SMALLTALK ON A RISC-CMOS IMPLEMENTATION

by

C. C. Marino

Memorandum No. UCB/ERL M85/48

6 June 1985

(cover)

SMALLTALK ON A RISC-CMOS IMPLEMENTATION

by

C. C. Marino

Memorandum No. UCB/ERL M85/48

6 June 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Smalltalk on a RISC - CMOS Implementation

Christopher C. Marino

May 14, 1985

Smalltalk on a RISC - CMOS Implementation

Abstract

A new 32 bit CMOS Reduced Instruction Set Computer (RISC) has been designed to execute the Smalltalk-80 programming language efficiently. This processor, Smalltalk On A RISC (SOAR) was designed using a 3 micron CMOS technology and is being fabricated. The CMOS implementation is pin for pin compatible to the NMOS version of SOAR. This version was designed using the Hawk-Squid system and took advantage of some of the advanced IC design features that are available through Hawk. This represents the first Berkeley RISC processor that uses CMOS and can show differences in compatible circuits using different technologies.

Acknowledgements

I must thank the following people for their support and encouragement over the past year. I would like to thank Richard Newton and David Patterson for giving me the opportunity to work on this project. Their suggestions and contribution to my work and the resources they provided are greatly appreciated. I must also thank Joan Pendelton who helped me tremendously in understanding SOAR and tolerated my persistence and what seemed like perpetual ignorance. I thank Mark Hofmann who taught me all about the Berkeley design tools and made me feel like I was a part of the group from the very start. Peter Moore for helping me with some tricky system bugs. even though he once described me as the fool with more guts than brains for taking on this project. The work of James Reed is also greatly appreciated. Most of all I thank Deirdre Ryan for keeping Hawk together and giving me the encouragement that I needed to finish SOAR. Her support made the months of hard work much less painful. This work was supported in part by DARPA under grant N00039-83-C-0107. Their support is gratefully acknowledged.

Table Of Contents

CHAPTER 1

1. Introduction

In 1980 Professor David Patterson at the University of California started the Berkeley Reduced Instruction Set Computer (RISC) projects [Patt81] [Fitz81]. The philosophy behind the RISC is to design a simple, fast microprocessor that has high performance characteristics. The performance of such a microprocessor is not gained through sophisticated circuit design nor through exotic processing techniques, but rather by designing the architecture of the processor so that only the essentials remain. A fast Arithmetic Logic Unit (ALU), program counter, registers, barrel shifter, and control logic is really all RISC I was. The high performance of RISC I was gained through reducing the clock cycle to the absolute minimum. A pipelined architecture and simple techniques to keep the pipeline full were used to attain maximum utilization of the chip circuitry. In 1980 when the RISC project began the techniques that RISC I used were already throughly understood and commonplace. How could simple components be integrated together on a single chip and attain an overall improvement in performance over complex computer architectures? If anything, the RISC projects represent a step backwards in computer architecture when compared to the VAX, Motorola 68000 and Intel iAPX-432. To understand fully the RISC philosophy it is necessary to present some historical observations and the progression of computers, their instruction sets, and architecture.

For many years there was a tendency to increase the size and complexity of a computer's instruction set. The motivation behind this was that richer instruction sets

---

VAX is a trademark of the Digital Equipment Corp.
68000 is a trademark of Motorola Corp.
iAPX-432 is a trademark of Intel Corp.

would simplify software and compilers and computers that were easier to use were thought to be better. Also, memory technology advanced so fast that additional microcode added almost nothing to the cost of a machine. Microinstructions run much faster than macroinstructions so every operation that could be put in microcode would result in a faster operation. Register-based load/store architectures were difficult to program, many addressing modes would further simplify the programs. These were all valid arguments when the evidence was examined that compared the two types of architectures. Richer instruction sets did yield better performance characteristics. But were these comparisons valid? As with all comparisons, they are only as valid as the assumptions they are based on. Using one comparison technique the memory to memory architecture is superior, using another the load/store architecture is better [Patt84].

With the validity of the architectural metrics questionable other things were examined as the basis for the comparison of architectures. The overall performance of a load/store architecture would be superior to that of a memory to memory architecture provided the registers could remain local to the processor. Most instructions in all computers are simple ones and are burdened by the overhead of complex microcode decoding. Since most instructions are simple, getting these simple instructions to run the fastest would increase the overall speed. Any instruction that increased the cycle time must provide a corresponding decrease in the number of instructions executed to be a valid performance enhancement.

An example of this is the implementation of the MicroVAX. One version contained the entire VAX architecture and instruction set, another implements only a subset. The full VAX implementation outperforms the MicroVAX by 20% but uses 5 to

---

CDC-6600 is a trademark of Control Data Corp.
MicroVAX is a trademark of Digital Equipment Corp.

10 times the resources [Patt84]. In the full implementation of the VAX architecture there are nine custom VLSI chips, in the MicroVAX there are only two. The performance gained through the complex instruction set was lost in chip to chip data transfer delays. The performance gains of VLSI are maximized when everything can remain on a single chip and not have to wait for chip to chip communications.

Complex addressing modes only lengthen the execution of simple instructions. Simple techniques of pipelining, using a delayed branch and simple hardware techniques to keep the pipeline full can increase performance tremendously. Evidence of this is the fact that the CDC-6600, one of the fastest computers in its time was a register-based load/store architecture using pipelined execution in its function units.

These observations, and other similar observations, were the basis for the RISC processors. SOAR is the third project in the family of RISC processors developed at UC Berkeley. RISC I and RISC II served to show the validity of the RISC philosophy. SOAR takes this philosophy one step further and attempts to map an experimental programming environment on a RISC machine. Smalltalk is an object-oriented language that is heavily burdened with complex overhead operations [Patt83]. SOAR expands the concept of the RISC to determine if simple hardware speed-ups can serve to increase the performance of a Smalltalk system [Patt83b]. SOAR differs from a traditional RISC in that it's control is a substantial part of the circuit and there are multiple-cycle instructions. However, these instructions pay for themselves in the increase in performance they provide. SOAR uses a tagged architecture to aid in the execution of Smalltalk and requires more control circuitry than either RISC I or RISC II. The tagged architecture is essential for efficient Smalltalk execution and the added control circuitry is necessary for tag support. Both CMOS and NMOS implementations of SOAR proceeded in parallel. The NMOS version was completed first and, where appropriate, comparisons are made between the two implementations.

In this report the CMOS implementation of the SOAR microprocessor is described. A brief architectural overview of SOAR is presented and then the approach taken in the design is described. The processor was designed using the Hawk-Squid system and a brief description of that system is also included. The simulation and performance results for CMOS SOAR, including critical path, and overall speed estimation are also included.

CHAPTER 2

2. Architectural Overview

The SOAR processor was designed to execute the Smalltalk language efficiently. Smalltalk was developed by Xerox in the late 1970's and is a highly productive software programming environment. It is implemented as a window-based system and allows for the the use of a pointing device. Smalltalk is an object-orientated language and programs are characterized by many calls **(sends)** and returns which result in a great deal of overhead. The goal of the SOAR project is to have the SOAR processor run Smalltalk as fast than an ECL mini-computer such as the Xerox Dorado [Deu83].

A problem with object-oriented languages is that objects are being created constantly. Often, these objects are used only once or twice and then become inactive. Objects are checked to see if they are still active and inaccessible objects are discarded. This memory reclamation process is called **generation scavenging.** This is a very time consuming process and contributes to the medocre performance of Smalltalk on conventional microprocessors. SOAR uses a tagged architecture to keep track of the age of an object which makes generation scavenging a much more efficient process. Observations of Smalltalk programs and the lifetime of objects indicate that, in general, new objects do not live long and old ones live forever. For example, the **square root** object will always be needed but the object that was just created to perform a local trivial task won't be used more than a few times [Unga83b]. The improved generation scavenging technique and efficient call and return mechanism allow SOAR to achieve its high performance.

Dorado is a trademark of Xerox Corp.

2.1. SOAR Architecture

Smalltalk-80 [DeS84] [Deu83] is an object-oriented software system which references objects through pointers called Object Oriented Pointers (OOP's). These OOP's point to a variety of objects and SOAR uses a tagged architecture to identify a few important cases. Both the pointer and its tag are contained in the same 32-bit data word which simplifies the hardware that distinguishes the different kinds of objects. The tag not only identifies the object but also contains a field that represents the age of an object.

If a program attempts to access an inappropriate object type an exception condition occurs and the condition is detected in hardware by examining the pointer's tag bits. If such a tag mismatch occurs SOAR traps and jumps to a software routine to service the exception. Special hardware in SOAR supports these traps and improves the performance of Smalltalk on SOAR. Smalltalk programs generally consist of deeply nested calls sends. They are also characterized by needing few local registers and few arguments. For this reason, the register file of RISC II was retained but the number of new registers allocated to a new window was reduced from sixteen to eight. Studies show that this eight window mechanism can cover 94% of all sends and 8 registers per window can cover 97% of all sends [Blak83]. The register window, global registers, and special registers are labeled 0-31 and are shown in Table 2.1.

SOAR uses a technique of in-line caching that optimizes the search operations that occurs on all sends. Measurements show that this optimization works 95% of the time [DAmb83]. This optimization technique is worthwhile because Smalltalk objects have many classes; if the caller's class is not the same as the callee's class Smalltalk climbs up the chain of classes to search for a match. This is a very time-consuming process. SOAR assumes that the classes are the same and traps if a mismatch occurs.

REGISTER GROUP	REG NUM	CONTENTS
GLOBAL	r31	Scratch
	r30	Scratch
	r29	Scratch
	r28	Scratch
	r27	Scratch
	r26	Scratch
	r25	Scratch
	r24	Scratch
SPECIAL	r23	PSW-Program Status Word
	r22	CWP-Current Window Pointer
	r21	TB-Trap Base register
	r20	SWP-Saved Window Pointer
	r19	SHA-Shadow register A
	r18	SHB-Shadow register B
	r17	PC-Program Counter
	r16	RZERO-Always 0
HIGH	r15	return address for this method
	r14	receiver/return value
	r13	arg1/local
	r12	arg2/local
	r11	arg3/local
	r10	arg4/local
	r9	arg5/local
	r8	arg6/local
LOW	r7	return address for called methods
	r6	receiver/return value
	r5	arg1
	r4	arg2
	r3	arg3
	r2	arg4
	r1	arg5
	r0	arg6

Table 2.1 SOAR register window

This describes briefly what makes SOAR different from RISC. The tags and traps that are supported by SOAR make it well-suited to execute Smalltalk-80 programs. SOAR can operate in a non-tagged mode so that it can also support the "C" and Pascal languages. In addition, SOAR has the following unique architectural features that are described before the hardware implementation of SOAR is presented.

SOAR has 32 bit address and data busses. These are non-multiplexed input and outputs on the chip. The actual addresses of SOAR's memory data are only 28 bits wide; bits 28 to 31 of the address bus are ignored by the external circuitry. The data words are 32 bits wide. SOAR operates on a three-stage pipeline the details of which are described in Section 2.4. SOAR is not a byte oriented processor and arbitrary shifts are not supported. This simplifies the hardware implementation by eliminating the need for a barrel shifter and byte aligner. Byte oriented operations are supported for compatibility with other software environments by including a byte inserter/extractor which can insert or extract a byte from the 32 bit data word. When SOAR detects a trap it will jump to an address specified by logically ORing the opcode of the offending instruction and the condition of the trap and the value contained in the Trap Base register (TB). The TB serves as the base that is offset by the trap logic in SOAR. This vectored trapping is a convenient method of handling many exceptional conditions. The hardware that implements the tags and traps of SOAR represent a large part of the circuitry in this processor. Some of these hardware features are described in more detail in the following sections.

2.2. Tags

SOAR tags are an essential part of each OOP and contain information about the object. The four tag bits are located in the 4 MSB's of each OOP. Small integers have only one tag bit (bit 31) to allow for the largest integer. In Smalltalk-80's virtual machine definition, all objects (except small integers) are referenced by pointers. The hardware distinguishes between the types of tags and handles each accordingly.

A small integer object is recognized by a zero in bit 31 of the OOP. All other tags fields are 4 bits wide and are shown in Table 2.2. The purpose of the tags is to keep track of the age of an object to support efficient generation scavenging. SOAR objects are divided into four 'tenure' groups that reflect the number of memory scavenging

OBJECT POINTED TO	32-BIT POINTER (OOP)	
	TAG BITS	WORD BITS
Small Integer	0	SmallInt
Assistant Object	1000	OOP
Associate Object	1001	OOP
Full Object	1010	OOP
Emeritus Object	1011	OOP
Context Object	1111	OOP

Table 2.2 SOAR Tag Fields

operations that an object has survived. The other main use of the tags is to distinguish a context object OOP from other OOP's. These objects in Smalltalk-80 do not follow a LIFO stack convention. These objects must be identified specifically so that they are not removed in LIFO order. The context tag "1111" generates a trap whenever a context object OOP is stored in memory so that it can be marked as non-LIFO [Samp84].

2.3. Traps

SOAR's Trap Base register gets concatenated with both the instruction that caused the trap and the vector for that trap condition. This provides a very flexible trap and interrupt mechanism that replaces the typical execute-on-condition instructions and allows for more hardware support for Smalltalk. This mechanism allows very efficient data structure updates and memory reclamation (generation scavenging).

The trap conditions are listed in Table 2.3. The vector field is specified by the type of trap. The trap base address is user specified by loading the Trap Base register

Name	Vector	Pri Class	Explanation
ILL	0	A	illegal opcode(i<31>=1 | i<31>=0&i<28:23>=unused)
TT	1	B	tag trap: illegal tags or overflow
SWI	2	B	software interrupt
WO	3	C	window overflow (calls)
WU	4	C	window underflow (on returns)
DPF	5	C	data page fault
TI	6	C	trap instruction
GS	7	D	GS trap(store new into old, store context, nonLIFO ret, context ret)
IPF	8	E	instruction page fault
I/O	9	F	I/O request

Table 2.3 SOAR Trap Conditions

with an arbitrary address, the offending opcode makes up the lower six bits of the trap vector.

These trap instructions were designed to provide flexibility in user defined single cycle test instructions. Since the trap is taken only on the satisfied condition the program is only penalized one cycle for the test. This feature can be taken advantage of if the condition is checked frequently, has a short instruction sequence to compute the condition and has a global action to be performed when the condition is satisfied. A more complete description of SOAR's tags and traps is included in [Unga83] and [Samp84].

2.4. The SOAR Pipeline

There are hardware features of the architecture that aid in the execution of Smalltalk-80 programs. SOAR operates on a three-stage pipeline. Each instruction requires three cycles to execute: Instruction Fetch, Operate, and Register Write. Load

and store instructions are exceptions to this rule and require an extra cycle to read and write to main memory. This allows the pipeline to execute an instruction every cycle. Trap and return instructions take longer since they flush the pipeline. The instruction execution and SOAR operation is shown in Figure 2.1. Each cycle consists of three non-overlapping clock phases and each operation is shown in Table 2.4.

Cycle 1	PHI1	Instruction Fetch
	PHI2	Instruction Fetch
	PHI3	Instruction Fetch
Cycle 2	PHI1	Pre Charge
	PHI2	Register Read
	PHI3	ALU Operation
Cycle 3	PHI1	
	PHI2	
	PHI3	Register Write

Table 2.4 SOAR Clock Phases

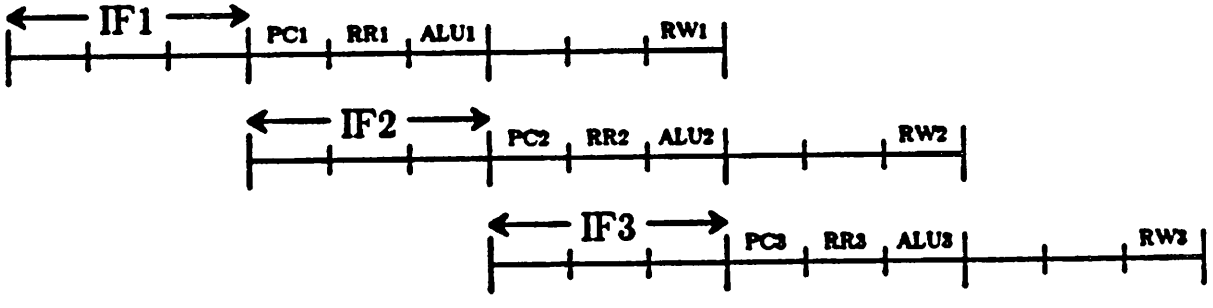


Figure 2.1 SOAR Pipeline Execution

SOAR's pipeline is three stages; if the source of any instruction is the destination of the previous instruction there is a problem. Examination of Figure 2.1 shows that the results of instruction IF1 are not written to the register file until Phase 3 of its third cycle of execution. This is long after the next instruction reads its own operands, during Phase 2 of the previous cycle. The second instruction must know that the value at the source is not yet valid. This data dependency is eliminated by the interlock and register-forwarding logic.

This special logic compares the source of the second and destination of the first instructions and, if equal, passes the destination value to both the register file and the input to the ALU. This keeps the pipeline full and eliminates the need for a special compiler to sequence the instructions to eliminate the data dependency. The special purpose compiler is the approach taken with the Stanford MIPS processor (Microprocessor without Interlocked Pipeline Stages). Note that this forwarding mechanism works with destinations that specify the register file and not the special registers.

SOAR makes further attempts to reduce the number of bubbles in the pipeline by including some hardware to control the **FastShuffle** line. On calls and jumps the address of the next logical instruction is not same as the next physical instruction in memory. When a call is executed (second stage of the pipeline), the next instruction has already been fetched. But this is not the next instruction that should be executed. Ordinarily the execution of this instruction would have to be suppressed and a bubble would be formed. SOAR, however, uses the FastShuffle circuitry to check the incoming instruction to see if it is a call or jump in the instruction fetch stage. If it is a jump it outputs the target address in the very next cycle without waiting for the execution stage. This eliminates the need for a delayed branch to keep the pipeline full. A similar problem exists in the execution of the return instruction except that it is complicated by the fact that the target address is not known until after the completion of the

second cycle of execution. Here the FastShuffle mechanism cannot prevent the bubble. The execution of the instruction in the first stage of the pipeline is suppressed by forcing a no-op into the pipeline.

2.5. Loads and Stores

No distinction is made between references to addresses on-chip or off-chip in the Saved Window Space. Therefore, hardware is provided so that the 28 bit address can be recognized as an on chip register. A typical load/store instruction execution is shown in Figure 2.2. This is the usual execution, this consists of the instruction fetch (Cycle 1), effective address calculation (Cycle 2), memory access (Cycle 3), and on-chip register write (Cycle 4).

If the effective address references an on-chip register this sequence is not as simple as described above. A check for on chip register reference must be made on each load/store execution without slowing down the overall cycle time. This is done by routing six bits from the ALU output (containing the window number of the effective address) to the register decoding logic. If the reference is on chip (as determined by a comparison of the SWP and effective address) the effective address is used to select the on chip data and the **RD_WR*** line stays invalid for the duration of the memory access cycle.

2.6. Register Window Control

The overlapping register window scheme of RISC II is preserved in SOAR but there are some distinctions, as described in Section 3.8. SOAR has three hardware pointers: the Current Window Pointer (CWP), Current Window Pointer-1 (CWP-1), and the Saved Window Pointer (SWP). These three pointers aid in the register window management system. Register windows that are written to main memory are saved in an area of main memory pointed to by the SWP. The first window saved in this area is saved at the highest address location, other windows are added at lower memory

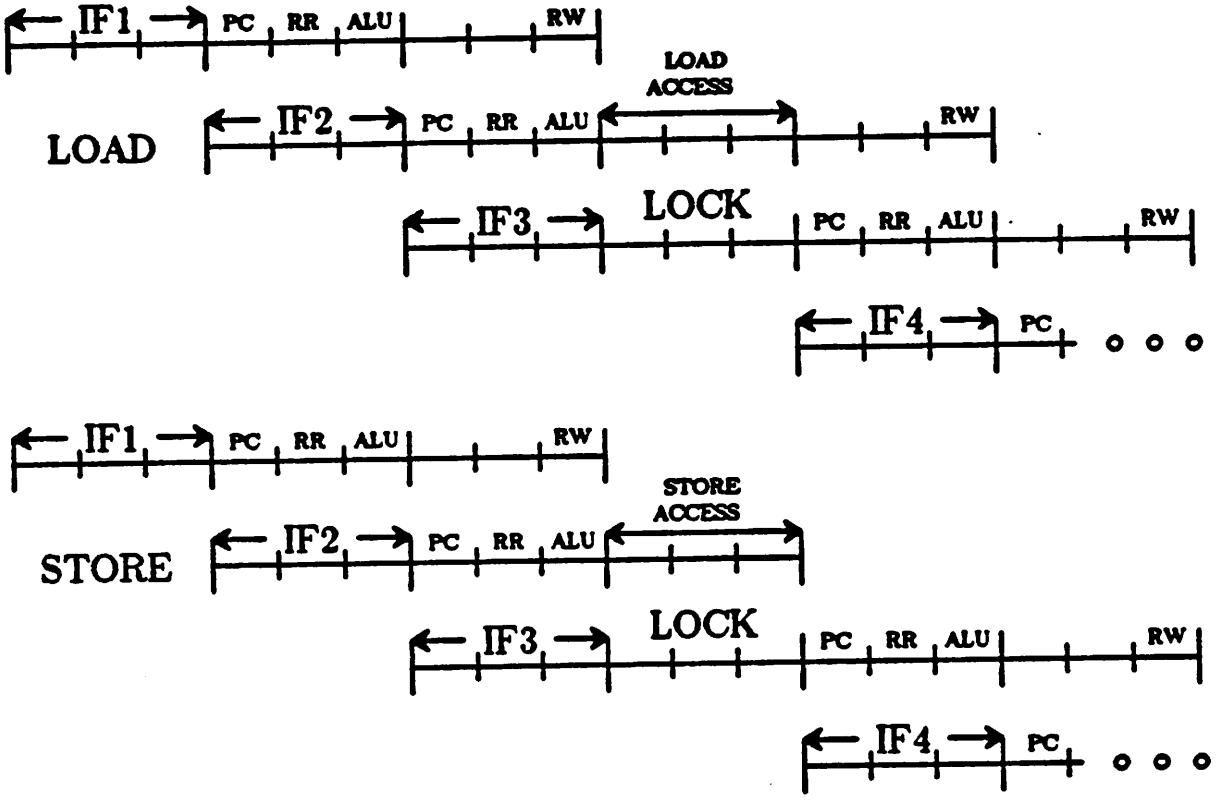


Figure 2.2 Load/Store Execution Timing

address locations. With each window overflow, the SWP gets updated and indicates the division between the saved register space in main memory and memory that is currently in the register file. Bits 6-4 of the SWP contain the window number of the next window to be transferred on-chip if a window underflow condition occurs.

The CWP is a pointer that points to the high registers in the window that is currently active in the register file. CWP-1 points to the low registers that are currently active.

A window overflow occurs when a call instruction is executed and CWP-SWP=1, a window underflow occurs when a return instruction is executed and SWP-CWP=1. The diagram of SOAR's saved register space is shown in Figure 2.3. Also, there is a more detailed description of the interaction of the CWP, SWP and the windows in Appendix A.

2.7. Shadow Registers

A part of SOAR's architecture that is essential to the trapping mechanism is the shadow registers. Two of these shadow registers are contained in the PSW special register (shown in Figure 2.4) the other two are special registers: r19 and r18. These registers, Shadow Opcode, Shadow Destination, Shadow A and Shadow B serve to save the necessary information so that an instruction, aborted during a trap, can complete after the trap has been serviced. These registers are loaded on each cycle provided interrupts are enabled. When a trap occurs interrupts are automatically disabled this captures the necessary information in the shadow registers.

This is a brief overview of the architecture of SOAR and is by no means complete. For further information see [Samp84] and [Unga83].

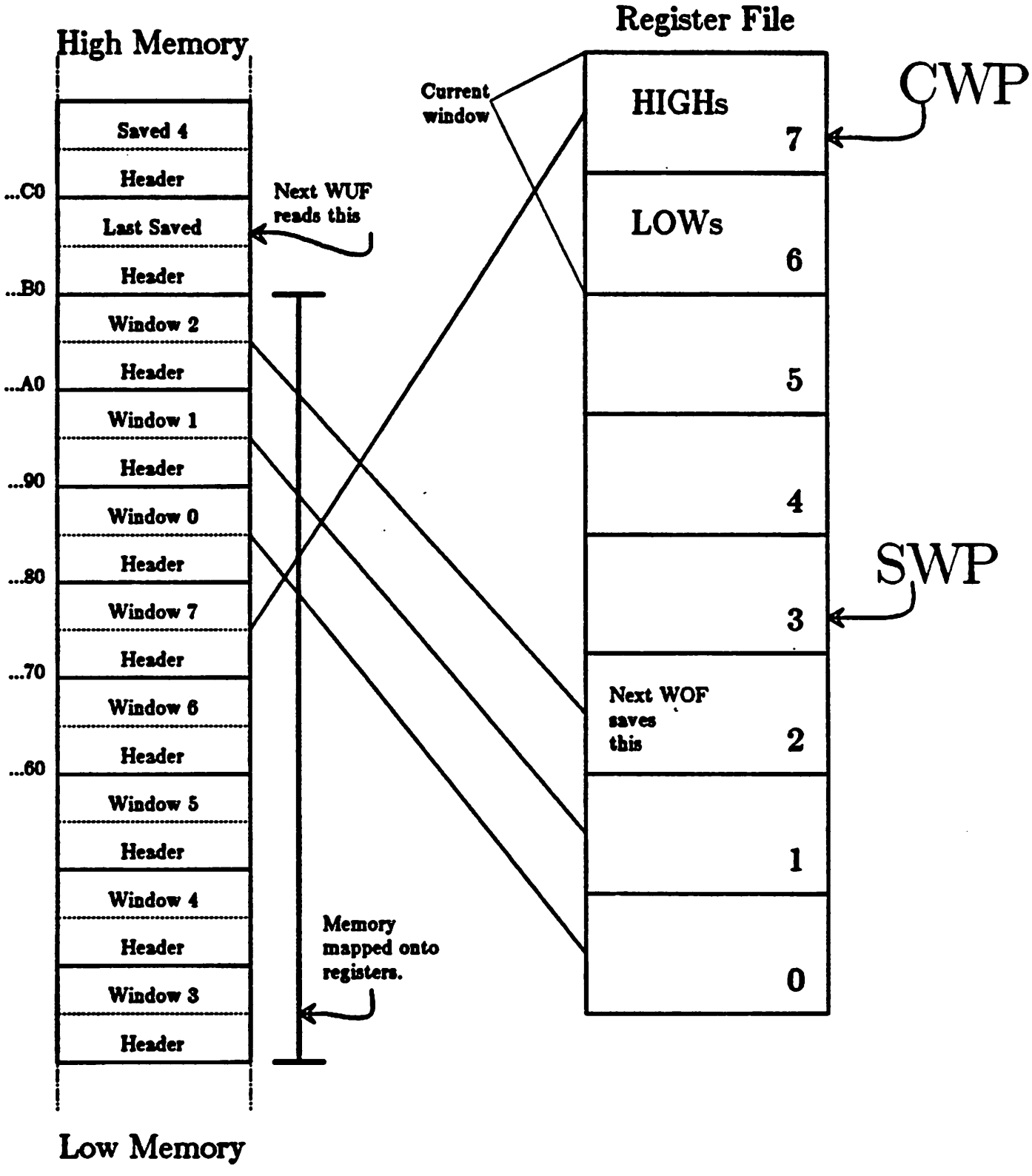


Figure 2.3 SOAR Saved Register Space

Figure 3 Format of the PSW Register

Figure 2.4 SOAR PSW Special Register

CHAPTER 3

3. Design Approach

At the time my work began in the Spring semester 1984 the SOAR processor consisted of the set of modules designed in the winter quarter CS290 class of 1983. These modules resulted from the first pass through the design of each the major component. The Input section, Register File, ALU and PC sections formed the datapath. The source and destination registers were complete and so were the shadow and control pipe (CPIPE) registers. The interlock logic was complete and the control PLA's had been generated. What remained to be implemented was the remaining control circuitry and the integration of all of the components. Compared to what had already had been done this seemed like a modest task; certainly this could be done in a few short weeks and the chip would be complete!

This was the attitude of many people involved with the project. What was not anticipated were some of the setbacks that were encountered and the design flaws that had to be corrected. Of all the modules built at the start of this project the only one that remains intact is the six-transistor RAM cell that makes up the register file. Everything else had to be modified, redesigned, or thrown out. This included the all the PLA's, interlock logic, source and destination registers, Current Window Pointer, Saved Window Pointer, ALU, Memory Address Latch and CPIPE. Even the original pads had to be replaces because of area limitations. The components in SOAR now bear little resemblance to those that were available when this work began. Add to this this fact that the control circuitry was redesigned after it was laid out and simulated it is not surprising that the chip was not fabricated until March 1985.

Rather than going into the details of every design iteration the approach taken to implement the control circuitry and the major changes that were made in the existing logic are explained here. Beyond that, a description of some of the important components in SOAR are included. The unique domino ALU, and the circuits that make up some of the critical paths such as the register file decoder, the Saved Window Pointer and program counter circuits are examined. The final CPIPE implementation is described because of its hardwired opcodes that allow multiple cycle instructions, and the process that generated the PLA's is also described here.

3.1. SOAR Implementation

From the appearance of the chip when the this work began it was obvious that there would be area problems on the right side of the chip. The pitch of the bits in the datapath were determined by the ALU. All cells had to match this cells pitch and, as a result, the entire datapath appears sparse. This is most evident in the register file. Had a different aspect ratio been chosen for the ALU there may not have been the same area limitations and the method for implementation may have been different. The floorplan of SOAR was critical and a poor one would prevent the chip fitting in the available space. The floorplan had to provide for associated components to be near each other and for simple interconnect routing. Fortunately these two features are usually not competing; a dense, structured floorplan is usually the result of judicious placement with high regard for the length of interconnect.

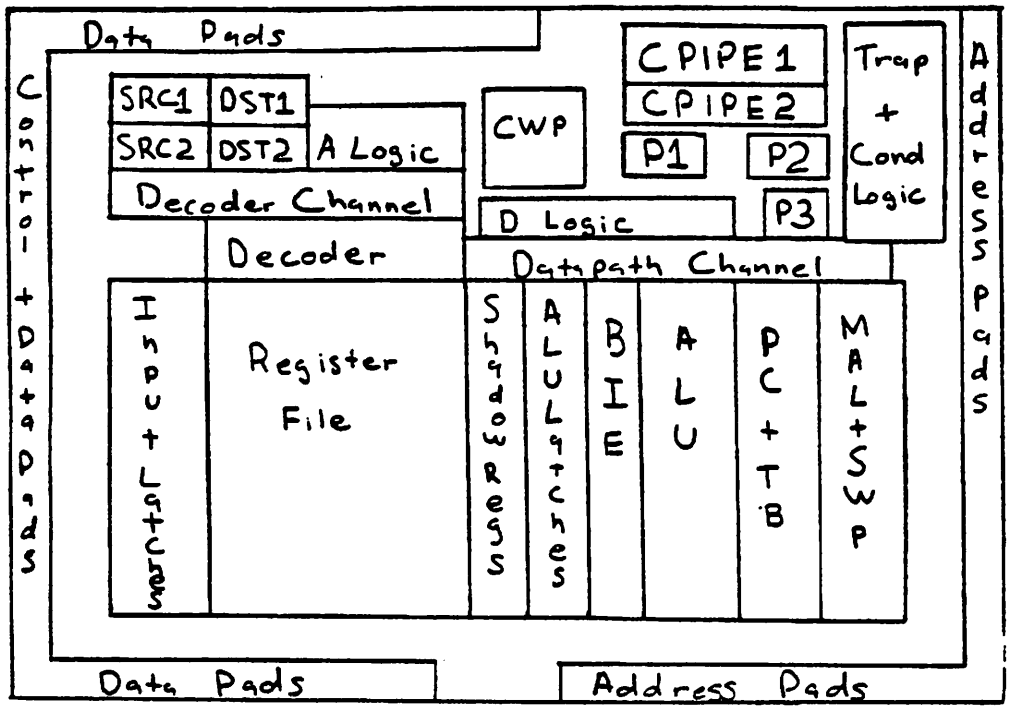
3.2. Floorplan

The floorplan was not determined at once but rather it evolved into its present state (Figure 3.1). It is surprisingly similar to what was originally conceived except for some rotating and mirroring or slight movement to allow for interconnect. Many things were simple to place, others not so easy, some required a placement that had no room for error. So even though the component was where it was supposed to be the

final placement was by no means trivial. The source and destination registers were connected together as a group (SRCDST) and obviously had to be placed near the register file decoder. The exact placement and rotation was not as obvious and decisions of this nature were postponed as long as possible. The address PLA's (apla, apla1, and apla2) were also easy to place. These PLA's and the SRCDST made up one of the large channels and their exact placement could be varied to achieve the highest channel density. The placement of the shadow registers (shadow opcode and shadow destination) was not as easy. The output of these registers gets written to the S-bus which runs vertically through the datapath in polysilicon. The inputs are the bits from the SRCDST or CPIPE register outputs. It was decided to place the shadow opcode and shadow destination registers as close to the S-bus as possible. These lines would take some time to drive and the SRCDST and CPIPE outputs would be long metal lines, so, extending them a little more would not cause too much more delay.

The two largest PLA's, cpla1 and xcpla1, make up the heart of the control logic. The speed through these PLA's is critical. These PLA's had to be as close as possible to the output of the CPIPE. Here some tradeoffs had to be made. In this area the final floorplan does not match the original one. The placement of the PLA's is the same but their orientation is different. Originally these two PLA's were placed such that the outputs faced the CPIPE and the interconnect ran across the length of the CPIPE. This design was laid out, the interconnect wired to the components, and it resulted in very short CPIPE nets. Unfortunately, this layout had to be abandoned since it was too tall to be placed above the datapath. Instead, the PLA's were rotated as they are now and the layout had to be redone. This rotation also required a redefinition of the location of the nets that entered the large channel containing the random logic.

The Trap logic was placed at the far right of the chip above the condition code PLA since this location was the only large area that remained and only a few new nets



SRC1	Source 1 Register
SRC2	Source 2 Register
DST1	Destination Register (current instruction)
DST2	Destination Register (previous instruction)
CWP	Current Window Pointer
P1	Instruction Decode PLA 1 (cpla1)
P2	Instruction Decode PLA 2 (xcpla1)
P3	Trap Condition PLA (tpla)
CPIPE1,2	Instruction Register
PC+TB	Program Counter, Trap Base Register
MAL+SWP	Memory Address Latch, Saved Window Pointer

Figure 3.1 SOAR Floorplan

had to be routed to this area.

With the placement of the major blocks completed the floorplan routing began and followed the procedure outlined in Section 3.3. It was soon obvious that there would be severe area limitations above the datapath where the CPIPE was placed. For this reason the placement of the logic gates, PLA's and the inputs to the channel to get the highest possible density was important. The channel router YACR2 [Reed85] was rerun many times after simply altering the placement of 2 or 3 input nets to the channel. Conversations with James Reed, who was working on the routing algorithm and implementation, gave insight to the workings of YACR2 and allowed the channel to be routed with the highest possible density. In fact, the interface of YACR2 to Hawk had to be redesigned by Deirdre Ryan to squeeze out the space between the tracks in the routed channel. Otherwise the chip would not have fit into the area available.

Above the datapath, in the vertical dimension, there is little usable silicon area. The nets that runs horizontally below the CPIPE have no contacts in them and do not cross each other, which was by no means a coincidence. The order was predetermined, knowing where each net originated and terminated, so that this would be the case. The new layout influenced the placement of the components and the entry points of nets in the channel below. This work was necessary because even after all the organizational work, redefinition of the channel for compactness, and designing smaller pads there was still only a 100 micron margin between the chip layout and the hard limit of MOSIS's fabrication capabilities.

3.3. Random Logic

Although most of the random logic in SOAR is implemented in PLA's many signals must be gated with clock signals or lie in a critical path and cannot be placed in a PLA. This random logic was implemented using a **structured-custom** approach to speed up the design process and reduce the layout time.

Each of the control signals in the datapath and registers have a small amount of random logic associated with them. Gating the signal on a clock phase or ANDing a few signals together before the result sets up the control point is typical of this logic. The logic requires two and three input NAND gates two and three input NOR gates and inverters. The first approach taken to implementing the logic was to place a gate near where the signal was needed, connect power and ground, and connect the signals. It soon became clear that this was a very inefficient approach. Power and ground were almost impossible to route without overlapping one another and signal input and outputs quickly became unmanageable. In addition, the logic functions were constantly changing during the layout and much of the work had to redone.

The solution to this was to use the automatic channel router YACR2 and define channels with the random gates. The signals to the datapath need complimentary signals for the pass gates. At each control point buffers were placed to generate the signal and its compliment. These tall thin buffers made gaps that were of almost no use. By designing the logic gates to fit into these gaps this area was was used and a channel was defined. Whenever possible, logic gates were placed in these gaps so that they could be routed together to satisfy the logic equations. Additional gates were placed opposite the datapath when these gaps were full or the gap was too far from where the signal was needed. This defined a channel that satisfied the logic function that allowed easy power and ground routing and allowed the interconnection to be completed automatically using YACR2.

With all the gates in place in the channel above the datapath there was still substantial room left on top of the channel. This space was filled by placing PLAs and one of the shadow registers along the top. This required designing new gates so that the power and ground connections were easier. The PLA's that were placed in the channel had critical inputs that originate from within the datapath. The byte

insert/extract PLA controls the byte insert/extractor directly and gets its inputs from bits 30 and 31 of the A-bus. Critical inputs to the condition code PLA are the two most significant bits of the A and B busses. The tag-compare and tpla PLAs use these as inputs to check the operands for type consistency. These four PLA's were the best candidates for use in the channel. The shadow opcode register writes to the S-bus and could be placed directly over the S-bus on the top of the channel. The four PLA's and the shadow register completely filled the remaining space left in the channel.

The setup time for this channel was substantial. Making sure that all the terminal names were correct and judicious placement of the input nets for highest density was non-trivial. There is no question that if this logic were placed and routed **once** by hand it would have been faster than the approach taken. There were however, many benefits of taking this structured-custom approach to the layout. Power and ground were simple to route, the empty spaces between the buffers were utilized, the layout of the chip became highly structured and regular and the layout of this section was more compact than if it were done by hand. More important than these reasons are that YACR2 guarantees that the channel is routed correctly and the turn-around time for updates in the logic was cut from days to minutes. The logic did, in fact, change several times during the implementation which proved this approach definitely was the correct method of design.

The structured-custom approach was first used on the large channel above the datapath (see Figure 3.1). Its success allowed this technique to be used wherever possible on the chip, there are a total of five areas that were laid out using YACR2. The input section (far left of datapath) had lots of logic that filled the area completely. This made a small channel that was laid out very quickly. The CPIPE and source and destination register control signal logic layout was also implemented using this structured-custom approach. The address PLA's and the more complex register for-

warding logic also use this technique. These channels, although less complex than the datapath channel, still exhibited the advantages seen in the large channel implementation. In fact these channels were much easier to layout because many bugs were eliminated from the YACR2 interface to Hawk and the system of defining the channel had been refined.

3.4. Domino ALU

The ALU is one of the most complex components in the datapath. The ALU was designed by Mark Hofmann at the beginning of the project and has changed little since it was first designed. The ALU was designed using Domino logic [Kram82]. This was because the ALU outputs are only needed on Phase 3 and so the ALU has an entire clock phase to process the operands. The ALU consists of the Byte Insert/Extractor (BIE), the Complimenter and the ALU itself.

The BIE is a simple section that can take the low byte of the A-bus and insert it into any of the other three bytes on the insert operation. An extract operation performs the reverse, byte 1,2, or 3 of the A-bus is extracted and placed in the low byte position. The circuit is shown in Figure 3.2. The signal **EX-INSpass** from the control PLA is ANDed with bits 0 and 1 from the B-bus which selects the byte to be operated on. This control function was implemented in a PLA for simplicity. The benefits gained by a hand layout were insignificant. The speed of this circuit is not critical but it must be noted that bits 0 and 1 of the B bus run up through the datapath in polysilicon and are driven only by the inverter in the ALU input latch. These devices have been made more robust but this still represents substantial loading.

The ALU needs both the true and complimented signal. Rather than running both these signals through the BIE, they are complimented and buffered at the Complimenter. The signals enter after going through the BIE which contains only n-channel devices. This did not present a problem since the busses are precharged on either side

of the Complimenter on Phase 1. These devices only have to pass a logic 0. The function of the complimenter is trivial. The Complimenter circuit does not use Domino logic and is shown in Figure 3.3.

The ALU itself uses Domino logic and a carry by-pass circuit that accelerates the carry propagation. The 4-bit carry by-pass achieves the greatest speed-up with the worst case carry propagation. In SOAR the worst case is an LSB generated carry that must propagate to the MSB in tagged mode. This is 30 bits of propagation, which means that the carry propagates through 4 bits of the carry chain and through 6 nibbles of the carry by-pass. This is the worst case because any other generated carry would be closer to the MSB and would have a shorter distance to propagate. The worst case propagate occurs in approximately 83ns [Hofn83]

Domino logic doesn't use full complimentary CMOS logic. The core devices implement the logic function as a pull-down and a clocked PMOS transistor provide the pre-charge pull-up. Since the ALU is only a single stage of logic, only n-channel core devices are used. This simplifies the clocking scheme, prevents race conditions and makes a more compact ALU. The output of the core devices are fed to static inverters. These inverters are critical to the next stage in a full Domino circuit that contain n-channel core devices in the following stage.

The ALU was extensively simulated and has been fabricated. It performs within specification and it has a worst-case add time of about 120ns. The circuit is shown in Figure 3.4. For a complete description of Domino and other CMOS logic types see [Kram82] and [Gonc83]. This ALU is more completely documented in [Hofn83].

3.5. Program Counter

The program counter can be a very slow component in the datapath, the time for the carry to ripple up the counter can be on the order of the ADD in the ALU. Unfortunately, the expense of a carry look ahead cannot be justified. The program counter

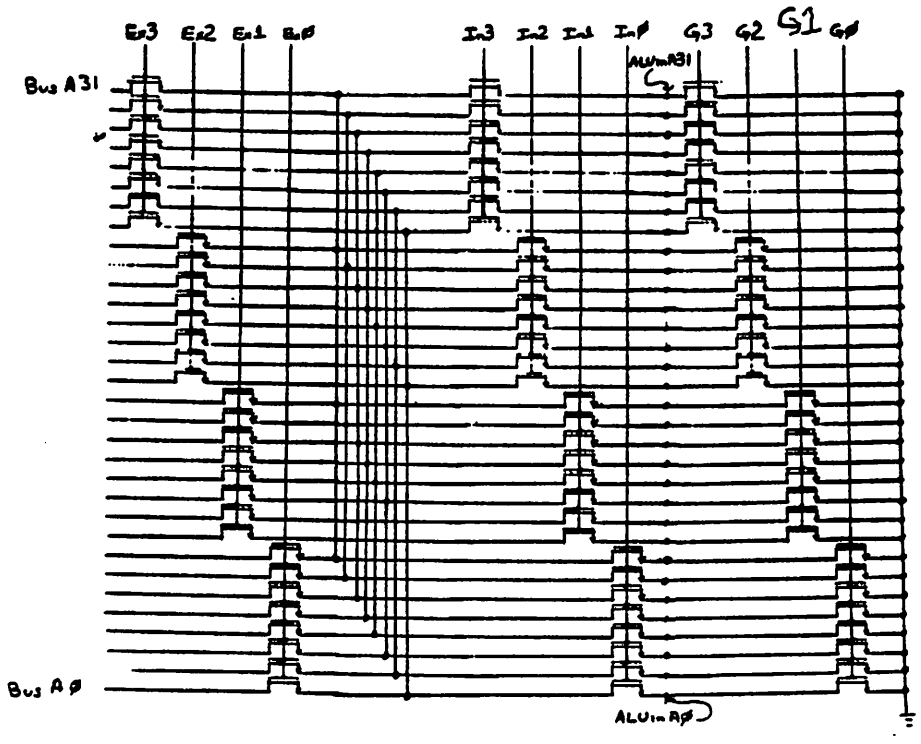


Figure 3.2 Byte Insert/Extract Circuit

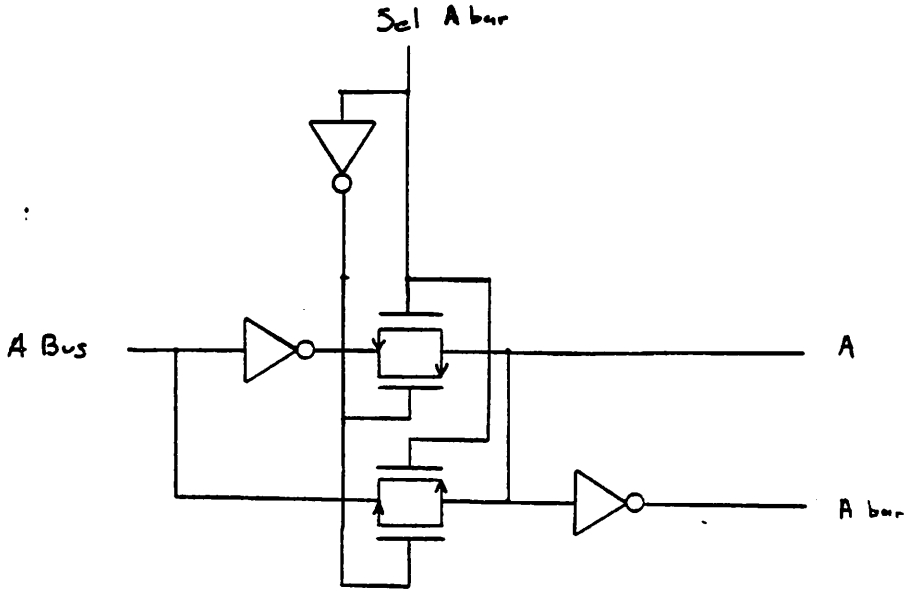


Figure 3.3 ALU Complementer Circuit

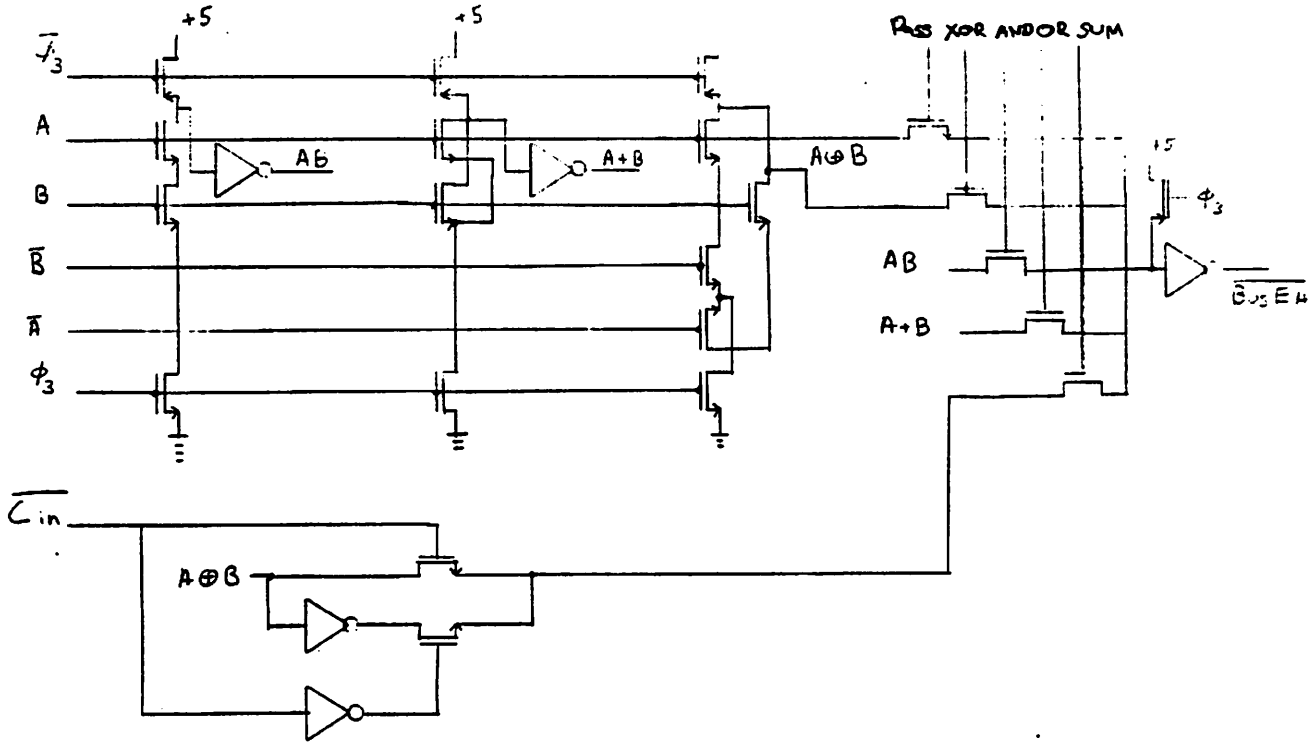


Figure 3.4 Domino ALU Circuit

in SOAR takes advantage of the fact that the contents are incremented by one and never more. A carry can be generated in the LSB and other bits can only propagate the carry. A very simple carry look-ahead scheme can be used to cut the time through this circuit in half. A 16-input AND gate is used to generate the carry in for the upper 16 bits of the program counter. Only if the lower 16 bits are 1 will there be a carry. By adding this one gate the propagation time is cut in half: diminishing returns are quickly seen using this technique for four gates would be required to cut the delay in half again. A schematic of the program counter is shown in Figure 3.5. This circuit has been simulated and worst case Program Counter increment is about 90ns.

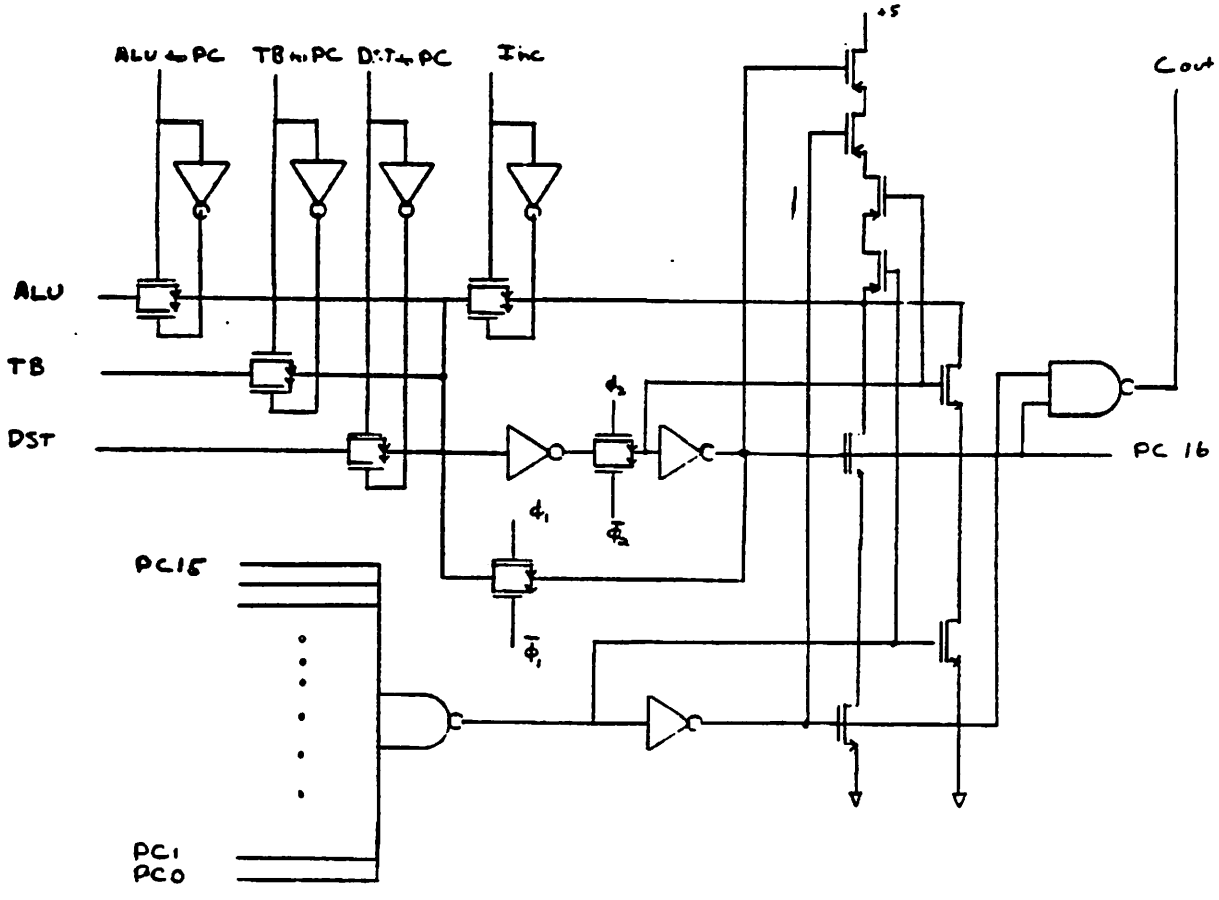


Figure 3.5 SOAR Program Counter

3.6. Saved Window Pointer

The Saved Window Pointer is an important component in the datapath and is active during Phase 1. The SWP itself is simply a register that points to the place in the Saved Window Space where the next window to be swapped out must go. This

register gets compared with the address bus on loads and stores to see if the memory location being accessed is on chip in the register file.

The test that is performed is SWP-ADD-1. If this value is zero the memory access is on chip and the signal **PTRtoREG_H** (pointer to register) becomes active. This is a complex operation to perform. Fortunately, it exhibits some of the same properties as the program counter. The subtraction is performed as it normally would but the borrow for bit 18 (the midpoint of the SWP) is generated by ORing the lower bits of the SWP together. This works because if these bits are zero there must be a borrow in the upper bits. This gate cuts the delay through the circuit in half. This circuit has been analyzed using **Crystal** [Oust83] and shows a delay of about 132ns.

3.7. PLA's

Much of the random logic for SOAR was implemented in Programmable Logic Arrays (PLA's). There are thirteen PLA's used that make up most of the control circuitry. Of the 13 PLA's, the outputs of the CPIPE are the inputs to six of them. These PLA's determine, from the opcode, what the instruction is, whether it is legal or not, and if there is a trap. They then set up the control points on the datapath accordingly. The function of the others vary; three are used to decode the source and destination of the instruction operands. Others determine the cause of a trap, the status of the condition codes and check the tag bits. Also implemented in the CMOS SOAR are two PLA's that are not found in the NMOS version. One is a 3-bit incrementer used for the Current Window Pointer (see Section 2.6) the other is used to set up the control signals for the byte inserter/extractor. PLA's were used because the logic in these PLA's is simple and dense and little benefit can be realized using fully-static CMOS for these functions.

Synthesis of PLA's is a simple procedure using the PLA generation tools available at Berkeley. The logic information first must be extracted from the SLANG description

of SOAR. This is done by using a program called SPLAT (Slang to PLA Truth table). The output of this program is a "C-like" format of logic expressions. The logic equations are then mapped into an AND-OR PLA truth table using the program **eqntott** (equations to truth table). The output of this program is a truth table satisfying the input equations that can be manipulated by logic minimization programs.

At the start of this project **pop** was the only logic minimization program available. Later in the project a new minimization tool was available, **Espresso,** [Rude85] which was run on these truth tables and a further reduction in area was achieved.

Once the truth table is minimized the physical layout of the PLA can be generated. This is conceptually simple because a 0 (1) in the AND plane means the true (inverted) input is used in that product term. A "-" is a don't care input. In the OR plane a 1 (0) means that the product term is used (not used) to generate the output. Each of these functions as well as input and output buffers are represented as **tiles** that are arranged according to the truth table by a program called **tpla.** A **template,** a set of tiles that make up a PLA, is designed once for a particular technology and tpla modifies the template with additional tiles to make up a PLA from any truth table input.

There were two 3 micron CMOS PLA templates available for SOAR: p-CS3cis.tp and p-CS3trans.tp [Mah84]. These templates are identical except that the cis version has both input and outputs on the same side, the trans version has the input and outputs on opposite sides.

The output of tpla is a complete layout of the PLA in the Caesar format. This format is incompatible with the Hawk/Squid system and so a program that converts from Caesar to Squid must be run on the output to make the data suitable for Hawk.

Espresso uses an algorithm that is independent of the order of the inputs on the PLA (it uses a minterm reduction approach rather than attempt to rearrange the order of the minterms). Curiously this was not found to be the case always. For ease of layout some of the inputs to the condition code PLA were transposed. This resulted in a PLA that was reduced to 31 instead of 33 minterms. Richard Rudell believes that

This conversion is simple but does not retain the hierarchy of the original Caesar file. This was not a problem with the PLA's since they were non-hierarchical to begin with. Converting larger files that contained calls to other files would be very cumbersome and is not recommended at this time. A problem that caused some trouble was the conversion of the terminal labels. In Caesar there is no notion of a terminal. The labels in Caesar are converted to labels in Squid and since there is a label layer in Caesar this does not translate into a label on the same layer as the geometry in the Squid file. This resulted in labels on the contact layer that were to be associated with polysilicon input lines underneath and caused a problem during the simulation because each label could not be associated with its corresponding geometry. These labels were removed from the Squid files and replaced with Squid terminals using Hawk.

The original templates made for this technology were assumed to be correct. SPICE [Vlad81] was used to determine how fast the PLA's would run and to examine the rise and fall times with different loading on the output. But what was not done was to see if the logic function of the truth table was satisfied by the PLA layout. It turned out that the templates were incorrect. However, this was not discovered until SOAR was complete and the chip-level simulation phase had begun. The error was in the AND plane. The AND-OR topology of the PLA is implemented as NOT-NOR-NOR-NOT for speed. This means that the compliment of the input signal is used. The templates did not include this first NOT function. Since the details of designing a template were unclear the PLA's were modified by hand using Hawk. Fortunately this was not a major change since the solution was to flip the symmetric buffers on all the input signals to the PLA's.

The CMOS PLA's do not contain full complimentary logic and dissipate static power through PMOS pull-ups on each of the input lines. The largest PLA was simulated with a 2pf load on the output so the overall speed of the circuitry could be

estimated. The delay of 57ns is acceptable considering the conservative loading. Results on the final layout showed that the actual loading is typically 0.5pf. The SPICE results are included in Appendix B. The Crystal result (38ns) is not of any interest for this circuit.

3.8. Register Window Mechanism

Part of the RISC I design that was retained in the SOAR architecture was the register file and overlapping window scheme. SOAR's register windows are smaller than RISC I's. Studies indicate that 95% of all Smalltalk contexts need no more than eight registers [Blak83]. Based on this study the register window was reduced from sixteen to eight in SOAR. The number of windows contained in the chip is eight to allow a nesting depth of eight before the register file overflows.

The overlapping window scheme is a technique that speeds up procedure calls and returns. There are 16 local registers to a window but eight are shared by adjacent windows. As a result, eight new registers become available with every procedure call. Before a procedure is called the arguments to the called procedure are held in the low eight registers. When the call is executed the current window pointer (CWP) is incremented. This places the old low registers in the new window's high registers leaving it with eight new registers. Also contained in each window are the eight global registers and the eight special registers. This totals 32 registers that are contained in the window, the window is shown in Table 2.1.

Eight sets of eight registers plus the eight global registers make up the register file in SOAR. The special registers are scattered in throughout the datapath and are not described in this section.

The 72 32-bit registers are made up of a simple six transistor RAM cell shown in Figure 3.6. The pass gates that access the cell from the word line are not

is simply an anomaly of the heuristic approach of Espresso.

complimentary but consist of a single NMOS transistor to reduce the capacitance of the word line and increase the overall speed. The operation of the circuit is obvious. The performance of this not meaningful in itself and no detail analysis was performed on a register alone.

The decoding for the register file is done in two stages, first a three-to-eight NOR decoder and then two input NAND gates select the appropriate register. One of the inputs to the NAND gate is the Current Window Pointer (CWP) or the CWP-1 to select the either the low or high registers. The other input is the output of the three-to-eight decoder. Bit 3 of the source and and destination registers selects either the

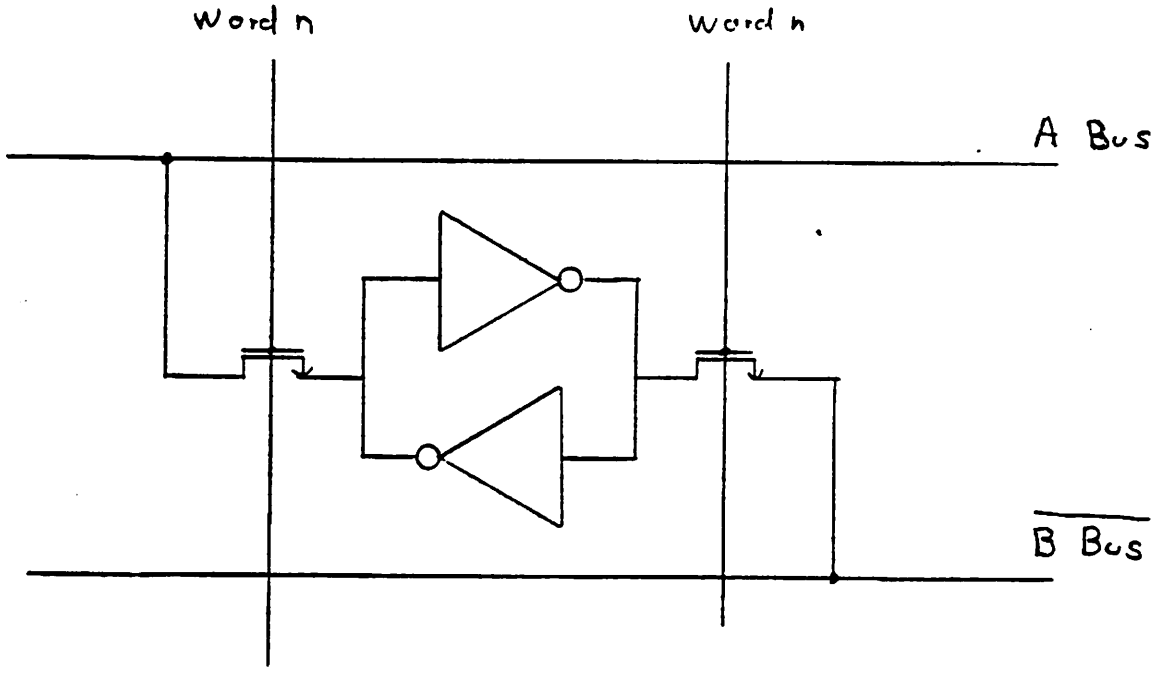


Figure 3.6 SOAR Register File Cell

CWP or CWP-1 and bits 0-2 select the register within the window. CWP and CWP-1 are 3 bits that are decoded and select the correct window.

Decoding for the global registers is handled separately. Bit 4 of the source and destination registers select the global registers and bits 0-2 select the register within the global set.

The layout of the decoder is highly structured due to its high degree of regularity. On top of the registers themselves are three banks of 72 two input NAND gates, one for the source1 source2 and destination (SRC1, SRC2, DST) addresses. On top of these banks are three sets of two three to eight decoders. One of these three-to-eight decoders decode the lower three bits of the source and destination and the other decodes the three bits of the CWP. The global registers don't have a three-to-eight decoder and are simply enabled by bit 4 of the source or destination. This scheme fully decodes the 72 registers in the register file.

SOAR has incorporated an automatic nulling of registers a procedure returns. This is achieved by enabling an entire window (specified by the destination) and nulling all registers in the window at once using the Zero Special Register. This feature is added by a slight modification of the destination three-to-eight decoder to make it activate all registers on the signal **nilonreturn.**

3.9. Opcode Latch design

The Opcode Latch (CPIPE) is a two stage latch the first (CPIPE1) is the input to most of the PLA's and their outputs set up the control points in the datapath. The next stage (CPIPE2) sets up the control associated with the last stage in the pipeline.

RISC I had no multiple cycle instructions so no mechanism was needed in its control to handle them. SOAR, however, has multiple loads and stores and a efficient method to handle them had to be added. This mechanism is simple and elegant, it resembles the mechanism that forces a flush op-code into the CPIPE on a return. The

output of CPIPE1 gets loaded on the signal **CPIPE1step_H.** This signal goes high depending on the op-code of the next instruction. If it is a normal instruction it is asserted. If not, it will stay low and another signal will be asserted. On a **loadm (storem)** instruction the **CPIPEloadm (CPIPEstorem)** signal gets asserted and forces into the CPIPE the op-code corresponding to the next step in the multiple cycle instruction. The op-code is 0010110XXX for **loadm** and 00010111XXX for **storem.** The unspecified values are determined by the destination latch output. These values start at 111 and get decremented to 000. This way they are automatically decremented and do not require a ROM for this small amount of microcode. This set up constitutes a small state machine that handles the problem of multiple cycle instructions very nicely (see Figure 3.7).

3.10. Pad Design

The original pads would have worked for SOAR but they used was too much silicon area, they had only guard rings, and they did not have any input protection. Pads could not be placed above the CPIPE and still the layout of the logic was very dense in that section. Not only were the pads too large but they would require additional interconnect to the internal circuitry that would have made the chip impossible to fabricate.

The output pads were redesigned with smaller pad area. Input protection was added, the guard rings were redesigned to be more compact, and the drive transistors, due to the layout of power and ground, were made twice as large as the old ones with no increase in overall area.

The input pads were laid out to be compatible with the output pads. They did not have the output drivers, but they did include the input protection and guard rings. Two type of input pads were designed: inverting and non inverting. The inverting pads include large inverters that drive the on chip load. The I/O pads required additional

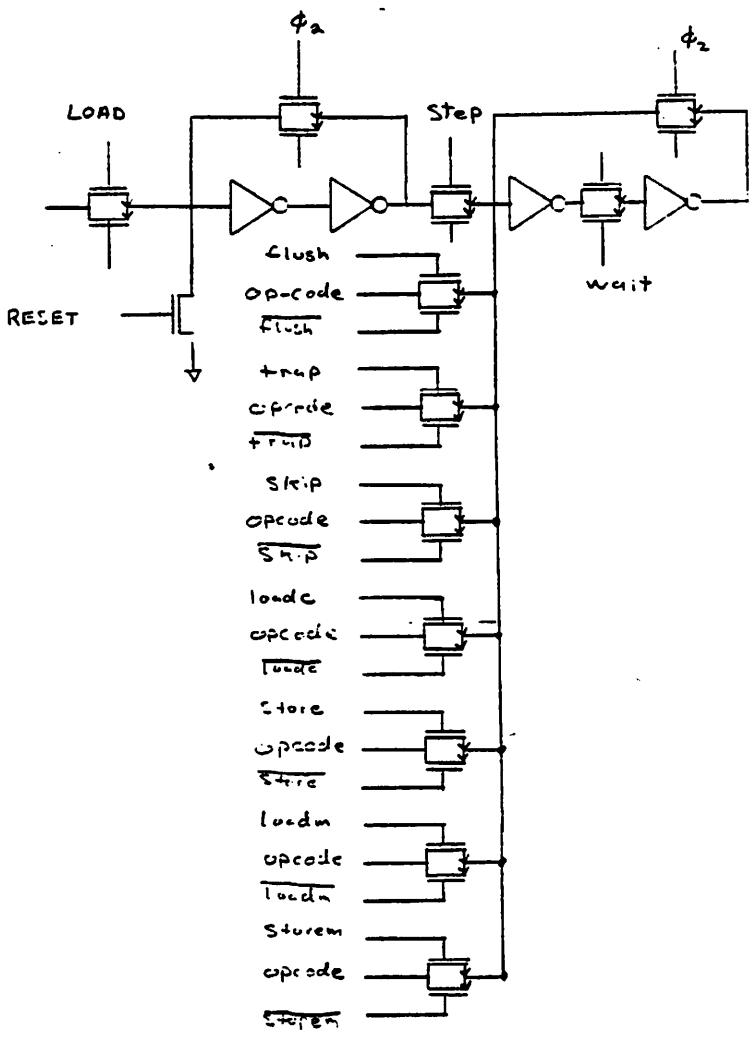


Figure 3.7 SOAR Opcode Latch Circuit

circuitry that reduced the pad interconnect in half. The old I/O pads had a separate connection for the input and output nets and a multiplexing circuit at the pad. This required routing two nets to each I/O pad. which was impossible. There were two alternative solutions to this problem. The final I/O pad circuitry is shown in Figure 3.8. The pass gate that isolates the pad from the input must be physically at the pad

because the input goes to more than one place on the chip. The multiplexing circuit must be at the datapath so the interconnect can be reduced. The pad driver can be placed either at the pad or the datapath. Placing it at the datapath would mean that the drive current (to drive two STTL loads) would have to go through the three micron metal net that ran to the pad, and through the input protection resistor. Although the net could support this current density this seemed like a poor solution. Placing the driver at the pad would eliminate this problem but would still have to drive the output through the input protection resistor. The solution was to duplicate the multiplexing function at the datapath and the pad so that the input protection resistors appeared only in the input path and the drivers did not have to drive a long metal net. This meant additional work and area and the output enable control line, **OE**, was heavily loaded, but the pads were spaced far enough apart to accommodate both the multiplexing and the tri-state circuitry for the I/O pads. The OE line could be buffered to increase speed.

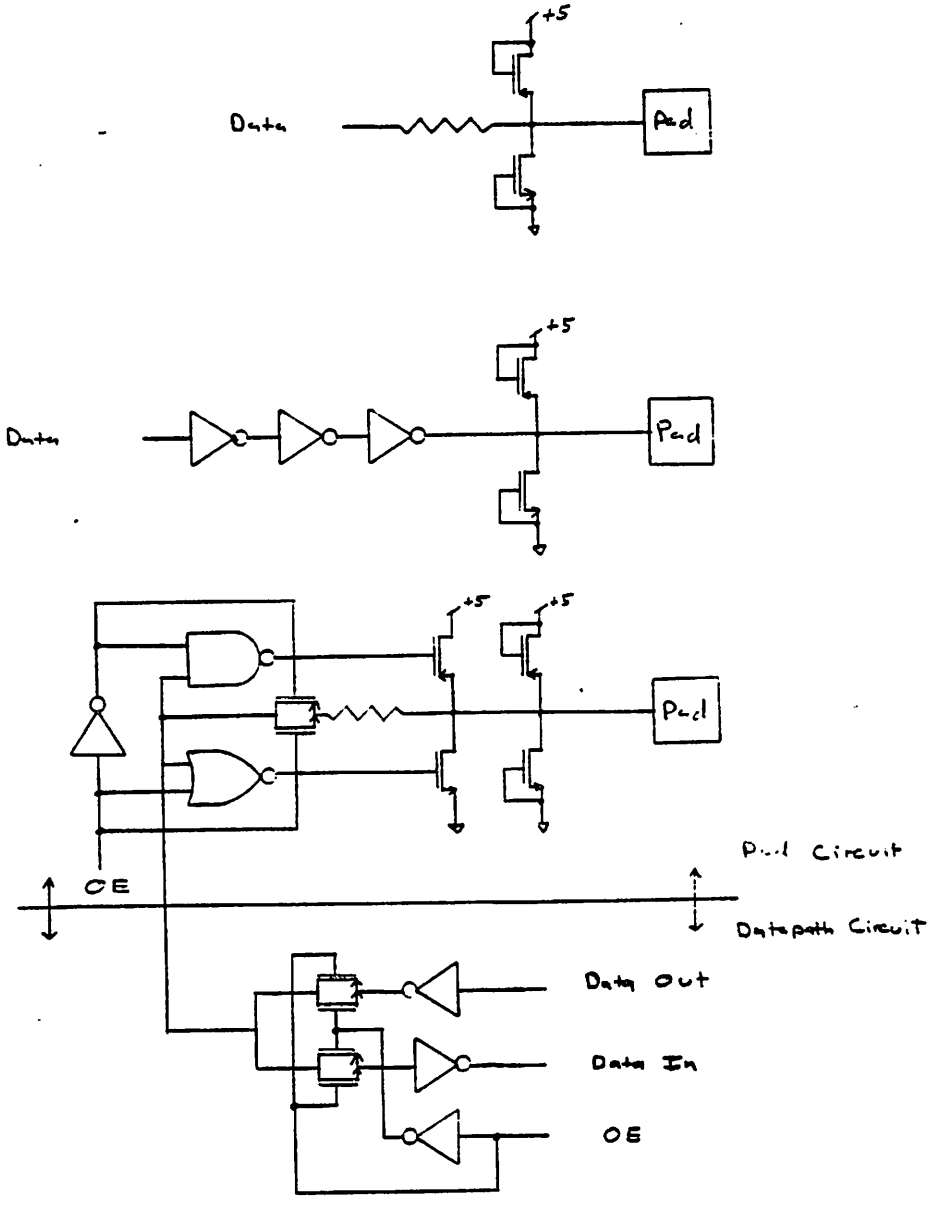


Figure 3.8 SOAR Pad Circuits

CHAPTER 4

4. Design Environment

4.1. Hawk/Squid

The SOAR microprocessor was laid out using the Hawk system [Kell84] run on a Tektronix 4113 color graphics terminal. The data is contained in the Squid database, [Kell84], which supports multiple views of a circuit in a hierarchical fashion. The microprocessor was broken down into small components called **cells.** Typically a cell would contain a register cell, NAND gate or a bit slice of the ALU. Each cell would be laid out by hand and form a Squid cell's **physical** view. The cell corresponds to a directory in the UNIX file structure and each view of that cell is stored as a file in that directory. The convention used was that the physical view contained the physical representation of the geometries that made up that cell. Other views might be a **schematic** view showing the schematic of the cell or a **symbolic** view which is similar to the physical view except that the detail of the structure is suppressed and only protection frames and terminals are used to represent the connectivity information and the size of the cell [Kell84].

The storage format of a Squid view is simple: each line represents a terminal and net list information, a geometry, or a call or instance of another Squid circuit view. At the time SOAR was designed, the format was stored as free text rather than as a binary data representation. This feature proved very useful under many circumstances in this project. For example, the symbolic view (generated by the **framer**) was substituted for the physical view in many places where the details of the structure were unnecessary. When the physical view was needed, the view (file) was edited using a text editor and the symbolic view was replaced by the physical view. This was

much faster and more efficient that doing the same thing using Hawk. This was most useful when editing a large views that contained many instances of other views as it cut down greatly on read-in and redisplay time. This technique was taken to extreme at some times where an empty view was defined that required no read-in or redisplay time.

Another advantage of ASCII Squid database format is seen when the cell is being checked for design rule violations (DRV's) in a large view. The Lyra layer (layer that indicates DRV's in layout) geometries that indicate DRV's are three to five microns wide. In a view that is thousands of microns long these geometries are invisible. By searching the Squid file for geometries on the Lyra layer the location of the DRV can be pinpointed exactly and found immediately on the graphics terminal.

The fact that the files are kept in an ASCII form is because of the nature of the Berkeley research environment. Since Hawk/Squid is a research project itself, it is constantly under development to enhance or add features. With the files in ASCII they are easier to maintain and track down bugs in Hawk. If Hawk reaches a steady state with good reliability, binary files could be used to achieve the speed-ups that a text editor provides. In fact, earlier versions of Squid did use a binary data structure that was about 50 times faster than the present implementation.

The Hawk system has many features that aided in the design and layout of SOAR. A basic but valuable feature is the multiple-window "desktop". By separating the screen into different windows laying out long interconnect by hand is made much simpler. One end can be magnified in one window and the other end in another window. The resolution of the pointing device is increased and entire nets can be placed at once. Different views can be put in each window and each window has a stack that can contain additional views. This is very useful when editing a large number of cells that are all contained in a larger cell. It was particularly useful when the final touches

were being made to the SOAR layout where many views had to fit together exactly and each cell had to be changed slightly.

Along these lines another very useful feature is the **subedit** or **edit-in-context** command. When trying to get things to fit together it is very useful to see the boundaries of the other cells as they relate to the cell being edited. There is a problem with this command though. When a cell undergoes a rotation or mirroring and is edited-in-context the movements to not go through the same translation. That is, a horizontal movement for a cell that was placed sideways would move vertically. This bug was observed early and a work-around was employed.

Hawk is not simply a CIF editor; it has, in addition to the graphics capabilities mentioned, some advanced features that make it very useful in the design of large digital systems such as SOAR. A simple but valuable tool is the **arraymaker** [Kell84]. This tool takes as an input a script that describes a linear array. The elements of the array can be mirrored or rotated and each can be unique. The Squid view is automatically generated without having to place each cell by hand using Hawk. This tool was used in every register in SOAR's datapath and the ALU. The register file was generated by first generating a column array and then using this cell as the elements of another array that made up the complete register file.

The **programmer** was also a useful tool in the design of SOAR. This tool tailored Squid views to perform a specific function. The byte insert/extractor, ALU complimenter and register file decoder banks all used this tool to change a common cell to be slightly different from the others. This usually meant that a contact would be placed at a particular location to customize the cell.

Additional tools were developed during the implementation of SOAR. These tool were motivated partially by there usefulness in the layout of the chip. Because of the tight area restrictions in the layout the placement of components was experimented

with many times. This required placing a cell routing it up and seeing what it looked like. This often took hours of layout time and usually had to be redone. This drove the development of the **interactive routing toolbox** by Deirdre Ryan [Ryan85].

This toolbox contains some programs that aid in the routing of cells in Hawk. The tool that was used the most was **pitchchange** tool. This tool selects a set of nets and extends them and sets a new pitch. Another tool performed a similar function and was called **L-turn.** A group of nets would be selected and would turn a corner and reset the pitch. These tools proved invaluable while laying out SOAR. The placement of a cell was not left where it was because it had been completed and moving it would be time consuming. It could be moved to the optimum position because its interconnections could be re-implemented in a very short time. The interconnect was generated as a cell itself and could be removed very easily.

These tools allowed for the optimum placement of components and made the dense floorplan possible. Another tool, the **cable** command, allowed nets to be defined at the beginning and end with terminals and an arbitrary path between them would be routed automatically. This command would permute the nets to achieve proper conductivity and included an algorithm to achieve maximum density.

4.2. YACR2

One of the most valuable tool for the design of SOAR was the YACR2 automatic channel router [Reed85]. The interface of YACR2 to the Hawk/Squid system was implemented by Deirdre Ryan and Richard Rudell. The operation of the router is straight forward. The channel is defined by sets of colinear terminals (additional terminals can be added at one end). The boundaries of the channel are pointed to after invoking the channel router command through Hawk. Hawk searches the perimeter of the region defined by the input points and makes up the input file for the YACR2 algorithm. YACR2 takes over and routes the channel and returns the file that describes

the solution. Hawk uses this file and translates it into the geometries that correspond to the technology that is currently being used. This tool is technology independent and can be changed my modifying the **.cadrc** file that specifies the present technology. This tool also provides useful information that is helpful when rough calculations are made on the delay through the channel. Longest net in both polysilicon and metal, signal name and the number of tracks in the channel are provided by YACR2. Also, if there is too much room allocated to a channel the interface will say by how much the channel can shrink. Likewise, if the channel cannot fit the YACR2 interface will tell the user how much more room is needed.

Originally, the spacing between tracks in the Squid view that YACR2 created were set by the worst-case situation (two contacts next to each other in adjacent tracks). This added two microns per track in the CMOS technology. It turned out that situation rarely occurred. The main channel in SOAR contained 52 tracks and 400 contacts and this condition occurred only twice. As a result, 100 microns of channel width were completely wasted along the entire length of the channel. Under the tight requirements of the chip such a circumstance was unacceptable. At my request, the YACR2 algorithm and the Squid interface were modified to add the extra space to a track only when necessary. This saved 100 microns by 4000 microns in area and provided a little breathing room for the other components above the channel.

The benefits of these tools are numerous but they are not without flaws. At the start, the pitch change command would only work when going from left to right and would create net of infinite length and width when used in the other directions. The **L-turn** command had similar flaws, the majority rule selection process was flakey at first and the resultant nets would often be displayed on the wrong layer although they were in fact correct. The **pitchchange** command's limit of **36** nets was exceeded and caused Hawk to crash. All these problems were solved easily and were tolerable. The

most irritating bug was seen when terminals were being moved. One terminal could be moved once without any problem. But, if the same terminal was moved again without deselecting it first Hawk would crash. The **select-by-rectangle** command would occasionally miss a terminal within the selected region and that terminal would become un-selectable. These bugs were very troublesome when defining the channel and experimenting with different placements of components because the read-in time for those files was long. Because of the close relationship to the CAD tool designer these bugs were discovered and fixed quickly. In fact many of these "problems" may not be considered bugs but simply limitations that were exceeded in this application. This feedback is essential for the rapid development of useful CAD tools.

Hawk also provided interactive design rule checking (DRC) through **lyra.** A region can be specified and lyra would check it. This was very convenient when the cells were being designed but proved less useful when the routing began. The batch DRC **Squidlyra** was used to check for DRV's on large Squid views and the violations corrected at a later time all at once.

The **framer** [Kell84] was used to generate the symbolic representations of the circuits. This program would take the physical view and suppress the detail within the boundaries and leave only a set of protection frames that would define the area that is occupied by a particular layer. The symbolic view would be used as the view that was placed in a higher level of the circuit. This would speed up the read in and display time. The framer would retain all the terminal information within the view. This turned out to be very cumbersome dragging all this information around due to the slow graphics speed. The **.cadrc** file was changed to remove much of the unwanted information and only the edge terminals were retained in the symbolic views.

4.3. Data Representation and Conversion

Hawk uses the Squid database to represent the geometries of the layout. The plotting program **cifplot** and the transistor extraction program **mextra** use the Caltech Intermediate Format (CIF) as the input data representation. The PLA tools generate cells in the Caesar format. All these formats must be interchangeable. The conversion from Caesar to Squid, Caesar to CIF, CIF to Squid and Squid to CIF must be possible. Caesar to Squid is done by a shell script called **caesartosquid.** This script invokes the Caesar graphics editor and pipes commands to it to generate the CIF output. This CIF file is then flattened using the **flatcif** command and then the program Squidtocif is run using that file. This program is used only for PLA's and since they are already flat the flatcif step is not essential. Other conversions are done in a similar fashion and are straightforward. There were some problems with the data conversion process when the entire chip had to be converted from Squid to CIF for the cifplot and mextra programs. But they were overcome with the help of Peter Moore and Tom Laidig.

CHAPTER 5

5. Simulation

The verification and simulation of a circuit the size of SOAR is not an easy task. The complexity of the system demands a large number of input test vectors and these can be very long. The circuit simulation is only a small part of the verification process in the system development. In this section the entire simulation process is described starting from the architectural verification to the detail circuit simulation of critical paths on the chip.

The architecture was defined around the Smalltalk programming language. Included in its definition are features that are intended to increase performance of the system. How can it be made certain that these features do what the architect thinks they will do? As with all designs, the architecture must be simulated. The simulation of an architecture begins with a complete description containing all the features and functions. At Berkeley, all large complex digital circuits are described at a functional level using SLANG [Scot84] (Simulation LANGuage). SLANG is a multilevel system for logic specification and and simulation. It allows the architect to specify the function of the circuit at an arbitrarily high (function unit) or low (transistor) level. For example, the ALU can be described with just a few lines of code that define the inputs, outputs and control signals. This is a very high level description. SLANG also has the capability of describing low level system functions such as pass gates. This range of functional descriptions makes SLANG a valuable high level simulator.

SLANG serves as a architecture simulator and from this description diagnostics can be run to see if the architecture does what it was intended to do. This is a logic simulation only and there is no notion of delay included in the model. The value of

SLANG is in its high level simulator. This description and the diagnostics run on it are the basis for all further simulations. Smalltalk code is used to write the diagnostics and this code is compiled to SOAR opcodes and control signals and is converted to an input that is compatible with SLANG. This compiled code is also converted into the esim format so that the same diagnostics that were run on the architectural description make up the test vectors that test the circuit at the lowest level. This closes the loop in the system verification process. By using the same diagnostics on both the highest and lowest level descriptions system integrity is maintained.

This describes only the logic verification and no such process is defined for the timing analysis. That is, SLANG has no timing information and allows a node to switch instantaneously. There is no concept of physical proximity of nodes either. This represents a flaw in the present Berkeley design system. The functions are described in the highest level, SLANG, and then this description is translated (by hand) to a circuit. From these circuits a layout is implemented without ever doing a simulation on the transistor level description. This is an important step since only at the transistor level can there be any consideration for the delay through a device. This timing verification is postponed until after the layout is complete; essentially the timing verification is the very last step in the process before the chip is fabricated. This step should be performed before any layout begins and should be part of the architectural verification. If an architecture calls for a 360ns cycle, the architecture should provide for that and not assume that it can be built to arbitrary specifications.

This is what happened in the SOAR project. The architecture was described in SLANG and diagnostic programs were written to test it functionally. From here the circuits were designed and the chip laid out. Once the layout was complete, the transistor information was extracted and and the logic was verified using esim. Then the timing analysis was performed and resulted in an estimated clock cycle that was

10 to 15 times what the architecture specified. This was not the result of poor circuit design, but poor architecture. The interlock logic that controls the pipeline was designed poorly; under some conditions the control PLAs had to be evaluated 2 or three times in a single clock cycle. There was no way of knowing this under the present design procedures. Once this problem was detected, the architecture had to be redefined and the control circuitry had to be re-implemented. This error was detected in the NMOS version of the chip because that version was farther along than the CMOS version. If there was a way to estimate the timing of the system, even rough estimates, this redesign and second time through the loop could be avoided.

This describes the system simulation of SOAR. But most of the actual simulation was done after the SLANG was defined and successfully running. Getting the SLANG to run is quite a bit easier than getting the circuit to work. How is the simulation started? Where do you begin debugging such a large system? The diagnostics test a working chip: They were not written to test a nonworking chip, they were not intended as debugging aids. For this reason a short manageable diagnostic that tests key features is essential. Once this short diagnostic program runs successfully it is reasonable to assume that most of the logic functions properly. After that it is a question of running diagnostics that exercise some of the isolated, less important logic. The first diagnostic run is called **test.patch** this set of vectors would be almost meaningless as a SLANG diagnostic, all it does is reset the processor and loads the Current Window Pointer, Program Counter, Trap Base, and Process Status Word. Then it provides an illegal opcode and generates a trap and returns. With this short diagnostic test vector the ALU, CPIPE, CWP, PC, TB, MAL, SWP, SRC1, SRC2, DST the input and output circuitry, PLA's and addressing can be verified. This represents a substantial amount of SOAR's circuitry.

Debugging SOAR with this vector was quite difficult. The first thing to be debugged was the CPIPE. Getting this to load was a complex and tedious task. To get these latches to load the input circuitry had to be debugged. At the start there were very many bugs, after one was found (that didn't take very long) the circuit had to be updated converted from Squid to CIF and extracted (see Section 4.3 for a description of this conversion process). What made this especially difficult was that the tri-state pads' output enable line is generated by the control PLA's output. The input to these PLA's is the CPIPE output. It took many iterations to correct these bugs and load the CPIPE successfully. After this the next most painful operation to debug was the addressing. This is because it is the most complex portion of the processor. The register forwarding logic and pipeline interlock logic must work for any address to be unique. If the address is not unique all busses become undefined. These were long and difficult problems to solve. Once they were solved, all the other problems were minor and straightforward to solve, although there were many of them! These problems were faster to fix because each was independent and so many bugs could be solved in parallel for each extraction.

5.1. Performance

The Architects of SOAR planned for a cycle time of 360ns. An 80ns bus precharge was assumed for Phase 1, a 130ns register read operation during Phase 2 and a 90ns ALU operation during Phase 3. Each phase requires a 20ns underlap. This totals 360ns, see Table 2.4. This cycle time was based on estimates of ALU speed, size of busses and register cell drive capability. These estimates were very far off. What was not accounted for was the complex decoding of the source and destination, which lengthens Phase 1 and the pointer to register logic which lengthens Phase 3 considerably.

Once SOAR was debugged to the point where it performed most of its functions a timing analysis was begun. The method of determining how fast a large system such as SOAR will run is a complex task. The timing tool Crystal [Oust83] was used for this analysis. This tool uses a simple model to calculate the delay through a circuit. Crystal knows nothing about the values of the nodes in the circuit. It can follow impossible paths trying to find which path is the longest. Once Crystal has been run, the path it believes to be the longest can be analyzed to determine if in fact it is a reasonable path. Often this process requires many iterations. Analysis of all phases starts with the setting up the pads as inputs and letting Crystal perform its analysis. In general, the results will be in error but this first analysis provides a starting point for the circuit setup phase that must then be performed. For the SOAR design, analyzing Phase 1 was the easiest. During Phase 1 no complex circuits are active. What Crystal found first was a critical path of 1200ns that ended at the A-bus. At first, this seemed totally unreasonable. Nothing ever reaches the busses on Phase 1; all the busses are precharged on Phase 1. Further analysis of Crystal's output showed that, although the path it found to the busses was impossible, there were, in fact, other legitimate paths to the busses on Phase 1. It turned out that the logic to one of the drivers of the A and B busses was changed to prevent a race condition (detected by esim) and the solution to the race condition allowed the busses to lose charge on Phase 1. This was a valid timing error that Crystal detected. Once this problem was fixed the critical path was almost the same except that the path stopped before the busses. A detailed examination of the path showed that it was an impossible path, which illustrates one of the limitations of Crystal. The sign extender circuit has a multiplexing function that allows either the raw data or the sign extended data to enter the datapath. The signals that drive the multiplexor are compliments of each other, if one is on the other must be off. Crystal doesn't know this; it knows almost nothing about the state of the circuit, it only sees paths through it. The path it saw as the worst was zig-zagging

through the sign extender and up and down the input register. This was obviously an invalid path. The solution to eliminate this path was to fix the node driving the multiplexor to 1 which defined a legitimate path. It must be noted that care must be taken when assigning values in Crystal because the user can possibly mask true critical paths. The values that were fixed were such that the longest path remained open.

With these nodes set the next run of Crystal produced legitimate paths. Unfortunately, the worst was one that was unanticipated. It ended in driving the output enable line **OE**. the delay along this path was over 1000ns. The path is reasonable since it originates at the CPIPE where the output is loaded on Phase 1, the signals propagate through the control PLA's and the output drives the **OE** line. This line drives all the pads and the multiplexor in the datapath (see Section 3.10). This path was shortened by buffering it at shorter intervals.

The next longest path was the output of the decoders. There are 72 equivalent paths, one for each register. The path is from the destination latch through the decoders, and its delay time was 235ns. This is not really the

critical path because the destination is not used until Phase 3 for the register write so it has plenty of time to set up. An identical path exists for the source1 and source2 latches. This is the true critical path and Crystal shows it to be 230ns. SPICE was run on this path and it determined the actual delay to be 360ns. The path is illustrated in Figure 5.1 and Crystal's analysis and SPICE simulation results are included in Appendix B.

Phase 2 was a little more difficult to analyze. The problem in the sign extender was also encountered in the byte insert/extractor but this time the fix was much faster. After this was fixed the worst case path included the control PLA's; this was impossible since they are evaluated on Phase 1. It turned out that the cause was the refresh in the CPIPE, which is performed during Phase 2. Crystal doesn't know about

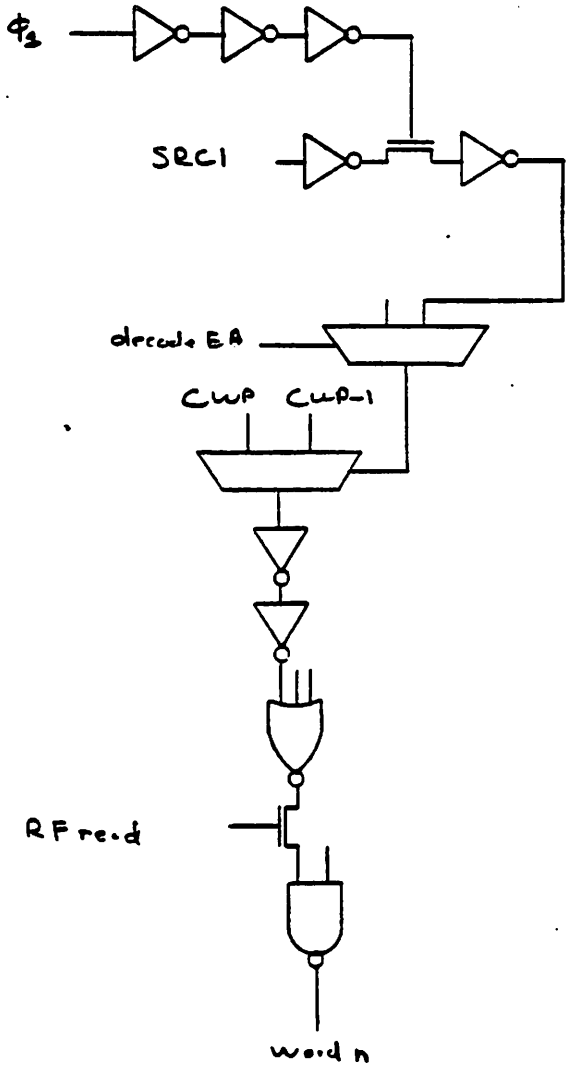


Figure 5.1 Phase 1 Critical Path

refreshing and assumes new data gets loaded. All the refresh circuits had to be turned

off to eliminate this problem. The critical path for Phase 2 ends with valid data applied to the ALU input. This data is normally applied from the register file but the worst case occurs when the input to the latches is the output of the ALU from the execution of the previous instruction. The register forwarding operation takes the output of the ALU, puts it on the D bus and then routes the data back to the ALU input latches. Since the D bus is larger than the A or B-busses this is the critical path for Phase 2. Crystal estimated the delay along this path to be 195ns, which agrees with the SPICE analysis. The critical path is shown in Figure 5.2 and Crystal's analysis and SPICE simulations are included in Appendix B.

Phase 3 was the most difficult to analyze. This is because of the complex domino ALU that is evaluated on Phase 3 and because the program counter and SWP are loaded on Phase 3. All of these circuits have carry chains that Crystal doesn't handle very well. Once all the bugs were found a legitimate Phase 3 critical path was found. The path was not exactly where it was assumed it would be originally. It did include the ALU evaluate but it also included the program counter, memory address latch (MAL) and the pointer to register logic. What was not anticipated was the condition where the ALU output gets loaded in the program counter on both load and store operations. This placed the PC, the MAL and SWP in the path. The carry chain of the PC is not included because the next PC value is not needed until the next cycle. Loading the three latches does not take long but when the SWP gets loaded the **PTRtoREG** signal gets evaluated (see Section 3.6). Recall that generating this signal requires determining SWP-ADD-1. The Crystal output and SPICE simulation are included in Appendix B. This is not the critical path because the SWP has until the next cycle to evaluate the comparison. Therefore the critical path for Phase 3 is the ALU evaluate. The critical path is shown on Figure 5.3 and is 243ns by Crystal's estimation. SPICE was run on this path and it determined the actual delay to be 348ns. These results are included in Appendix B.

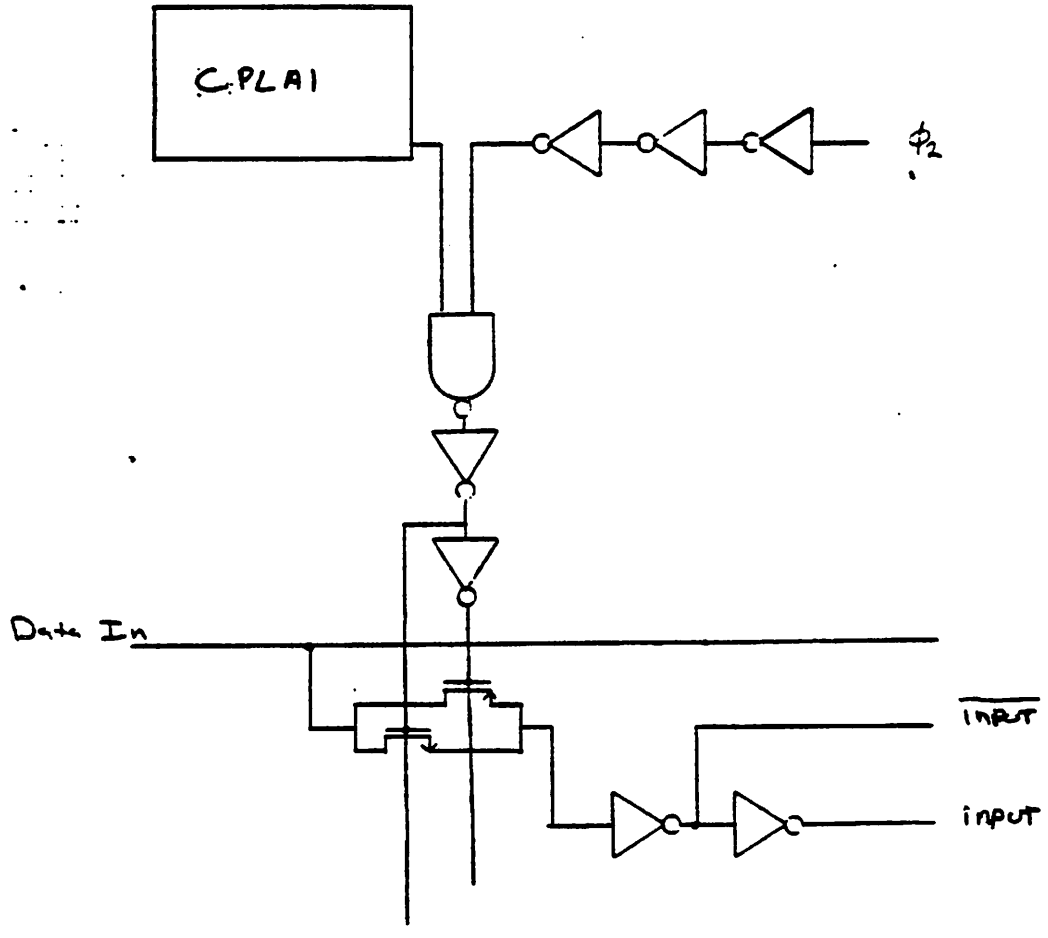


Figure 5.2 Phase 2 Critical Path

These results from Crystal compare favorably with the Crystal results for the NMOS version of SOAR. These results are based on parameters derived from previous MOSIS 3u CMOS processes. The parameters (listed in Appendix C) that Crystal uses are only the resistance and capacitance per unit area for polysilicon, diffusion and metal. The capacitance from run to run doesn't change drastically but the diffusion

resistance can vary tremendously and a conservative value was used. This along with the conservative method Crystal uses for determining the critical path the simulation results are very pessimistic. Table 5.1 shows a comparison of the SPICE results and the Crystal results for some of the paths in SOAR.

Table 5.1 SPICE and Crystal Comparison		
Path	SPICE	Crystal
Phase 1	370ns	230ns
Phase 2	200ns	195ns
Phase 3	340ns	243ns
Underlap	80ns	60ns
PLA	57ns	38ns

The overall speed of SOAR can now be estimated. The cycle time consists of the sum of the three phases plus the underlap between phases. This underlap is critical between Phase 2 and 3. The register is being read on Phase 2 and written to in Phase 3. If the underlap is insufficient the word lines for the source could still be active and the data written to the destination could corrupt the source register. To eliminate this problem the underlap between Phase 2 and 3 must be long enough for the **RFidle** line (active on not Phase 2 and not Phase 3) to discharge the word lines. The circuit is shown in Figure 5.4. Crystal estimates the necessary delay to be 60ns but the SPICE estimate is 80ns. A 20ns underlap is sufficient between Phase 1 and 3 and Phase 1 and 2. Using SPICE estimates, the total cycle time is 370 + 200+ 340 + 120 = 1030ns. Note that these results are pessimistic.

One on the goals of SOAR was to build a Smalltalk system whose speed can compare with that of a Dorado, an ECL minicomputer. The original estimates of a 360ns

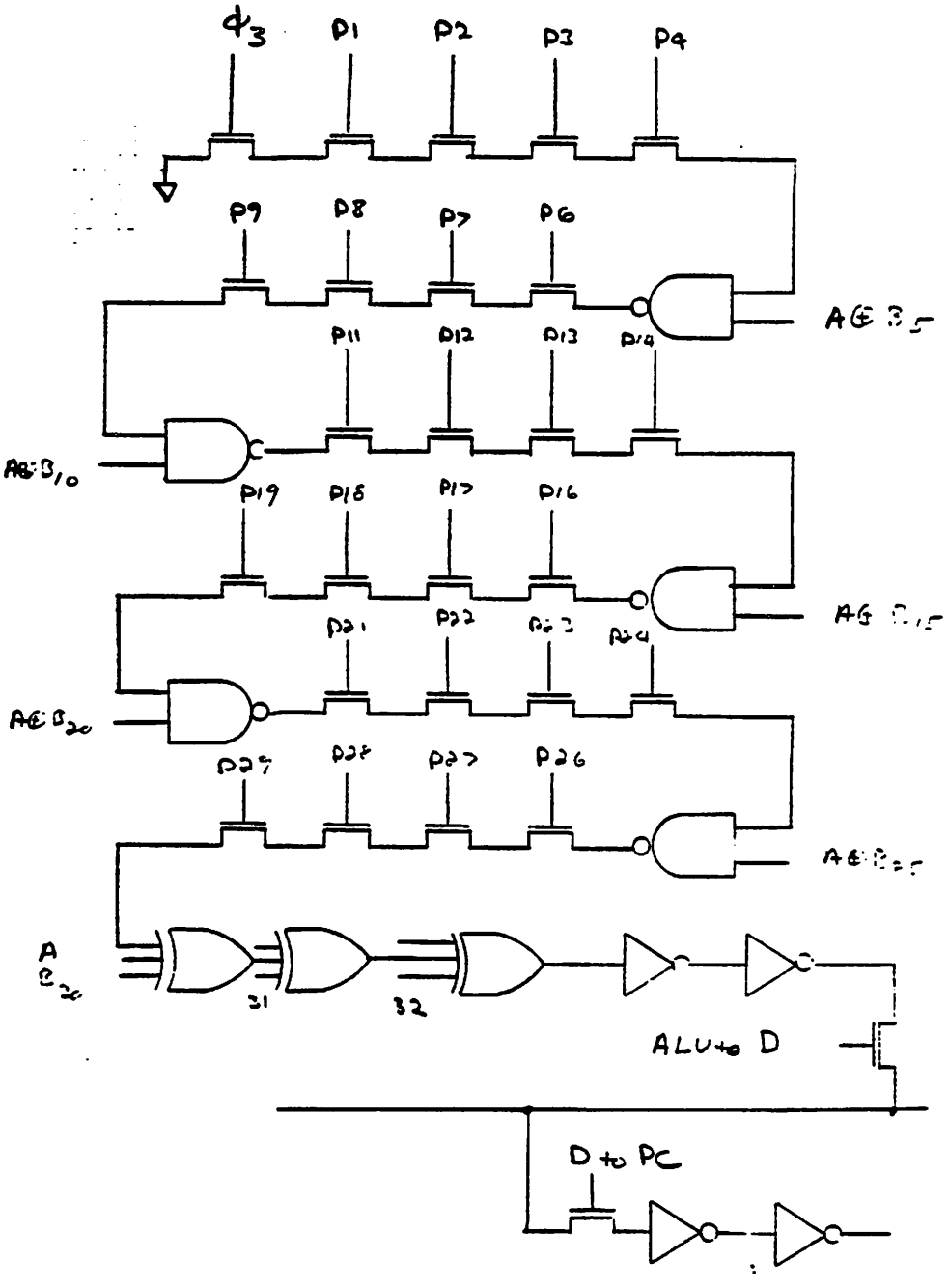


Figure 5.3 Phase 3 Critical Path

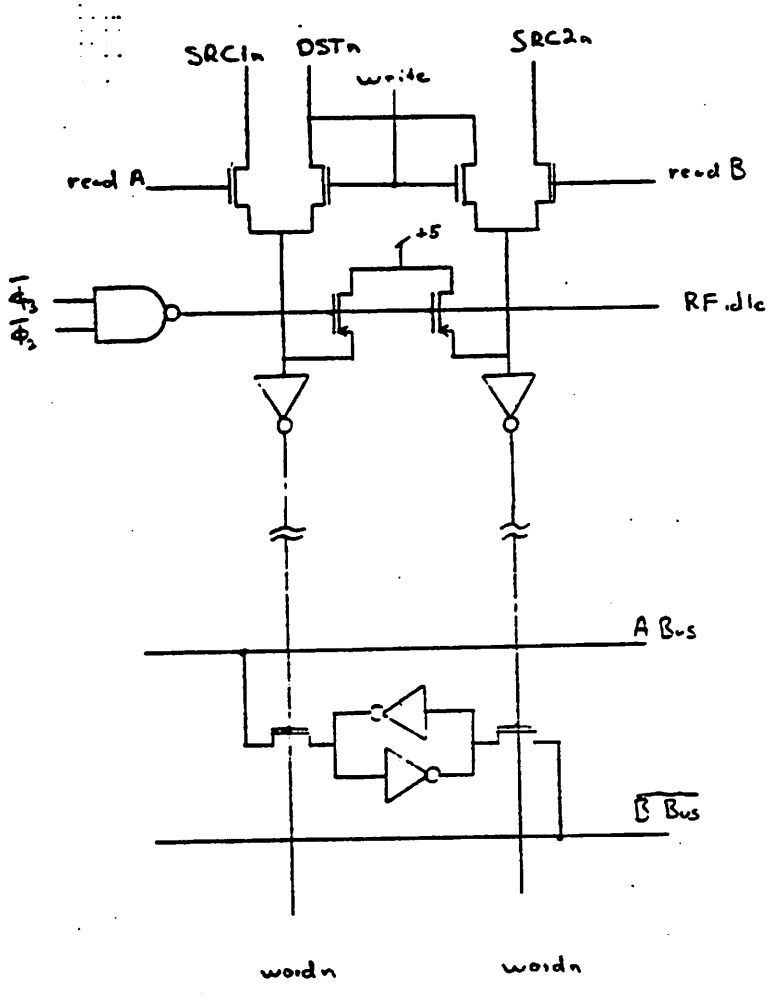


Figure 5.4 Phase 2 Phase 3 Underlap Requirements

cycle time would result in performance, based on a set of benchmarks, equal to that of a Dorado. With the estimates of Crystal and SPICE greater than the original goal of 360ns and the belief that this estimates are very pessimistic it is reasonable to assume that SOAR will execute Smalltalk as fast as a Dorado.

CHAPTER 6

6. Conclusions

The CMOS version of the SOAR microprocessor has been described. The architecture consists of a 32 bit tagged-data microprocessor that was designed to execute Smalltalk-80 efficiently. The methods used to achieve this efficiency were described in the architectural overview section. The tags and traps are the main support of for Smalltalk. They provide a variety of interrupts and a flexible service mechanism that allow for efficient memory management. The register file system serves to speed up calls to subroutines and reduce the memory bandwidth normally seen in pure load/store architectures. The RISC philosophy was maintained in SOAR and the emphasis in this project was to reduce the clock cycle to its absolute minimum.

The method of chip design for CMOS SOAR was described. The Hawk/Squid system was used to lay out this chip and many of the advanced IC design features provided by this CAD system were used. A structured-custom approach was used to implement the random logic. Routing tool, developed by Deirdre Ryan, were used route most of this logic. This method allowed for quick turn-around time and guaranteed logic correctness in the routing. The performance figures suggest that this approach did not adversely affect overall circuit performance.

The actual performance of this chip is unclear. Since the reliability of timing analysis tools such as Crystal for the new CMOS technology is not known. The models used for timing analysis were based on previous CMOS runs fabricated by MOSIS. The SPICE parameters are for a worst case design and these parameters degrade performance drastically. The actual parameters for the run that this chip goes through will have a great bearing on the overall performance of the processor.

This processor represents only one aspect of the SOAR project. A board that interfaces SOAR with a workstation has also been developed and the Berkeley Smalltalk system will be run on this combination. The CMOS version of SOAR is pin-for-pin compatible with the NMOS version. The speed of Smalltalk on SOAR is estimated to be about that of Smalltalk running on a Dorado. This processor is an application of an expermental programming environment on a RISC. The final results results will not be verified until the working chips are tested but from all the indications available and the findings in this paper it is reasonable to conclude that SOAR will be as fast as any Smalltalk system available at this time.

APPENDIX A

Interaction of CWP and SWP

Appendix A - Interaction of CWP and SWP

The interaction of the register file, CWP and SWP is not difficult to grasp intuitively. Getting all of the numbers to work out in detail, however, is a wheel of a different magnitude. This appendix will hopefully prevent others from having to reinvent this wheel.

Assume the CWP is initialized to 7, and the SWP is initialized to some value, the last seven bits of which are zero (note below that only the last eight bits are dispayed: x is a binary digit and w is the other binary digit). Assume that we have a root program P1 with procedures P2, P3, ...

CWP	ACTION	CWP (after action)	SWP (after action)	w7	w6	w5	w4	w3	w2	w1	w0
?	initialize	7	x000 0000	1H	1L	–	–	–	–	–	–
7	P1 calls P2	6	x000 0000	1H	2H	2L	–	–	–	–	–
6	P2 calls P3	5	x000 0000	1H	2H	3H	3L	–	–	–	–
5	P3 calls P4	4	x000 0000	1H	2H	3H	4H	4L	–	–	–
4	P4 calls P5	3	x000 0000	1H	2H	3H	4H	5H	5L	–	–
3	P5 calls P6	2	x000 0000	1H	2H	3H	4H	5H	6H	6L	–
2	P6 calls P7	1	x000 0000	1H	2H	3H	4H	5H	6H	7H	7L
1	P7 calls P8	1	x000 0000	1H	2H	3H	4H	5H	6H	TH	TL
	The overflow trap handler can use only globals to spill the window.										
	decrement SWP, store w7(1H) into (w111 0000)										
		0	w111 0000	8L	2H	3H	4H	5H	6H	7H	8H
0	P8 calls P9	7	w111 0000	TH	TL	3H	4H	5H	6H	7H	8H
	decrement SWP, store w6(2H) into (w110 0000)										
		7	w110 0000	9H	9L	3H	4H	5H	6H	7H	8H
7	P9 calls Pa	6	w110 0000	9H	TH	TL	4H	5H	6H	7H	8H
	decrement SWP, store w5(3H) into (w101 0000)										
		6	w101 0000	9H	aH	aL	4H	5H	6H	7H	8H
6	Pa returns to P9	7	w101 0000	9H	9L	–	4H	5H	6H	7H	8H
7	P9 returns to P8	0	w101 0000	8L	–	–	4H	5H	6H	7H	8H
0	P8 returns to P7	1	w101 0000	–	–	–	4H	5H	6H	7H	7L
1	P7 returns to P6	2	w101 0000	–	–	–	4H	5H	6H	6L	–
2	P6 returns to P5	3	w101 0000	–	–	–	4H	5H	5L	–	–
3	P5 returns to P4	4	w101 0000	–	–	–	4H	4L	–	–	–
4	P4 returns to P3	4	w101 0000	–	–	–	4H	TL	–	–	–
	read (w101 000)(3H) into w5, increment SWP, increment CWP										
		5	w110 0000	–	–	3H	3L	–	–	–	–
5	P3 returns to P2	5	w110 0000	–	–	3H	TL	–	–	–	–
	read (w110 000)(2H) into w6, increment SWP, increment CWP										
		6	w111 0000	–	2H	2L	–	–	–	–	–
6	P2 returns to P1	6	w111 0000	–	2H	TL	–	–	–	–	–
	read (w111 000)(1H) into w7, increment SWP, increment CWP										
		7	x000 0000	1H	1L	–	–	–	–	–	–
	And we are back to the original state. Of course, any more returns after this point would result in a major system error.										

0

APPENDIX B

SPICE and Crystal Output

1********05/06/85 ******** SPICE 2G.6 3/15/83 ********21:40:23*****

0* 3U P WELL CMOS MODELS ---- BK

0***** INPUT LISTING TEMPERATURE = 27.000 DEG C

0**********

```
.MODEL N1 NMOS(LEVEL=2 TOX=65N NSUB=15E15 VTO=1.1 XJ=0.35U LD=0.25U
+              JS=1.24E-4 PB=0.80 UO=526 UCRIT=3.97E4 UEXP=0.08
+              UTRA=0.25 GAMMA=1 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
+              VMAX=5E4 NEFF=3 RSH=30)
.MODEL N NMOS(LEVEL=2 TOX=50N NSUB=10E15 VTO=0.93 XJ=0.45U LD=0.24U
+              JS=1.24E-4 PB=0.80 UO=381 UCRIT=99E4 UEXP=0.001
+              UTRA=0 LAMBDA=0.025 CGBO=4.0E-10 TPG=1
+              CGDO=5.2E-10 CGSO=5.2E-10 CJ=3.2E-4 CJSW=9.0E-10
+              VMAX=5.5E4 NEFF=1.0E-2 RSH=25 DELTA=1.47 NFS=3.73E11)
.MODEL N3 NMOS(LEVEL=2 TOX=55N NSUB=5E15 VTO=0.5 XJ=0.6U LD=0.4U
+              JS=1.24E-4 PB=0.80 UO=1053 UCRIT=3.97E4 UEXP=0.08
+              UTRA=0.25 GAMMA=0.9 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
+              VMAX=5E4 NEFF=3 RSH=10)
.MODEL P1 PMOS(LEVEL=2 TOX=65N NSUB=5E15 VTO=-1.1 XJ=0.35U LD=0.25U
+              JS=7.75E-5 PB=0.88 UO=210 UCRIT=4.14E4 UEXP=0.16
+              UTRA=0.25 GAMMA=0.6 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
+              VMAX=5E4 NEFF=3 RSH=100)
.MODEL P PMOS(LEVEL=2 TOX=50N NSUB=2.97E14 VTO=-0.844 XJ=0.0258U LD=0.512U
+              JS=7.75E-5 PB=0.88 UO=100 UCRIT=18500 UEXP=0.145
+              GAMMA=0.723 LAMBDA=0.0527 CGBO=4.0E-10 TPG=-1
+              CGDO=4.0E-10 CGSO=4.0E-10 CJ=2.0E-4 CJSW=4.0E-10
+              VMAX=10E4 NEFF=.01 RSH=95 DELTA=2.19 NFS=1.62E12)
.MODEL P3 PMOS(LEVEL=2 TOX=55N NSUB=.3E15 VTO=-0.5 XJ=0.6U LD=0.4U
+              JS=7.75E-4 PB=0.88 UO=421 UCRIT=4.14E4 UEXP=0.16
+              UTRA=0.25 GAMMA=0.4 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
+              VMAX=5E4 NEFF=3 RSH=50)
* NODES 4-5 CORRESPOND TO ALUINA#30_H (SEE GATE AT 3544,2111)
* NODES 6-7 CORRESPOND TO ALUINA#30_L (SEE GATE AT 3537,2121)
* NODES 8-9 CORRESPOND TO COMPINA#30_H (SEE GATE AT 3510,2128)
* NODES 10-11 CORRESPOND TO BUSA#30_H (SEE GATE AT 3209,2884)
* NODES 12-13 CORRESPOND TO IN#6 (SEE SOURCE AT 4575,2534)
* NODES 14-15 CORRESPOND TO BUSA#31_H (SEE GATE AT 3194,2884)
* NODES 16-17 CORRESPOND TO COMPINA#31_H (SEE GATE AT 3508,2218)
* NODES 18-19 CORRESPOND TO PHI1_L#7 (SEE GATE AT 3505,2255)
* NODES 20-21 CORRESPOND TO PHI_H (SEE GATE AT 2854,2740)
* NODES 22-23 CORRESPOND TO PHI_L (SEE GATE AT 2859,2723)
* NODES 24-25 CORRESPOND TO PHASE1 (SEE GATE AT 4138,3532)

R1 22 23 23639
C1 23 0 0.569PF
R2 12 13 1793
C2 13 0 0.126PF
R3 16 17 1405
C3 17 0 0.196PF
R4 14 15 12893
C4 15 0 0.804PF
R5 4 5 4039
C5 5 0 0.179PF
R6 6 7 1172
C6 7 0 0.056PF
R7 10 11 13393
C7 11 0 0.782PF
R8 8 9 1405
C8 9 0 0.164PF
R9 18 19 16798
C9 19 0 1.049PF
R10 20 21 52265
C10 21 0 2.477PF
M1 1 7 4 1 P L=3.0U W=11.0U
M2 0 7 4 0 N L=3.0U W=8.0U
M3 0 9 6 0 N L=3.0U W=8.0U
M4 6 9 1 1 P L=3.0U W=11.0U
M5 11 1 8 0 N L=3.0U W=4.0U
M6 10 1 13 0 N L=3.0U W=4.0U
```

```
 M7 15 1 12 0 N L=3.0U W=4.0U
 M8 17 1 14 0 N L=3.0U W=4.0U
 M9 16 19 1 1 P L=3.0U W=4.0U
 M10 0 21 18 0 N L=3.0U W=27.0U
 M11 1 21 18 1 P L=3.0U W=59.0U
 M12 1 23 20 1 P L=2.8U W=58.9U
 M13 0 23 20 0 N L=3.0U W=30.0U
 M14 22 25 0 0 N L=3.0U W=27.0U
 M15 22 25 1 1 P L=3.0U W=59.0U

 * INITIAL CONDITIONS:
 .IC V(22)=5.000000
 .IC V(23)=5.000000
 .IC V(12)=0.000000
 .IC V(13)=0.000000
 .IC V(16)=0.000000
 .IC V(17)=0.000000
 .IC V(14)=0.000000
 .IC V(15)=0.000000
 .IC V(4)=0.000000
 .IC V(5)=0.000000
 .IC V(6)=5.000000
 .IC V(7)=5.000000
 .IC V(10)=0.000000
 .IC V(11)=0.000000
 .IC V(8)=0.000000
 .IC V(9)=0.000000
 .IC V(18)=5.000000
 .IC V(19)=5.000000
 .IC V(20)=0.000000
 .IC V(21)=0.000000

 VDD 1 0 5.0
 VIN 25 0 PULSE(0 5 0NS 0NS 0NS)
 .TRAN 5.00NS 1000NS
 .PLOT TRAN V(5) (0,5)
 .END
1****************05/06/85 ********************** SPICE 2G.6    3/15/83 ******************

0* 3U P WELL CMOS MODELS ---- BK
0****                    MOSFET MODEL PARAMETERS                          TEMPERATURE =   27

0*********************************************************************************************


                  N1          N          N3          P1          P          P3
0TYPE           NMOS        NMOS       NMOS        PMOS        PMOS       PMOS
0LEVEL         2.000       2.000      2.000       2.000       2.000      2.000
0VTO           1.100       0.930      0.500      -1.100      -0.844     -0.500
0KP         2.79d-05    2.63d-05   6.61d-05    1.12d-05    6.91d-06   2.64d-05
0GAMMA         1.000       0.834      0.900       0.600       0.723      0.400
0PHI           0.716       0.695      0.660       0.660       0.514      0.514
0LAMBDA     2.00d-02    2.50d-02   2.00d-02    2.00d-02    5.27d-02   2.00d-02
0PB            0.800       0.800      0.800       0.880       0.880      0.880
0CGSO       5.70d-10    5.20d-10   5.70d-10    5.70d-10    4.00d-10   5.70d-10
0CGDO       5.70d-10    5.20d-10   5.70d-10    5.70d-10    4.00d-10   5.70d-10
0CGBO       5.70d-10    4.00d-10   5.70d-10    5.70d-10    4.00d-10   5.70d-10
0RSH          30.000      25.000     10.000      100.000      95.000     50.000
0CJ         6.00d-04    3.20d-04   6.00d-04    4.10d-04    2.00d-04   4.10d-04
0CJSW       5.64d-10    9.00d-10   5.64d-10    3.85d-10    4.00d-10   3.85d-10
0JS         1.24d-04    1.24d-04   1.24d-04    7.75d-05    7.75d-05   7.75d-04
0TOX        6.50d-08    5.00d-08   5.50d-08    6.50d-08    5.00d-08   5.50d-08
0NSUB       1.50d+16    1.00d+16   5.00d+15    5.00d+15    2.97d+14   3.00d+14
0NFS        0.  d+00    3.73d+11   0.  d+00    0.  d+00    1.62d+12   0.  d+00
0TPG           1.000       1.000      1.000       1.000      -1.000      1.000
0XJ         3.50d-07    4.50d-07   6.00d-07    3.50d-07    2.58d-08   6.00d-07
0LD         2.50d-07    2.40d-07   4.00d-07    2.50d-07    5.12d-07   4.00d-07
0UO           526.000     381.000   1053.000     210.000     100.000    421.000
0UCRIT      3.97d+04    9.90d+05   3.97d+04    4.14d+04    1.85d+04   4.14d+04
0UEXP          0.080       0.001      0.080       0.160       0.145      0.160
0UTRA          0.250       0.         0.250       0.250       0.         0.250
0VMAX       5.00d+04    5.50d+04   5.00d+04    5.00d+04    1.00d+05   5.00d+04
0NEFF          3.000       0.010      3.000       3.000       0.010      3.000
0DELTA         0.          1.470      0.          0.          2.190      0.
1****************05/06/85 ********************** SPICE 2G.6    3/15/83 ******************
```

0* 3U P WELL CMOS MODELS ---- BK
0**** INITIAL TRANSIENT SOLUTION TEMPERATURE = 27

0**********

NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE
(1)	5.0000	(4)	0.0000	(5)	0.0000	(6)	5.0000	(7)	5.0000
(10)	-0.0000	(11)	-0.0000	(12)	0.0000	(13)	0.0000	(14)	0.0000
(17)	0.0000	(18)	5.0000	(19)	5.0000	(20)	0.0000	(21)	0.0000
(25)	0.								

VOLTAGE SOURCE CURRENTS

NAME	CURRENT
VDD	-3.005d-09
VIN	0. d+00

TOTAL POWER DISSIPATION 1.50d-08 WATTS
1**********05/06/85 ********** SPICE 2G.6 3/15/83 **********

0* 3U P WELL CMOS MODELS ---- BK
0**** OPERATING POINT INFORMATION TEMPERATURE = 27

0**********

0
0**** MOSFETS

0	M1	M2	M3	M4	M5	M6	M7	M8	
0MODEL	P	N	N	P	N	N	N	N	P
ID	3.84e-10	1.25e-24	-2.97e-14	-2.59e-24	-1.62e-27	-1.62e-27	-1.62e-27	3.40e-26	-1.
VGS	5.000	5.000	-5.000	-5.000	5.000	5.000	5.000	5.000	
VDS	5.000	-0.000	-5.000	-0.000	-0.000	-0.000	-0.000	0.000	
VBS	5.000	-0.000	-5.000	0.	-0.000	-0.000	-0.000	-0.000	

0	M13	M14	M15
0MODEL	N	N	P
ID	2.72e-23	2.04e-12	4.48e-24
VGS	5.000	0.	-5.000
VDS	-0.000	5.000	-0.000
VBS	-0.000	0.	0.

1**********05/06/85 ********** SPICE 2G.6 3/15/83 **********

0* 3U P WELL CMOS MODELS ---- BK
0**** TRANSIENT ANALYSIS TEMPERATURE = 27

0**********

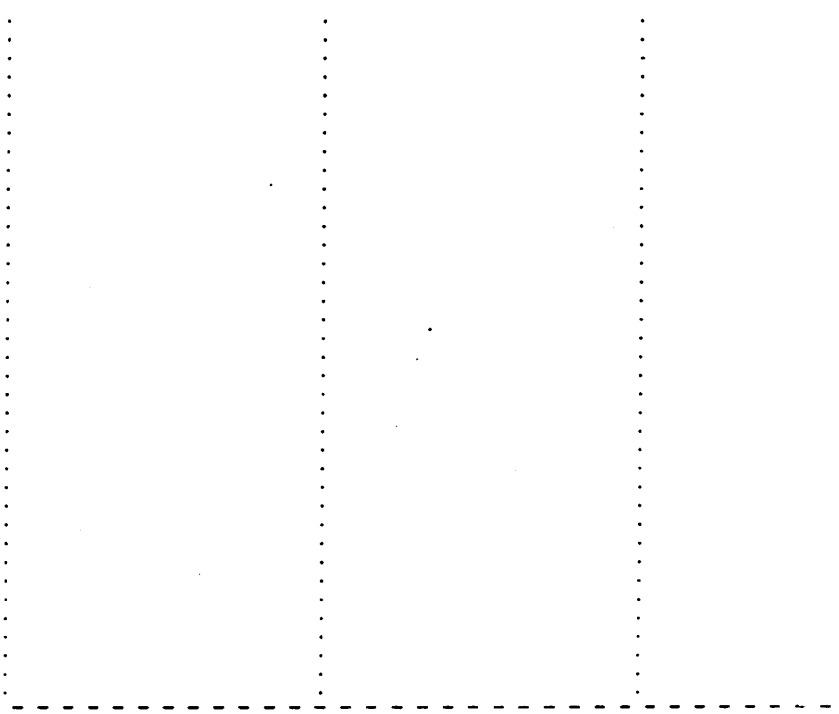
X

TIME	V(5)	0. d+00	1.250d+00	2.500d+00
0. d+00	9.674d-14	*	.	.
5.000d-09	-4.276d-06	*	.	.
1.000d-08	-1.327d-06	*	.	.
1.500d-08	-4.418d-08	*	.	.
2.000d-08	2.965d-07	*	.	.
2.500d-08	3.224d-07	*	.	.
3.000d-08	1.915d-07	*	.	.
3.500d-08	3.547d-07	*	.	.

4.000d-08	5.033d-07	•
4.500d-08	6.619d-07	•
5.000d-08	7.049d-07	•
5.500d-08	7.182d-07	•
6.000d-08	7.063d-07	•
6.500d-08	7.078d-07	•
7.000d-08	7.382d-07	•
7.500d-08	8.014d-07	•
8.000d-08	8.645d-07	•
8.500d-08	8.962d-07	•
9.000d-08	9.251d-07	•
9.500d-08	9.539d-07	•
1.000d-07	9.828d-07	•
1.050d-07	9.965d-07	•
1.100d-07	1.009d-06	•
1.150d-07	1.021d-06	•
1.200d-07	1.034d-06	•
1.250d-07	1.297d-06	•
1.300d-07	1.584d-06	•
1.350d-07	1.870d-06	•
1.400d-07	2.156d-06	•
1.450d-07	2.881d-06	•
1.500d-07	3.646d-06	•
1.550d-07	4.410d-06	•
1.600d-07	4.477d-06	•
1.650d-07	4.283d-06	•
1.700d-07	4.373d-06	•
1.750d-07	4.686d-06	•
1.800d-07	4.518d-06	•
1.850d-07	3.782d-06	•
1.900d-07	3.045d-06	•
1.950d-07	1.207d-06	•
2.000d-07	-1.059d-05	•
2.050d-07	-2.240d-05	•
2.100d-07	-3.420d-05	•
2.150d-07	-7.229d-05	•
2.200d-07	-1.399d-04	•
2.250d-07	-2.075d-04	•
2.300d-07	-2.751d-04	•
2.350d-07	-3.147d-04	•
2.400d-07	-3.496d-04	•
2.450d-07	-3.845d-04	•
2.500d-07	-4.197d-04	•
2.550d-07	-4.552d-04	•
2.600d-07	-4.906d-04	•
2.650d-07	-5.261d-04	•
2.700d-07	-5.532d-04	•
2.750d-07	-5.740d-04	•
2.800d-07	-5.948d-04	•
2.850d-07	-6.156d-04	•
2.900d-07	-1.634d-04	•
2.950d-07	6.491d-04	•
3.000d-07	1.462d-03	•
3.050d-07	2.274d-03	•
3.100d-07	9.995d-03	•
3.150d-07	2.298d-02	•
3.200d-07	3.596d-02	.•
3.250d-07	4.894d-02	.•
3.300d-07	2.630d-01	. •
3.350d-07	6.060d-01	. •
3.400d-07	9.489d-01	. •
3.450d-07	1.292d+00	. •
3.500d-07	1.892d+00	. •
3.550d-07	2.657d+00	. •
3.600d-07	3.423d+00	.
3.650d-07	4.188d+00	.
3.700d-07	4.647d+00	.
3.750d-07	4.911d+00	.
3.800d-07	5.174d+00	.
3.850d-07	5.082d+00	.
3.900d-07	4.984d+00	.
3.950d-07	4.937d+00	.
4.000d-07	4.983d+00	.
4.050d-07	5.028d+00	.
4.100d-07	5.012d+00	.
4.150d-07	4.995d+00	.
4.200d-07	4.991d+00	.
4.250d-07	4.996d+00	.

4.300d-07	5.000d+00
4.350d-07	5.005d+00
4.400d-07	5.004d+00
4.450d-07	5.002d+00
4.500d-07	4.999d+00
4.550d-07	4.997d+00
4.600d-07	4.997d+00
4.650d-07	4.999d+00
4.700d-07	5.000d+00
4.750d-07	5.002d+00
4.800d-07	5.001d+00
4.850d-07	5.000d+00
4.900d-07	5.000d+00
4.950d-07	4.999d+00
5.000d-07	4.999d+00
5.050d-07	5.000d+00
5.100d-07	5.000d+00
5.150d-07	5.000d+00
5.200d-07	5.000d+00
5.250d-07	5.000d+00
5.300d-07	5.000d+00
5.350d-07	5.000d+00
5.400d-07	5.000d+00
5.450d-07	5.000d+00
5.500d-07	5.000d+00
5.550d-07	5.000d+00
5.600d-07	5.000d+00
5.650d-07	5.000d+00
5.700d-07	5.000d+00
5.750d-07	5.000d+00
5.800d-07	5.000d+00
5.850d-07	5.000d+00
5.900d-07	5.000d+00
5.950d-07	5.000d+00
6.000d-07	5.000d+00
6.050d-07	5.000d+00
6.100d-07	5.000d+00
6.150d-07	5.000d+00
6.200d-07	5.000d+00
6.250d-07	5.000d+00
6.300d-07	5.000d+00
6.350d-07	5.000d+00
6.400d-07	5.000d+00
6.450d-07	5.000d+00
6.500d-07	5.000d+00
6.550d-07	5.000d+00
6.600d-07	5.000d+00
6.650d-07	5.000d+00
6.700d-07	5.000d+00
6.750d-07	5.000d+00
6.800d-07	5.000d+00
6.850d-07	5.000d+00
6.900d-07	5.000d+00
6.950d-07	5.000d+00
7.000d-07	5.000d+00
7.050d-07	5.000d+00
7.100d-07	5.000d+00
7.150d-07	5.000d+00
7.200d-07	5.000d+00
7.250d-07	5.000d+00
7.300d-07	5.000d+00
7.350d-07	5.000d+00
7.400d-07	5.000d+00
7.450d-07	5.000d+00
7.500d-07	5.000d+00
7.550d-07	5.000d+00
7.600d-07	5.000d+00
7.650d-07	5.000d+00
7.700d-07	5.000d+00
7.750d-07	5.000d+00
7.800d-07	5.000d+00
7.850d-07	5.000d+00
7.900d-07	5.000d+00
7.950d-07	5.000d+00
8.000d-07	5.000d+00
8.050d-07	5.000d+00
8.100d-07	5.000d+00
8.150d-07	5.000d+00

8.200d-07	5.000d+00	.
8.250d-07	5.000d+00	.
8.300d-07	5.000d+00	.
8.350d-07	5.000d+00	.
8.400d-07	5.000d+00	.
8.450d-07	5.000d+00	.
8.500d-07	5.000d+00	.
8.550d-07	5.000d+00	.
8.600d-07	5.000d+00	.
8.650d-07	5.000d+00	.
8.700d-07	5.000d+00	.
8.750d-07	5.000d+00	.
8.800d-07	5.000d+00	.
8.850d-07	5.000d+00	.
8.900d-07	5.000d+00	.
8.950d-07	5.000d+00	.
9.000d-07	5.000d+00	.
9.050d-07	5.000d+00	.
9.100d-07	5.000d+00	.
9.150d-07	5.000d+00	.
9.200d-07	5.000d+00	.
9.250d-07	5.000d+00	.
9.300d-07	5.000d+00	.
9.350d-07	5.000d+00	.
9.400d-07	5.000d+00	.
9.450d-07	5.000d+00	.
9.500d-07	5.000d+00	.
9.550d-07	5.000d+00	.
9.600d-07	5.000d+00	.
9.650d-07	5.000d+00	.
9.700d-07	5.000d+00	.
9.750d-07	5.000d+00	.
9.800d-07	5.000d+00	.
9.850d-07	5.000d+00	.
9.900d-07	5.000d+00	.
9.950d-07	5.000d+00	.
1.000d-06	5.000d+00	.



Y
0

JOB CONCLUDED
0 TOTAL JOB TIME 85.88

1*******05/10/85 ******** SPICE 2G.6 3/15/83 ********20:34:15*****

0* 3U P WELL CMOS MODELS ----- BK

0**** INPUT LISTING TEMPERATURE = 27.000 DEG C

0**********

```
 .MODEL N1 NMOS(LEVEL=2 TOX=65N NSUB=15E15 VTO=1.1 XJ=0.35U LD=0.25U
+               JS=1.24E-4 PB=0.80 UO=526 UCRIT=3.97E4 UEXP=0.08
+               UTRA=0.25 GAMMA=1 LAMBDA=0.02 CGBO=5.7E-10
+               CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
+               VMAX=5E4 NEFF=3 RSH=30)
 .MODEL N NMOS(LEVEL=2 TOX=50N NSUB=10E15 VTO=0.93 XJ=0.45U LD=0.24U
+               JS=1.24E-4 PB=0.80 UO=381 UCRIT=99E4 UEXP=0.001
+               UTRA=0 LAMBDA=0.025 CGBO=4.0E-10 TPG=1
+               CGDO=5.2E-10 CGSO=5.2E-10 CJ=3.2E-4 CJSW=9.0E-10
+               VMAX=5.5E4 NEFF=1.0E-2 RSH=25 DELTA=1.47 NFS=3.73E11)
 .MODEL N3 NMOS(LEVEL=2 TOX=55N NSUB=5E15 VTO=0.5 XJ=0.6U LD=0.4U
+               JS=1.24E-4 PB=0.80 UO=1053 UCRIT=3.97E4 UEXP=0.08
+               UTRA=0.25 GAMMA=0.9 LAMBDA=0.02 CGBO=5.7E-10
+               CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
+               VMAX=5E4 NEFF=3 RSH=10)
 .MODEL P1 PMOS(LEVEL=2 TOX=65N NSUB=5E15 VTO=-1.1 XJ=0.35U LD=0.25U
+               JS=7.75E-5 PB=0.88 UO=210 UCRIT=4.14E4 UEXP=0.16
+               UTRA=0.25 GAMMA=0.6 LAMBDA=0.02 CGBO=5.7E-10
+               CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
+               VMAX=5E4 NEFF=3 RSH=100)
 .MODEL P PMOS(LEVEL=2 TOX=50N NSUB=2.97E14 VTO=-0.844 XJ=0.0258U LD=0.512U
+               JS=7.75E-5 PB=0.88 UO=100 UCRIT=18500 UEXP=0.145
+               GAMMA=0.723 LAMBDA=0.0527 CGBO=4.0E-10 TPG=-1
+               CGDO=4.0E-10 CGSO=4.0E-10 CJ=2.0E-4 CJSW=4.0E-10
+               VMAX=10E4 NEFF=.01 RSH=95 DELTA=2.19 NFS=1.62E12)
 .MODEL P3 PMOS(LEVEL=2 TOX=55N NSUB=.3E15 VTO=-0.5 XJ=0.6U LD=0.4U
+               JS=7.75E-4 PB=0.88 UO=421 UCRIT=4.14E4 UEXP=0.16
+               UTRA=0.25 GAMMA=0.4 LAMBDA=0.02 CGBO=5.7E-10
+               CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
+               VMAX=5E4 NEFF=3 RSH=50)
* NODES 4-5 CORRESPOND TO AIN0 (SEE SOURCE AT 3276,1760)
* NODES 6-7 CORRESPOND TO 73686 (SEE SOURCE AT 3276,2152)
* NODES 8-9 CORRESPOND TO 69988 (SEE SOURCE AT 3276,2057)
* NODES 10-11 CORRESPOND TO 69678 (SEE DRAIN AT 3276,2057)
* NODES 12-13 CORRESPOND TO 65696 (SEE SOURCE AT 3276,1940)
* NODES 14-15 CORRESPOND TO 65319 (SEE SOURCE AT 3276,1918)
* NODES 16-17 CORRESPOND TO 61120 (SEE SOURCE AT 3276,1823)
* NODES 18-19 CORRESPOND TO 60716 (SEE DRAIN AT 3276,1823)
* NODES 20-21 CORRESPOND TO 56520 (SEE SOURCE AT 3276,1706)
* NODES 22-23 CORRESPOND TO 56113 (SEE SOURCE AT 3276,1684)
* NODES 24-25 CORRESPOND TO 51954 (SEE SOURCE AT 3276,1589)
* NODES 26-27 CORRESPOND TO 51576 (SEE DRAIN AT 3276,1589)
* NODES 28-29 CORRESPOND TO 47316 (SEE SOURCE AT 3276,1472)
* NODES 30-31 CORRESPOND TO 46939 (SEE DRAIN AT 3276,1472)
* NODES 32-33 CORRESPOND TO 42678 (SEE SOURCE AT 3276,1355)
* NODES 34-35 CORRESPOND TO 42305 (SEE SOURCE AT 3276,1333)
* NODES 36-37 CORRESPOND TO 35510 (SEE SOURCE AT 3276,1163)
* NODES 38-39 CORRESPOND TO 35137 (SEE DRAIN AT 3276,1163)
* NODES 40-41 CORRESPOND TO 30910 (SEE SOURCE AT 3276,1046)
* NODES 42-43 CORRESPOND TO 30543 (SEE DRAIN AT 3276,1046)
* NODES 44-45 CORRESPOND TO 26349 (SEE SOURCE AT 3276,929)
* NODES 46-47 CORRESPOND TO 25939 (SEE DRAIN AT 3276,929)
* NODES 48-49 CORRESPOND TO 21742 (SEE SOURCE AT 3276,812)
* NODES 50-51 CORRESPOND TO 21376 (SEE DRAIN AT 3276,812)
* NODES 52-53 CORRESPOND TO 17178 (SEE SOURCE AT 3276,695)
* NODES 54-55 CORRESPOND TO 16802 (SEE DRAIN AT 3276,695)
* NODES 56-57 CORRESPOND TO 12720 (SEE SOURCE AT 3276,578)
* NODES 58-59 CORRESPOND TO 12338 (SEE DRAIN AT 3276,578)
* NODES 60-61 CORRESPOND TO 8382 (SEE SOURCE AT 3276,461)
* NODES 62-63 CORRESPOND TO 7994 (SEE DRAIN AT 3276,461)
* NODES 64-65 CORRESPOND TO 4102 (SEE SOURCE AT 3276,344)
* NODES 66-67 CORRESPOND TO 3766 (SEE DRAIN AT 3276,344)
* NODES 68-69 CORRESPOND TO BUSA#30_H (SEE GATE AT 3209,2884)
* NODES 70-71 CORRESPOND TO IN#6 (SEE SOURCE AT 4575,2534)
* NODES 72-73 CORRESPOND TO BUSA#31_H (SEE GATE AT 3194,2884)
* NODES 74-75 CORRESPOND TO 74115 (SEE GATE AT 3186,2185)
* NODES 76-77 CORRESPOND TO 74211 (SEE GATE AT 3145,2191)
* NODES 78-79 CORRESPOND TO BUSD_H#31 (SEE GATE AT 3019,2186)
```

```
* NODES 80-81 CORRESPOND TO BUFIN_H#27 (SEE GATE AT 3131,2255)
* NODES 82-83 CORRESPOND TO BUSDTOINA_H#2 (SEE GATE AT 3135,2256)
* NODES 84-85 CORRESPOND TO 76982 (SEE DRAIN AT 3154,2237)
* NODES 86-87 CORRESPOND TO PHI2_H#4 (SEE GATE AT 3171,2652)
* NODES 88-89 CORRESPOND TO PHI2_L#2 (SEE GATE AT 3166,2669)
* NODES 90-91 CORRESPOND TO PHI2 (SEE GATE AT 4260,3550)

C1 89 0 0.365PF
C2 71 0 0.104PF
C3 29 0 0.058PF
C4 75 0 0.054PF
C5 73 0 0.753PF
C6 41 0 0.058PF
C7 21 0 0.058PF
C8 69 0 0.737PF
C9 53 0 0.058PF
C10 61 0 0.058PF
C11 33 0 0.058PF
C12 83 0 0.059PF
C13 45 0 0.058PF
C14 25 0 0.058PF
C15 81 0 0.704PF
C16 37 0 0.076PF
C17 13 0 0.058PF
C18 9 0 0.058PF
C19 5 0 1.322PF
C20 65 0 0.058PF
C21 57 0 0.058PF
C22 77 0 0.078PF
C23 17 0 0.058PF
C24 49 0 0.058PF
C25 87 0 3.006PF
M1 0 69 5 0 N L=3.0U W=4.0U
M2 7 73 1 1 P L=3.0U W=8.0U
M3 9 69 7 1 P L=3.0U W=8.0U
M4 11 0 9 1 P L=3.0U W=8.0U
M5 13 0 11 1 P L=3.0U W=8.0U
M6 15 0 13 1 P L=3.0U W=8.0U
M7 17 0 15 1 P L=3.0U W=8.0U
M8 19 0 17 1 P L=3.0U W=8.0U
M9 21 0 19 1 P L=3.0U W=8.0U
M10 23 0 21 1 P L=3.0U W=8.0U
M11 25 0 23 1 P L=3.0U W=8.0U
M12 27 0 25 1 P L=3.0U W=8.0U
M13 29 0 27 1 P L=3.0U W=8.0U
M14 31 0 29 1 P L=3.0U W=8.0U
M15 33 0 31 1 P L=3.0U W=8.0U
M16 35 0 33 1 P L=3.0U W=8.0U
M17 37 0 35 1 P L=3.0U W=8.0U
M18 39 0 37 1 P L=3.0U W=8.0U
M19 41 0 39 1 P L=3.0U W=8.0U
M20 43 0 41 1 P L=3.0U W=8.0U
M21 45 0 43 1 P L=3.0U W=8.0U
M22 47 0 45 1 P L=3.0U W=8.0U
M23 49 0 47 1 P L=3.0U W=8.0U
M24 51 0 49 1 P L=3.0U W=8.0U
M25 53 0 51 1 P L=3.0U W=8.0U
M26 55 0 53 1 P L=3.0U W=8.0U
M27 57 0 55 1 P L=3.0U W=8.0U
M28 59 0 57 1 P L=3.0U W=8.0U
M29 61 0 59 1 P L=3.0U W=8.0U
M30 63 0 61 1 P L=3.0U W=8.0U
M31 65 0 63 1 P L=3.0U W=8.0U
M32 67 0 65 1 P L=3.0U W=8.0U
M33 5 0 67 1 P L=3.0U W=8.0U
M34 69 1 71 0 N L=3.0U W=4.0U
M35 73 1 71 0 N L=3.0U W=4.0U
M36 73 75 1 1 P L=3.0U W=23.0U
M37 73 75 0 0 N L=3.0U W=11.0U
M38 0 77 75 0 N L=3.0U W=11.0U
M39 1 77 75 1 P L=3.0U W=11.0U
M40 1 81 77 0 N L=3.0U W=4.0U
M41 1 83 81 1 P L=3.0U W=59.0U
M42 0 83 81 0 N L=3.0U W=27.0U
M43 85 1 83 0 N L=3.0U W=8.0U
M44 0 87 85 0 N L=3.0U W=8.0U
M45 1 87 83 1 P L=3.0U W=8.0U
M46 1 89 87 1 P L=2.8U W=58.9U
```

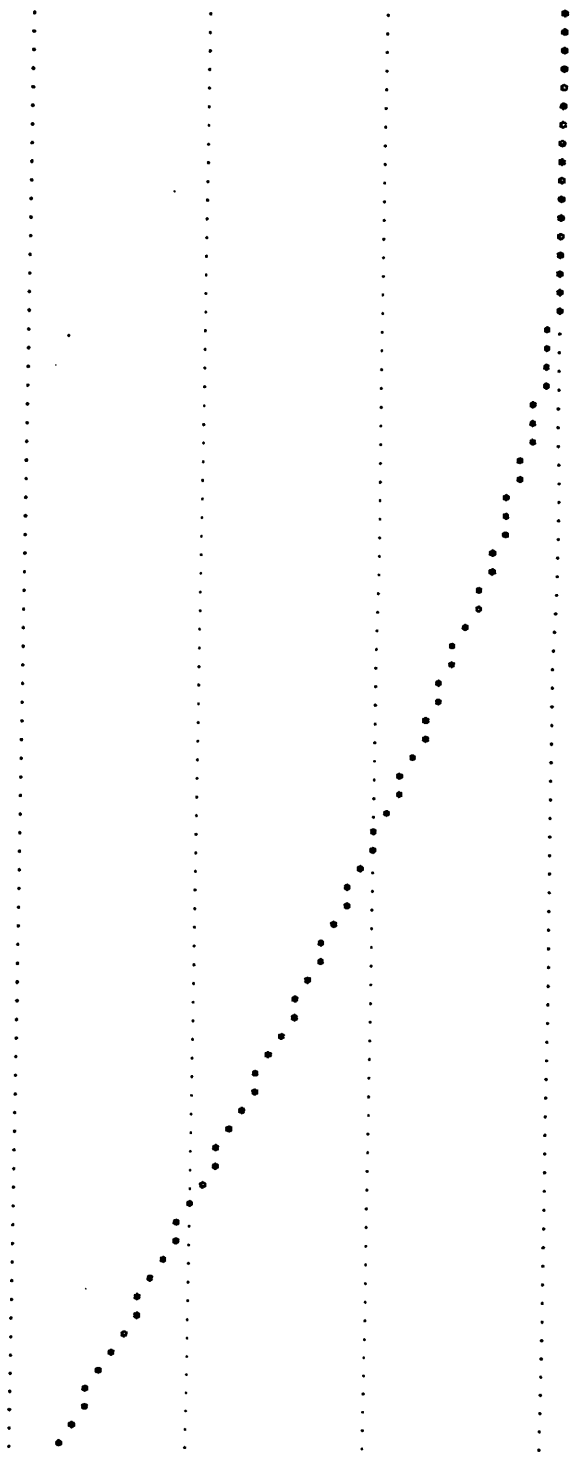
```
 M47 0 89 87 0 N L=3.0U W=30.0U
 M48 89 91 0 0 N L=3.0U W=27.0U
 M49 89 91 1 1 P L=3.0U W=59.0U

 * INITIAL CONDITIONS:
 .IC V(29)=5.000000
 .IC V(71)=0.000000
 .IC V(89)=5.000000
 .IC V(35)=5.000000
 .IC V(11)=5.000000
 .IC V(41)=5.000000
 .IC V(73)=0.000000
 .IC V(59)=5.000000
 .IC V(75)=5.000000
 .IC V(27)=5.000000
 .IC V(7)=5.000000
 .IC V(15)=5.000000
 .IC V(21)=5.000000
 .IC V(47)=5.000000
 .IC V(39)=5.000000
 .IC V(53)=5.000000
 .IC V(69)=0.000000
 .IC V(33)=5.000000
 .IC V(61)=5.000000
 .IC V(67)=5.000000
 .IC V(83)=5.000000
 .IC V(31)=5.000000
 .IC V(19)=5.000000
 .IC V(85)=5.000000
 .IC V(25)=5.000000
 .IC V(45)=5.000000
 .IC V(5)=5.000000
 .IC V(37)=5.000000
 .IC V(13)=5.000000
 .IC V(63)=5.000000
 .IC V(81)=0.000000
 .IC V(9)=5.000000
 .IC V(57)=5.000000
 .IC V(51)=5.000000
 .IC V(65)=5.000000
 .IC V(17)=5.000000
 .IC V(77)=0.000000
 .IC V(49)=5.000000
 .IC V(55)=5.000000
 .IC V(43)=5.000000
 .IC V(23)=5.000000
 .IC V(87)=0.000000

 .WIDTH OUT=80
 VDD 1 0 5.0
 VIN 91 0 PULSE(0 5 0NS 0NS 0NS)
 .TRAN 2.00NS 400NS
 .PLOT TRAN V(5) (0,5)
 .END
1********05/10/85 ******** SPICE 2G.6    3/15/83 ********20:34:15*****

0* 3U P WELL CMOS MODELS ---- BK

0****      MOSFET MODEL PARAMETERS          TEMPERATURE =   27.000 DEG C

0**********************************************************************
```

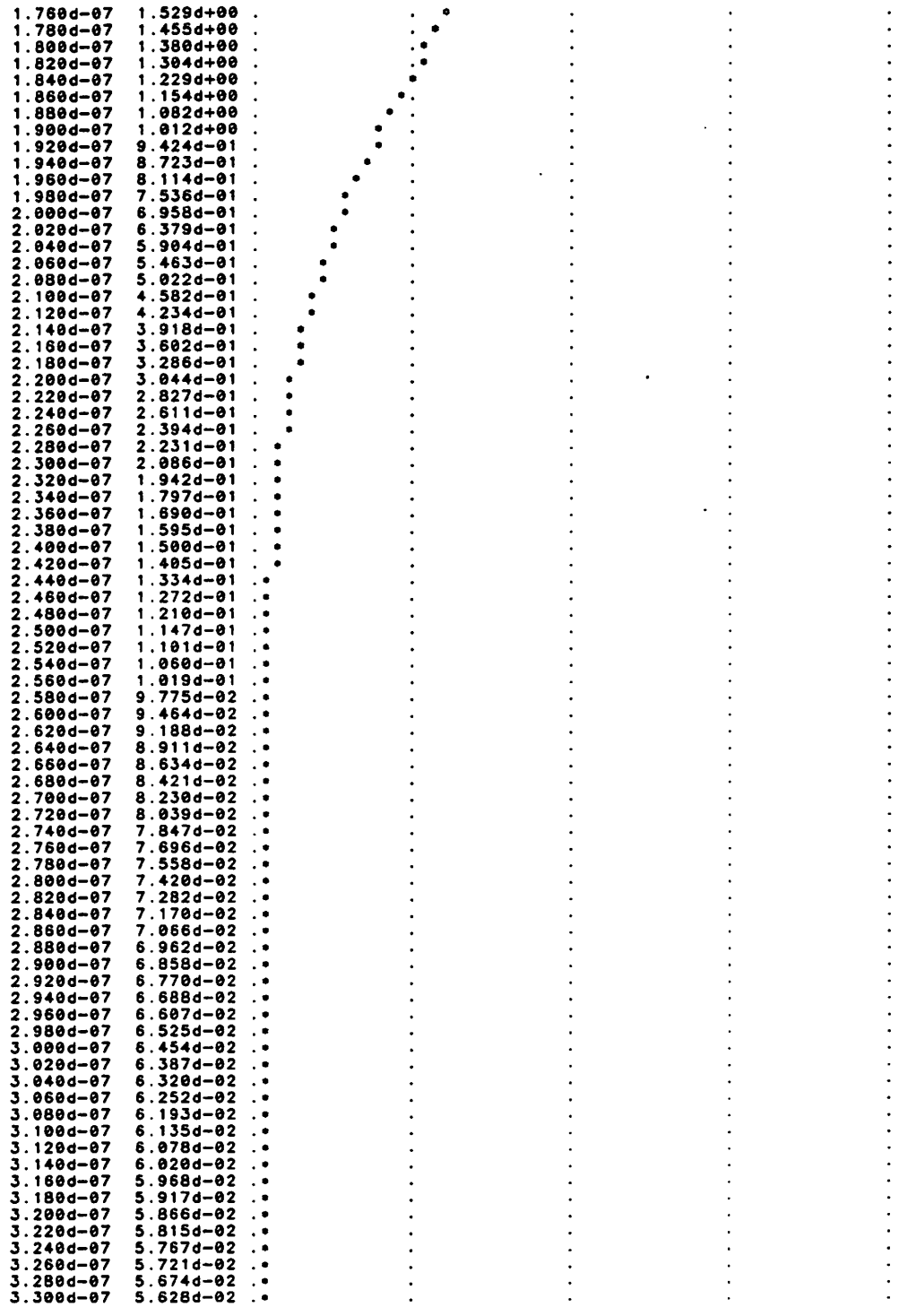
	N1	N	N3	P1	P	P3
0TYPE	NMOS	NMOS	NMOS	PMOS	PMOS	PMOS
0LEVEL	2.000	2.000	2.000	2.000	2.000	2.000
0VTO	1.100	0.930	0.500	-1.100	-0.844	-0.500
0KP	2.79d-05	2.63d-05	6.61d-05	1.12d-05	6.91d-06	2.64d-05
0GAMMA	1.000	0.834	0.900	0.600	0.723	0.400
0PHI	0.716	0.695	0.660	0.660	0.514	0.514
0LAMBDA	2.00d-02	2.50d-02	2.00d-02	2.00d-02	5.27d-02	2.00d-02
0PB	0.800	0.800	0.800	0.880	0.880	0.880
0CGSO	5.70d-10	5.20d-10	5.70d-10	5.70d-10	4.00d-10	5.70d-10
0CGDO	5.70d-10	5.20d-10	5.70d-10	5.70d-10	4.00d-10	5.70d-10
0CGBO	5.70d-10	4.00d-10	5.70d-10	5.70d-10	4.00d-10	5.70d-10
0RSH	30.000	25.000	10.000	100.000	95.000	50.000

```
0CJ          6.00d-04  3.20d-04  6.00d-04  4.10d-04  2.00d-04  4.10d-04
0CJSW        5.64d-10  9.00d-10  5.64d-10  3.85d-10  4.00d-10  3.85d-10
0JS          1.24d-04  1.24d-04  1.24d-04  7.75d-05  7.75d-05  7.75d-04
0TOX         6.50d-08  5.00d-08  5.50d-08  6.50d-08  5.00d-08  5.50d-08
0NSUB        1.50d+16  1.00d+16  5.00d+15  5.00d+15  2.97d+14  3.00d+14
0NFS         0.  d+00  3.73d+11  0.  d+00  0.  d+00  1.62d+12  0.  d+00
0TPG            1.000     1.000     1.000     1.000    -1.000     1.000
0XJ          3.50d-07  4.50d-07  6.00d-07  3.50d-07  2.58d-08  6.00d-07
0LD          2.50d-07  2.40d-07  4.00d-07  2.50d-07  5.12d-07  4.00d-07
0UO           526.000   381.000  1053.000   210.000   100.000   421.000
0UCRIT       3.97d+04  9.90d+05  3.97d+04  4.14d+04  1.85d+04  4.14d+04
0UEXP           0.080     0.001     0.080     0.160     0.145     0.160
0UTRA           0.250     0.        0.250     0.250     0.        0.250
0VMAX        5.00d+04  5.50d+04  5.00d+04  5.00d+04  1.00d+05  5.00d+04
0NEFF           3.000     0.010     3.000     3.000     0.010     3.000
0DELTA          0.        1.470     0.        0.        2.190     0.
1*******05/10/85 ********  SPICE 2G.6    3/15/83 ********20:34:15*****

0* 3U P WELL CMOS MODELS ---- BK

0****      INITIAL TRANSIENT SOLUTION       TEMPERATURE =   27.000 DEG C

0**********************************************************************

  NODE   VOLTAGE     NODE   VOLTAGE     NODE   VOLTAGE     NODE   VOLTAGE

(  1)    5.0000    (  5)    5.0000    (  7)    5.0000    (  9)    5.0000

( 11)    5.0000    ( 13)    5.0000    ( 15)    5.0000    ( 17)    5.0000

( 19)    5.0000    ( 21)    5.0000    ( 23)    5.0000    ( 25)    5.0000

( 27)    5.0000    ( 29)    5.0000    ( 31)    5.0000    ( 33)    5.0000

( 35)    5.0000    ( 37)    5.0000    ( 39)    5.0000    ( 41)    5.0000

( 43)    5.0000    ( 45)    5.0000    ( 47)    5.0000    ( 49)    5.0000

( 51)    5.0000    ( 53)    5.0000    ( 55)    5.0000    ( 57)    5.0000

( 59)    5.0000    ( 61)    5.0000    ( 63)    5.0000    ( 65)    5.0000

( 67)    5.0000    ( 69)   -0.0000    ( 71)    0.0000    ( 73)    0.0000

( 75)    5.0000    ( 77)    0.0000    ( 81)    0.0000    ( 83)    5.0000

( 85)    5.0000    ( 87)    0.0000    ( 89)    5.0000    ( 91)    0.


    VOLTAGE SOURCE CURRENTS

    NAME        CURRENT

    VDD       -5.523d-09

    VIN         0.   d+00

    TOTAL POWER DISSIPATION   2.76d-08  WATTS
1*******05/10/85 ********  SPICE 2G.6    3/15/83 ********20:34:15*****

0* 3U P WELL CMOS MODELS ---- BK

0****      OPERATING POINT INFORMATION      TEMPERATURE =   27.000 DEG C

0**********************************************************************

0
0**** MOSFETS
```

```
0                M1          M2          M3          M4          M5          M6          M7
0MODEL          N           P           P           P           P           P           P
 ID          -1.33e-14   2.34e-26    2.12e-26    2.09e-26    2.09e-26    2.09e-26    2.09e-26
 VGS            -5.000      -5.000      -5.000      -5.000      -5.000      -5.000      -5.000
 VDS            -5.000       0.000      -0.000      -0.000       0.000      0.           0.000
 VBS            -5.000      0.          -0.000      0.           0.000      0.          0.

0                M8          M9          M10         M11         M12         M13         M14
0MODEL          P           P           P           P           P           P           P
 ID           2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26
 VGS            -5.000      -5.000      -5.000      -5.000      -5.000      -5.000      -5.000
 VDS            -0.000      0.          0.          0.          0.          0.          0.
 VBS            -0.000      0.          0.          0.          0.          0.          0.

0                M15         M16         M17         M18         M19         M20         M21
0MODEL          P           P           P           P           P           P           P
 ID           2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26
 VGS            -5.000      -5.000      -5.000      -5.000      -5.000      -5.000      -5.000
 VDS            0.          0.          0.          0.          0.          0.          0.
 VBS            0.          0.          0.          0.          0.          0.          0.

0                M22         M23         M24         M25         M26         M27         M28
0MODEL          P           P           P           P           P           P           P
 ID           2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26    2.08e-26
 VGS            -5.000      -5.000      -5.000      -5.000      -5.000      -5.000      -5.000
 VDS            0.          0.          0.          0.          0.          0.          0.
 VBS            0.          0.          0.          0.          0.          0.          0.

0                M29         M30         M31         M32         M33         M34         M35
0MODEL          P           P           P           P           P           N           N
 ID           2.08e-26    2.08e-26    2.08e-26    2.06e-26   -2.64e-24   -1.31e-27    3.23e-22
 VGS            -5.000      -5.000      -5.000      -5.000      -5.000       5.000       5.000
 VDS            0.          0.          0.          0.          -0.000      -0.000       0.000
 VBS            0.          0.          0.          0.          0.          -0.000      -0.000

0                M36         M37         M38         M39         M40         M41         M42
0MODEL          P           N           N           P           N           P           N
 ID          -8.36e-10    3.21e-22   -4.22e-14    3.69e-26    1.95e-12    2.18e-09    2.18e-23
 VGS            -0.000       5.000      -5.000      -5.000       0.000       5.000       5.000
 VDS            -5.000       0.000      -5.000       0.000       5.000       5.000      -0.000
 VBS            0.          0.          -5.000       0.000      -0.000       5.000      -0.000

0                M43         M44         M45         M46         M47         M48         M49
0MODEL          N           N           P           P           N           N           P
 ID           1.93e-12   -2.95e-14   -5.49e-27    2.48e-09    2.72e-23    2.04e-12    4.48e-24
 VGS             0.000      -5.000      -5.000       5.000       5.000      0.          -5.000
 VDS            -0.000      -5.000       0.000       5.000      -0.000       5.000      -0.000
 VBS            -5.000      -5.000       0.000       5.000      -0.000      0.          0.
1*******05/10/85 ********   SPICE 2G.6     3/15/83 ********20:34:15*****

0* 3U P WELL CMOS MODELS ---- BK

0****       TRANSIENT ANALYSIS                TEMPERATURE =    27.000 DEG C

0***********************************************************************


X
     TIME        V(5)
                         0.    d+00      1.250d+00      2.500d+00      3.750d+00   5.000d+00
                               - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 0.    d+00  5.000d+00 .              .              .              .              *
 2.000d-09   5.000d+00 .              .              .              .              *
 4.000d-09   5.000d+00 .              .              .              .              *
 6.000d-09   5.000d+00 .              .              .              .              *
 8.000d-09   5.000d+00 .              .              .              .              *
 1.000d-08   5.000d+00 .              .              .              .              *
 1.200d-08   5.000d+00 .              .              .              .              *
 1.400d-08   5.000d+00 .              .              .              .              *
 1.600d-08   5.000d+00 .              .              .              .              *
 1.800d-08   5.000d+00 .              .              .              .              *
```

2.000d-08	5.000d+00
2.200d-08	5.000d+00
2.400d-08	5.000d+00
2.600d-08	5.000d+00
2.800d-08	5.000d+00
3.000d-08	5.000d+00
3.200d-08	5.000d+00
3.400d-08	5.001d+00
3.600d-08	5.001d+00
3.800d-08	5.001d+00
4.000d-08	5.001d+00
4.200d-08	4.999d+00
4.400d-08	4.997d+00
4.600d-08	4.991d+00
4.800d-08	4.985d+00
5.000d-08	4.973d+00
5.200d-08	4.961d+00
5.400d-08	4.944d+00
5.600d-08	4.924d+00
5.800d-08	4.904d+00
6.000d-08	4.878d+00
6.200d-08	4.849d+00
6.400d-08	4.821d+00
6.600d-08	4.793d+00
6.800d-08	4.758d+00
7.000d-08	4.721d+00
7.200d-08	4.684d+00
7.400d-08	4.647d+00
7.600d-08	4.605d+00
7.800d-08	4.562d+00
8.000d-08	4.518d+00
8.200d-08	4.475d+00
8.400d-08	4.427d+00
8.600d-08	4.379d+00
8.800d-08	4.330d+00
9.000d-08	4.281d+00
9.200d-08	4.229d+00
9.400d-08	4.176d+00
9.600d-08	4.123d+00
9.800d-08	4.071d+00
1.000d-07	4.015d+00
1.020d-07	3.959d+00
1.040d-07	3.903d+00
1.060d-07	3.846d+00
1.080d-07	3.788d+00
1.100d-07	3.729d+00
1.120d-07	3.670d+00
1.140d-07	3.611d+00
1.160d-07	3.551d+00
1.180d-07	3.489d+00
1.200d-07	3.428d+00
1.220d-07	3.367d+00
1.240d-07	3.304d+00
1.260d-07	3.241d+00
1.280d-07	3.178d+00
1.300d-07	3.115d+00
1.320d-07	3.050d+00
1.340d-07	2.985d+00
1.360d-07	2.920d+00
1.380d-07	2.855d+00
1.400d-07	2.789d+00
1.420d-07	2.722d+00
1.440d-07	2.656d+00
1.460d-07	2.589d+00
1.480d-07	2.521d+00
1.500d-07	2.453d+00
1.520d-07	2.385d+00
1.540d-07	2.317d+00
1.560d-07	2.247d+00
1.580d-07	2.177d+00
1.600d-07	2.107d+00
1.620d-07	2.037d+00
1.640d-07	1.966d+00
1.660d-07	1.894d+00
1.680d-07	1.822d+00
1.700d-07	1.750d+00
1.720d-07	1.677d+00
1.740d-07	1.603d+00




1.760d-07 1.529d+00
1.780d-07 1.455d+00
1.800d-07 1.380d+00
1.820d-07 1.304d+00
1.840d-07 1.229d+00
1.860d-07 1.154d+00
1.880d-07 1.082d+00
1.900d-07 1.012d+00
1.920d-07 9.424d-01
1.940d-07 8.723d-01
1.960d-07 8.114d-01
1.980d-07 7.536d-01
2.000d-07 6.958d-01
2.020d-07 6.379d-01
2.040d-07 5.904d-01
2.060d-07 5.463d-01
2.080d-07 5.022d-01
2.100d-07 4.582d-01
2.120d-07 4.234d-01
2.140d-07 3.918d-01
2.160d-07 3.602d-01
2.180d-07 3.286d-01
2.200d-07 3.044d-01
2.220d-07 2.827d-01
2.240d-07 2.611d-01
2.260d-07 2.394d-01
2.280d-07 2.231d-01
2.300d-07 2.086d-01
2.320d-07 1.942d-01
2.340d-07 1.797d-01
2.360d-07 1.690d-01
2.380d-07 1.595d-01
2.400d-07 1.500d-01
2.420d-07 1.405d-01
2.440d-07 1.334d-01
2.460d-07 1.272d-01
2.480d-07 1.210d-01
2.500d-07 1.147d-01
2.520d-07 1.101d-01
2.540d-07 1.060d-01
2.560d-07 1.019d-01
2.580d-07 9.775d-02
2.600d-07 9.464d-02
2.620d-07 9.188d-02
2.640d-07 8.911d-02
2.660d-07 8.634d-02
2.680d-07 8.421d-02
2.700d-07 8.230d-02
2.720d-07 8.039d-02
2.740d-07 7.847d-02
2.760d-07 7.696d-02
2.780d-07 7.558d-02
2.800d-07 7.420d-02
2.820d-07 7.282d-02
2.840d-07 7.170d-02
2.860d-07 7.066d-02
2.880d-07 6.962d-02
2.900d-07 6.858d-02
2.920d-07 6.770d-02
2.940d-07 6.688d-02
2.960d-07 6.607d-02
2.980d-07 6.525d-02
3.000d-07 6.454d-02
3.020d-07 6.387d-02
3.040d-07 6.320d-02
3.060d-07 6.252d-02
3.080d-07 6.193d-02
3.100d-07 6.135d-02
3.120d-07 6.078d-02
3.140d-07 6.020d-02
3.160d-07 5.968d-02
3.180d-07 5.917d-02
3.200d-07 5.866d-02
3.220d-07 5.815d-02
3.240d-07 5.767d-02
3.260d-07 5.721d-02
3.280d-07 5.674d-02
3.300d-07 5.628d-02

```
 3.320d-07   5.585d-02 .*
 3.340d-07   5.542d-02 .*
 3.360d-07   5.500d-02 .*
 3.380d-07   5.458d-02 .*
 3.400d-07   5.417d-02 .*
 3.420d-07   5.378d-02 .*
 3.440d-07   5.338d-02 .*
 3.460d-07   5.299d-02 .*
 3.480d-07   5.261d-02 .*
 3.500d-07   5.224d-02 .*
 3.520d-07   5.187d-02 .*
 3.540d-07   5.149d-02 .*
 3.560d-07   5.114d-02 .*
 3.580d-07   5.078d-02 .*
 3.600d-07   5.043d-02 .*
 3.620d-07   5.007d-02 .*
 3.640d-07   4.973d-02 .*
 3.660d-07   4.939d-02 .*
 3.680d-07   4.905d-02 .*
 3.700d-07   4.870d-02 .*
 3.720d-07   4.838d-02 .*
 3.740d-07   4.806d-02 .*
 3.760d-07   4.774d-02 .*
 3.780d-07   4.742d-02 .*
 3.800d-07   4.711d-02 .*
 3.820d-07   4.681d-02 .*
 3.840d-07   4.651d-02 .*
 3.860d-07   4.620d-02 .*
 3.880d-07   4.591d-02 .*
 3.900d-07   4.562d-02 .*
 3.920d-07   4.533d-02 .*
 3.940d-07   4.504d-02 .*
 3.960d-07   4.476d-02 .*
 3.980d-07   4.448d-02 *
 4.000d-07   4.420d-02 *
                       ---------------------------------------

Y
0

        JOB CONCLUDED
0        TOTAL JOB TIME            274.62
```

1*******05/10/85 ******** SPICE 2G.6 3/15/83 ********20:34:28*****

0* 3U P WELL CMOS MODELS ---- BK

0**** INPUT LISTING TEMPERATURE = 27.000 DEG C

0**********

```
 .MODEL N1 NMOS(LEVEL=2 TOX=65N NSUB=15E15 VTO=1.1 XJ=0.35U LD=0.25U
 +              JS=1.24E-4 PB=0.80 UO=526 UCRIT=3.97E4 UEXP=0.08
 +              UTRA=0.25 GAMMA=1 LAMBDA=0.02 CGBO=5.7E-10
 +              CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
 +              VMAX=5E4 NEFF=3 RSH=30)
 .MODEL N NMOS(LEVEL=2 TOX=50N NSUB=10E15 VTO=0.93 XJ=0.45U LD=0.24U
 +              JS=1.24E-4 PB=0.80 UO=381 UCRIT=99E4 UEXP=0.001
 +              UTRA=0 LAMBDA=0.025 CGBO=4.0E-10 TPG=1
 +              CGDO=5.2E-10 CGSO=5.2E-10 CJ=3.2E-4 CJSW=9.0E-10
 +              VMAX=5.5E4 NEFF=1.0E-2 RSH=25 DELTA=1.47 NFS=3.73E11)
 .MODEL N3 NMOS(LEVEL=2 TOX=55N NSUB=5E15 VTO=0.5 XJ=0.6U LD=0.4U
 +              JS=1.24E-4 PB=0.80 UO=1053 UCRIT=3.97E4 UEXP=0.08
 +              UTRA=0.25 GAMMA=0.9 LAMBDA=0.02 CGBO=5.7E-10
 +              CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
 +              VMAX=5E4 NEFF=3 RSH=10)
 .MODEL P1 PMOS(LEVEL=2 TOX=65N NSUB=5E15 VTO=-1.1 XJ=0.35U LD=0.25U
 +              JS=7.75E-5 PB=0.88 UO=210 UCRIT=4.14E4 UEXP=0.16
 +              UTRA=0.25 GAMMA=0.6 LAMBDA=0.02 CGBO=5.7E-10
 +              CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
 +              VMAX=5E4 NEFF=3 RSH=100)
 .MODEL P PMOS(LEVEL=2 TOX=50N NSUB=2.97E14 VTO=-0.844 XJ=0.0258U LD=0.512U
 +              JS=7.75E-5 PB=0.88 UO=100 UCRIT=18500 UEXP=0.145
 +              GAMMA=0.723 LAMBDA=0.0527 CGBO=4.0E-10 TPG=-1
 +              CGDO=4.0E-10 CGSO=4.0E-10 CJ=2.0E-4 CJSW=4.0E-10
 +              VMAX=10E4 NEFF=.01 RSH=95 DELTA=2.19 NFS=1.62E12)
 .MODEL P3 PMOS(LEVEL=2 TOX=55N NSUB=.3E15 VTO=-0.5 XJ=0.6U LD=0.4U
 +              JS=7.75E-4 PB=0.88 UO=421 UCRIT=4.14E4 UEXP=0.16
 +              UTRA=0.25 GAMMA=0.4 LAMBDA=0.02 CGBO=5.7E-10
 +              CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
 +              VMAX=5E4 NEFF=3 RSH=50)
 * NODES 4-5 CORRESPOND TO 100072 (SEE DRAIN AT 4492,3422)
 * NODES 6-7 CORRESPOND TO 98266 (SEE GATE AT 4490,3410)
 * NODES 8-9 CORRESPOND TO 98884 (SEE GATE AT 4489,3322)
 * NODES 10-11 CORRESPOND TO 98304 (SEE GATE AT 4657,3323)
 * NODES 12-13 CORRESPOND TO 98634 (SEE GATE AT 4665,3315)
 * NODES 14-15 CORRESPOND TO 99764 (SEE GATE AT 4700,3549)
 * NODES 16-17 CORRESPOND TO OUT#77 (SEE GATE AT 4490,3752)
 * NODES 18-19 CORRESPOND TO 104688 (SEE DRAIN AT 4507,3752)
 * NODES 20-21 CORRESPOND TO INB#2 (SEE GATE AT 4507,3752)
 * NODES 22-23 CORRESPOND TO 87074 (SEE GATE AT 4441,3049)
 * NODES 24-25 CORRESPOND TO 91691 (SEE GATE AT 4443,2973)
 * NODES 26-27 CORRESPOND TO 85756 (SEE GATE AT 4570,2971)
 * NODES 28-29 CORRESPOND TO OUT#61 (SEE GATE AT 4570,2707)
 * NODES 30-31 CORRESPOND TO IN#6 (SEE SOURCE AT 4575,2534)
 * NODES 32-33 CORRESPOND TO BUSA#30_H (SEE GATE AT 3209,2884)
 * NODES 34-35 CORRESPOND TO 72620 (SEE GATE AT 3186,2119)
 * NODES 36-37 CORRESPOND TO 72601 (SEE GATE AT 3147,2119)
 * NODES 38-39 CORRESPOND TO BUSA30 (SEE GATE AT 3557,2548)
 * NODES 40-41 CORRESPOND TO 71960 (SEE GATE AT 1671,2132)
 * NODES 42-43 CORRESPOND TO RWBUSA_H#104 (SEE GATE AT 1684,2211)
 * NODES 44-45 CORRESPOND TO 76328 (SEE GATE AT 1677,2243)
 * NODES 46-47 CORRESPOND TO 80179 (SEE SOURCE AT 1672,2331)
 * NODES 48-49 CORRESPOND TO DSTDEC_L#32 (SEE SOURCE AT 1672,2292)
 * NODES 50-51 CORRESPOND TO RFWRITE (SEE GATE AT 834,2360)
 * NODES 52-53 CORRESPOND TO OUT#32 (SEE GATE AT 829,2363)
 * NODES 54-55 CORRESPOND TO 81217 (SEE DRAIN AT 774,2398)
 * NODES 56-57 CORRESPOND TO PHASE3 (SEE GATE AT 4122,3489)

C1 55 0 0.006PF
R1 30 31 1793
C2 31 0 0.124PF
R2 36 37 774
C3 37 0 0.098PF
R3 50 51 41045
C4 51 0 2.331PF
C5 47 0 0.004PF
R4 40 41 601
C6 41 0 0.081PF
```

```
R5 32 33 13393
C7 33 0 0.774PF
R6 14 15 5266
C8 15 0 0.353PF
R7 44 45 1825
C9 45 0 0.065PF
R8 24 25 702
C10 25 0 0.121PF
R9 22 23 2119
C11 23 0 0.295PF
R10 28 29 1341
C12 29 0 0.049PF
R11 20 21 6865
C13 21 0 0.253PF
R12 34 35 1279
C14 35 0 0.063PF
R13 52 53 1829
C15 53 0 0.066PF
R14 42 43 20869
C16 43 0 0.563PF
R15 6 7 1276
C17 7 0 0.090PF
R16 38 39 11877
C18 39 0 1.634PF
C19 19 0 0.006PF
R17 10 11 1755
C20 11 0 0.071PF
R18 4 5 3108
C21 5 0 0.140PF
R19 26 27 6516
C22 27 0 0.233PF
R20 8 9 1260
C23 9 0 0.159PF
R21 48 49 1241
C24 49 0 0.074PF
R22 16 17 1117
C25 17 0 0.051PF
R23 12 13 2347
C26 13 0 0.074PF
M1 4 7 1 1 P L=3.0U W=24.0U
M2 0 7 4 0 N L=3.0U W=15.0U
M3 0 9 6 0 N L=3.0U W=4.0U
M4 6 0 1 1 P L=6.0U W=4.0U
M5 8 0 1 1 P L=6.0U W=4.0U
M6 8 11 0 0 N L=3.0U W=4.0U
M7 10 13 0 0 N L=3.0U W=12.0U
M8 10 13 1 1 P L=2.7U W=22.0U
M9 1 15 12 1 P L=3.0U W=24.0U
M10 12 15 0 0 N L=3.0U W=12.0U
M11 14 17 0 0 N L=3.0U W=8.0U
M12 14 17 1 1 P L=3.0U W=8.0U
M13 16 21 1 1 P L=3.0U W=8.0U
M14 19 21 0 0 N L=3.0U W=8.0U
M15 16 1 19 0 N L=3.0U W=8.0U
M16 0 23 20 0 N L=3.0U W=15.0U
M17 1 23 20 1 P L=3.0U W=24.0U
M18 22 0 1 1 P L=6.0U W=4.0U
M19 0 25 22 0 N L=3.0U W=4.0U
M20 0 27 24 0 N L=3.0U W=4.0U
M21 24 0 1 1 P L=6.0U W=4.0U
M22 1 29 26 1 P L=3.0U W=24.0U
M23 26 29 0 0 N L=3.0U W=12.0U
M24 31 0 28 1 P L=3.0U W=8.0U
M25 33 0 30 1 P L=3.0U W=8.0U
M26 32 35 0 0 N L=3.0U W=11.0U
M27 32 35 1 1 P L=3.0U W=23.0U
M28 1 37 34 1 P L=3.0U W=11.0U
M29 0 37 34 0 N L=3.0U W=11.0U
M30 39 0 36 1 P L=3.0U W=4.0U
M31 0 1 40 0 N L=3.0U W=15.0U
M32 41 43 38 0 N L=3.0U W=7.0U
M33 1 45 42 1 P L=3.0U W=68.0U
M34 0 45 42 0 N L=3.0U W=31.0U
M35 0 1 47 0 N L=3.0U W=7.0U
M36 47 1 48 0 N L=3.0U W=7.0U
M37 49 51 44 0 N L=3.0U W=4.0U
M38 1 53 50 1 P L=3.0U W=59.0U
M39 0 53 50 0 N L=3.0U W=27.0U
```

```
 M40 55 1 0 0 N L=3.0U W=8.0U
 M41 52 57 55 0 N L=3.0U W=8.0U
 M42 1 57 52 1 P L=3.0U W=8.0U

 * INITIAL CONDITIONS:
 .IC V(55)=0.000000
 .IC V(30)=5.000000
 .IC V(31)=5.000000
 .IC V(36)=5.000000
 .IC V(37)=5.000000
 .IC V(50)=0.000000
 .IC V(51)=0.000000
 .IC V(47)=0.000000
 .IC V(40)=0.000000
 .IC V(41)=0.000000
 .IC V(32)=5.000000
 .IC V(33)=5.000000
 .IC V(14)=5.000000
 .IC V(15)=5.000000
 .IC V(44)=5.000000
 .IC V(45)=5.000000
 .IC V(24)=5.000000
 .IC V(25)=5.000000
 .IC V(22)=0.000000
 .IC V(23)=0.000000
 .IC V(28)=5.000000
 .IC V(29)=5.000000
 .IC V(20)=5.000000
 .IC V(21)=5.000000
 .IC V(34)=0.000000
 .IC V(35)=0.000000
 .IC V(52)=5.000000
 .IC V(53)=5.000000
 .IC V(42)=0.000000
 .IC V(43)=0.000000
 .IC V(6)=5.000000
 .IC V(7)=5.000000
 .IC V(38)=5.000000
 .IC V(39)=5.000000
 .IC V(19)=0.000000
 .IC V(10)=5.000000
 .IC V(11)=5.000000
 .IC V(4)=0.000000
 .IC V(5)=0.000000
 .IC V(26)=0.000000
 .IC V(27)=0.000000
 .IC V(8)=0.000000
 .IC V(9)=0.000000
 .IC V(48)=0.000000
 .IC V(49)=0.000000
 .IC V(16)=0.000000
 .IC V(17)=0.000000
 .IC V(12)=0.000000
 .IC V(13)=0.000000

 VDD 1 0 5.0
 VIN 57 0 PULSE(0 5 0NS 0NS 0NS)
 .WIDTH OUT=80
 .TRAN 2.00NS 400NS
 .PLOT TRAN V(5) (0,5)
 .END
1*******05/10/85 ********  SPICE 2G.6    3/15/83 ********20:34:28*****

0* 3U P WELL CMOS MODELS ---- BK

0****      MOSFET MODEL PARAMETERS          TEMPERATURE =   27.000 DEG C

0*****************************************************************
```

	N1	N	N3	P1	P	P3
0TYPE	NMOS	NMOS	NMOS	PMOS	PMOS	PMOS
0LEVEL	2.000	2.000	2.000	2.000	2.000	2.000
0VTO	1.100	0.930	0.500	-1.100	-0.844	-0.500
0KP	2.79d-05	2.63d-05	6.61d-05	1.12d-05	6.91d-06	2.64d-05
0GAMMA	1.000	0.834	0.900	0.600	0.723	0.400

0PHI	0.716	0.695	0.660	0.660	0.514	0.514
0LAMBDA	2.00d-02	2.50d-02	2.00d-02	2.00d-02	5.27d-02	2.00d-02
0PB	0.800	0.800	0.800	0.880	0.880	0.880
0CGSO	5.70d-10	5.20d-10	5.70d-10	5.70d-10	4.00d-10	5.70d-10
0CGDO	5.70d-10	5.20d-10	5.70d-10	5.70d-10	4.00d-10	5.70d-10
0CGBO	5.70d-10	4.00d-10	5.70d-10	5.70d-10	4.00d-10	5.70d-10
0RSH	30.000	25.000	10.000	100.000	95.000	50.000
0CJ	6.00d-04	3.20d-04	6.00d-04	4.10d-04	2.00d-04	4.10d-04
0CJSW	5.64d-10	9.00d-10	5.64d-10	3.85d-10	4.00d-10	3.85d-10
0JS	1.24d-04	1.24d-04	1.24d-04	7.75d-05	7.75d-05	7.75d-04
0TOX	6.50d-08	5.00d-08	5.50d-08	6.50d-08	5.00d-08	5.50d-08
0NSUB	1.50d+16	1.00d+16	5.00d+15	5.00d+15	2.97d+14	3.00d+14
0NFS	0. d+00	3.73d+11	0. d+00	0. d+00	1.62d+12	0. d+00
0TPG	1.000	1.000	1.000	1.000	-1.000	1.000
0XJ	3.50d-07	4.50d-07	6.00d-07	3.50d-07	2.58d-08	6.00d-07
0LD	2.50d-07	2.40d-07	4.00d-07	2.50d-07	5.12d-07	4.00d-07
0UO	526.000	381.000	1053.000	210.000	100.000	421.000
0UCRIT	3.97d+04	9.90d+05	3.97d+04	4.14d+04	1.85d+04	4.14d+04
0UEXP	0.080	0.001	0.080	0.160	0.145	0.160
0UTRA	0.250	0.	0.250	0.250	0.	0.250
0VMAX	5.00d+04	5.50d+04	5.00d+04	5.00d+04	1.00d+05	5.00d+04
0NEFF	3.000	0.010	3.000	3.000	0.010	3.000
0DELTA	0.	1.470	0.	0.	2.190	0.

1*******05/10/85 ******** SPICE 2G.6 3/15/83 ********20:34:28*****

0* 3U P WELL CMOS MODELS ---- BK

0**** INITIAL TRANSIENT SOLUTION TEMPERATURE = 27.000 DEG C

0**********

NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE
(1)	5.0000	(4)	0.0000	(5)	0.0000	(6)	5.0000
(7)	5.0000	(8)	0.0000	(9)	0.0000	(10)	5.0000
(11)	5.0000	(12)	0.0000	(13)	0.0000	(14)	5.0000
(15)	5.0000	(16)	0.0000	(17)	0.0000	(19)	0.0000
(20)	5.0000	(21)	5.0000	(22)	0.0000	(23)	0.0000
(24)	5.0000	(25)	5.0000	(26)	0.0000	(27)	0.0000
(28)	5.0000	(29)	5.0000	(30)	5.0000	(31)	5.0000
(32)	5.0000	(33)	5.0000	(34)	0.0000	(35)	0.0000
(36)	5.0000	(37)	5.0000	(38)	5.0000	(39)	5.0000
(40)	0.0000	(41)	0.0000	(42)	0.0000	(43)	0.0000
(44)	5.0000	(45)	5.0000	(47)	0.0000	(48)	0.0000
(49)	0.0000	(50)	0.0000	(51)	0.0000	(52)	5.0000
(53)	5.0000	(55)	0.0000	(57)	0.		

VOLTAGE SOURCE CURRENTS

NAME	CURRENT
VDD	-6.500d-05
VIN	0. d+00

TOTAL POWER DISSIPATION 3.25d-04 WATTS

1*******05/10/85 ******** SPICE 2G.6 3/15/83 ********20:34:28*****

0* 3U P WELL CMOS MODELS ---- BK

0**** OPERATING POINT INFORMATION TEMPERATURE = 27.000 DEG C

0**********

0
0**** MOSFETS

0	M1	M2	M3	M4	M5	M6	M7
0MODEL	P	N	N	P	P	N	N
ID	-8.73e-10	5.06e-24	-1.33e-14	-2.68e-24	-3.25e-05	5.52e-09	1.98e-12
VGS	-0.000	5.000	-5.000	-5.000	-5.000	5.000	0.000
VDS	-5.000	-0.000	-5.000	-0.000	-5.000	0.000	5.000
VBS	0.	-0.000	-5.000	0.	0.	0.	0.

0	M8	M9	M10	M11	M12	M13	M14
0MODEL	P	P	N	N	P	P	N
ID	-9.80e-25	8.71e-10	3.35e-22	1.96e-12	-2.65e-24	-2.74e-10	6.63e-27
VGS	-5.000	5.000	5.000	0.000	-5.000	-0.000	5.000
VDS	-0.000	5.000	0.000	5.000	-0.000	-5.000	0.000
VBS	0.	5.000	0.	0.	0.	0.	0.

0	M15	M16	M17	M18	M19	M20	M21
0MODEL	N	N	P	P	N	N	P
ID	1.07e-22	-5.89e-14	6.96e-25	-3.25e-05	-5.52e-09	-1.33e-14	-2.68e-24
VGS	5.000	-5.000	-5.000	-5.000	5.000	-5.000	-5.000
VDS	0.000	-5.000	0.000	-5.000	-0.000	-5.000	-0.000
VBS	-0.000	-5.000	0.000	0.	-0.000	-5.000	0.

0	M22	M23	M24	M25	M26	M27	M28
0MODEL	P	N	P	P	N	P	P
ID	8.71e-10	3.35e-22	2.15e-26	2.13e-26	1.98e-12	-1.94e-24	3.84e-10
VGS	5.000	5.000	-5.000	-5.000	0.000	-5.000	5.000
VDS	5.000	0.000	0.000	-0.000	5.000	-0.000	5.000
VBS	5.000	0.	0.000	-0.000	0.	0.	5.000

0	M29	M30	M31	M32	M33	M34	M35
0MODEL	N	P	N	N	P	N	N
ID	1.66e-24	8.16e-28	-2.51e-25	-2.56e-14	2.52e-09	2.85e-23	-1.77e-26
VGS	5.000	-5.000	5.000	-5.000	5.000	5.000	5.000
VDS	-0.000	-0.000	-0.000	-5.000	5.000	-0.000	-0.000
VBS	-0.000	-0.000	-0.000	-5.000	5.000	-0.000	-0.000

0	M36	M37	M38	M39	M40	M41	M42
0MODEL	N	N	P	N	N	N	P
ID	-1.69e-26	-1.33e-14	2.18e-09	2.18e-23	-2.30e-26	1.96e-12	6.01e-28
VGS	5.000	-5.000	5.000	5.000	5.000	-0.000	-5.000
VDS	-0.000	-5.000	5.000	-0.000	0.000	5.000	0.000
VBS	-0.000	-5.000	5.000	-0.000	0.	-0.000	0.000

1*******05/10/85 ******** SPICE 2G.6 3/15/83 ********20:34:28*****

0* 3U P WELL CMOS MODELS ---- BK

0**** TRANSIENT ANALYSIS TEMPERATURE = 27.000 DEG C

0**********

x

TIME V(5)

	0.	d+00	1.250d+00	2.500d+00	3.750d+00	5.000d+00
0. d+00	2.823d-13	•	.	.	.	.
2.000d-09	2.027d-05	•	.	.	.	.
4.000d-09	2.655d-06	•	.	.	.	.
6.000d-09	-4.659d-06	•	.	.	.	.
8.000d-09	-2.780d-06	•	.	.	.	.
1.000d-08	-2.056d-06	•	.	.	.	.
1.200d-08	-1.605d-06	•	.	.	.	.
1.400d-08	-1.083d-06	•	.	.	.	.

1.600d-08	-8.066d-07	•
1.800d-08	-4.644d-07	•
2.000d-08	-1.043d-07	•
2.200d-08	-1.719d-08	•
2.400d-08	1.629d-07	•
2.600d-08	4.211d-07	•
2.800d-08	6.793d-07	•
3.000d-08	7.235d-07	•
3.200d-08	7.640d-07	•
3.400d-08	8.046d-07	•
3.600d-08	8.451d-07	•
3.800d-08	9.567d-07	•
4.000d-08	1.069d-06	•
4.200d-08	1.182d-06	•
4.400d-08	1.279d-06	•
4.600d-08	1.250d-06	•
4.800d-08	1.221d-06	•
5.000d-08	1.192d-06	•
5.200d-08	1.174d-06	•
5.400d-08	1.244d-06	•
5.600d-08	1.315d-06	•
5.800d-08	1.385d-06	•
6.000d-08	1.442d-06	•
6.200d-08	1.395d-06	•
6.400d-08	1.349d-06	•
6.600d-08	1.302d-06	•
6.800d-08	1.267d-06	•
7.000d-08	1.326d-06	•
7.200d-08	1.385d-06	•
7.400d-08	1.444d-06	•
7.600d-08	1.484d-06	•
7.800d-08	1.432d-06	•
8.000d-08	1.381d-06	•
8.200d-08	1.329d-06	•
8.400d-08	1.296d-06	•
8.600d-08	1.349d-06	•
8.800d-08	1.403d-06	•
9.000d-08	1.456d-06	•
9.200d-08	1.491d-06	•
9.400d-08	1.441d-06	•
9.600d-08	1.390d-06	•
9.800d-08	1.340d-06	•
1.000d-07	1.308d-06	•
1.020d-07	1.361d-06	•
1.040d-07	1.413d-06	•
1.060d-07	1.466d-06	•
1.080d-07	1.481d-06	•
1.100d-07	1.414d-06	•
1.120d-07	1.346d-06	•
1.140d-07	1.395d-06	•
1.160d-07	1.409d-06	•
1.180d-07	1.394d-06	•
1.200d-07	1.395d-06	•
1.220d-07	1.400d-06	•
1.240d-07	1.403d-06	•
1.260d-07	1.403d-06	•
1.280d-07	1.404d-06	•
1.300d-07	1.405d-06	•
1.320d-07	1.407d-06	•
1.340d-07	1.408d-06	•
1.360d-07	1.410d-06	•
1.380d-07	1.412d-06	•
1.400d-07	1.410d-06	•
1.420d-07	1.408d-06	•
1.440d-07	1.406d-06	•
1.460d-07	1.412d-06	•
1.480d-07	1.465d-06	•
1.500d-07	1.460d-06	•
1.520d-07	1.423d-06	•
1.540d-07	1.407d-06	•
1.560d-07	1.405d-06	•
1.580d-07	1.403d-06	•
1.600d-07	1.401d-06	•
1.620d-07	1.400d-06	•
1.640d-07	1.400d-06	•
1.660d-07	1.399d-06	•
1.680d-07	1.399d-06	•
1.700d-07	1.400d-06	•

1.720d-07	1.402d-06	•
1.740d-07	1.403d-06	•
1.760d-07	1.405d-06	•
1.780d-07	1.406d-06	•
1.800d-07	1.405d-06	•
1.820d-07	1.405d-06	•
1.840d-07	1.405d-06	•
1.860d-07	1.405d-06	•
1.880d-07	1.406d-06	•
1.900d-07	1.407d-06	•
1.920d-07	1.408d-06	•
1.940d-07	1.408d-06	•
1.960d-07	1.409d-06	•
1.980d-07	1.409d-06	•
2.000d-07	1.409d-06	•
2.020d-07	1.410d-06	•
2.040d-07	1.410d-06	•
2.060d-07	1.410d-06	•
2.080d-07	1.411d-06	•
2.100d-07	1.411d-06	•
2.120d-07	1.410d-06	•
2.140d-07	1.410d-06	•
2.160d-07	1.410d-06	•
2.180d-07	1.409d-06	•
2.200d-07	1.409d-06	•
2.220d-07	1.408d-06	•
2.240d-07	1.408d-06	•
2.260d-07	1.407d-06	•
2.280d-07	1.407d-06	•
2.300d-07	1.407d-06	•
2.320d-07	1.407d-06	•
2.340d-07	1.407d-06	•
2.360d-07	1.407d-06	•
2.380d-07	1.407d-06	•
2.400d-07	1.407d-06	•
2.420d-07	1.406d-06	•
2.440d-07	1.405d-06	•
2.460d-07	1.405d-06	•
2.480d-07	1.404d-06	•
2.500d-07	1.403d-06	•
2.520d-07	1.402d-06	•
2.540d-07	1.401d-06	•
2.560d-07	1.400d-06	•
2.580d-07	1.403d-06	•
2.600d-07	1.405d-06	•
2.620d-07	1.408d-06	•
2.640d-07	1.411d-06	•
2.660d-07	1.412d-06	•
2.680d-07	1.411d-06	•
2.700d-07	1.411d-06	•
2.720d-07	1.411d-06	•
2.740d-07	1.421d-06	•
2.760d-07	1.432d-06	•
2.780d-07	1.443d-06	•
2.800d-07	1.496d-06	•
2.820d-07	1.583d-06	•
2.840d-07	1.598d-06	•
2.860d-07	1.534d-06	•
2.880d-07	1.446d-06	•
2.900d-07	1.392d-06	•
2.920d-07	1.373d-06	•
2.940d-07	1.355d-06	•
2.960d-07	1.336d-06	•
2.980d-07	1.332d-06	•
3.000d-07	1.344d-06	•
3.020d-07	1.356d-06	•
3.040d-07	1.368d-06	•
3.060d-07	1.378d-06	•
3.080d-07	1.385d-06	•
3.100d-07	1.392d-06	•
3.120d-07	1.400d-06	•
3.140d-07	1.408d-06	•
3.160d-07	1.516d-06	•
3.180d-07	2.171d-06	•
3.200d-07	2.089d-05	•
3.220d-07	7.077d-05	•
3.240d-07	6.215d-05	•
3.260d-07	-2.908d-04	•

```
 3.280d-07 -1.372d-03 *
 3.300d-07 -3.239d-03 *
 3.320d-07 -5.911d-03 *
 3.340d-07 -8.115d-03 *
 3.360d-07 -3.370d-03 *
 3.380d-07  2.593d-02 *
 3.400d-07  1.747d-01 .*
 3.420d-07  9.841d-01 .  *
 3.440d-07  2.226d+00 .   *
 3.460d-07  3.934d+00 .    *
 3.480d-07  4.728d+00 .     *
 3.500d-07  4.957d+00 .     *
 3.520d-07  4.992d+00 .     *
 3.540d-07  4.998d+00 .     *
 3.560d-07  4.999d+00 .     *
 3.580d-07  4.999d+00 .     *
 3.600d-07  5.000d+00 .     *
 3.620d-07  5.000d+00 .     *
 3.640d-07  4.999d+00 .     *
 3.660d-07  5.000d+00 .     *
 3.680d-07  5.000d+00 .     *
 3.700d-07  5.000d+00 .     *
 3.720d-07  5.000d+00 .     *
 3.740d-07  5.000d+00 .     *
 3.760d-07  5.000d+00 .     *
 3.780d-07  5.000d+00 .     *
 3.800d-07  5.000d+00 .     *
 3.820d-07  5.000d+00 .     *
 3.840d-07  5.000d+00 .     *
 3.860d-07  5.000d+00 .     *
 3.880d-07  5.000d+00 .     *
 3.900d-07  5.000d+00 .     *
 3.920d-07  5.000d+00 .     *
 3.940d-07  5.000d+00 .     *
 3.960d-07  5.000d+00 .     *
 3.980d-07  5.000d+00 .     *
 4.000d-07  5.000d+00 .     *
```

Y
0

JOB CONCLUDED
0 TOTAL JOB TIME 317.85

APPENDIX C

SPICE Parameters

```
* 3u p well cmos models ----- bk
.MODEL N1 NMOS(LEVEL=2 TOX=65N NSUB=15E15 VTO=1.1 XJ=0.35U LD=0.25U
+              JS=1.24E-4 PB=0.80 UO=526 UCRIT=3.97E4 UEXP=0.08
+              UTRA=0.25 GAMMA=1 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
+              VMAX=5E4 NEFF=3 RSH=30)
.MODEL n NMOS(LEVEL=2 TOX=50N NSUB=10E15 VTO=0.93 XJ=0.45U LD=0.24U
+              JS=1.24E-4 PB=0.80 UO=381 UCRIT=99E4 UEXP=0.001
+              UTRA=0 LAMBDA=0.025 CGBO=4.0E-10 TPG=1
+              CGDO=5.2E-10 CGSO=5.2E-10 CJ=3.2E-4 CJSW=9.0E-10
+              VMAX=5.5E4 NEFF=1.0E-2 RSH=25 DELTA=1.47 NFS=3.73E11)
.MODEL N3 NMOS(LEVEL=2 TOX=55N NSUB=5E15 VTO=0.5 XJ=0.6U LD=0.4U
+              JS=1.24E-4 PB=0.80 UO=1053 UCRIT=3.97E4 UEXP=0.08
+              UTRA=0.25 GAMMA=0.9 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=6.0E-4 CJSW=5.64E-10
+              VMAX=5E4 NEFF=3 RSH=10)
.MODEL P1 PMOS(LEVEL=2 TOX=65N NSUB=5E15 VTO=-1.1 XJ=0.35U LD=0.25U
+              JS=7.75E-5 PB=0.88 UO=210 UCRIT=4.14E4 UEXP=0.16
+              UTRA=0.25 GAMMA=0.6 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
+              VMAX=5E4 NEFF=3 RSH=100)
.MODEL p PMOS(LEVEL=2 TOX=50N NSUB=2.97E14 VTO=-0.844 XJ=0.0258U LD=0.512U
+              JS=7.75E-5 PB=0.88 UO=100 UCRIT=18500 UEXP=0.145
+              GAMMA=0.723 LAMBDA=0.0527 CGBO=4.0E-10 TPG=-1
+              CGDO=4.0E-10 CGSO=4.0E-10 CJ=2.0E-4 CJSW=4.0E-10
+              VMAX=10E4 NEFF=.01 RSH=95 DELTA=2.19 NFS=1.62E12)
.MODEL P3 PMOS(LEVEL=2 TOX=55N NSUB=.3E15 VTO=-0.5 XJ=0.6U LD=0.4U
+              JS=7.75E-4 PB=0.88 UO=421 UCRIT=4.14E4 UEXP=0.16
+              UTRA=0.25 GAMMA=0.4 LAMBDA=0.02 CGBO=5.7E-10
+              CGDO=5.7E-10 CGSO=5.7E-10 CJ=4.1E-4 CJSW=3.85E-10
+              VMAX=5E4 NEFF=3 RSH=50)
```

REFERENCES

[Blak83] J. Blakken. "Register Window for SOAR" Proceedings of CS290R, Smalltalk on a RISC - Architectural Investigations, Computer Science Division, Univ. of Cal., Berkeley, April, 1983.

[Vlad81] A.Vladimirescu, K.Zhang, A.R.Newton, D.O.Pederson, A.Sangiovanni-Vincentelli "SPICE Version 2G Users Guide" Department of Electrical Engineering and Computer Sciences Univ. of Cal., Berkeley CA. Aug 10,1981.

[DAmb83]B. D'Ambrosio "Smalltalk-80 Language Measurements - Dynamic use of Compiled Methods" Proceedings of CS290R, Smalltalk on a RICS - Architectural Investigations, Computer Science Division, Univ. of Cal. Berkeley, EECS, April, 1983.

[Deu81] L.P. Deutsch, "Measurement of the Dorado Smalltalk-80 Systems", Berkeley Computer Systems Seminar, Fall 1981.

[Deu83] L.P. Deutsch, "The Dorado Smalltalk-80 Implementation: Hardware Architecture's Impact on a Software Architecture", Addison Wesley, Sept 1983.

[DeS84] L.P. Deutsch and A.M Shiffman, "Efficient Implementations of the Smalltalk-80 System", Proceedings of the 11th Annual ACM SIGACT News-SIGPLAN notices Symposium on Principles of Programming Languages, Salt Lake City, Utah, Jan. 1984.

[Fitz81] D. Fitzpatrick, J. Foderaro, M. Ketevenis, H. Lardman, D. Patterson, J. Peek, Z. Peshkess, C. Sequin, R. Sherburne, K. VanDyke. "VLSI Implementations of a Reduced Instruction Set Computer" VLSI Systems and Computations, Carnegie Mellon Univ. Conf., Computer Science Press, pp.327-336 Oct 1981.

[Gonc83] N. Gonclaves, H.J. DeMann "NORA: A Race Free Dynamic CMOS Structure Technique for Pipelined Logic Structures." IEEE Journal of Solid State Circuits Vol SC-9, No. 5, pp. 272-285.

[Hofm83] M. Hofmann. "Aspects of Design and Layout of a CMOS ALU for SOAR", CAD Group Internal Memorandum, Dec, 1983.

[Kell84] Keller, K., "An Electric Circuit CAD Framework," Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, June, 1984, Memo No. UCB/ERL M84/54.

[Kram82] R.H. Krambeck, C.M. Lee, H.S. Law. "High Speed Compact Circuits with CMOS." IEEE Journal of Solid State Circuits Vol. SC-17 No.3, pp. 118-126.

[Mah84] G. Mah, "PANDA: A PLA Generator for Multiply-Folded PLAs", Proc. IEEE Int'l Conf on CAD, Santa Clara, CA. Nov 1984 pp. 122,124

[Oust83] J. Ousterhout. "Using Crystal for Timing Analysis" in "Berkeley VLSI Design Tools More Works by the Original Artists" Computer Science Division, EECS University of California, Berkeley, CA Sept, 1983.

[Patt81] D. Patterson, C. Sequin "RISC I A Reduced Instruction Set Computer" Proc. of the 8th. Annual Symposium on Computer Architecture. ACM SIGARCH 9.3 pp 443-457 May 1981.

[Patt83] Patterson, David, "Proceedings of CS290R, Smalltalk on a RISC - Architectural Investigations", Computer Science Division, Univ. of Cal., Berkeley, April, 1983.

[Patt83b] Patterson, David, "Second SOAR" Internal Memorandum Computer Science Division, EECS, Univ. of California, Berkeley. Feb. 1983.

[Patt84] Patterson, David, "Reduced Instruction Set Computers" Communications of the ACM Dec. 1984. pp.8-21.

[Reed85] Reed, James, "YACR2. Yet Another Channel Router 2" MS Report, Univ. of Cal., Berkeley, Dept. of EECS, May 1985.

[Rude85] R. Rudell, "ESPRESSO-IIC Users Manual", CAD group manual, Univ. of Cal., Berkeley, Dept. of EECS, May 1985.

[Ryan85] D. Ryan "An Interactive Routing toolbox" MS Report, Univ. of Cal. Berkeley, Dept of EECS MS Report, May 1985.

[Samp84] D. Samples, M. Kline, D. Foley "SOAR Architecture" Univ. of Cal. Berkeley, EECS Internal Memorandum Sept, 1983.

[Unga83] D. Ungar.,R. Blau, P. Foley, D. Samples, D. Patterson, "Architecture of SOAR Smalltalk on a RISC" Proceedings of the 11th Annual Symposium on Computer Architecture. Anarbor MI. June 1984.