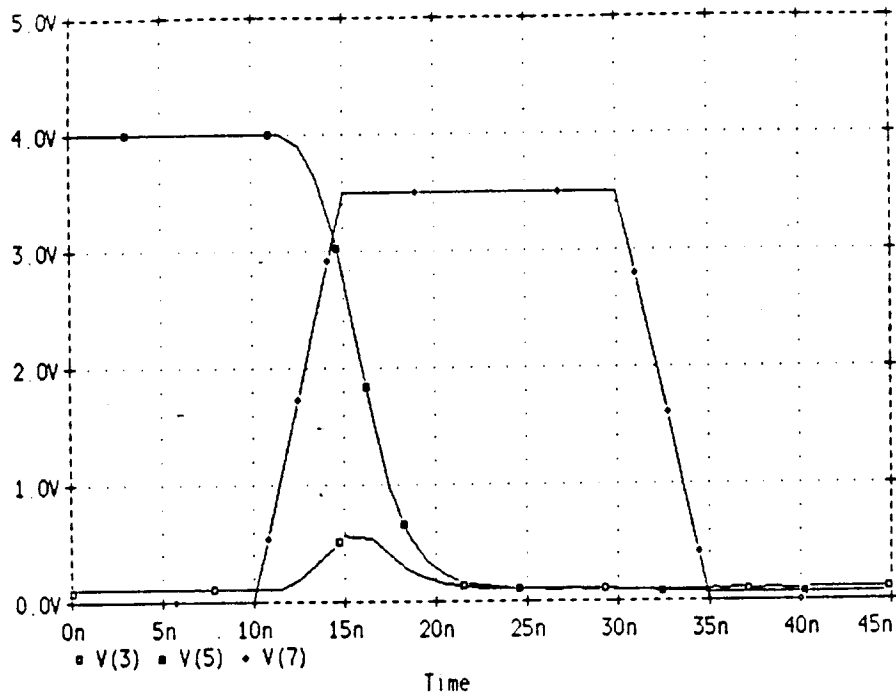


Figure 4-4c. SPICE simulation on NMOS 6T dual-port R/W memory cell - read

having all NMOS transistor cell can be used for this extra line. The gates must be pulsed during the write cycle and periodically to refresh the state of a cell. Without refreshing, the state of a cell can be altered by the leakage current out of storage node.

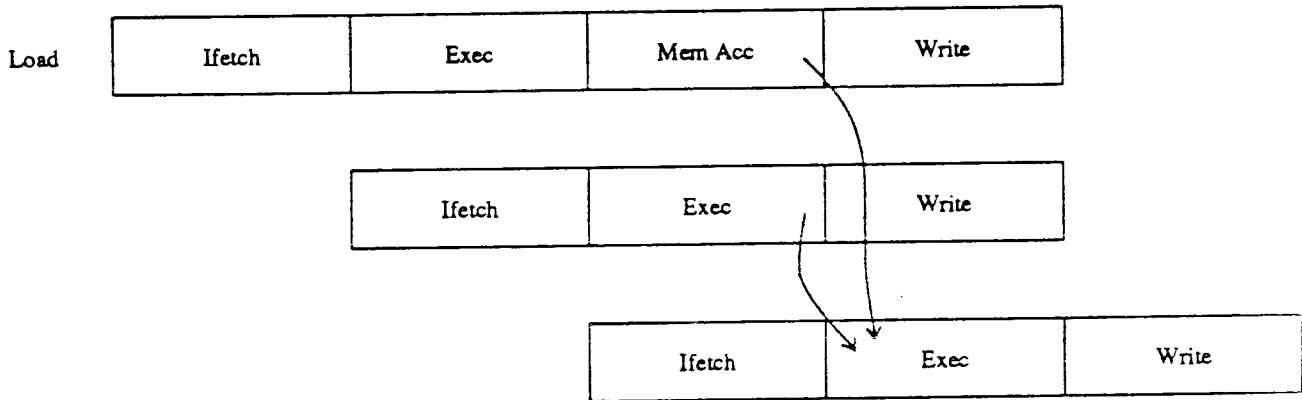


Figure 4-5. 3-stage pipeline with double writable register file

If the register file is implemented using this dual port R/W memory cell, the pipeline of R-R instruction need not be stretched to the 4-stage to avoid register write conflict, resulting in a 3-stage pipeline except for those instructions that reference the memory. However, double internal

forwarding will still be needed for the special sequence of instructions. That is, register references of instruction 2 cycles after the LOAD may require a double internal forwardings as shown in Figure 4-5. On the other hand, other instruction sequences require only single internal forwarding.

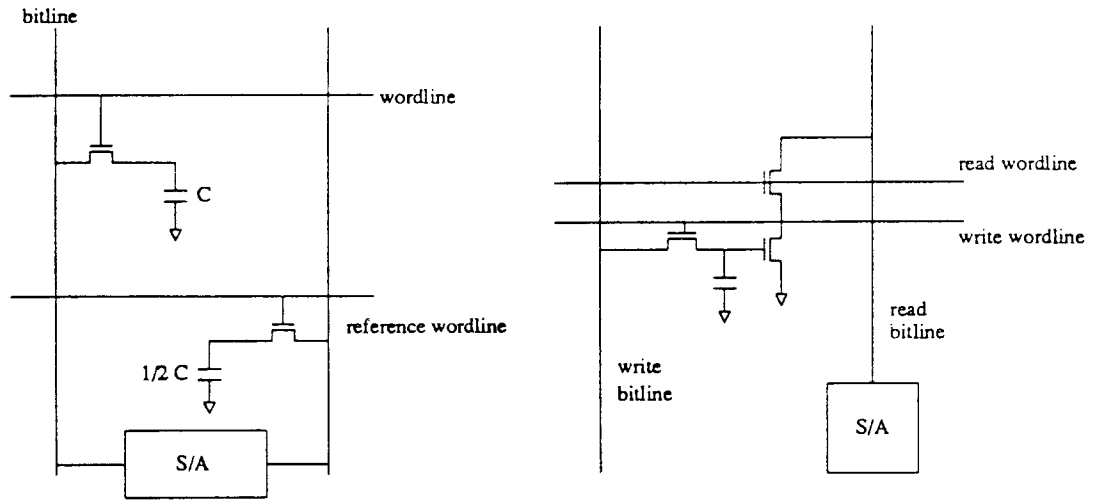
4.2 An On-chip Dynamic Memory

The motivation for having on-chip dynamic memory comes from the need for sizable amounts of memory for program and data storage. The effective storage access time can be reduced and high throughput is achieved. Several dynamic memory designs that can tolerate the variations of process technology have been investigated and can be found in [Tham85], [Chil85], and [Tran84]. The purpose of this subsection is to briefly review the limiting factors in designing an on-chip DRAM, and then to discuss what we would have done if we chose to implement DRAM on the SPUR CPU chip.

The processes used to fabricate high speed and high density DRAM's usually offer special features such as double layers of polysilicon, and extra implants. First of all, the absence of these special features in standard processes for the microprocessor chips makes it hard to implement on-chip dynamic memory. The area of a dynamic memory cell using a standard process would be two or more times larger than it would be using a state of the art DRAM process.

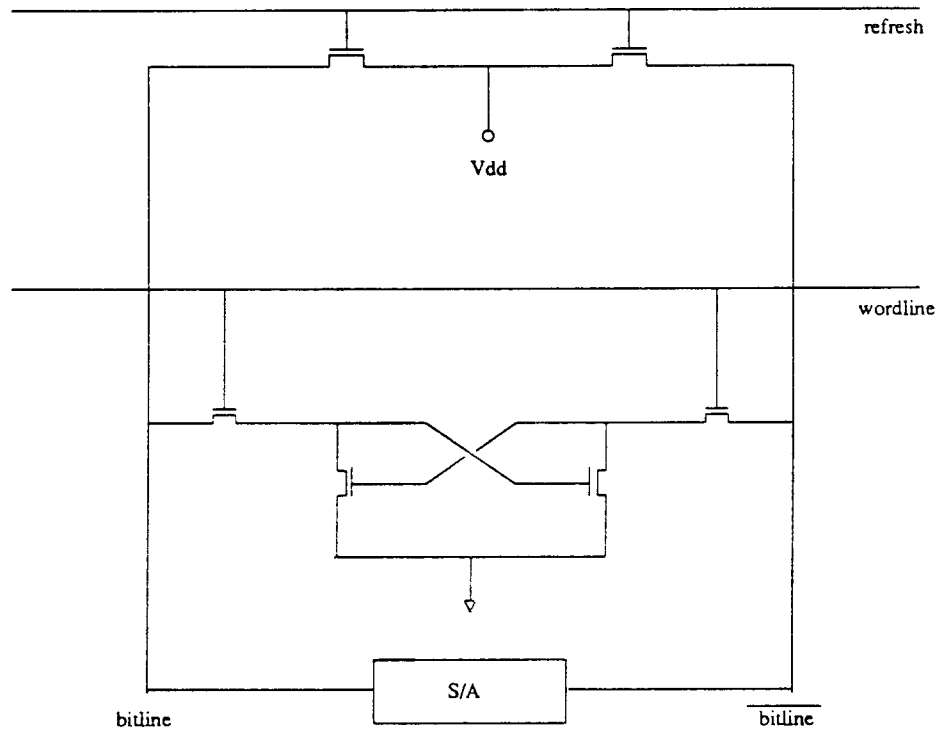
Junction leakage, subthreshold conduction, and soft errors induced by alpha particles can be other major concerns regarding the standard process. To be tolerable to the variations of these, a certain minimum storage capacitance is necessary. However, this will in turn result in larger cell area and hence lead to the longer access time as well as longer refreshing time.

In addition to the above problems, complexity of the circuit controlling the operation of DRAM's have kept us away from implementing on-chip dynamic memory. Process independent, self-timed control circuits for DRAM were investigated and presented in [Tham85]. Even with such circuits, any dynamic memory should be refreshed periodically, and this may disturb the processor's execution stream. If it is time to refresh all dynamic memories on chip, the control of the processor must stop and wait until all refreshings are done, degrading the processor's performance. In typical case this is necessary about every 2-3msec.



1T dynamic memory cell

3T dynamic memory cell



4T dynamic memory cell

Figure 4-6. Dynamic RAM cells

Typical dynamic cells using 1T, 3T, and 4T are shown in Figure 4-6. All these cells can be laid out in much smaller area than a 6T SRAM cell. The advantage of 3T or 4T dynamic memory cells over 1T cell is that they are less sensitive to process variations and faster in access time. While the 1T cell requires a certain minimum storage capacitance due to the undesired effects mentioned above, more area is needed for the extra transistors in 3T or 4T cells. However, the presence of a multi-layer metal process makes it easy to interconnect these extra transistors. Therefore, the 3T or 4T dynamic cells are more feasible than 1T cell, to be used for an on-chip dynamic memory with standard microprocessor process available today. As technology advances, however, the 1T cell will become more attractive due to its high density.

Process dependencies of dynamic memory cells may be overcome by using 3T or 4T cells. However, a small (about 1% of cycle time) refreshing overhead still exists. The on-chip dynamic memory is needed for an on-chip cache for instruction or data storage. If we add extra features to this cache, we can effectively eliminate the overhead associated with refreshing of dynamic memory used for the caches. Figure 4-7 shows one possible cache implementation with some extra circuits. The operation of this cache is as follows:

- (1) The refresh timer generates a clock with frequency determined by the refreshing interval, e.g. typically 2-4msec, or to be safe for the worst case, a half of the required refreshing time. The clock high time can be very short and can be coincided with time that processor does not use cache, not to disturb the execution stream of processor.
- (2) Initially, this clock sets node N1 of each word.
- (3) Whenever a word is accessed, the node N1 of that word is cleared. The signal on the node N1 along with word valid bit is then fed into the Domino circuit, which is evaluated once in every refreshing interval.
- (4) When refresh clock goes high, the domino circuit is evaluated. If any word has not been accessed and not been invalidated for the last period, the node N2 (Invalidate*) will remain high indicating all valid words have been accessed (refreshed) and hence no need for the refreshing, and the processor's execution stream can go on without any disruption.

- (5) If the Invalidate* is discharged to GND, it indicates that there are words that need to be refreshed. In this case, we can invalidate the whole cache or interrupt the processor's execution and refresh all words in a cache.

Study is needed to decide when to invalidate the whole cache rather than refresh it. The size and organization of the cache may be predominant factors to assess the tradeoffs involved here. Since such a study is out of the scope of this report, it is not pursued. The circuits required for this scheme are very simple, fast, and easy to implement.

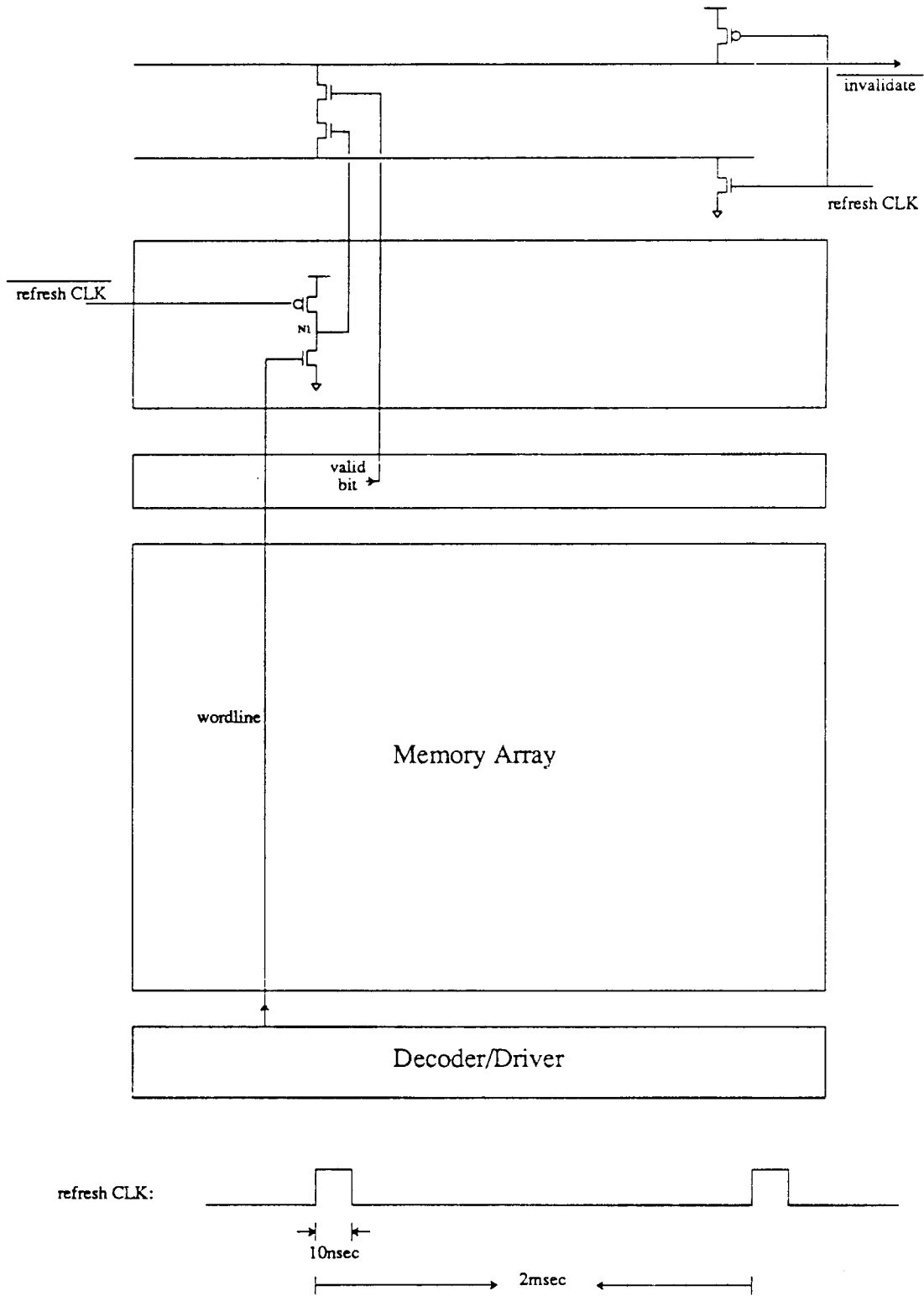


Figure 4-7. On-chip cache implemented with dynamic memory cells

5. CONCLUSION AND SUMMARY

There are many ways to implement a data path for high performance VLSI processors. In this report, implementation issues for a particular data path have been presented. It has focused on the data path of the SPUR CPU's execution unit. As in the RISC ancestors at U.C. Berkeley, the SPUR CPU includes a large register file and minimal functional units on the chip. As process technology has advanced, however, more silicon area has become available on the processor chip. An on-chip instruction cache is added on the SPUR CPU chip. The data path has been split into two parts. The upper data path, for the instruction address calculation and the special register reference, is implemented around the instruction cache. The lower data path, for the general computation on the 40-bit tagged registers, is implemented around the register file.

In section 2, a brief architectural overview was given, to help illustrate the implementation details presented in the following sections. In section 3, the implementation of the lower data path has been described in detail. It included the register file, the internal forwarding logic, and the functional units. Testing results showed that the clock cycle time can be reduced to 100 nsec from the initial estimate of 140 nsec. The critical delay of the register file operation is below 20nsec, since this delay defines the width of the phases in the cycle.

Design alternatives have been closely examined in section 4. First, dual-port memory cells were considered. Multi-port memory can have a great impact on processor performance since it provides a higher bandwidth between the register file and the functional units. Moreover, multi-port memory can affect the pipeline organization of the processor. Secondly, on-chip implementation of dynamic memory was examined. As integrated circuit technology advances further, a process-independent, high-density, fast on-chip dynamic memories will become more feasible. A possible implementation of dynamic memory as an on-chip cache was also presented.

There are many design tradeoffs to be considered in the design of data paths for a high performance microprocessors. The tradeoffs made during the design process, however, must be re-evaluated when the final processor is used in the completed computer system.

ACKNOWLEDGEMENTS

First of all, I would like to thank my research advisers, Professors David Hodges and Randy Katz, for directing this research work and their invaluable comments regarding this report.

I would also like to acknowledge all the people on the SPUR project who made it an exciting project to work on. In particular, I would like to thank the CPU design team: Shing Kong, Wook Koh, and Rich Duncombe. Thanks also go to Ken Lutz for setting up the test bench for the SPUR CPU testchip.

REFERENCES

- [Beyers81] J.Beyers, L.Dohse, J.Fucetola, R.Kochis, C.Lob, G.Taylor and E.Zeller, *A 32-Bit VLSI CPU Chip*, IEEE Journal of Solid-State Circuits, Vol. sc-16, no.5 pp.537-541 October 1981.
- [Chil85] Brian Childers: *On-Chip Memory For Microprocessors*, MS Report, EECS, U.C.Berkeley, Jan. 1985.
- [DKJeong86] D.K. Jeong, Gaetano Borriello, David A. Hodges and Randy H.Katz, *Design of PLL-Based Clock Generation Circuits*, Submitted to IEEE Journal of Solid-State Circuits, 1986.
- [Hill85] M.D. Hill et. al.: *SPUR: A VLSI Multiprocessor Workstation*, Report No. UCB/CSD 86/273, Computer Science Division (EECS), U.C.Berkeley, December 1985.
- [Joan85] Joan M. Pendlton, *A Design Methodology For VLSI Processor*, Ph.D. Dissertation, Memorandum No. UCB/ERL M85/88, U.C.Berkeley, November 1985.
- [Kate83] M.G.H. Katevenis, *Reduced Instruction Set Computer Architectures for VLSI*, Ph.D. Dissertation, Report No. UCB/CSD 83/141, U.C.Berkeley, October 1983.
- [Kong85] S. Kong, *Some Design Techniques for High Performance MOS Circuits*, February 1985, M.S. Report, EECS, U.C.Berkeley.
- [Kong86] S. Kong, R. Duncombe, D. Lee, W. Koh: *The SPUR CPU: An Architectural Description*, Version 2.0, Computer Science Division (EECS), U.C.Berkeley, June 1986.
- [MeCo80] C.A.Mead and L.A.Conway, *Introduction to VLSI Systems*, Addison Wesley Publishing Co. 1980.
- [Mohsen79] A.M.Mohsen and C.A.Mead, *Delay-Time Optimization for driving and Sensing of Signals on High-Capacitance Paths of VLSI Systems*, IEEE Journal of Solid-State Circuits, Vol. sc-14, pp.462-470, April 1979.
- [Nora83] Nelson F. Goncalves and Hugo J. DeMan, *NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures*, IEEE Journal of Solid-State Circuits, Vol. sc-18 No.3, June 1983, pp.261-266.
- [Sher84] Robert W. Sherburn, *Processor Design Tradeoffs in VLSI*, Ph.D. Dissertation, Report No. UCB/CSD 84/173, U.C.Berkeley, April 1984.
- [Taylor85] G.Taylor, *SPUR Instruction Set Architecture*, Computer Science Division (EECS), U.C.Berkeley, May 1985.
- [Tham85] K.S. Tham: *A Self-Timed One Transistor Dynamic Ram*, MS Report, EECS, U.C.Berkeley, Jan. 1985.
- [Tran84] L.V.Tran, *A 32K Bit High Speed On-chip DRAM For Digital Signal Processor*, Proceedings of IEEE Custom Integrated Circuits Conference, pp.166-169, June 1985.

APPENDIX

Overlapped with parent		Overlapped with child	
OWP <5>	11111	OWC <5>	01111
OWP <4>	11110	OWC <4>	01110
OWP <3>	11101	OWC <3>	01101
OWP <2>	11011	OWC <2>	01011
OWP <1>	11010	OWC <1>	01010
Local Registers		Global Registers	
Local <9>	11001	Global <9>	01001
Local <8>	11000	Global <8>	01000
Local <7>	10111	Global <7>	00111
Local <6>	10110	Global <6>	00110
Local <5>	10101	Global <5>	00101
Local <4>	10100	Global <4>	00100
Local <3>	10011	Global <3>	00011
Local <2>	10010	Global <2>	00010
Local <1>	10001	Global <1>	00001
Local <0>	10000	Global <0>	00000

Appendix 1. 5-bit Register Addresses

```
*****  
*           TYPICAL Device parameters for the HP CMOS40 Process           *  
*                                                                 *  
*           NOTE:  These parameters are intended for digital design only. *  
*                                                                 *  
*****
```

***** Use N and P models for W >= 4U and L <= 2U *****

```
.MODEL NMOS NMOS LEVEL=2 VTO= 0.75 KP=76.0U GAMMA=.40 LAMBDA=.025 TOX=25N  
+ NSUB=4E16 TPG=+1 XJ=.25U LD=.20U UEXP=.16 VMAX=5.5E4 JS=1000U  
+ CGSO=220P CGDO=220P CJ=230U CJSW=260P CGBO=400P
```

```
.MODEL PMOS PMOS LEVEL=2 VTO=-0.75 KP=27.0U GAMMA=.50 LAMBDA=.045 TOX=25N  
+ NSUB=2.0E16 TPG=-1 XJ=.20U LD=.05U UEXP=.15 VMAX=9.0E4 JS=1000U  
+ CGSO=220P CGDO=220P CJ=670U CJSW=215P CGBO=400P
```

```
*****  
*           TYPICAL Device parameters for the XEROX 2um CMOS Process           *  
*                                                                 *  
*           NOTE:  These parameters are intended for digital design only. *  
*                                                                 *  
*****
```

```
.MODEL PMOS PMOS LEVEL=2 VTO=-0.70 KP=46.0U GAMMA=.49 LAMBDA=.04 TOX=30N  
+ NSUB=1.0E16 TPG=-1 XJ=.5U LD=.375U UEXP=.44 VMAX=9.0E4  
* TYPICAL CAPS  
+ CGSO=4.4E-10 CGDO=4.4E-10 CJ=3.1E-4 CJSW=3.0E-10 CGBO=1.4E-10  
* MAX CAPS  
CGSO=5.1E-10 CGDO=5.1E-10 CJ=3.4E-4 CJSW=3.8E-10 CGBO=1.5E-10
```

```
.MODEL NMOS NMOS LEVEL=2 VTO= 0.70 KP=54.0U GAMMA=.44 LAMBDA=.04 TOX=30N  
+ NSUB=1.0E16 TPG=+1 XJ=.3U LD=.225U UEXP=.12 VMAX=5.2E4  
* TYPICAL CAPS  
+ CGSO=2.7E-10 CGDO=2.7E-10 CJ=1.5E-4 CJSW=2.5E-10 CGBO=1.4E-10  
* MAX CAPS  
CGSO=3.1E-10 CGDO=3.1E-10 CJ=3.0E-4 CJSW=1.9E-10 CGBO=1.5E-10
```

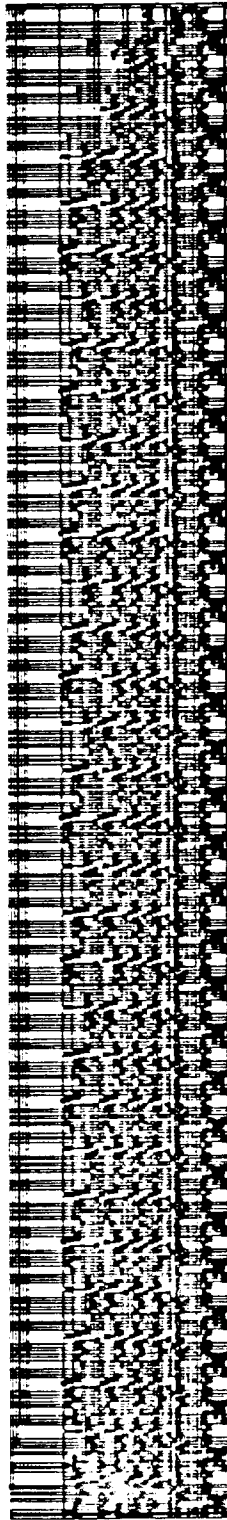


Figure A-1: Shifter Lyout Plot

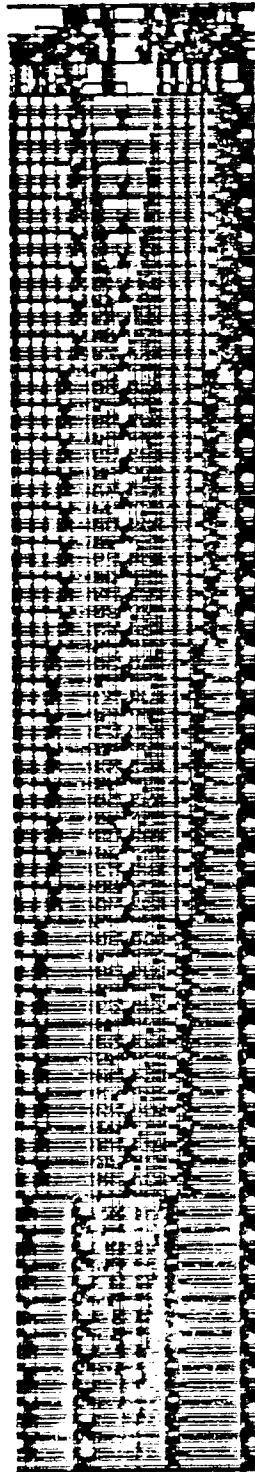


Figure A-2:Byte Extractor/Inserter Layout Plot

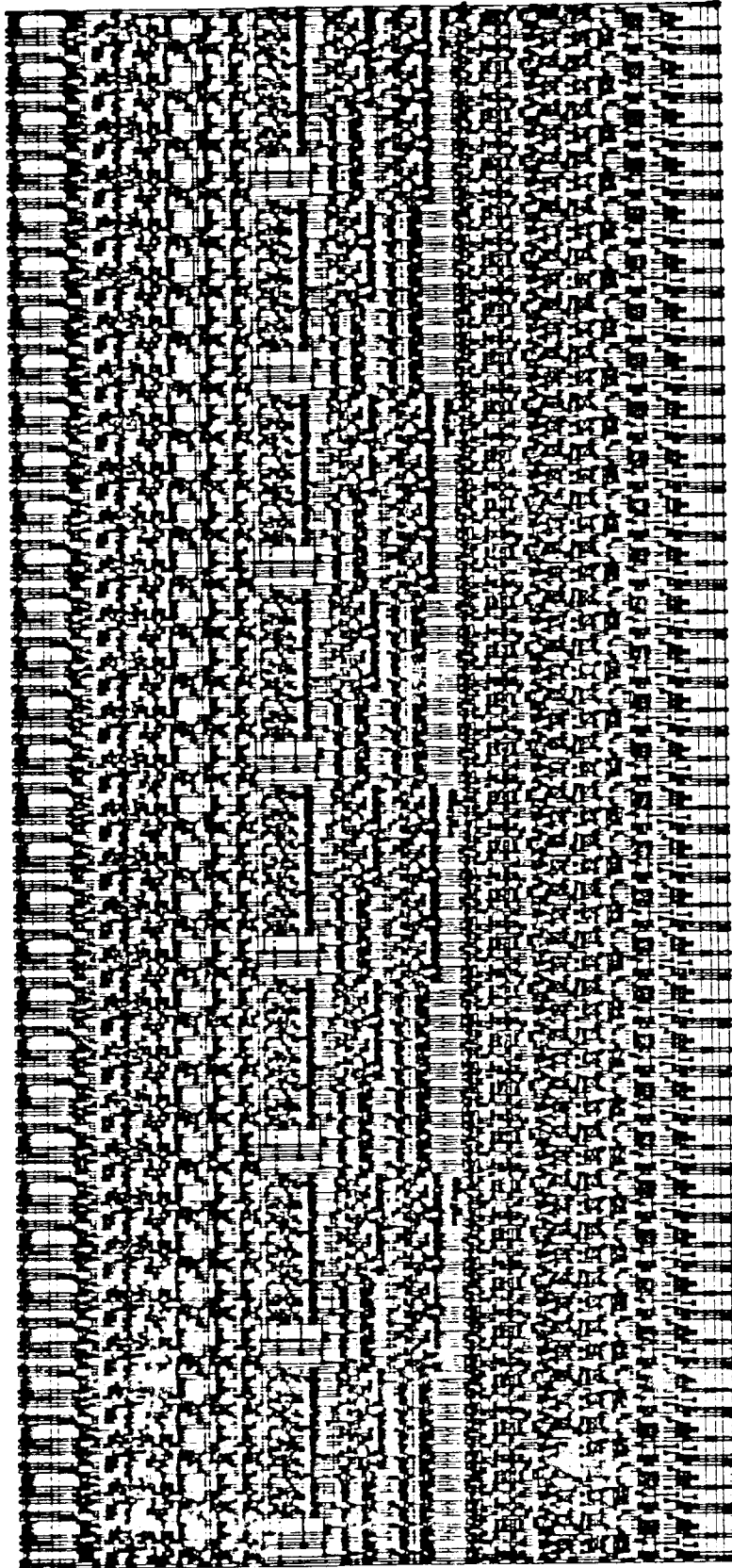


Figure A-3:ALU Layout Plot