A Response to the Star LSI Workstation Goals

Version 1.0
November 1978

Draft of 1:00 P.M. November 3, 1978

The *Star LSI Workstation Goals* document sets guidelines for the implementation of a low cost workstation (terminal, processor, and peripherals) for the OIS Star product. This document responds to those goals by the:

1. Enumeration and brief analysis of several alternative technical approaches.
2. Elaboration and analysis of the Wildflower approach.

Although Wildflower, as described in the documents currently available, will not be adequate for the workstation design; we find that a the use of a bipolar microprocessor which is carefully synchronized to the memory and I/O controllers does offer the best approach to the workstation implementation.

Version 1.0 represents current thinking concerning the design of the Star LSI Workstation. The reader is encouraged to send comments to Bob Belleville (Belleville @ ParcMaxc or 8*923*4520). This file is stored on [IRIS] <Architecture> LsiWS 1.0 > LsiWS-Part1.press and Part2.press.

Prepared by: Robert L. Belleville
Ronald C. Crane
Robert B. Garner
J. Pitts Jarvis
E. A. Miller
Roy C. Ogus
Stephen C. Purcell
C. Richard Snow

Approved by Robert M. Metcalfe
SDD System Architecture

Approved by David E. Liddle
SDD

XEROX

Systems Development Department
3408 Hillview Avenue
Palo Alto, California 94304

Table of Contents

Xerox Private Data

A Response to the Star LSI Workstation Goals

1. Introduction, Conclusions, and Recomendations

The *Star LSI Workstation Goals* document [reference 1] sets guidelines for the implementation of a low cost workstation (terminal, processor, and peripherals) for the OIS Star product. This document responds to those goals by the:

1. Enumeration and brief analysis of several alternative technical approaches.
2. Elaboration and analysis of the Wildflower approach.

This document was written during three weeks of intensive study by a team of 8 engineers. It should not be represented as an exhaustive examination of all the available alternatives, but rather, a careful look at the most promising ones.

1.1 Introduction

Wildflower [reference 2] is a high performance microprocessor implemented with the 2901, bipolar, 4-bit cpu slice. Like the Alto and the D0, Wildflower shares this processor among the peripheral controllers to realize simple and compact controllers. Wildflower simplifies the approach taken in the Alto and D0 in two crucial ways:

1. Wildflower places a strict limit on the number and kind of peripherals it can support. Specifically, only four are allowed - full page display, rigid disk, Xerox Wire, and low speed microprocessor bus which handles serial communication, auxiliary media, printers, and floppy disk.
2. The Wildflower processor is shared among these peripheral controllers in a strict time synchronous way, not simply on demand as in the Alto and D0. This allows each controller to be designed with a known I/O latency. As we shall see, this results in a dramatic simplification of the entire system over the Alto or the D0; however, the performance is comparable to the D0 in the execution of Mesa bytecodes.

It is likely that a synchronous processor like Wildflower would result in the lowest cost workstation; however, this type of design leaves little flexibility for either configuration changes or for variations in the specifications or availability of integrated circuit components. This increases the risk in conjunction with a tight development schedule.

A more traditional bus organized microprocessor design is presented in chapter 5 as the primary alternative to a Wildflower-like approach. The higher estimated cost of this approach is offset by increased flexibility in configuration and in the fact that individual components can be defined and developed in parallel during the development cycle.

1.2 Conclusions

Chapter 5 introduces five workstation alternatives:

- o Cost-reduced Alto
- o Cost-reduced D0
- o Custom LSI
- o Bus organized microprocessor
- o Synchronous multi-tasking microprocessor

Xerox Private Data

The number of modifications needed to efficiently implement the Mesa/Pilot environment on a cost-reduced Alto eliminate this as a viable approach.

There is little that can be done to dramatically reduce the cost of the D0 because of its complex controllers, processor, and memory system.

Custom LSI, built within Xerox or outside, requires a fully tested architecture consistent with the LSI technology, which we do not now have, and more lead time than we have for the LSI workstation IMO. ED's Standard Silicon Bus looks very attractive; however, we have no experience with the use of this technology for the construction of an OIS processor.

The cost estimates in section 11 and the cost figures in the *Star LSI Workstation Goals* show that only about $1100 can be used for the processor and the peripheral controllers. This includes the cost of the PWB, connectors, and and memory. These costs are essentially the same in either a MSI or LSI implementation. At most LSI implementation can save a few hundred dollars in the total workstaion UMC.

The bus-organized microprocessor is attractive. However, the use of even the most advanced 16-bit monolithic microprocessor would require extensive changes to the Mesa development environment which would be both time-consuming to develop and difficult to maintain.

The use of a custom-designed bipolar microprocessor is essential to the efficient use of the Mesa/Pilot environment. Careful examination of the use of this class of processor in the bus organized approach results in the observation that it is the synchronous relationship of the memory, processor, and I/O that gives the most attractive design.

The multi-tasking synchronous processor SUMIT divides the use of the high performance microprocessor amoung the four I/O devices (disk, display, X-wire, and low speed bus) to realize simple, low chip count controllers. The type of processor is also well suited to the execution of the Mesa byte stream.

1.3 Recomendations

So, we conclude that a bipolar microprocessor which is relatively tightly coupled to the the I/O controllers is the only viable design alternative for the implementation of the Star Workstation. A processor like SUMIT is the most likely to meet both the cost and schedule constraints. Furthermore, we recommend that the design of such a processor begin within SDD as soon as possible so that the proposed schedules can be met.

With respect to the implementation of the workstation with LSI technology, we conclude that because longer schedules will be needed we should start now to develop such a machine. However, since the cost of the workstation is dominated by the peripherals and the cabinetry, the UMC cost reduction of such a station will not be very dramatic.

2. Mandatory References

[1] Ruebel, R. et. al., *Star LSI Workstation Goals*, Xerox Internal Memo, 11 September 1977.

[2] Lampson, B. W., *Wildflower Manual (Draft)*, Xerox Palo Alto Research Center, 18 August 1978. Filed on [IVY] <WF> WFManual.Press and [IVY] <WF> WFDrawings.Press.

3. Informational References

These background references provides more depth on many of the subjects covered in this document.

[3] *The OIS Processor Principles of Operation*, Systems Development Department, April 9, 1977.

[4] *ALTO: A Personal Computer System Hardware Manual*, Xerox Palo Alto Research Center, February 1978.

[5] *D0 Processor Functional Specificaton*, Electronic Operations Department, January 16, 1978.

[6] Ogus, Roy C., *Xerox Wire Specification*, Systems Development Department, April 5, 1978.

[7] *Microprocessor Data Manual II*, Electronic Design 21, October 11, 1978.

[8] Crane, R. C., *Star Grounding*, Systems Development Department, September 13, 1978.

[9] Wick, J., *D0 Timing Simulation*, Systems Development Department, October 18, 1978.

4. Assumptions

The following is a list of the assumptions used in the rest of the document:

1. The Star LSI workstation processor must be Mesa source level compatible with existing D0 and Alto systems. The only exception to this rule will be the very lowest level of Mesa I/O drivers.
2. None of the D0 micorcode or microdiagnostics will be used.
3. The Star LSI workstation will not be developed to emulate the Alto nor will it run existing Alto programs. Nor will it support Alto peripherals.
4. The configuration of the Star Workstation will be essentially as described in this document up to IMO.
5. Pilot can be modified or extended to support the simplified organization of the I/O devices and the firmware supported virtual memory system.

5. The Alternatives

The following are possible alternative implementations for the Star LSI workstation:

- o Cost-reduced Alto
- o Cost-reduced D0
- o Custom LSI
- o Bus organized microprocessor
- o Synchronous multi-tasking microprocessor

5.1 Cost-reduced Alto

Some time ago SPG worked out a plan to cost reduce the Alto and to make it ready for volume production. The machine would be exactly an Alto down to the microcode. This is not a viable alternative because:

- o Display at Alto resolution (606 x 808)
- o No virtual memory and 16 bit addressing
- o Control store size limitation
- o No hardware stack
- o Not an efficient Mesa processor
- o Model 31 disk
- o Cost too high UMC ~$5000

5.2 Cost-reduced D0

The question of scaling down the D0 to meet the goals is likely to come up. We could find no significant ways to reducing the size or cost of the D0.

5.3 Custom LSI

The key problem with custom LSI developed by Xerox or elsewhere is the definition of a basic architecture which is known to meet the performance goals (for the execution of a Mesa byte stream and for the handling of a large, bit mapped, imaginal display) and to be compatible with the technology. Currently, we have no such architecture and the use of custom LSI for this version of the Star LSI workstation seems impossible within the schedule.

Xerox should take steps to develop custom LSI concurrently with this project. The next two alternatives to custom LSI can be used as stepping stones to the effective use of custom LSI by Xerox.

5.4 Bus organized microprocessor

Two needs of the workstation guide the design of the processor.

1. The workstation needs a full page imaginal display for the presentation of various fonts, graphics, and other symbolic and pictorial information. The workstation must be powerful enough to maintain the image on the monitor and to modify the underlying bit map quickly.
2. The high level Mesa language is needed to support the wide range of comprehensive workstation software. The processor must be able to execute the instructions produced by the compiler and be fast enough to provide adequate performance.

One additional requirement shapes the workstation architecture.

1. While the display places a continous drain of about 40M bits per second on the system, the rigid disk (Shugart 1000) and the Xerox Wire place intermittent loads of 4M bits per second and 10M bits per second respectively. In addition to these three high bandwidth devices, the workstation needs a host of low speed devices such as the mouse, keyboard, serial communication, printers, etc. The total cost of the system will be a strong function, not of the cost of the central processor, but of the combined cost of the peripheral controllers and the processor.

Recent development of 16 bit monolithic LSI microprocessors are interesting from two points of view - the power and speed of the CPU, and the level of integration of the support devices such as controllers.

Compare the specification of 3 of the best 16 bit machines (see reference 7):

	Intel 8086	Motorola 68000	Zilog 8000
Address Space	2^{20} (bytes)	2^{23} (bytes)	2^{23} (words)
Instruction Size	1-6 bytes	2-6 bytes	2-6 bytes
Fastest Operation	400 ns.	500 ns.	750 ns.
Package	40 pin dip	64 pin dip	40 or 48 pin
Power	5V/275 ma.	5V/300 ma.	5V/300 ma.
Price (100)	$93.75	not available	not available

As you can see each of these processors is capable of instructions in the 1M instructions per second (ips) range. Execution of Mesa on this class of processor could be done in a number of ways.

First, an emulator for the "Princops Mesa" byte code stream could be written for the microprocessor. With this technique there would be no modification to the compiler, but the performance would be limited to 25K to 100K ips range. (For reference, the Alto runs at about 200K ips with the display on.)

Or, the code generators of the compiler could be extended to produce code for the microprocessor. This would allow performance in the 100K to 500K ips range but it is likely that the size of the compiled programs would be increased by a factor of 2 or so. This is acceptable performance; however, the cost to the software environment is unknown - but probably high.

It is relatively simple to design a microprocessor based on MSI or bipolar slice microprocessors (2901, 2903, 3000 etc) to emulate the Mesa byte codes. This approach is not significantly more expensive than the LSI microprocessor; however, its performance would be quite satisfactory without the need to modify the software environment.

Support devices provide comprehensive support for the rest of the workstation. Each of the following devices is contained in a single 40 pin dip (or smaller) and is compatible with the Intel 8086 CPU.

Universal Peripheral Interface (actually a small computer)
Programmable Communication Interface
Programmable Interval Timer
Programmable Peripheral Interface

Programmable Interrupt Controller
Programmable DMA (Direct Memory Access) Controller
Programmable Floppy Disk Controller
Programmable HDLC/SDLC Controller
Programmable Alphanumeric CRT Display Controller
Programmable Keyboard Interface
GPIB (IEEE 488) Talker/Listener
GPIB Controller
Data Encryption Unit (NBS standard)

Figure 5.0a shows the organization of a Star LSI workstation configured from a combination of commercially available LSI parts and MSI/LSI modules which contains the devices specificed in the goals document [1].

full page display. By implementing an autonomous display controller which accesses a private memory bank, most the the high bandwidth requirements on the system can be localized to the display section. The memory would be available to the rest of the system through the system busses. This memory would be "dual ported" so that the display would be maintained without the need for processor intervention.

rigid disk controller. This is a traditional disk controller. If the disk drive changes, a new controller can be developed and eventually the controller could be replaced by a custom LSI controller.

Xerox Wire controller. This controller is like the disk controller.

BitBlt processor. If a monolithic LSI microprocessor was used in the system a special hardware controller would be needed to handle the bit oriented shifting and masking needed to move patches of the display bitmap from one section of memory to another (for example placing character bitmaps on the display). This function would be done in the slice CPU if that approach is taken.

CPU. Commercial LSI, commercial slice microprocessor, or custom LSI could be used here. Use of the slice processor would allow the BitBlt functions to be handled by the CPU. With this exception, the only tasks the CPU has to do is to emulate the Mesa machine, all the controllers are autonomous.

Main Memory. Completely conventional.

Other Low Speed Devices. The flexible organization of the bus organized microprocessor allows the use of the commercial LSI controllers listed above as well as costom LSI.

This alternative has the following disadvantages:

- o The parts count and cost are higher than the alternative in section 5.5 below.

This alternative has the following advantages:

- o Each of the components can be developed in parallel with relatively high confidence that the integerated system will operate correctly.
- o There is considerable flexibility in the configuration.
- o The separation of the controllers allow the use of custom LSI as it can be developed.

- The CPU can be configured from components of various size and performance to meet very low cost, limited performance future requirements.

- A CPU configured from bipolar slice microprocessor technology can be of conservative design and still result in satisfactory Mesa emulation performance because this is the only task the processor must do.

5.5 Synchronous multi-tasking microprocessor

In the next chapter, we will look at an implementation of a synchronous multi-tasking microprocessor.

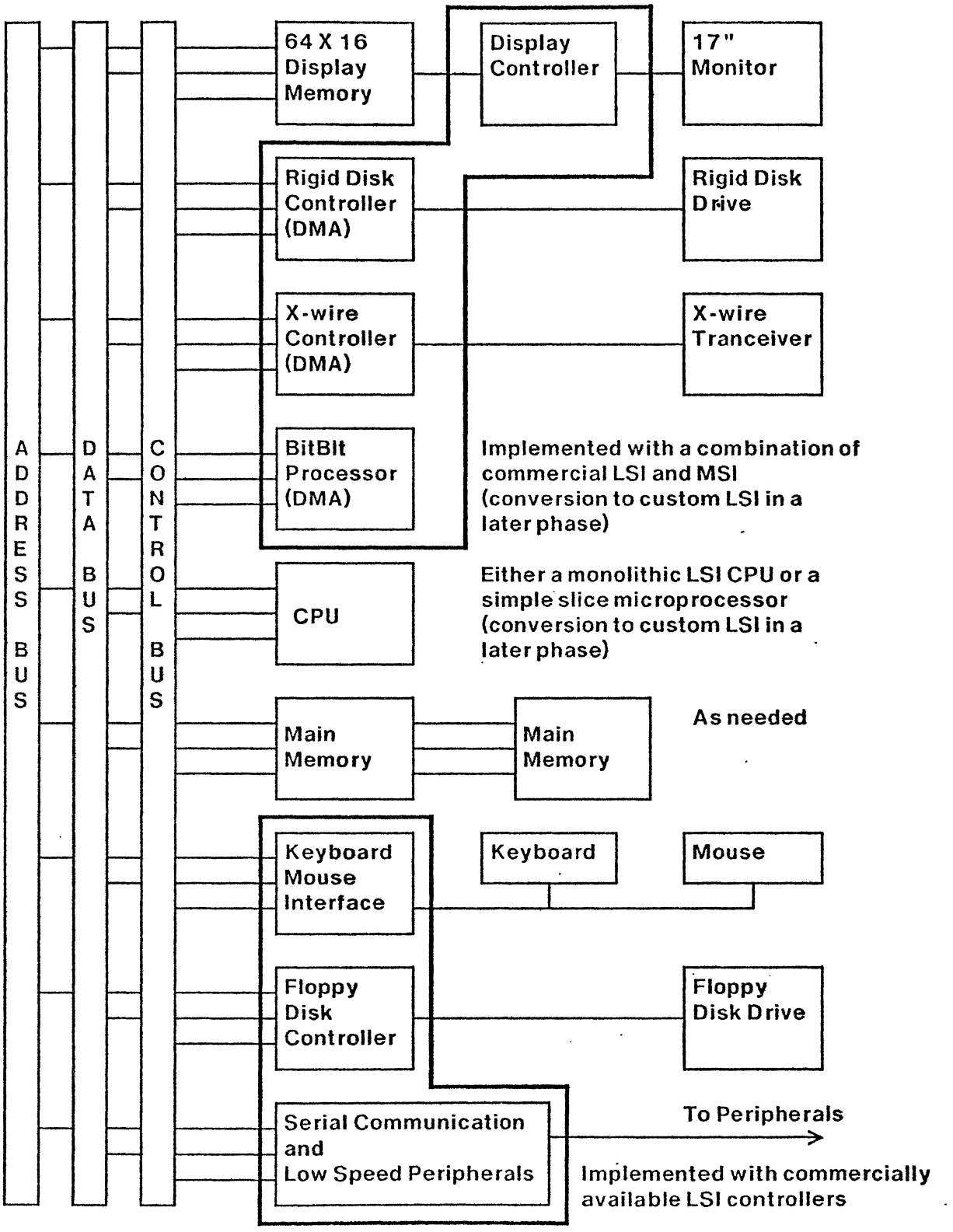


Figure 5.0a Bus Organized Workstation Microprocessor

6. A Synchronous Multi-Tasking Microcomputer

6.1 Introduction

A Synchronous Multi-Tasking (SUMIT) architecture consists of a structure which has been closely interwoven with the I/O devices that it services. The primary design concept is a single synchronously driven memory whose access is shared among several I/O controllers and the emulator in a round-robin fashion. Each of the high bandwidth devices is allocated a fixed time slot during which it can access the memory. How the devices are allocated slots and the width of the time slot determines the overall design. In addition, the micro-programmable microprocessor holds the memory addresses and other state information of the I/O controllers, thereby dramatically reducing the size of the controllers. This design is based on the Wildflower machine. [ref 2] This chapter will present the major details and discuss the primary deviations. The rest of this chapter will look at the implementation of the I/O devices specifed by the Goals document. Figure 6.1a shows an overall block diagram for a workstation based on a SUMIT architecture.

The SUMIT concept is based on a fixed microcode tasking scheme, a synchronously operating memory, some synchronously driven I/O devices, and a simple ALU (such as the 2901 4-bit slice). A block diagram of the basic data paths is shown in figure 6.1b.

6.1.1 Tasking Mechanism

The machine provides ALU services to the Mesa emulator and five I/O tasks: display, Xerox Wire, rigid disk, and a low speed peripheral bus (LSP Bus). The microprocessor services these devices in a set order during a single *round.* A round consists of a fixed number of *clicks* where each device executes during its click if the hardware requested service. If the I/O device does not run during its click, then the emulator runs instead. Clicks are divided into three *cycles* during which three micro-instructions are executed and one main memory cycle is completed.

Several click allocation schemes are possible where each differs by how much of the machine is used by the display and the length of a click. The original Wildflower design calls for eight 300 nanosecond clicks per round of which 3 are for the display. Another possible scheme is 16 clicks per round and 7 allocated for the display. Based on a conservative timing estimate for the machine cycle time, the following click allocation scheme is assumed in this report:

Click	*Task*
0	Display
1	rigid Disk/LSP Bus
2	Display
3	Xerox Wire/LSP Bus

With this allocation scheme, a cycle lasts 120 nanoseconds, a click is therefore 360 nsec, and a round 1.44 μsec (4 clicks). These times are based on conservative timing estimates. Appendix B shows a timing analysis of some of Wildflower's electrical paths: Note that all are much less than 120 nanoseconds.

Given this click allocation scheme, one can determine the refresh rate for a full page display (1024 x 808 bits). The display transfers 32 bits per click and requires 32 clicks to paint a 1024 bit scan line. Giving 4 rounds for horizontal retrace and assuming an 875 line video implies 25.2 msec per frame or a 40 Hertz refresh rate. See section 6.3 for more discussions on the full page display.

6.1.2 Data Paths

Most data paths of the machine are 16 bit, however, some widths are determined from bandwidth considerations. The following table shows the width of important data paths in the machine as it is currently arranged and some maximum obtainable bandwidths given the current allotment of clicks. (Note that, unlike the D0, maximum obtainable bandwidth becomes guaranteed bandwidth once a particular I/O controller is designed). See section 6.7 for more performance estimates.

from	*to*	*width (bits)*	*bandwidth (Mbits/sec)*
mem address reg	memory	17	
memory	instruction buffer	32	
Y bus	LSP Bus	8	(see 6.6)
memory	display	32	44
Y bus	Xerox Wire	16	11
Y bus	SA 1000 Disk	8	5.6

6.1.3 Memory

The memory consists of 34 64K word dynamic RAM chips for a total of 128K words and parity for each word. The virtual memory space is currently defined by a 20-bit word (see section 6.2.5 for discussion of virtual memory).

At the first click of each round a memory reference is always started. The sixteen least significant address bits come from the Y bus and the upper four bits come from the RH auxiliary memory. If the reference is a store, the second click must supply the data on the Y bus and execute a MEMOUT← function, otherwise the operation is assumed to be a fetch. Since a memory access can occur once per 360 nanosecond click, maximum processor to memory bandwidth is 44 MBits/sec. The memory operates 32 bits wide and the least significant address bit selects the appropriate half.

6.1.4 Control

The machine will have 2K or 4K words of micro-instruction memory addressed by a next instruction address register (NIA) which is loaded from a 12 bit field of the micro-instruction (INIA). Note that a RAM control store implementation will have a significant hardware cost versus a PROM control store (which the current Wildflower design calls for). The D0 control store consists of 33 4K by 1 bit Intel 70 nS RAMs and a small loader PROM is necessary. The current Wildflower design calls for 6 2K by 8 bit MMI 70 nSec PROMs. Since it is much less expensive to make field changes to a RAM than a PROM control store (such as to upgrade performance or fix bugs), it is expected that the work station will contain a writeable control store (the engineering model certainly will).

Branching is accomplished by ORing into the least significant bit of INIA of the next instruction. There are several flavors of dispatching which are accomplished by ORing subfields of the X or Y bus with INIA of the next instruction. The function NextMacro ORs the next byte of the instruction buffer into the least 8 bits of INIA, and is used to dispatch on the byte code in the last cycle of a click.

Subroutine linkage is accomplished via three link registers which can be loaded from NIA (call) and ORed into INIA (return). Like dispatches, they take effect in the cycle after they are specified.

Currently the microinstruction size is 42 bits: 18 control the 2901 and RH, 4 control the Stack, U and W bus, 12 are INIA, and 8 are two function fields. (There probably will be 3 function fields

since these fields do much of the work).

6.1.5 I/O Latency

The I/O devices are designed with the assumption that they can receive a known amount of service in an small amount of time, i.e., commensurate with byte or word transmission times. Each device has a *request* signal which is synchronized and tested during the second cycle of the click preceding the device's click. If true, the device runs during its click. In addition, there must be a mechanism to reset the request signal which avoids lost service requests.

With this scheme, the minimum latency a device can expect is 4t nanoseconds for input and 5t for an output device. The maximum is 13t nanoseconds for an output device which is always not requesting service (t is the cycle time, 120 nSec). Thus, the amount of buffering a device needs depends on this maximum latency and the synchronization time between the processor and the device.

6.1.6 Clocks

In order to maintain a synchronous display, the "2/4" allocation scheme used in this report (DKDX) requires a master clock with a 7.5ns period (133 MHz). This clock is divided by 3 to obtain the display bit duration (22.5ns), and by 16 to obtain the cycle time (120ns). The "7/16" allocation scheme requires a master clock with an 8.75ns period (117 MHz). The "3/8" 100ns Wildflower requires a 40 MHz clock.

6.1.7 Design Considerations

The SUMIT philosophy will lead to some interesting design tradeoffs. Since the display controller is totally synchronous to the processor clock, the refresh rate can only be adjusted proportionately with the cycle time. Also, since device buffering is very small, the processor clock cannot be appreciably varied otherwise the I/O will lose data. Thus, the SUMIT processor must be brought up with a reasonable cycle time in order to debug the whole system. For these reasons it is believed that 120 nanoseconds is a reasonable cycle time, and in fact the machine will have a good speed margin. Also, the associated 40 frames/second is an good display refresh rate.

6.1.8 Risks

The current design assumes a National Semiconductor IDM 2901A-1 4-bit bipolar microprocessor slice with a "D" setup time of 40 ns. Common 2901's are much slower, for example, AMD's 2901 has a "D" setup of time of 100 ns.

If 64K RAMs aren't available in time, then 136 16K RAMs are required for a 128K main memory instead of 34 chips. This will have a dramatic effect on the packaging.

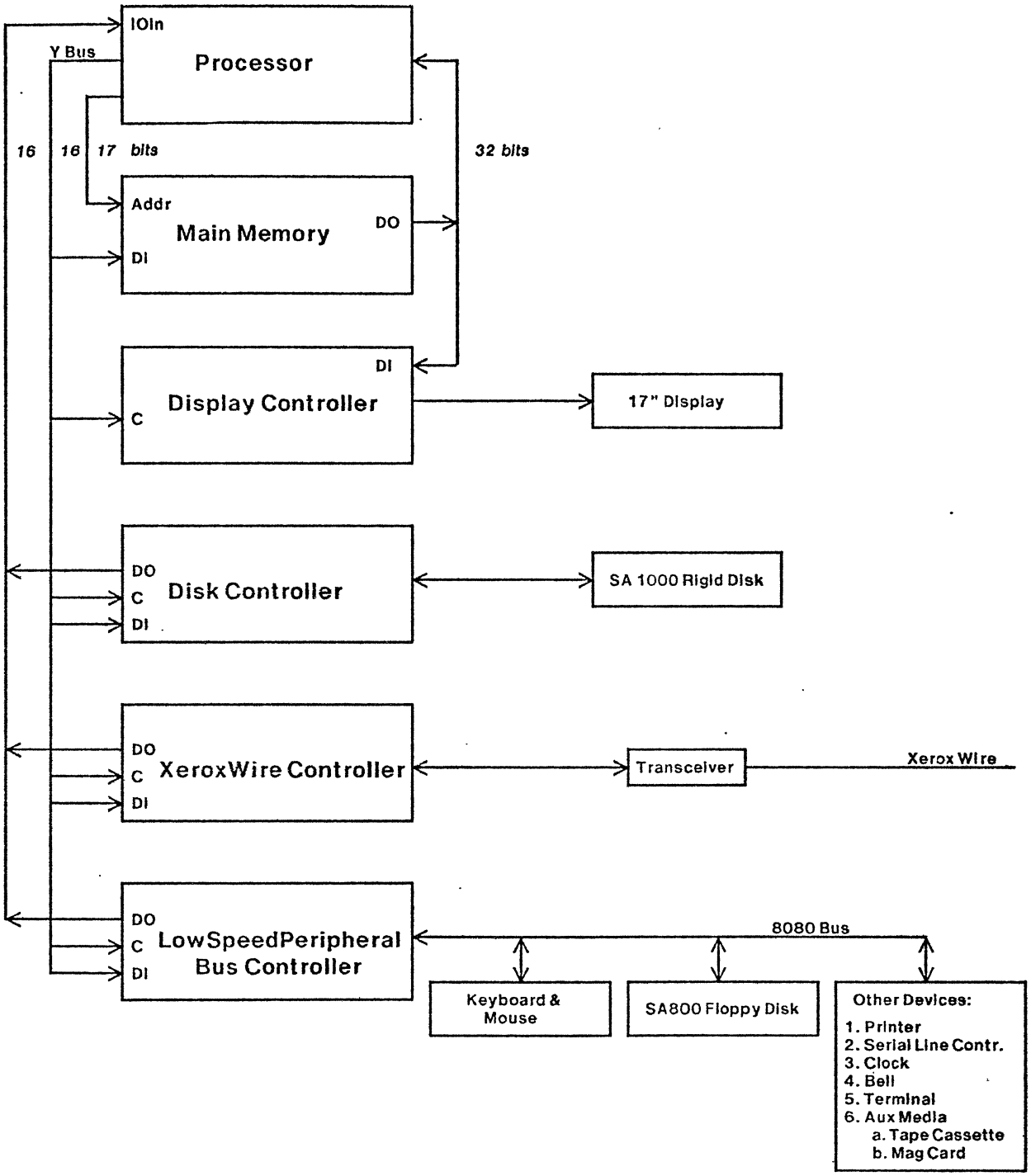


Figure 6.1a SUMIT Overall Organization

wfoverall.sil

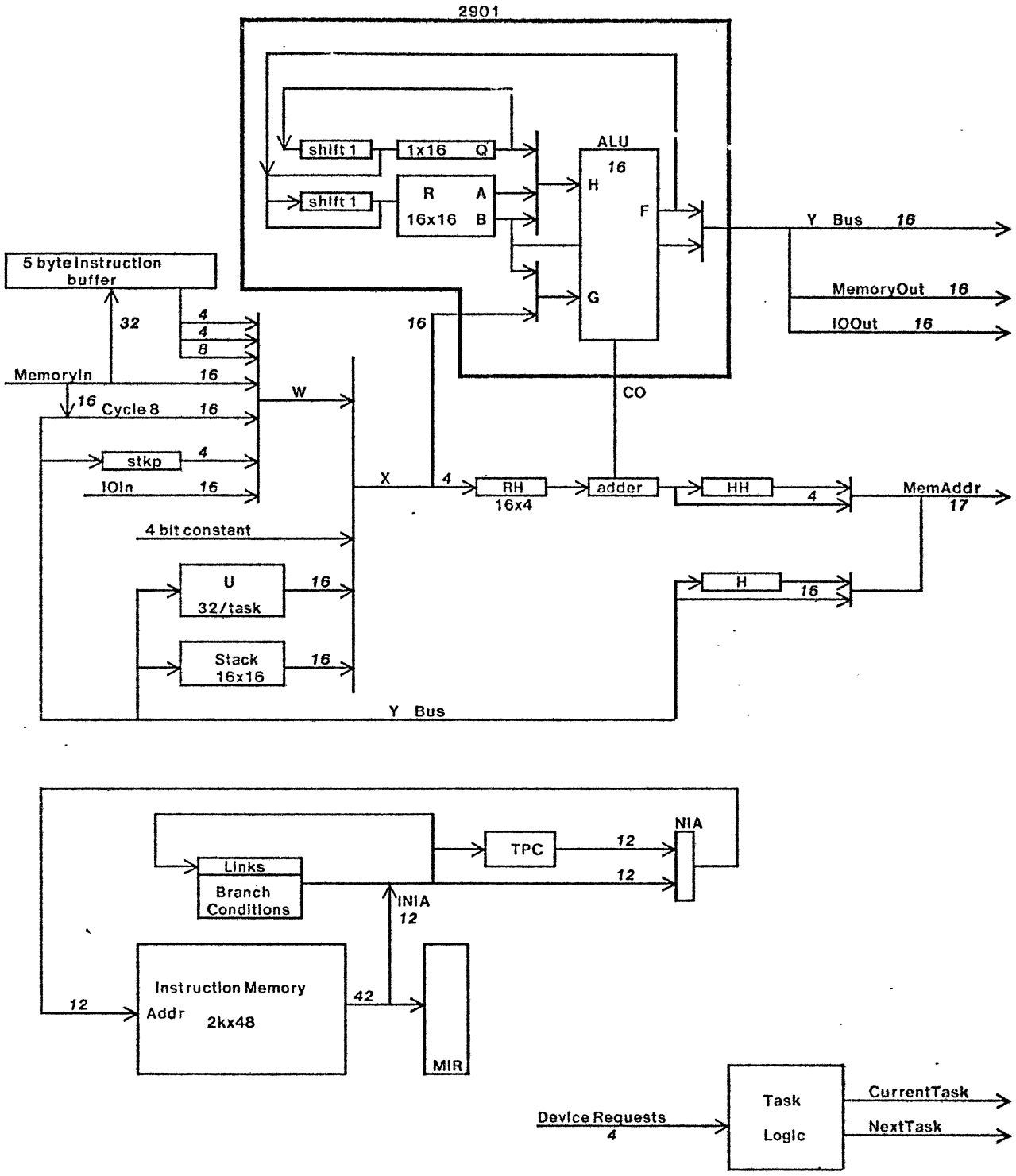


Figure 6.1b SUMIT Processor Data Paths wfdatapaths.sll

6.2 The Mesa Emulator

This machine is designed to emulate Mesa efficiently. An instruction buffer, a stack, a split adder and an extra bit of program counter are provided to speed up the emulation of Mesa bytecodes.

6.2.1 Instruction Buffer

There is a 5-byte instruction buffer (IB) which holds the next few bytes of the Mesa instruction stream. The last click of each instruction dispatches on a byte from the IB or traps to refill it when it is near empty. Refill requires only one click to fetch four bytes, and emulation continues. At buffer refill times, the emulator checks for interrupts from devices or the timer maintained by the display task.

6.2.2 Stack

A special set of 16 registers is provided to hold the Mesa evaluation stack, together with a 4-bit register (STKP) to address them. By programming convention the top of the stack is kept in an R register (called TOS), and STKP points to the S register containing the second word on the stack. This style makes the top two words immediately available for a (non-arithmetic) binary operation.

6.2.3 Split Adder

The SplitAdder feature allows slightly different data sources for the high and low bytes of arithmetic/logic input. The motivation is to generate in one cycle virtual addresses of the form TOS + (MDS + IB) where MDS supplies the high byte and IB provides the low byte.

6.2.4 PC.16

The machine contains a single-bit register PC.16 which is intended to be used for the least significant bit of the Mesa byte program counter; an R register called PC will be used for the remainder of the program counter. A function is provided to increment these registers as if they were one 17 bit counter.

6.2.5 Memory Map and Page Faults

Since virtual memory is implemented with a map located in the main memory and microcode, the emulator tries to cache real page numbers for important data structures such as local and global frames and the program counter. Pointers to these base locations are mapped as part of Xfer. Instructions which fetch though general pointers must use an extra click to do their own mapping. It is estimated that less than 10% of memory references will suffer the overhead of mapping. Hardware assist is provided for fetching the map entry of an address instead of the normal contents; a "MapRef" fetch builds an address at the top of memory from the page number in a virtual address. Map entries contain a carefully aligned (11 bit) real page number and PrincOps flag bits encoded for fast detection of simple cases.

The simplest virtual memory accesses require no update of the map entry and take one click. Two or three clicks are required to update the referenced or dirty bit. Page faults are detected in the instruction causing them, so unwinding side effects should be simple.

6.2.6 *Performance Estimate: 420 Kips*

Trial microcode has been written by Roy Levin to emulate a representative sample of mesa's basic byte instructions (excluding Blt, BitBlt, Xfer, etc). The following is a speed analysis based on Levin's microcode and dynamic frequencies of groups of related bytecodes (see: D0 Timing Simulation memo from Wick to Liddle 10/18/78 - reference 9). Memory mapping times are included. The instruction fetch overhead is calculated as one click per four bytes.

Instruction	freq.	time	size	time	size (weighted)
Local/Global	40%	1 click	1 byte	.4	.4
Constants	10%	1 click	2 byte	.1	.2
PUSH like	5%	1 click	1 byte	.05	.05
Arithmetic	10%	2 click	1 byte	.2	.1
Field Access	5%	4 click	2 byte	.2	.1
Jumps	15%	3 click	2 byte	.4	.3
Pointer Access	5%	4 click	2 byte	.2	.1
Other	5%	4 click	2 byte	.2	.1
Subtotal	95%	average 2 clicks =		1.7+(1.3 *.25)	
Xfer	5%	24 click	1 byte	1.2	(.05)
Total	100%	av: 3.3 clicks =		2.9 + .4 (IB clicks)	

A click is 360ns and the emulator gets half of all clicks when the display is running but not other devices. Therefore the effective time of emulator clicks is 720ns, and ordinary instructions run at 700 thousand instructions per second (700 Kips). When Xfer type instructions are included as 5% of the dynamic mix the speed becomes 420 Kips. When the display is off these figures are 1.4 Mips and 840 Kips. The machine speed is quite sensitive to the proportion of Xfers executed since it is such a slow instruction; 5% is a conservative figure, twice what Apex and Desktop generate. (Note that the Wildflower design with a 100ns cycle has combined-display-on rate of 600 Kips.)

For comparison here is a chart of display overheads and performance estimates in Kips (Thousand instructions per second). The figures for Alto and D0 were measured and the Workstation estimated, so caution is advised. The display overhead for D0 is currently about 40% but will hopefully be reduced to 20%. The best that the D0 could be improved to is about 600 Kips for the easy mix with display off:

Kips	Alto	D0	Workstation
Display:	50%	20%	50%
--Display On			
Easy mix:	210	340	700
Hard mix:	130	200	420
Xfer(RET):	40	40	50
BitBlt 9x12:	2	3	4
--Display Off			
Easy mix:	430	420	1400
Hard mix:	265	265	840
Xfer(RET):	80	50	100
BitBlt 9x12:	4	3	8

Xfer -- 12 mapped memory references, 24 clicks (@720ns), or 17 uSec

Xfer has not been trial coded but the speed may be estimated by counting memory references, since the Alto experience has been that Xfer is memory bound. The most common form of Xfer (return) requires 65 alto cycles or 12 memory references.

BitBlt -- 2 or 2.5 times faster than Alto BitBlt

BitBlt has not been coded for Wildflower, but it references memory 3 times faster than Alto and shifts 3 times faster. The Alto requires 2x140 ns per bit shifted, while the Wildflower requires only 120ns per bit.

Rich Johnson has timed Alto and D0 BitBlt on the task of painting a 9x12 bit character (word packed). Alto required 185-340 usec; D0 required 310-350 usec. The D0 is more constant because it shifts any amount in constant time. The rough estimate made above indicates that the same task on a workstation would take 130-170 usec.

6.2.7 Microcode Register and Size Estimate: 8 R registers, 1.2 K instructions

Experience from Alto and D0 indicates that the mesa emulator will require approximately 1.2K words of control store. The emulator uses eight R registers and the Q register for MDS, TOS, PC, L, C, G, Temp, TTemp and Q=VA.

6.3 Full Page Display

6.3.1 General architecture

The system dedicates a contiguous portion of the virtual space to store the bit map. One page of 256 words stores exactly four lines with 1024 pixels per line. During horizontal retrace, the processor maps virtual pages into physical pages. To save physical storage, any unmapped pages are displayed as background. No special hardware is necessary to support a cursor; however, a microcoded cursor implementaion increases the size and complexity of the display microcode.

The cursor is two double words, 64 bits wide. On the lines that contain the cursor, the microcode fetches two double words from the cursor area rather than from the bit map. This scheme allows the illusion of a 16 bit wide cursor without alignment restrictions. No special hardware is necessary to support a cursor; however, a microcoded cursor implementaion increases the size and complexity of the display microcode.

The display task performs functions other than display refresh. The system uses the horizontal retrace interval to maintain a system clock and to perform the necessary memory refresh cycles. The TMS4164 requires 256 refresh cycles every four milliseconds or once every 15.6 microsecconds; Lampson's Wildflower design scanned a line in 31.2 microseconds. By coincidence, exactly two refresh cycles per line suffice to keep the memory refreshed. Any change in the design which increases the scanning period necessitates three refresh cycles per line.

6.3.2 Microcode

Appendix A contains sample microcode to scan one line. The code performs several functions during the horizontal retrace interval:

1. turn on horzontal blanking
2. perform memory refresh cycles

3. check for vertical retrace
4. map virtual addresses
5. maintain processor clock
6. check for cursor
7. fudge for interlace
8. wake up Xerox Wire task.

6.3.3 Hardware Interface

Changing the click allocation strategy from the "3/8" to the "2/4" scheme eliminates the auxilliary buffer from the hardware and simplifies the microcode. The distinction between A, B, and C clicks vanishes and the microcode therefore need not identify the type of click to the hardware.

6.4 SA1000/SA4000 Rigid Disks

6.4.1 General architecture

The Wildflower design used the Shugart SA4000, while the LSI Workstation Goals specify an SA1000 as the rigid disk drive. The SA1000 has slower data rates, 4.3 megabits per second, and smaller capacity, 5.3-10.7 megabytes, than the SA4000 which transfers at 7.1 megabits per second and has a capacity of 7.2-28.8 megabytes. The lower bit rate means that the SA1000 requires only one click per round and an eight bit buffer in the hardware controller. Should users require a more powerful workstation with an SA4000, it seems prudent to retain the same one click per round service while doubling the buffer size to sixteen bits. The cost of the two or three chips required for the extra buffering is negligible compared to the incremental cost of the SA4000.

6.4.2 Microcode

Inner loops for transferring data to and from the drive one byte at a time are included in appendix A. The inner loop calculates sixteen bit wide longitudinal parity for each field of a sector, assembles bytes into sixteen bit words, and transfers data between the drive and central storage. These three functions require dedicating three of the sixteen available R-registers to the disk task.

6.4.3 Hardware Interface

Adding a CRC chip saves two dedicated R-registers in the microcode. The code no longer needs a register to accumulate parity/checksum; furthermore, since the code no longer needs to compute the parity, it is possible to do byte assembly using a dedicated U-register and temporary R-register. The temporary R-register does not contain useful data between clicks and can be used in a similar manner by other tasks. The addition of a CRC chip makes the disk and Xerox Wire hardware very similar in function.

6.4.4 Assumptions/Requirements:

The interface of an SA1000 is similar to a SA4000 interface; for the present time, it is reasonable to assume the same controller hardware works for either drive.

This microcode assumes that the processor can load IOOut from MemoryIn in the same cycle. This can be implemented by tri-state buffers on MemoryIn with outputs on Y.

The microcode also assumes it can load KData from H.0-7, the high order byte, instead of H.8-15, the low order byte.

6.4.5 Performance and Impact on Rest of Processor

Can LSP bus work with some other click allocation scheme if SA4000 uses 8 bit data paths?

6.4.6 Present status of above and tasks necessary for complete design

Will timing prohibit loading IoOut from MD?

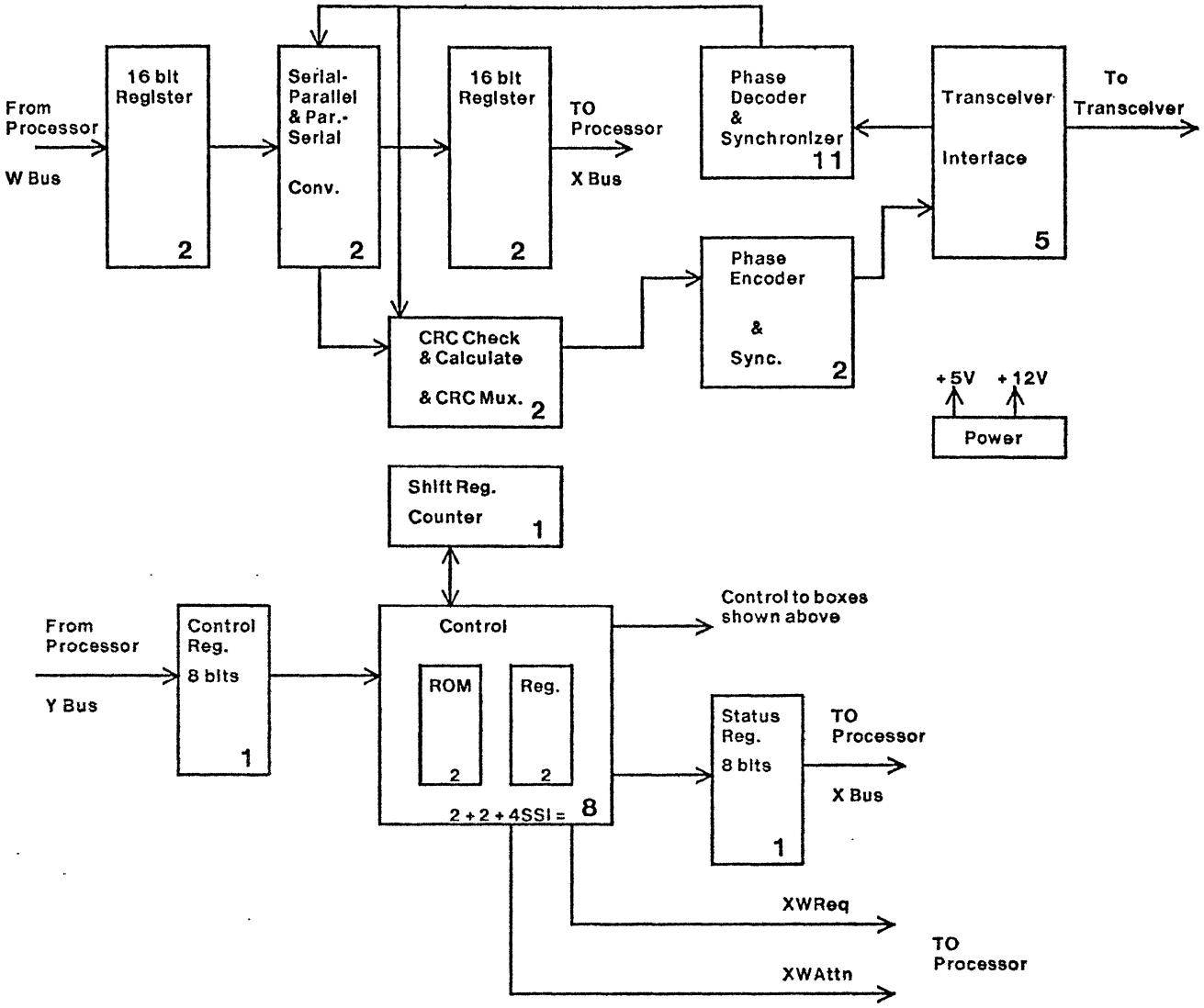
6.5 Xerox Wire

6.5.1 General architecture:

The Xerox Wire is allocated 1 click/round (DKDX). Due to bandwidth requirements, a 16-bit data path is needed for the interface. The interface will be half-duplex and will consist of serial to parallel and parallel to serial conversion hardware, together with circuits for phase-encoding and decoding. A phase-lock loop will probably replace the sampling approach shown for Ethernet due to speed considerations. The Xerox Wire output is not synchronous with the 120 nS cycles of the processor. Thus, an extra clock and synchronizer will be necessary to generate 10 MHz data rate.

6.5.2 Assumptions/Requirements:

- o Interface must not lose packets during transitions between transmit and receive.
- o Interface will not be able to receive packets back to back. (to receive, packet spacing must be 12 clicks = 17.3 μS. (5 clicks of post processing of old packet and 7 clicks for preprocessing to set up for next packet)
- o Machine host number comes from software.
- o Address recognition will be done by microcode.
- o The CSB and IOCB's are located in the first 64K of virtual memory.
- o Buffer addresses present in the IOCB's are real addresses, not virtual, and buffers should be contiguous in real memory.
- o Buffers cannot cross a 64K space boundary.
- o One IOCB is used per packet.
- o Each IOCB contains two buffers each of size at least two words.
- o The software driver will be woken up after every packet.
- o The initial retransmission interval on output packets is always zero.



Note:

1. Numbers in lower right hand corner are chip estimates.
2. 8 SSI chips whose functions are not yet specified are not shown

Figure 6.5a Xerox Wire Controller Block Diagram wsxwire.sil

6.5.3 Hardware Interface

Below is a description of the parts of the interface. Refer to the block diagram for perspective.

Interface to Transciever

The transciever interface has balanced signals and ECL levels. 5 chip equivalents are consumed by ECL chips and discrete components.

Phase Decoder (Analog or Digital)

A digital phase decoder can be implemented if there a 1Kx4 ROM and register combination can be found which will work with a 60 nS period. It would take about 6 chips to implement this phase decoder and it would lock in one bit time. The analog phase locked loop takes about 11 chips including synchronizer, and takes about 10 bit times to acquire phase lock.

Phase Encoder

This includes the output clock, synchronizer, XOR gate at output of shift register, and gating. This uses 5 SSI chips.

Serial-Parallel Conversion

16 bits at a time. This takes 4 chips (2 shift registers, 1 counter, and 1 CRC chip) + control

Buffering

1 word for input, 1 word for output and some gates for clock inputs. This is 5 chips.

Status and Control Register

Status and control registers which can be read and written by the processor require 2 8-bit register chips and 1 SSI chip for gating.

Control

Control of the unit is done using a ROM (read only memory) and a register running on 120 nS cycles and using 4 chips. 5 more SSI chips will likely be used as well. This is an estimate only as details have not yet been worked out.

Miscellaneous SSI

6 more chips for interconnecting various functions would probably be used.

6.5.4 Microcode

6.5.4.1 Interface with Mesa software

The Xerox Wire microcode would provide the facility to queue packets on both input and output. In addition, a limited scatter/gather mechanism would be implemented to enable the easy separation of the header and data parts of a packet. This should improve the performance of the Xerox Wire, especially for situations such as swapping over the Wire.

Two types of data structure blocks would exist for communication between the software and the microcode. The *Controller Status Block (CSB)* provides the basic link between the emulator and the Xerox Wire microcode. It has the following fields: host address (2 words), receive queue start pointer (1 word), receive wakeup mask (1 word), transmit queue start pointer (1 word), and transmit wakeup mask (1 word). The host address contains the 32-bit host address of the machine, or is zero indicating a promiscuous host. The queue start pointers point to the first IOCB (see below) of the appropriate buffer queue. There is only one input queue. A zero pointer value indicates the absence of a queue. The wakeup masks are used by the microcode to notify the software process after each packet has been processed.

The transmit and receive queues would be made up of *Input/Output Control Blocks (IOCB)*, each of which describe the buffers available for a single packet. Each IOCB would have two buffers associated with it, one for the packet header, the other for the data part. The IOCB has the following fields: pointer to next IOCB (1 word), pointer to buffer 0 (2 words), size of buffer 0 (1 word), pointer to buffer 1 (2 words), size of buffer 1 (1 word), buffer status (1 word). The buffer pointers are long pointers with the high 4 bits of the pointer right justified in the high word. The status word will be used to indicate the number of words received on input, or the state of the transmission on output.

No wakeups from the emulator are required once the CSB has been set up. The queues may be added to or packets removed on the fly. The display task will wakeup the Xerox Wire task periodically so that the latter can check if there is any work to do.

6.5.4.2 Interface with Xerox Wire hardware

The hardware is half-duplex with one word of buffering. The microcode can store commands and data into the hardware using the XCtl← and XData← F-field directives. To read a word of data or the hardware status, the F-field values, ←XData and ←XStatus should be used. rXReq clears the hardware wakeup request flip-flop. In addition, a special branch condition, XBranch, is needed to enable the microcode to implement a fast multi-way branch based on hardware conditions. A possible enumeration of the XBranch conditions is: interface has/wants a word of data, interface did not set XReq (i.e. task was started by display), and the interface has just stopped transmitting/receiving a packet.

6.5.4.3 Operation of microcode

A reasonably large part of the microcode has been informally written to determine the potential difficulties and to discover the necessary resource usage. Some of the more important parts can be found in appendix A. The Xerox Wire hardware buffering (1 word) requires that during transmission and reception a word has to be transferred between the interface and the processor every click. As a result several tricks are needed in order to enable the necessary functions to be performed on the fly. This section will discuss the various portions of the microcode.

Idle check: This portion is executed when the Xerox Wire is idle and the microcode is looking for work to do. It checks whether there are transmit and receive queues present. If a transmit queue is there then the buffer setup code is entered. If there is a receive queue then first buffer is set up and the arrival of a packet is awaited.

Receive buffer setup: This code sets up the input buffers for the packet by loading U-registers with the buffer pointers and the buffer counts form the IOCB. Six memory references are needed to set up the buffer. In addition, the host address has to be loaded into U-registers as well, but

this can be done at less frequent intervals. If the host is promiscuous (host address zero), then the address recognition microcode will be different, since it will not have to provide filtering.

Address recognition: When a packet arrives this code is executed to determine whether the incoming packet has a matching host address, or has destination address zero indicating a broadcast packet. If the packet is accepted, it is not possible, due to the hardware latency requirements, to store the first two words of the packet into memory on the fly. These two words are saved in U-registers and are stored in the packet buffer *after* the entire packet has been received.

The case of a promiscuous host is handled separately. Different address recognition code is executed, and the first two words need not be stored away in U registers.

Receive main loop: The receive main loop executes one click per word, during which the Xerox Wire word is read into the memory buffer and tests made for both the end of the buffer and the end of the packet. The latter test is performed using the XBranch F-field value. When the end of the first buffer is reached, the second buffer of the packet has to be setup. A second main loop is used for executing with the second buffer since the end conditions are different for the two buffers.

Post input packet processing: After the packet has been received, some proceesing still has to be done before the next packet can be accepted by the microcode. This processing includes the storing of the 2 words which had to be saved, as described above (2 memory operations), the computation of the number of words received and storing this value in the IOCB (together with a packet status word - 1 or 2 memory operations), the wakeup of the software driver (1 memory operation), and the setting up of the next buffer (7 memory operations). Thereafter, the microcode is ready for the next packet. The timer to perform this processing constitutes a window during which any packet arriving will be missed.

Transmit buffer setup: A similar buffer setup procedure is required for transmit packets. Again a minimum of 6 memory operations are needed for this.

Transmit main loop: The transmit main loop is similar to the receive counterpart. Again, one word per click has to be transferred to or from the interface. In order to make this possible, it is necessary to make a hardware modification to the Wildflower processor as it presently exists. At present, the output bus IOOUT is a slow sink, which means that it cannot accept data from arithmetic operations, or from SU or MEMIN. The latter restriction prevents data transfers directly from memory (via MEMIN) to the I/O device (via IOOUT) in a single instruction. If this cannot be done, then single click output main loops will be difficult (if not impossible) to realize. The hardware fix envisaged would connect the W-bus (or just MEMIN) directly to IOOUT via a set of drivers. This change would be needed for the disk microcode as well.

Post output packet processing: The post processing consists of storing the packet status in the IOCB (1 memory operation), waking up the software driver (1 memory operation), and setting up the next buffer (7 memory operations). If a collision occured during the packet then the current packet would have to be set up again for re-transmission.

6.5.5 Performance, Cost, and Impact on Rest of Processor

6.5.5.1 Performance

The Xerox Wire has one click per round assigned to it with three clicks between service clicks (1.4 μS). The maximum service latency is 1.92 μS. At 10 Mbps, the Xerox Wire requires service every 1.6 μS, and with two words of hardware buffering, the maximum service latency tolerated is 3.2 μS. The usage of clicks for the basic data transfer is 10/11, since the maximum bandwidth of the data path is 11 Mbps.

Address recognition is done on the fly and if a packet is rejected, then the interface will be immediately ready for the next packet. Two clicks are needed to reject a packet. If a packet is accepted, then the cost is two clicks after the packet has been received completely in order to store the first two words which had been saved in U-registers.

After a packet has been completely received, the time until the next packet can be accepted is a function of the amount of post-processing required. This can be lower-bounded by the number of memory operations needed. About 12 memory operations are needed for this processing (12 clicks), which results in in a latency of 17.3 μS, before the next packet can be received. Thus, back-to-back packets (1.6 μsec apart) cannot be received by a host. The probability of two back-to-back packets for a single host is, however, small.

On output, about 9 memory operations are needed for post processing, which means that the minimum inter-packet spacing on transmission is about 13 μS.

When the microcode is idling about 4 clicks are needed to determine if there is any work to be done.

6.5.5.2 Resources used

Clicks: 91% of the assigned clicks per word during transfer.
R-registers: one permanent, one temporary (used only within a click).
U-registers: approximately 16.
Microcode size: approximately 200 instructions.

6.5.5.3 Hardware Size

The total chip count is 48 chips. Note that this is an estimate based on a block diagram design where some of the blocks have not undergone detailed design. The chips fall into the following categories.

SSI 25 (16 TTL & 9 ECL)
MSI 12
LSI 3
Discrete Components 8 chip equivalents

6.5.5.4 Changes to Processor from that Described in Wildflower Manual

A connection between the MemIn and the IoOut needs to be accomplished.
One of the F fields needs to be able to look at an attention bit from the Xerox Wire interface.

6.5.6 Present status of above and tasks necessary for complete design

All of above have yet to be done.

6.6 Low Speed Peripheral Bus

6.6.1 General architecture:

The low speed peripheral bus is an 8080 type system bus intended to support low speed peripherals (1.5 Mbit/sec. and less). There is an 8 bit data path to and from the microbus. The microbus gets zero to two clicks per round (DKDX) since it is second in priority behind the disk and Xerox wire (i.e. if they do not use their clicks, the microbus gets them).

Two possible methods of handling data transfers to and from the microbus are described.

Bare Bones Approach

The first uses microcode to toggle each of the control lines on the data bus to effect a transfer. (see Figure 6.6a) This requires a minimum of hardware, but the maximum transfer rate would be slow (3.7 Mbit/sec. with two clicks/round). It would take 3 clicks to perform a single input or output operation of one byte. This is due to the speed of the microbus (approx. 330 nS minimum bus clock period). Within 6 clicks (= 18 cycles distributed over 4.3 microseconds), the microcode would fetch a 16 bit word to be transferred, make 2 1-byte transfers to the interface, perform 1 byte swap, move the control lines to the bus, increment address, do an end condition test, and perform some background tasks in the remaining cycles. Startup would involve polling a status register which would point to the source of an interrupt as well as checking for presence of next byte to be transferred in U registers. This operation would take about one click.

Bus Handshaking and DMA Support

A second approach involves use of some extra logic to perform the bus handshaking and DMA support operations. (see Figure 6.6b) This would permit an interface to the processor which would be loaded with the controller address, a read/write command, and data (if write). The entire operation would occur within one round, thus consuming only one click per byte of data transferred for inner loop microcode. That is a data rate of 11.1 Mb/s. For this scheme, the I/O address space is only 6 bits, but can easily be made larger if there is a need. Only the read and write control lines are implemented in this scheme as they are the only ones needed by the chips being used.

In this scheme, the bus operates with two transfer modes (mesa and DMA) for low and medium speed devices. The bus is supported by device independent microcode for mesa I/O instructions and for DMA block transfers. The bus is designed to have device drivers written only in mesa. "Slow" interupts become mesa naked notifies to be serviced using mesa I/O primitives for transferring single bytes of data over the bus. The latency or overhead of mesa process switching is about 100uSec (assuming display on), after which transfers take about 3 mesa instruction (load, load, output)(load, input, store) or 5 uSec per byte. Data rates are proportional to the number of bytes transfered per interrupt. At one byte per interrupt, 9.6 Kbps can be serviced with about 12% of the emulator cycles (6% of processor). Keyboards, clocks, teletype and mouse use negligable cycles. To accomodate medium speed devices such as floppy disk controllers, printers and high speed communications (~500Kbps), there is a direct memory access (DMA) mode of block transfer which runs concurrently with mesa causing naked notifies only at the completion of

long blocks of transfer. Four workstation tasks implement four independent DMA channels, servicing devices as the Intel 8257 DMA chip would. The bus is shared between DMA channels and mesa I/O instructions. Mesa device drivers initialize DMA transfers by writing device control registers and passing parameters to the DMA microcode in an IOCB. The DMA task maintains an address pointer and a byte count, buffering bytes as it converts between words of memory and bytes for device data.

Devices on the bus can only cycle in 2 or 3 clicks, but the microcode for servicing a byte is only one click long. To minimize processor overhead, then, a bus control state machine grants bus requests, initiates reads and completes writes, generating task switches exactly as needed. This extra bus control costs only a couple chips and keep the processor overhead to one click per byte (a factor of at least three improvement over the minimum chip scheme).

Xerox Private Data

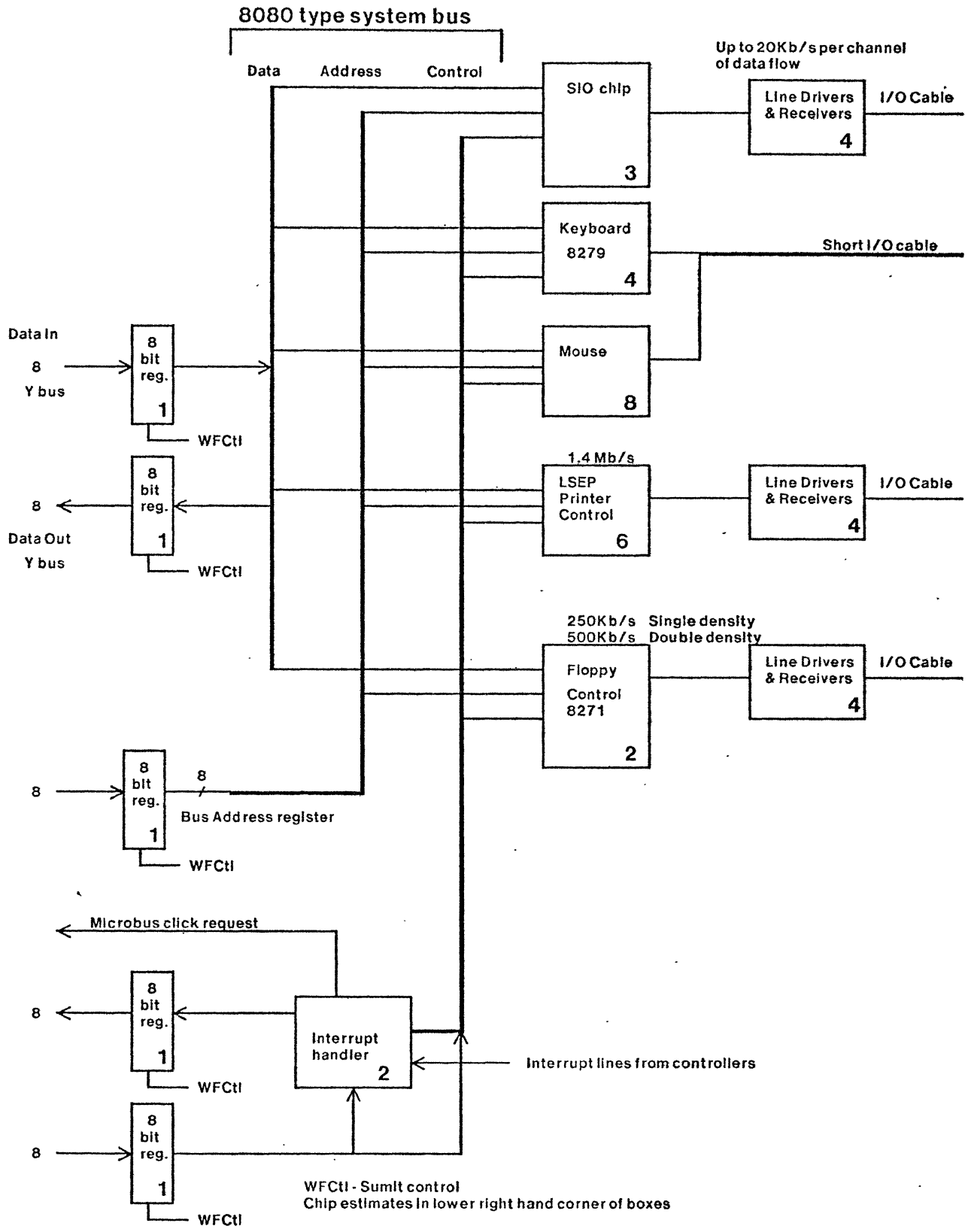
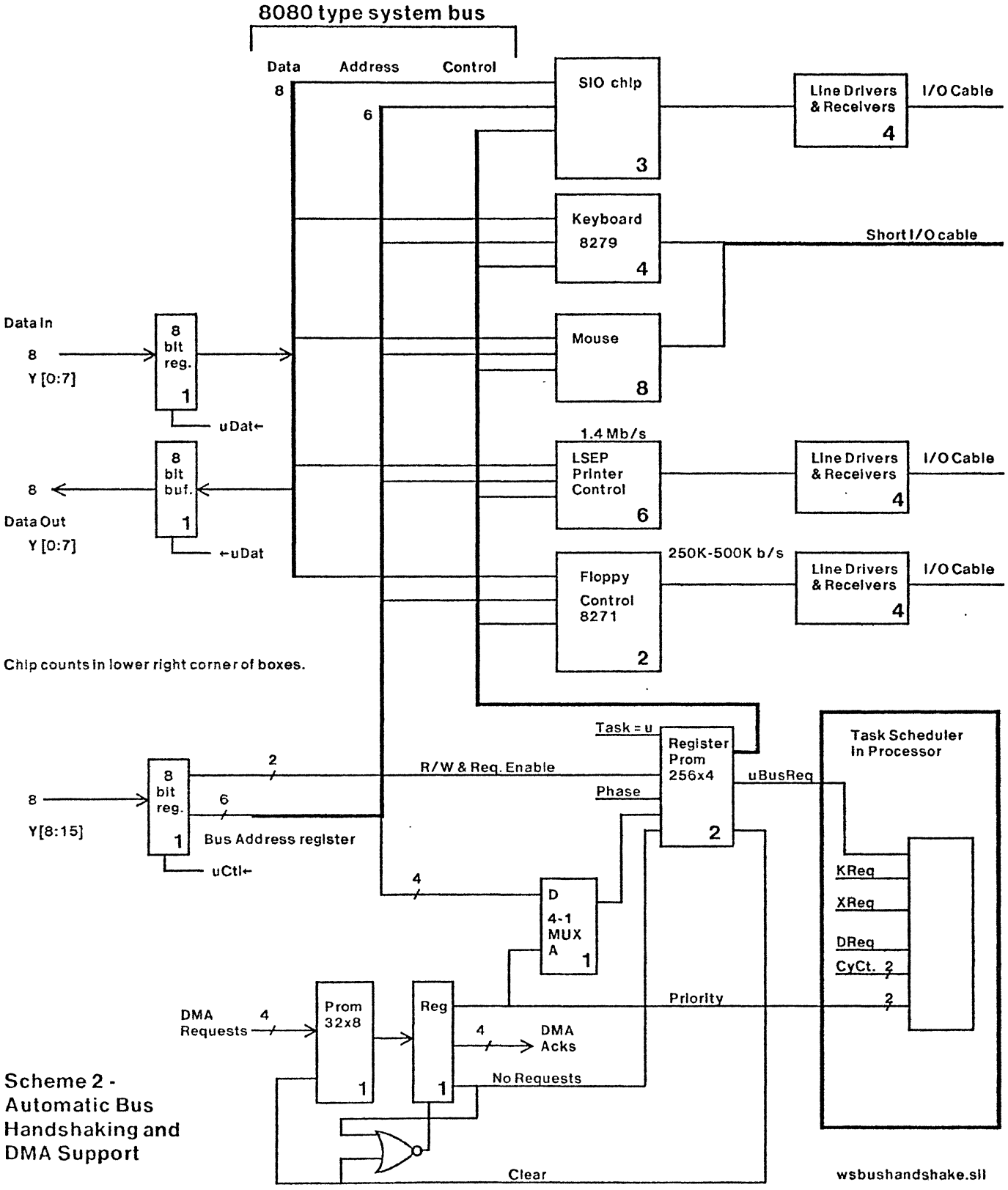


Figure 6.6a Workstation MicroComputer Bus
Scheme 1 - Bare Bones Register Control

wssimplebus.sil



Scheme 2 - Automatic Bus Handshaking and DMA Support

Figure 6.6b Workstation MicroComputer Bus

6.6.2 Assumptions/Requirements:

- o Bus will be an 8080 type bus, but unused control lines are omitted.
- o Bus will not extend outside of cabinet.

6.6.3 Hardware Interface

Bus-Processor Interface (see figure 6.6a)

The base interface described in the first approach has 7 chips; 5 1-byte registers handling data in, data out, address out, control out, and control in, and 2 for gating and synchronizing requests. The second approach uses different chips with a net gain of 2 chips to perform the bus handshaking and DMA suppport.

Floppy Disk controller (Intel 8271 FD Ctl. or Western Digital 1791 double density)

This controller is an LSI chip, address decode and line drivers and receivers making a total of 6 chips. This chip would be supported by the DMA in scheme 2.

Serial Line controller (Zilog SIO, Intel 8251 USART, or 8273 SDLC)

Also LSI, similar to the floppy disk controller, this would take about 7 chips per RS-232 type channel termination. This chip would be supported by the DMA in scheme 2.

Auxiliary media (Tape cassette, IBM Mag Card, Xerox Mag Card, IBM compatible Floppy)

Interfaces to the 850 series peripherals could be done through an 8080 type interface similar to the one employed in the 850 itself.

Clock (time of day)

The output of a low power (with battery backup) clock chip could be gated to the bus through a simple buffer. Timers might also be included, including a watchdog timer which caused a boot if it expires.

Keyboard & mouse (8212 latch for mouse U/D ctr., maybe 8279 interface for keyboard)

LSEP (printer, high speed = 1.4 MB/s)

Estimated chip count is 6 if on microbus. This chip would be supported by the DMA in scheme 2.

Bell

This is a simple audio output device taking 1 chip (address decode) and a transducer.

6.6.4 Microcode

The elementary microbus operations of read and write require 3 clock cycles of the bus. During the first cycle the device address is placed on the bus. In the second cycle, the operation (read or write) is specified. In the third cycle, the operation takes place.

Microcode to perform the above could perform 1 bus cycle operation per click for the first approach. With handshaking hardware added, all 3 operations could be specified in a click, with the hardware taking the time between clicks (.72 μS) to actually do the operation.

Read and write I/O register - Mesa I/O instructions could be made to perform these in about 3 clicks each.

DMA Support via Mesa - A simple, but slow scheme would be to have mesa software keep track of DMA word counts and addresses. It would take about 100 microseconds to transfer 2 bytes using interrupts to mesa, most of the time doing process switching.

6.6.5 Performance, Cost, and Impact on Rest of Processor

Bandwidth

The bandwidth numbers mentioned earlier are peak inner loop bandwidths. They are 3.7 Mb/s, and 11.1M b/s for schemes 1 and 2 respectively. Time must be added for sorting out the source of interrupts and communicating with mesa software and the emulator. The overhead for this function depends on the block size to which it is allocated. If the fastest device on the simple bus is 250K b/s, and only one or two are running at a time, there should be no problem, but this needs to be analyzed in more detail.

Processor Resources % of clicks used, registers, etc.

For running the floppy disk at 250K b/s, about 8% of the available clicks would be used during the transfer phase for the first scheme. 3% would be used in the second scheme.

Hardware Size

The hardware items will be itemized below, first listing the 3 schemes, each adding to the other, and then the peripheral interfaces plugged into the bus.

Section	Total	SSI	MSI	LSI
Scheme 1 Bare bones	7 chips	2	5	
Scheme 2 Auto handshaking	+2 chips	1	6	2
SIO chip & interface	7 chips	5	1	1
Keyboard interface	4 chips	3		1
Mouse interface	8 chips	2	6	
LSEP (printer) interface on bus	6 chips	6	3	1
Floppy controller	6 chips	4	1	1

Microcode Size

Not enough microcode has been written to get a good estimate of size.

Registers Used

Register usage will fall into two classes. The first is the basic set of registers necessary to support polling the interface and storage of task state. In addition, a second class will need an address register and word count for each active transfer in progress. Scheme 2 uses only U registers.

6.6.6 Present status of above and tasks necessary for complete design

All of above yet to be done

6.7 Performance Estimates

Tables 6.7.1 and 6.7.2 give a summary of some statistics describing how the processor cycles are shared between the I/O devices and the emulator. The device bandwidth (BW) numbers are intended to represent maximum rate data transfers by the devices, and do not take in to account any overhead associated with the data transfer. These calculations assume the (DKDX) click scheduling arrangement, with the low-speed peripheral bus taking unused rigid disk and Xerox Wire clicks. The cycle time is assumed to be 120 nS; a click is thus 360 nS, and a round is 4 clicks or 1.44 μS. The tables are intended only to provide an overall picture of the processor sharing, and the estimates are not to be considered as highly accurate.

From Tables 6.7.1 and 6.7.2 we can compute the following workstation load estimates. The Mesa performance estimates assume the instruction mix which includes Xfers, as described in section 6.2.6. The Mesa performance values are pessimistic since they assume the simultaneous, full-bandwidth activity of each device. Except for the display, most of the devices have a relatively low duty cycle.

Load	Processor usage by I/O [%]	Processor for emulator [%]	Mesa Performance [Kips]
No I/O load	0	100	840
Display only	50	50	420
Display, SA-1000 disk	69	31	260
Display, SA-1000 disk, Xerox Wire	92	8	70
Display, SA-1000 disk, Xerox Wire, floppy disk	93	7	60

Table 6.7.1 Device sharing of processor cycles

Device	Clks/round (Max. latency) [clicks (uS)$^{\$}$]	Path width [bits]	Max. data path BW [Mbps]	Max. device BW [Mbps]	Processor usage [%]$^{\#}$
Display/memory	2 (.72)	32	44.4	44.4	50
Disk/processor					
SA-4000	1 (1.92)	16	11.1	7.1	16
SA-1000	1 (1.92)	8	5.6	4.3	19
XWire/processor	1 (1.92)	16	11.1	10	23
Low speed bus/processor	+	8	+	*	*

Table 6.7.2 Low-speed bus peripherals

Device	Max. device BW [Kbps]	Processor usage when active [%]
Floppy Disk (SA-800, Aux. media)	250	1.13
Serial line (2 channels @ 20Kbps, full duplex)	80	.36
Electrostatic Printer (200 SPI)	250	1.13
LSEP (300 SPI)	1400	6.25
Xerox Card (aux. media)	6.2	.03
IBM Card (aux. media)	8.3	.04
Xerox Tape (aux. media)	6.5	.03
Keyboard/Mouse	7	.03
Character printer (30 CPS Diablo)	.6	.003

Notes:

\+ The low-speed peripheral bus has no guaranteed bandwidth since no click in a round is permanently assigned to it. The low-speed bus will take unused rigid disk or Xerox Wire clicks for its service. It may be necessary in some cases to turn off the Xerox Wire in order to run a relatively high speed device such as the LSEP on the low-speed peripheral bus. Changing the allocation of clicks to the display to 7/16 instead of 1/2 (and adding appropriate display hardware buffering), and assigning the extra 1/16 click to the low-speed bus, will guarantee a bandwidth of 1.4 Mbps for the low-speed bus.

* Depends on device configuration on bus.

$ Calculated from $3Td + 7T$, T is cycle time, d the max. number of clicks between (potential) service clicks.

\# Percentage of processor cycles used. Calculated from:

(Fraction of clicks allotted) * [(Max. device BW) / (Max. data path BW)]

6.8 Packaging/EME/Environmental

6.8.1 General considerations:

Processor, power supply, & peripheral controllers are in a separate box.
Must meet UL (Underwriters Laboratories) and CSA (Canadian Standards Association) requirements.
Must meet FCC requirements.
Must meet Xerox multinational requirements (which supposedly include above)

6.8.2 Packaging:

Power switch and boot button location should be such that they are not inadvertently actuated. A possibility is a keyswitch for power.
If two board configuration, one board can have the peripheral controllers and other parts which change with differing configurations.

6.8.3 EME considerations:

Use shielded cable on all signal cables.
Balanced signalling will be used for signals going to equipment not powered by local power supply.
Single ended signalling can be used for signals going to external boxes operating from local power supply and not having any other external connections.

6.8.4 Environment

All units should operate in $50^{o}C$ ambient.
Fans, if used, should be quiet ($\leq$40 dBA ?)
Impact printer should have acoustic cover and be designed not to overheat with cover in place.

6.8.5 Impact on Cost and Performance of Workstation

Balanced drivers require more chips and board space.

6.8.6 Present status of above and tasks necessary for complete design

All of the above need to be examined in more detail. Signalling to meet EME must be addressed early in the design effort. Environmental considerations can be addressed once the circuit size and physical partitioning have been settled.

7. Development Tools

7.1 Introduction

This section describes the software and hardware development tools that are considered necessary or desirable in the development of the LSI Workstation. Consideration is given to the various forms of prototype machine which may be required, together with the programming tools which will be used. Some recommendations are put forward as to the desirability of using modified forms of existing tools compared with the effort required in constructing completely new tools.

The hardware and the software tools will be treated in separate sections.

7.2 Hardware

7.2.1 Engineering Design Models

As the Workstation is in the process of being designed it may desirable to build some prototype hardware. Since the design process is essentially incomplete, this hardware is very likely to change. It would therefore be undesirable to make firm allocations of chips to geometric positions on the boards, or even decisions such as which chips are to be placed on which boards. The stitchweld technique as used in the D0 would be appropriate for this version of the Workstation, which will be amenable to changes in architecture and detailed design at all times. The well-established technique of using the programs SIL to draw the circuit diagrams, Analyze to map the connections onto pin numbers, and Route to build the wiring schedule would again be applicable to this design.

Similarly, the microcode will be subject to change, and hence it will be necessary to build the control store in RAM, rather than the proposed PROM. Thus it will be possible to make changes to the microcode in order to test out the various parts of the firmware system.

For debugging purposes also, it would be desirable to have access to certain information within the machine, such as the contents of registers, of the control and main memory busses. This implies that some means of carrying this information out to an accessible point, for instance the backplane, should be provided so that these signals may be used by such devices as a logic analyser.

A number of other micro-code debugging aids could most conveniently be implemented in hardware. These include the ability to halt the execution of micro-instructions when certain conditions arise, such as the execution of a particular micro-instruction, the execution of a micro-instruction at a particular control store address (hardware breakpoint), on alteration of a particular register or the raising of a pre-defined condition. Only the registers which are available between chips will of course be available in this way, which precludes the inspection in hardware of Wildflower's R-registers. They can of course be inspected by the use of suitable micro-code (of which more later). The stack, the U-registers, the memory address register and bus, and the I/O busses will however be available given suitable hardware hooks in the Engineering Design Model.

One further hardware provision which should be made to aid debugging both of the hardware design and the micro-code is the so-called umbilical cord. This is the technique currently in use whereby D0 microprograms may be run under the supervision of the MIDAS program running in an Alto. It is not necessary for special hardware to be provided for this facility, as this connection may be established through one of the regular ports on the Workstation (for example, the low speed peripheral bus). All that is required is that a suitable kernel be written in micro-code to handle the communication with the host machine.

Such hardware hooks may also be useful when attempts are made to measure the performance of the Workstation. If the registers which are open to inspection include the Mesa program counter and the Mesa instruction buffer it will be possible to measure (say) the speed and frequency of execution of certain Mesa instructions. Measurements of the hit-rate of the memory map, and the frequently executed portions of Mesa code can also be detected. At a lower level, measurements of the way the various micro-code tasks are executing can be made, as can the frequency with which the disk and Xerox-wire tasks relinquish their clicks to the Mesa emulator. With these measurements some assessment of the effect of these devices on the overall performance of the system can be ascertained.

7.3 Software

7.3.1 Assembly and Instruction Placement

The language in which microprograms are specified would probably look somewhat similar to the microprogramming language of either the D0 or the Alto. Examples of Wildflower micro-code are available written in a dialect of the Alto microprogramming language. This code then has to pass through some kind of assembly process.

Micro is an example of a general purpose micro-programming language assembler, which is tailored to a particular machine by the provision of a suitable file of macros. These macros provide the expansion into bit patterns of the various constructs provided in the language. In the case of Micro, a file called D0Lang is first read by Micro, and this is followed by the source language version of the microprogram. This approach works reasonably well, although there seems to be a problem with regard to producing informative error messages.

The output from Micro is then passed to MicroD, which is responsible for assigning control memory locations to the micro-instructions. In order to do this, MicroD, too, requires a description of the underlying machine. MicroD produces a binary output file which is suitable to run on the D0.

7.3.2 Run Time Environment

Once the microprogram has been written and processed by Micro and MicroD, it needs to be loaded into the machine and run. If the development work, (editing of source program, assembly etc to produce binary code) is to be done on a separate and different machine, some means whereby the program can be transferred to the control store of the Engineering Design version of the Workstation must be provided. This suggests that the umbilical cord approach is mandatory so that microprograms may be loaded down line to the machine. Once there, they can either be left to run, with very little being available to inform the programmmer whether the program is running correctly, or else some debugging aids may be called in to assist.

The D0 has associated with it a program called MIDAS, which runs on an Alto at the 'mother' end of the umbilical cord, and is able to monitor the progress of a microprogram running in the D0. In order to provide the necessary services of inserting breakpoints, inspecting registers and control and main memory, single stepping of instructions and instruction tracing, MIDAS itself has to carry a model of the D0 architecture.

In order for MIDAS to be able to communicate with the microprogram running in the target machine, some (hopefully small) piece of micro-code must be installed in the control store of the target machine. This microprogram is known as the kernel. Two problems seem to occur as a result of the kernel being present. Firstly, the kernel occupies control memory space, leaving less memory available for the microprogram under test. This problem may be solved by providing the development machine with more micro-code memory space than the final production model. The

second problem is that of sliding the kernel out of the micro-code, to leave a clean, properly debugged version of the micro-code behind, ready to be burned into the PROM.

7.3.3 Simulated Run Time Environment

Consideration has been given to the question of whether a simulator for the Workstation should be designed and built. The conclusion seems to be that such a simulator should be constructed for a number of reasons.

(a) It is likely that a simulator would be available before the Workstation hardware is complete. Thus, microprogrammers can obtain experience of how the machine should be programmed at an early stage of the project.

(b) The exercise of programming the Workstation could possibly identify faults in the hardware design, so that experience with the simulator could be fed back to the hardware designers before too much actual hardware had been laid down.

(c) A piece of software, such as a simulator, which runs on an available machine, such as an Alto, is more accessible to programmers than a new piece of hardware, and so there will be less contention for machine time if programs can be tested using the simulator.

(d) Provided the simulator and the hardware are at all times functionally equivalent, it should be possible to provide a system in which programs can be run on either the hardware or the simulator without any noticeable change in the user interface. In fact, it may be possible to run a program on both the hardware and the simulator in parallel and compare the results at suitable points. In this way, both the hardware and the simulator are continuously being checked against each other.

(e) Because the Wildflower architecture has as its overriding goal to keep the number of components as low as possible, the provision of a simulator will assist in ensuring that the design is complete as the micro-code is being produced. It will preempt any enlargement of the design as the project proceeds.

7.4 Recommendations

The difficulty experienced by D0 users in regard to the development software available (i.e. Micro, MicroD and MIDAS) is that each of these three programs requires a separate description of the D0 hardware on which to operate. There is therefore a considerable problem of consistency between these various descriptions. Exactly what is the cost in terms of programmer time of these inconsistencies is difficult to estimate, but it seems that a system in which all the components use a single description of the hardware would be a great advantage.

One other consideration at this point is that the existing Micro/MicroD/MIDAS system is relatively old and BCPL based. We would therefore propose that a new system be constructed, based on Mesa, which would be an integerated package allowing source microprograms to be assembled, linked, instructions placed, and run either in the real machine or on the simulator or both. The various parts of this integrated system would take their own model of the hardware from a single description data base, which would be kept up to date in complete synchrony with the development of the hardware design.

The user interface to this system should be very carefully designed, and in particular, the interface to the simulator/debugging system should be a common one. Thus the same set of commands

would be used to specify similar actions whether the microprogram was being run on the simulator or on the real hardware.

8. Field and Factory Diagnostics

8.1 Introduction

This section addresses the problem of diagnosing faults in the Workstation. We distinguish between fault diagnosis at the site of the Workstation, where faults would be corrected by replacing components of the system, and the repair of field replaceable items, which would take place in a factory environment.

8.2 Assumptions

The following assumptions are made about the physical and architectural features of the Workstation hardware. The Workstation will physically consist of:

Processor Card
Configuration Card
Display
Keyboard and Mouse
Rigid Disk
Floppy Disk
Xerox Wire
Serial Line Controller
Other devices

The processor card is assumed to be the same for all installations. The configuration card will vary from workstation to workstation according to the size of memory associated with the system, the devices that are attached and so on. The display, keyboard, rigid disk and floppy disk are all standard components and will have their own diagnostic procedures.

The assumption is also made that it would not be possible to make field modifications to these components, and that if any unit is found to be malfunctioning, then it will be returned to the factory for repair.

8.3 Field Diagnostics

In this section we make the further assumption that field engineering equipment and field engineer training is to be kept to a minimum. Also it is desirable that the field engineer will have sufficient information from the report of the fault, that he can be reasonably certain that his initial diagnosis is the correct one, and hence a single visit to the site is sufficient.

In the field, it is necessary to attempt to isolate the fault to within one of the components mentioned in 6.8.2. As stated above, it is not necessary for the field engineer to discriminate any further; this will be a task for the factory repair procedures. The field engineer will however require some diagnostic aids to ensure that it is the faulty component which is replaced.

In view of this last assumption, it must be possible for the operator of the workstation to be able to communicate the symptoms of the fault to the field engineer in such a way that the engineer can know to within a reasonable degree of confidence which component is faulty before he leaves his base, taking with him a replacement component to be substituted in the field site. What is required is some kind of programmable display, similar to the D0's maintenance panel numbers, to inform the operator of the status the system.

This diagnostic micro-code will be designed so that if it executes correctly, (i.e. the maintenance panel indicates a successful boot), then the processor and configuration cards may be assumed to be working correctly. That being the case, it should then be possible to write diagnostic programs to exercise the individual devices without changing the microprogram (i.e. diagnostic programs written in Mesa). In order to load and run such programs, without being sure of the reliability of any of the devices, and in particular the disk, the machine must be provided with alternative methods of performing an initial load of a program. It is assumed that the normal system booting (after initialisation of the machine) is achieved by reading certain information from the disk. However, it may be necessary for the field engineer (or operator) to force the micro-code to perform its initial load from some alternative place such as the Xerox-wire or floppy disk. This may be achieved by attempting a boot through one controller (e.g. the disk), and if that fails, then an alternative boot device is tried, such as the floppy disk Failing that, the micro-code can then attempt a boot from the Xerox-wire. The initial load can then be forced to take place from a particular controller simply by truning off the other devices.

8.4 Factory Repair

Once the faulty component has been identified and replaced, it may then be returned to the factory where a thorough and detailed examination may be carried out with a view to repairing the component. In the case of the processor and configuration cards it might be expected that repair of these components would consist of isolation and replacement of one or more faulty chips. Repair of faulty components at the factory will allow the use of much more sophisticated, programmable, test equipment to isolate the offending chip or chips.

8.5 Repair of Peripheral Devices

As stated in section 6.8.2, the peripheral devices, being standard well-tested pieces of equipment, will probably have their own diagnostic procedures. These devices would not necessarily need to be replaced as a unit in the case of their malfunctioning, but the precise action to be taken would depend totally on the device and the precise nature of the fault.

Once the fault has been diagnosed as being located within a particular device, the correct fault diagnosis and repair procedure must be ascertained from the specifications of the device.

9. OIS Architectural Impact

This chapter looks at how the SUMIT conforms to the *OIS Principals of Operation* [Reference 3] document and how Pilot might have to change.

9.1 Mesa Instruction Set

The microprocessor should be able to implement all the mesa instructions, including Xfer, BitBlt, and the process opcodes, and is thus OIS compatible in this sense (See section 6.2). As currently designed, there are no special restrictions as to placement of frames or code segments.

9.2 Virtual Memory Size

The machine is currently designed for a 20 bit virtual address, and therefore, out of a 128K word memory, 1/32 of it (16 pages) holds the map. The machine could be easily expanded to a 22 bit virtual address, but this would fill 1/8 of real memory (64 pages). The current PrincOps document specifies a virtual address size of 24 bits, and the D0 implements 22 bits.

9.3 Virtual Addresses: When are they mapped?

PrincOps currently states that the memory system is given vitual addresses and, for every reference, mapping hardware of some sort performs the translation to real addresses. This is obviously a very strict limitation, and it is not clear that it is necessary in order to implement a virtual memory system.

SUMIT maps only when necessary: once per local or global frame setup, once per MDS relative read/write opcodes (e.g. RB), once per long pointer opcodes, and when the Mesa program counter crosses a page boundary or jumps into a different page. This mapping scheme works fine for the emulator, but I/O devices present a special problem.

9.4 I/O and Virtual Addresses

The emulator can leisurely remap cached real addresses when they cross page boundaries, but the I/O microcode generally cannot while in inner transfer loops. This leads to the restriction that if any device must transfer more than one page of data, then the physical pages must be contiguous. The display, Xerox Wire, and possibly the micro-computer bus suffer these limitations. If the display were turned off in order to acquire more memory space then it must reclaim a contiguous set of pages, implying a remap of all of memory.

The I/O controllers have also been designed assuming that the Controller Status Blocks and I/O Control Blocks contain real addresses. This is radically different from the current PrincOp's IOCB design.

In summary, if PrincOps is to specify any kind of I/O architecture, it will not be below the level of CSB's and IOCB's, and even then the format will have to allow machine dependent encodings (such as virtual versus real addresses).

9.5 Other Aspects of PrincOps

In additon to Mesa and I/O, PrincOps also talks about things like double bit errors, restart registers, and interruptable instructions. SUMIT has none of these since it is not a D0. Of the list of traps on page 62 [of Reference 3], the Wildflower currently does not recognize a stack error (i.e., poping an empty stack or pushing a full one), nor does it notice a virtual address out of range.

9.6 Risks

With regards to PrincOps, there is essentially one risk involved: getting Pilot to run with the above modifications. Perhaps 20 per cent of Pilot must change. Note that we cannot continue to radically change Pilot for each next machine since the PrincOps document no longer makes sense.

10. Development Schedule

The estimate below reflects the use of the SUMIT processor. The estimate was developed in a meeting in Palo Alto on November 1, 1978.

Two software tools are central to the schedule below. First, a version of the D0 has to be developed to look exactly like the workstation both in hardware and software. This allows the parallel development of high level software and Mesa level device drivers.

Second, a software simulator for the workstations processor is needed to allow the parallel development of the workstations microcode. Neither of these two projects has been completely scoped.

Months from Start	Completed Tasks
	Completed before start are:
	Machine definition, via *OIS Processor Principles of Operation.* This means that the target machine organization is defined.
	Machine performance goals.
	Machine configuration and I/O interfaces.
0	Start of Design.
4	Microinstruction format and function defined to allow finalization of the support tools.
7	SIL drawings and microcode for the machine are completed.
8	Fabrication of the first machine is complete.
9	Unit test complete. This means that the stitchweld boards that makeup the machine pass the card tester.
13	Nucleus test complete. This means that the machine runs the diagnostic routines designed to verify the operation of the hardware.
	This is an integrated processor and major peripherals. Drawings now released for etch.
15	Engineering model now runs standalone Mesa programs.
16	End of first 3 month etch cycle.
17	Engineering model now runs Pilot 5.0 with the Arcturus Star software.
18	Engineering model now runs Pilot 5.0 with the Canopus Star software.
19	End of second 3 month etch cycle.
20	Engineering model now runs Pilot 6.0. This is the IMO software.
27	IMO.

11. Cost Estimate

Warning. The estimates here are rough. The items marked with an * make up the nucleus in the Star LSI Workstation Goals. The bulk of the workstation cost is dependent on peripherals. The nucleus subtotal is $2767 versus the $2155 in the goals document.

Processor and Integral Peripheral controllers (parts costs only)

Part	Proc.	Disp.	Disk	XWire	LSPB	Mem.	Total	$
SSI chips @$.30	40	10	10	25	21	3	109	$ 33
MSI chips@$1.00	90	10	11	12	17	11	151	$151
LSI chips @10.00	10	0	0	3	6	1	20	$200
64K Rams@$20						34	34	$680
PC card(s)@$100							3	$300
Other @$1.00				8			8	$ 8
Totals	140	20	21	48	44	49	322	$1372 *
							(262 over goals)	

Display

Display	$ 230 *
Covers & cable	95 *

Keyboard and mouse:

Keyboard, cover, cable	$ 160 *
Mouse, cover, cable	110 *

Power Supply

Pwr. for processor, FD, RD and XW xcvr. (assumes $2.00/watt & total power = 275W)	$ 550 * (350 over goals)

Cabinet

Cabinet and fans	$ 250 *

Rigid Disk

SA 1000	$ 480

Floppy Disk

SA 800	$ 200

Character Printer

Lombard	$ 830

Xerox Wire Transceiver

Transceiver, interface cable (vinyl)	$ 50
Coax cable (teflon)	$ 2.00 - 3.00 per foot installed

Appendix A. Microcode

Xerox Private Data

```
; Full Page Display Microcode
; wfdisplay.mc last edit by Jarvis on November 2, 1978  8:43 AM

; Display gets three clicks per round.  These clicks are named A, B, and C.
; When the A click starts, there are no pixels stored in the auxilliary
; buffer.  When the B click starts, there are eight pixels in the auxilliary.
; At the beginning of the C click four pixels remain in the auxilliary.

; There 39 clicks per scan line:  32 for data and 5 for retrace.  The last
; data comes on a C click.

; The code does not process a linked list of IOCB's, instead, it refreshes the
; display from a buffer whose location is fixed the virtual space.  Each page,
; 256 words, has exactly the storage required to represent four scan lines.
; Every fourth scan line during the retrace interval, the code performs a map
; reference to find the physical page used to represent the next four scan
; lines.  To save physical memory, the code could display unmapped virtual
; pages as background.

; Memory refresh:  The TMS 4164 requires 256 refresh cycles every 4 msec.
; or a cycle every 15.6 usec.  It takes 31.2 usec to scan one line, therefore
; memory refresh requires two refreshes per scan line.

; Tasks which must be performed during horizontal retrace
; 1  set horizontal blanking
; 2  2 memory refreshes
; 3  check for vertical retrace
; 4  map storage
; 5  maintain processor clock?
; 6  check for cursor?
; 7  fudge for interlace (add words per line into address register)

; R registers
;   DAdr holds address of next double word of bit map

; U registers  ** denotes constants precomputed during vertical retrace
;   uDFence  holds address of last word in line + 2
;   uCFence** holds address in bit map where cursor is inserted
;   uCAdr** holds address of first 32 bit segment of cursor
;   uCAdr1** holds address of second segment of cursor

; main loop for lines on which the cursor appears  It is not necessary to
; check for the end of line since the cursor is guaranteed to come before
; the end of line.
Ac:   MAR ← DAdr, DAdr ← DAdr+2;                {1}
      sink ← DAdr xor uCFence, F=0;             {2}
      Display ← uAConst, goto[Bc];  * or doCursor  {3}

Bc:   MAR ← DAdr, DAdr ← DAdr+2;                {1}
      sink ← DAdr xor uCFence, F=0;             {2}
      Display ← uBConst, goto[Cc];  * or doCursor  {3}

Cc:   MAR ← DAdr, DAdr ← DAdr+2;                {1}
      sink ← DAdr xor uCFence, F=0;             {2}
      Display ← uCConst, goto[Ac];  * or doCursor  {3}

; writes the two 32 bit segments of the cursor.
doCursor: MAR ← uCAdr;                          {1}
      DAdr ← DAdr + 4;                          {2} skip over words in bit map
      Display ← uC0;                            {3}

      MAR ← uCAdr1;                             {1}
      sink ← uCDispatch, YDispatch;             {2} fall into loop in phase
      Display ← uC1, goto[endCursor];           {3}
; uCDispatch is precomputed during vertical retrace handles the four possible cases:
; (1) cursor ends on A click, (2) cursor end on B click, (3) cursor ends on C click,
; and (4) cursor ends on C click and is at the end of line
```

```
; Main scan line loop if no cursor on line or past the cursor
A:    MAR ← DAdr, DAdr ← DAdr+2;                {1}
      nop;                                      {2}
      Display ← uAConst;                        {3}

B:    MAR ← DAdr, DAdr ← DAdr+2;                {1}
      nop;                                      {2}
      Display ← uBConst;                        {3}

C:    MAR ← DAdr, DAdr ← DAdr+2;                {1}
      sink ← DAdr xor uDFence, F=0;             {2}
      Display ← uCConst, goto[A]; * or hRetrace {3} always ends on C click

; Horizontal retrace -- use the low order bits of system clock for refresh address
; 7 clicks is all we get!
hRetrace: MAR ← uClock;                         {1} click 1
      nop;                                      {2}
      Display ← HorizontalBlankOn;              {3}

      MAR ← Rt ← uClock + 2;                    {1} click 2 impossible
      Rt ← Rt + 2;                              {2}
      uClock ← Rt;                              {3}
```

Xerox Private Data

```
; Rigid Disk Microcode
; last edit by Jarvis on October 23, 1978  2:53 PM
;; assumption:
;;   can load MD from loIn in same cycle and
;;   can load loOut from MD in same cycle  (accomplished by tri-state buffers
;;      on MemIn with outputs on Y?)

;; If the disk hardware has a CRC chip, I think the ucode
;;      can get away with only a single R register just like the ether.

; R register
;   KAdr     address of next word of field
;   KCks     checksum during read, transferred data during write
;   KAc      byte assembly
;   Rt       tempory, trash between clicks

; U registers
;   KFence   address of last word of field

; main disk write loop (order is low, high) to fix play games: KData ← H.0-7?
!1, 2, byte0ContOut, KCksOut;

byte0Out: MAR ← KAdr, KAdr xor KFence, F=0, rKReq;  {1}
        nop, goto[byte0ContOut];                    {2} KCksOut for F#0
byte0ContOut: KData ← KAc ← MD;                     {3}

        KCks ← KCks xor KAc, rKReq;                 {1} checksum
        KAc ← KAc lcy 8;                            {2}
        KData ← KAc, KAdr ← KAdr+1, goto[byte0Out]; {3}

KCksOut: KData ← KCks, rKReq;                       {3}

        KCks ← KCks lcy 8;                          {1}
        nop;                                        {2}
        KData ← KCks;                               {3}


; main disk read loop (order is high, low)
!1, 2, byte0ContIn, KCksIn;

byte0In: KAdr xor KFence, F=0, rKReq;               {1}
        KAc ← KData, goto[byte0Cont];               {2} KCksIn for F#0
byte0ContIn: KAc ← KAc lcy 8;                       {3}

        MAR ← KAdr, KAdr ← KAdr+1, rKReq;           {1}
        MD ← KAc ← KAc or KData;                    {2}
        KCks ← KCks xor KAc, goto[byte0In];         {3} checksum

KCksIn:                                             {3}
```

```
; Wildflower microcode for Xerox Wire.
; last edit by Roy Ogus on October 26, 1978  6:27 PM
; stored on: wfXWire.mc.
;; assumptions:
;;   can load MD from loIn in same cycle and
;;   can load loOut from MD in same cycle  (accomplished by tri-state buffers
;;      on MemIn with outputs on Y?)

;; Notes:
;;         Only key portions of the microcode are shown.
;;         Have to still work out XBranch dispatches.


; R registers
;         XAddr     address of next word of field
;         XTemp     temp register, not used between clicks

; U registers
;         XPtr0Lo  Pointer to buffer 0 (low)
;         XPtr0Hi  Pointer to buffer 0 (high)
;         XLast0   Address of last word of buffer 0 of packet
;         XPtr1Lo  Pointer to buffer 1 (low)
;         XPtr1Hi  Pointer to buffer 1 (high)
;         XLast1   Address of last word of buffer 1 of a packet
;         XHA0     Host address (low)
;         XHA1     Host address (high)
;         PktDest  Register marking whether broadcast or host match

;  Address recognition.
!1, 2, NotHA0, GoodSoFar
!1, 2, NotForMe, ForMe

; assume that RH is set up already.
; check first word in = HA0
RecWakeup:      XAddr ← XData, XBranch;                          {1}
        Sink ← XAddr xor XHA0, F=0;                              {2}
        GOTO[GoodSoFar], rXWReq;                                 {3}

; first word matched, check second word.
GoodSoFar:      XTemp ← PktForMe, XBranch;                       {1}
        Sink ← XData xor XHA1, F=0;                              {2}
        PktDest ← XTemp, GOTO[ForMe], rXWReq;                    {3}

; first word didn't match, check for broadcast
NotHA0: XTemp ← BCPkt, XBranch;                                  {1}
        Sink ← XData xor XAddr, F=0;                             {2}
        PktDest ← XTemp, rXWReq, GOTO[ForMe];                    {3}

; address match, or broadcast.
ForMe:  MAR ← XAddr ← XPtr0Lo, XBranch;                          {1}
        MD ← XData;                                              {2}
        XAddr ← XAddr + 1, rXWReq, GOTO[ILoop0];                 {3}

NotForMe:       {reject packet}

; INPUT main loop
!1, 2, IMore0, INone0;
!1, 2, IMore1, INone1;
!1, 2, IMore1, INone1;

; Input main loop while storing into buffer 0
ILoop0: MAR ← XAddr, XAddr ← XAddr + 1, XBranch, GOTO[IMore0];   {1}
IMore0: MD ← XData, GOTO [ICont0];                               {2}
ICont0: XAddr xor XLast0, F=0, rXWReq, goto[ILoop0];             {3}

INone0: MD ← XData;                                              {2}
        RH ← XPtr1Hi;                                            {3}
```

```
; Input main loop - transition to buffer 1
        MAR ← XAddr ← XPtr1Lo, XBranch;                          {1}
        MD ← XData;                                              {2}
        XAddr ← XAddr + 1;                                       {3}

; Input main loop while storing into buffer 1
ILoop1: MAR ← XAddr, XAddr ← XAddr + 1, XBranch, GOTO[IMore1];   {1}
IMore1: MD ← XData, GOTO [ICont1];                               {2}
ICont1: XAddr xor XLast1, F=0, rXWReq, goto[ILoop1];             {3}

INone1: MD ← XData;                                              {2}
        {check for buffer overrun}


; OUTPUT main loop
!1, 2, OLoop0, ONone0;
!1, 2, OLoop1, ONone1;

; Output main loop while storing into buffer 0
OLoop0:         MAR ← XAddr, XAddr ← XAddr + 1, XBranch;         {1}
        XAddr xor XLast0, F=0, rXWReq;                           {2}
        XWData ← MD, GOTO[OLoop0];                               {3}

; Output main loop - transition to buffer 1
ONone0:         MAR ← XAddr, XBranch;                            {1}
        RH ← XPtr1Hi, rXWReq;                                    {2}
        XWData ← MD;                                             {3}

        MAR ← XAddr ← XPtr1Lo, XBranch;                          {1}
        XAddr ← XAddr + 1, rXWReq;                               {2}
        XWData ← MD;                                             {3}

; Output main loop while storing into buffer 1
OLoop1:         MAR ← XAddr, XAddr ← XAddr + 1, XBranch;         {1}
        XAddr xor XLast1, F=0, rXWReq;                           {2}
        XWData ← MD, GOTO[OLoop0];                               {3}

ONone1:         MAR ← XAddr, XBranch;                            {1}
        NOP;                                                     {2}
        XWData ← MD;                                             {3}

        {end of packet}
```

Appendix B. Propagation Delay Estimates

B.1 Assumptions

Manufacturers' maximum device propagation times or minimum setup times over the full commercial temperature (0 to 100 °C) and power supply range range were used. If the maximums were only stated for 25 °C and +5 V, then they were increased by no more than 10% (brackets indicate the manufacture's given number at 25 °C).

In addition, 3 nanoseconds were added per chip interconnnect to account for propagation delays (except for connections to the 2902 look ahead carry unit). No assumptions were made here on number of loads or trace thickness.

B.2 Delays

The following delays are derived from the most recent Wildflower Sil diagrams [reference 2].

AB Arith (operations of form: Rreg ← Rreg + Rreg):

7	S74	MIR to A,B [7]
45	2901A-1	A,B to G/P
10	2901A-1	G/P to Cin
35	2901A-1	Cin setup

97		device total
3		1 trace delay

100		total

X Arith (operations of form: Rreg ← Rreg + IB):

7	S74	MIR to mux enable [7]. currently 25S18
23	S257	mux enable to X [21]
30	2901A-1	X to G/P
10	2901A-1	G/P to Cin
35	2901A-1	Cin setup

105		device total
6		2 trace delays

111		total

AB Logic (operations of form: Ureg ← Rreg OR Rreg):

7	S74	MIR to A,B [7]
50	2901A-1	A,B to Y
5	93422	data setup [5]
33	93422	min write pulse [30]

95		device total
6		2 trace delays

101		total

SU Logic (operations of form: Rreg ← Ureg OR Rreg):

18	25S09	MIR to SUaddr [17]
45	93422	SUaddr to X
40	2901A-1	X setup

103		device total
6		2 trace delays

109		total

F=0 AB Arith (operations of form: Rreg + 1 = 0 ?):

7	S74	MIR to A,B [7]
45	2901A-1	A,B to G/P
10	2901A-1	G/P to Cin
22	2901A-1	Cin to F3
8	S03	o.c. for Fdisp [7.5]
6	25S09	data setup [5.5]

98		device total
9		3 trace delays

107		total

F=0 SU Logic (operations of form: Ureg XOR Rreg = 0 ?):

7	S74	MIR to SUaddr [7]. currently 25S09
45	93422	SUaddr to X
32	2901A-1	X to F3
8	S03	o.c. for Fdisp [7.5]
6	25S09	data setup [5.5]

98		device total
12		4 trace delays

110		total

μInst (fetching next microinstruction):

18	25S09	MIR to NIA [17]
75	63S1681	NIA to data out [70]
5	25S18	data setup [5]

98		device total
6		2 trace delays

104		total

Y cycle 8 (operations of form: Rreg ← Rreg cycle 8):

7	S74	MIR to A,B [7]
40	2901A-1	A,B to Y
8	S257	Y to X [7.5]
40	2901A-1	X setup

95		device total
9		3 trace delays

104		total