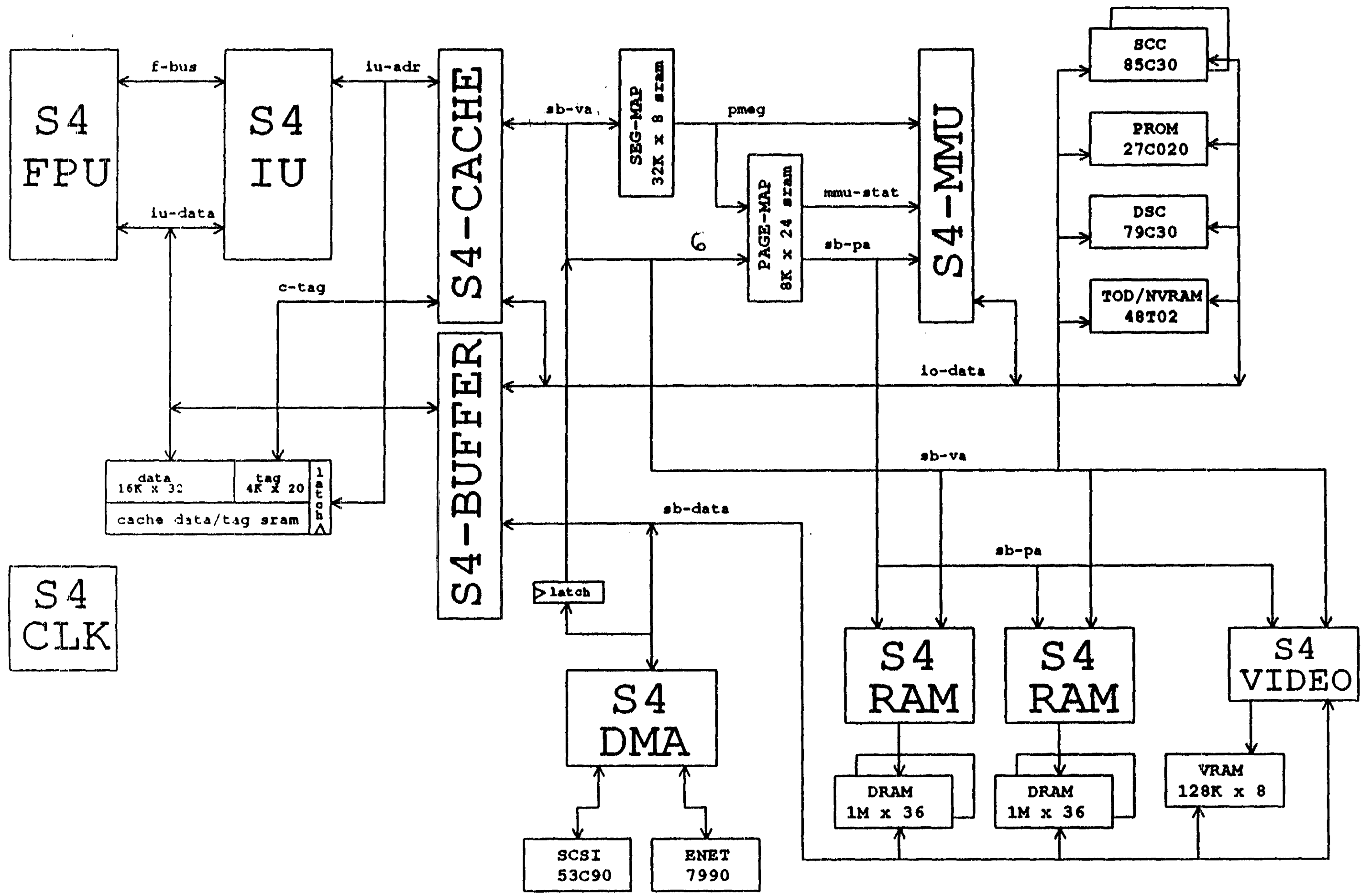


sun microsystems

TITLE: OFFCAMPUS Block Diagram
SUNPARTNR:
REV:
SHEET:
ENGINEER: AMH
DATE:

SPARCstation-1 Programmer's Model

Stephen A. Chessin

DRAFT 7
Version 8.3, 89/06/10

1. Introduction

This paper describes the programmer's view of SPARCstation-1: address spaces, caching and memory management, and interrupt levels. It is a synthesis of information contained in the hardware specifications, but organized to be useful to a programmer.

Where appropriate, comparisons with the "standard" Sun4 architecture are made.

WARNING: This document is a DRAFT and may contain errors. Please report all mistakes to the author for correction.

Major Changes Since Draft 1 (Version 1.7)

(1) The page size has changed from 8K to 4K.

(2) The size of a physical address has changed from 29 bits to 28 bits.

(3) The Sbus has moved from Type 0 space to Type 1 space, and there has been a major reorganization of the Type 1 addresses to accommodate this.

Changes Since Draft 2 (Version 2.4)

(4) Minor typographical and editorial changes.

(5) Better explanations.

Changes Since Draft 3 (Version 3.7)

(6) The Interrupt Register is used to clear level 15 interrupts.

(7) All Sbus devices are now described using relative offsets.

(8) More bits are used in the Auxiliary Input/Output Register. (Which used to be the Auxiliary Output Register.)

Changes Since Draft 4 (Version 4.7)

(9) The interrupt levels have been changed slightly. All Sbus devices, including the builtin ones, interrupt on Sbus IRQ levels only.

(10) The Auxiliary Input/Output Register has changed slightly.

(11) The definition of the DMA Write bit was backwards.

(12) The video subsystem is off the board, again.

Changes Since Draft 5 (Version 5.6)

Better explanations and addition of more examples.

Changes Since Draft 6 (Version 6.1)

(1) Added warnings that this is still a DRAFT document and may not be completely accurate.

(2) Described the bugs in various levels of hardware:

Synchronous parity errors cause asynchronous traps (fixed in P1.7)
SER records asynchronous errors (won't be fixed)
ASER and ASEVAR latch on synchronous memory errors (won't be fixed)
On cache fill errors, SEVAR may not have exact address of problem (won't be fixed)

ASER sometimes isn't set on asynchronous errors (won't be fixed)
ASEVAR isn't properly sign-extended on DVMA errors (won't be fixed)

(3) Audio/ISDN replaces Audio DAC.

(4) Level 8 interrupts can be masked.

(5) Video goes into slot 3.

(6) Sbus IRQ6 and IRQ7 now map to SPARC level 8 and 9, instead of 9 and 13, respectively.

(7) Miscellaneous corrections.

2. Address Spaces

The SPARC Architecture defines the existence of at least 4 address spaces. A given implementation may define more than 4 address spaces. Selection of a particular address space is done via the Address Space Indicator (ASI) field of the load and store alternate address space instructions. Ordinary load and store instructions automatically go to User or Supervisor Data space, depending upon the mode of the CPU. Instruction fetches by the CPU automatically go to User or Supervisor Instruction space, again depending upon the mode of the CPU.

The following table describes the address spaces defined by the Sun4 Architecture and the SPARCstation-1 implementation.

ASI	Sun4 Use	SPARCstation-1 Use	Comments
0x0	Reserved	Reserved	
0x1	Reserved	Reserved	
0x2	System Space	Same	Note 1
0x3	Segment Map	Same	
0x4	Page Map	Same	
0x5	Block Copy	Reserved	Note 2
0x6	Region Map	Reserved	Note 2
0x7	Flush Cache (Region)	Reserved	Note 2
0x8	User Instruction	Same	
0x9	Supervisor Instruction	Same	
0xA	User Data	Same	
0xB	Supervisor Data	Same	
0xC	Flush Cache (Segment)	Same	
0xD	Flush Cache (Page)	Same	
0xE	Flush Cache (Context)	Same	
0xF	Flush Cache (User)	Reserved	Note 3
0x10	Flush I-Cache (Segment)	Reserved	Note 2
0x11	Flush I-Cache (Page)	Reserved	Note 2
0x12	Flush I-Cache (Context)	Reserved	Note 2
0x13	Flush I-Cache (User)	Reserved	Note 2
0x14	Flush D-Cache (Segment)	Reserved	Note 2
0x15	Flush D-Cache (Page)	Reserved	Note 2
0x16	Flush D-Cache (Context)	Reserved	Note 2
0x17	Flush D-Cache (User)	Reserved	Note 2
0x1B	Flush I-Cache (Region)	Reserved	Note 2
0x1F	Flush D-Cache (Region)	Reserved	Note 2

Note 1. See System Space table (next section)
Note 2. SPARCstation-1 has no corresponding function.
Note 3. This is a change in the specification between Sunrise and Sunray.

User and Supervisor Instruction and Data spaces are collectively known as "Device Space". All accesses to Device Space go through the Memory Mangement Unit (MMU). All the other address spaces are collectively known as "Control Space". The non-System Space portions of Control Space all deal with Cache and MMU management, and are discussed in the section on "Contexts, Caching, and the MMU". System Space is discussed in the next section.

System Space (ASI = 2)

System Space is a portion of control space that is used to access various devices, as the following table indicates:

A31:28	Sun4 Use	SPARCstation-1 Use	Comments
0x0	ID Prom	Reserved	Note 1
0x1	Reserved	Reserved	
0x2	Reserved	Reserved	
0x3	Context Register	Same	
0x4	System Enable Register	Same	
0x5	Reserved	Reserved	
0x6	Bus Error Register	Bus Error Registers	Note 5
0x7	Diagnostic Register	Unused	Note 2
0x8	(D-)Cache Tags	Cache Tags	
0x9	(D-)Cache Data	Same	Note 3
0xA	I-Cache Tags	Reserved	Note 4
0xB	I-Cache Data	Reserved	Note 4
0xC	Reserved	Reserved	
0xD	Reserved	Reserved	
0xE	VME Interrupt Vector	Reserved	Note 4
0xF	Serial Port	Same	MMU bypass

Note 1. SPARCstation-1 does not have an ID Prom and a timeout will occur.
Note 2. SPARCstation-1 has no diagnostic register but a write to this address will just be ignored and not cause a timeout.
Note 3. This is a change in the specification between Sunrise and Sunray.
Note 4. SPARCstation-1 has no corresponding function.
Note 5. SPARCstation-1 has four Bus Error Registers, compared to Sun4's one.

The Context Register, Cache Tags, and Cache Data are described in the section on "Contexts, Caching, and the MMU". The rest of the registers in System Space are described below.

3.1. System Enable Register

The System Enable Register is referenced via byte loads and stores at location (ASI=0x2, A31:28=0x4). It has the following format:

```
 7 6 5 4 3 2 1 0
v---------------v
|N|0|S|C|0|R|0|D|
^---------------^
```

N	ENA_NOTBOOT	0 = all supervisor references go to EPROM 1 = normal MMU operation
-		Reserved (Enables I/O Cache in Sun4)
S	ENA_SDVMA	1 = all DVMA is enabled
C	ENA_CACHE	1 = Cache enabled
-		Reserved (Enables video display in Sun4)
R	ENA_RESET	1 = Reset the System (asserts SBRESET)
-		Reserved (Resets VMEbus in Sun4)
D	ENA_DIAG	Always 0 (Diagnostic/Monitor in Sun4)

All bits are initialized to zero by a reset. Setting ENA_RESET to one will cause a reset, and control will not be returned to the program that does so; rather, a reboot will occur. Software (or the boot PROM) should set ENA_NOTBOOT to one after initializing the MMU.

3.2. Bus Error Registers

There are four registers, divided into two sets of two, used to indicate the type and location of bus errors. One set is for synchronous errors, and the other for asynchronous errors. Synchronous errors are ose that occur due to the execution of the current instruction and are reported to the CPU by a trap at the end of that instruction's execution. All errors that cannot be associated with the execution of the current instruction, but are related to such things as DVMA activity, buffered writes, or cache write-back[1], are considered asynchronous and are reported via an interrupt on level 15. After servicing the level 15 interrupt, it is cleared by toggling bit 0 of the Interrupt Register.

There is an exception to the above rule. On machines prior to the P1.7 level, parity errors that occur (or any condition that causes SE_MEMERR, described below, to be set) during CPU memory accesses cause the reporting of both a synchronous and asynchronous error. For parity errors that occur during data fetches, the data-access trap occurs first and the level 15 interrupt remains pending. Software may clear the level 15 interrupt while processing the data-access trap. For parity errors that occur during instruction fetches, the level 15 interrupt occurs first and the text-access trap never occurs. Software can distinguish true asynchronous errors from instruction fetch errors by maintaining an invalid value in the SEVAR and comparing the SEVAR to the ASEVAR on asynchronous errors. If they compare equal, then this is an instruction-fetch error, otherwise it is a true asynchronous error. Software must remember to reload the SEVAR with the invalid value after processing all synchronous (including instruction-fetch) errors.

On P1.7 and later boards, memory errors during CPU memory accesses only cause the reporting of a synchronous error; a level 15 interrupt does not occur. (The asynchronous registers still latch on synchronous memory errors, however, and must be cleared; see the descriptions of the ASER and ASEVAR, below.)

The Bus Error Registers are all fullword in size, although they can be accessed via byte, halfword, or fullword loads and stores, just as memory is. They reside at the following addresses in ASI=2 space:

Address	Description
0x60000000	Synchronous Error Register
0x60000004	Synchronous Error Virtual Address Register
0x60000008	Asynchronous Error Register
0x6000000C	Asynchronous Error Virtual Address Register

Although in normal use the registers can be treated as read-only, they can be written for diagnostic purposes.

3.2.1. Synchronous Error Register

The Synchronous Error Register (SER) occupies four bytes at locations (ASI=0x2, A31:28=0x6, A3:0=0x0 to 0x3). Reading any portion of the register also clears that portion. It has the following format:

```
31                     23                     15                     7  6  5  4  3  2  1  0
v - - - - - - - - - - - v - - - - - - - - - - - v - - - - - - - - - - - v - - - - - - - - - - - v
|0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0|R|0  0  0  0  0  0  0|I |P |T |B |M |0 |S |W|
^ - - - - - - - - - - - ^ - - - - - - - - - - - ^ - - - - - - - - - - - ^ - - - - - - - - - - - ^
```

R	SE_WRITE	1 = Error during write cycle, 0 = read cycle
I	SE_INVALID	1 = Valid bit was zero in a page map entry
P	SE_PROTERR	1 = Protection error (see below)
T	SE_TIMEOUT	1 = Non-existent device was addressed
B	SE_SBERR	1 = bus error during Sbus master access
M	SE_MEMERR	1 = Memory (parity or ECC) error
S	SE_SIZERR	1 = Incorrect size transfer attempted
W	SE_WATCHDOG	1 = Restart due to IU error

The SER records all errors since it was last cleared. **This includes asynchronous errors as well; the SER must be read to clear it as part of asynchronous error processing.** The SE_WRITE bit records the type of access (read or write) of the last error.

[1] SPARCstation-1 does not have a write-back cache, but if it did it could cause asynchronous errors.

A protection error can be caused by an attempted write to a read-only page, or by a user-mode access to a supervisor-only page.

A timeout is reported on access to a non-existent device, except for accesses to non-existent physical memory. See the section "Type 0 Space," below.

The Memory Error Register must be inspected when a memory error occurs, to further isolate the cause of the error. Note that synchronous memory errors also cause the Asynchronous Error Register and Asynchronous Error Virtual Address Register to be latched; see the description of these registers below for more information.

Not all bus errors cause immediate traps. Due to pipelining, the CPU fetches instructions four cycles before they will be executed, so it is possible that the CPU will attempt to fetch an instruction that will not, in fact, be executed. To prevent spurious traps, the CPU does not trap on memory exceptions until it actually needs to execute the instruction that it was unable to fetch.

For example, suppose we have the following instruction sequence in virtual memory, where a, b, c, etc. represent miscellaneous instructions:

```
a
b
bz,a                    label
d
----                    page boundary
e <--this page is marked invalid
f
g
----                    page boundary
label:                  <--this page is valid
x
y
z
```

These instructions will advance through the pipeline as follows:

Time	1	2	3	4	5
Fetch	d	-	x	y	z
Decode	bz	d	-	x	y
Execute	b	bz	d	-	x
Write	a	b	bz	d	-

At time (2), the CPU wants to fetch e but the page is marked invalid, so the invalid bit is set in the SER and the instruction address is set in the SEVAR. However, the branch (if taken) means that e is never needed, so that it would be incorrect for the CPU to trap on a page fault due to the attempt to fetch e.

Now let's examine the following sequence:

```
a
b
st something to a read-only page
d
----    page boundary
e       <--this page is marked invalid
f
g
```

The pipeline now looks as follows:

Time	1	2	3	4	5
Fetch	d	-	-	x	y
Decode	st	d	-	-	x
Execute	b	st	d	-	-
Write	a	b	st	-	-

.ie attempt to fetch e from an invalid page at time (2) will turn on the SE_INVALID bit in the SER, but the CPU will not take an instruction access exception until it actually needs to execute e, at time (5). The store to a read-only page at time (3), however, does result in an immediate data access exception, and the CPU will find both the SE_INVALID bit and the SE_PROTERR bit on in the SER. (The exception results in a flush of the pipe, and instruction d never does get to the Write stage in step (4)).

A similar scenario, where the store is replaced by a branch (in user mode) to a supervisor-only page, can result in multiple bits being on for instruction access exceptions.

It is up to the software to determine the true cause of the exception when multiple bits are on in the SER. Here is one algorithm:

```
SEVAR = getsevar();
SER = SERsave = getser();
SER &= ~(SE_WRITE | SE_WATCHDOG);
if (data access exception)
    error_addr = SEVAR;
else if (instruction access exception)
    error_addr = old PC;
else
    /* CAN'T HAPPEN */;

if (SER & (SER - 1)) {
    /* multiple bits on; must manually probe the PME */
    pme = getpme(error_addr);
    if (pme valid) {
        if ((SER & SE_PROTERR) && (pme denies access)) {
            SER = SE_PROTERR;
        } else
            SER &= ~(SE_PROTERR|SE_INVALID);
    } else
        SER = SE_INVALID;
}

/*
 * Note: we could still have other multiple bits on (TIMEOUT,
 * MEMERR, SIZERR, SBERR), but we probably won't recover from
 * this condition anyway, so it really doesn't matter.
 *
 * But if you really wanted to, know you'd do something like
 * this:
 */

/* more than one of TIMEOUT, SBERR, MEMERR, or SIZERR */
(void) getser(); /* make sure it's clear */
if (on_fault())
    newSER = getser();
else {
    register int x;

    newSER = 0;
    x = *error_addr; /* probe the address to see what happens */
}
no_fault();
/* use newSER to figure out what the problem was, if any */
```

2.2. Synchronous Error Virtual Address Register

The Synchronous Error Virtual Address Register (SEVAR) occupies four bytes at locations (ASI=0x2, A31:28=0x6, A3:0=0x4 to 0x7). It contains the virtual address associated with the last synchronous bus error. It is not latched.

Note that on errors resulting from cache-fill operations, the SEVAR will contain the address that the CPU presented to the cache chip that triggered the cache-fill operation. This may or may not be the address of the word that actually caused the error.

The SEVAR has the following format:

```
31                                                                  0
v--------------v--------------v--------------v--------------v
|                   Virtual Address (A31:0)                   |
^--------------^--------------^--------------^--------------^
```

3.2.3. Asynchronous Error Register

The Asynchronous Error Register (ASER) occupies four bytes at locations (ASI=0x2, A31:28=0x6, A3:0=0x8 to 0xB). Reading any portion of the register also clears that portion. It has the following format:

```
31            23            15             7   5 4          0
v--------------v--------------v--------------v--------------v
|              undefined                     |W|0|T|D|0 0 0 0|
^--------------^--------------^--------------^--------------^
```

W	ASE_WBACKERR	1 = Valid bit was zero in a page map entry
T	ASE_TIMEOUT	1 = Non-existent device was addressed
D	ASE_DVMAERR	1 = bus error during DVMA access

The ASER latches (freezes) with the cause of an asynchronous error, ignoring subsequent asynchronous errors, until read and cleared. **It is also latched when a synchronous memory error (SE_MEMERR) occurs, and should be read to unlatch it as part of SE_MEMERR processing. Note that bits in the SER are set when bits in the ASER are set; thus the SER should be read to clear it as part of asynchronous error processing.**

A write-back error can occur on systems with write-back caches, and/or on systems that do buffered writes, when either the hardware malfunctions or the MMU mapping is changed without properly flushing the cache. In addition, certain devices (for example, frame buffers) will generate write-back errors under device-specific conditions when a store is attempted to them.

A timeout is reported on access to a non-existent device, except that accesses to non-existent physical memory may produce detectable behavior other than timeouts. (See the section "Type 0 Space," below.) For SPARCstation-1, this can only happen if the MMU is set up to map a non-existent device or if the hardware malfunctions.

The specific cause of a DMVA bus error must be determined by polling the possible sources to see which indicated the error. All possible sources of DVMA errors of this type must be recognizable in some way. For SPARCstation-1, the only possible source of DVMA bus errors is memory parity errors. These can be determined by examining the Memory Error Register, described below.

Due to a bug in the cache chip, the ASER is not always set when an asynchronous error occurs. In this event, the ASER can be reconstructed from the bits in the SER. SE_MEMERR should be on in the SER. In addition, SE_TIMEOUT indicates that ASE_TIMEOUT should have been reported, and SE_SBERR indicates that ASE_WBACKERR should have been reported. The address in the ASEVER is correct even when the ASER is not set. This bug is in all versions of the hardware, including P1.7's, and is not expected to be fixed.

3.2.4. Asynchronous Error Virtual Address Register

The Asynchronous Error Virtual Address Register (ASEVAR) occupies four bytes at locations (ASI=0x2, A31:28=0x6, A3:0=0xC to 0xF). It contains the (pseudo) virtual address associated with the asynchronous bus error described in the ASER. It is latched under the same conditions that the ASER is tched. It is unlatched when it is read, not when the ASER is read. Thus, the ASEVAR should be read to unlatch it as part of SE_MEMERR processing.

The ASEVAR has the following format:

```
31  29                                                          0
v--------------v--------------v--------------v--------------v
|S S|              Pseudo Virtual Address (A29:0)           |
^--------------^--------------^--------------^--------------^
```

S Bits 31:30 are copies of bit 29.

The address is called a "pseudo-virtual" address because the hardware only carries the low-order 30 bits of the virtual address onto the bus, and assumes that bits 31:29 are all the same. The ASEVAR reverses this process by copying bit 29 into bits 31 and 30 on asynchronous errors reported by the IU. **Due to a bug in the cache chip, bits 31 and 30 are zero on DBMA asynchronous errors (ASE_DVMAERR is on). Software must do the sign extension itself.**

Determining the context register value associated with an asynchronous error is usually straightforward; there is only one tricky case.

Since DVMA is always done using context 0, the address associated with a DVMA error will always be context 0.

Non-DVMA asynchronous errors are due to buffer chip activity. The buffer chip allows only one outstanding store; a subsequent store will stall the CPU in the middle of execution of the second store until the outsanding store completes. If it completes with an asynchronous error, the error will be reported to the CPU immediately after execution of the second store instruction finishes. (This is not necessarily completion of the second store itself, as it may itself be buffered. This is just completion of the store instruction from the CPU's point of view.) Unless the second store is a write to the context register, the address of the asynchronous error will be associated with the value in the context register (the current context).

If the second store does modify the context register, then the address of the asynchronous error is associated with the previous context, which must be determined by software. (If, for example, the first store was to a supervisor-only page, then the actual context is irrelevant as supervisor-only pages are mapped into all contexts.)

One can construct pathological cases where it would be impossible to determine that an asynchronous error is associated with the previous context (for example, a store to a user page, followed by a branch, with the store to the context register in the delay slot of the branch). It is up to software to avoid these pathologies.

3.2.5. Simultaneous errors

It is possible for both a synchronous and an asynchronous error to be reported simultaneously. Consider the following case:

```
st      %g0, [%l0]! this address causes an asynchronous timeout
st      %g0, [%l1]! this address causes a page fault
```

Depending upon the alignment of the instructions in the cache, it is possible for the IU to take the page fault trap (a synchronous error) first, and while it is disabled for traps but before the SER has been read, the asynchronous fault can be reported. This will turn on the MEMERR bit in the SER, which can lead software to believe that this is a synchronous memory error. Since the MEMERR is really asynchronous, there will be a level 15 interrupt pending. If software treats this error as synchronous, and diligently reads the SER, SEVAR, ASER, and ASEVAR to clear and/or unlatch them, then when traps are eventually enabled and the level 15 interrupt occurs software will discover that there is no information in either the ASER or the SER pertaining to the asynchronous interrupt.

Software can avoid this difficulty by comparing the ASEVAR to the SEVAR whenever MEMERR is set on a synchronous trap. If they are identical, then this is a true synchronous MEMERR. If they are different, then the MEMERR is associated with the asynchronous trap. Software should clear the pending vel 15 interrupt and process the asynchronous error, using the ASER and ASEVAR values, and being cognizant of the bugs in asynchronous error reporting described previously. The synchronous error can be

,nored, for it will recur if and when execution of the program is resumed.

3.2.6. Serial Port

The serial port is referenced by byte loads and stores at locations beginning at (ASI=0x2, A=0xF0000000). This access is provided so that the serial port may be used before the MMU has been initialized, for example by the PROM monitor. Software normally accesses the serial port via I/O space through the MMU.

See the section "Serial Ports" under "Type 1 Space", below, for more information on the serial port registers.

4. Physical Space

The MMU maps virtual addresses in Device Space to physical addresses in Physical Space. Physical space is further subdivided into four types, as indicated in the following table.

Type	Sun4 Use	SPARCstation-1 Use	Comments
0	Main Memory	Same	
1	I/O Space	Same	
2	VMEbus, 16-bit data	Unused	Note 1
3	VMEbus, 32-bit data	Unused	Note 1

Note 1. In SPARCstation-1, references to type 2 or 3 space cause a timeout.

The size of a physical address is 28 bits.

4.1. Type 0 Space

Type 0 space contains the main memory (RAM) in SPARCstation-1. Since PA27:0 are used for RAM device decoding, the Sbus can support a theoretical maximum of 256 Mbytes of RAM. However, e SPARCstation-1 implementation only supports a maximum of 64 Mbytes. In addition, individual SPARCstation-1 machines can be configured with as little as 4 Mbytes of memory. To explain what happens when non-existent RAM is addressed, the implementation must be explained and some terms defined.

The SPARCstation-1 memory subsystem contains two RAM controllers. Each RAM controller controls a "bank" of 32 Mbytes of address space. Each bank is made up of two "sets" spanning 16 Mbytes each. Each set contains four SIMMs (Single Inline Memory Modules) each. Each SIMM consists of 9 chips. Each chip is either a 1 Mbit or a 4 Mbit DRAM. All the chips in a SIMM are of the same type, and all the SIMMs in a set must be of the same type. A set of 1 Mbit DRAMs contains 4 Mbytes of memory, and a set of 4 Mbit DRAMs contains 16 Mbytes of memory. The SIMMs in one set can be of a different type than the SIMMs in another set, even in the same bank.

The RAM controllers require PA27 to be zero. If PA27=1, then no controller responds and a bus timeout occurs.

PA26:25 selects the appropriate RAM controller. One controller responds to 0x0, the other responds to 0x1. If PA26:25=2 or 3, then no controller responds and a bus timeout occurs.

PA24 selects one of the two sets of SIMMs controlled by a controller. If the selected set is not installed (a hole), then on writes the data is thrown away and on reads the bus lines remain high (subject to noise) and a characteristic bit pattern (normally all ones) is returned. Software can detect a hole by doing a store to followed by a load from a byte on 16 Mbyte boundary. If the data read does not agree with the data written, then a hole exists. If they agree, the same test with a different bit pattern should be used before concluding that real memory exists. (Note that parity checking should be disabled when doing these checks, as parity errors will be reported if the noise pattern contains bad parity and parity checking is enabled.)

If the selected set consists of 4 Mbit DRAMs, then all 16 Mbytes of address space spanned by that set are valid and correspond to unique memory locations. If the selected set consists of 1 Mbit DRAMs, then only 4 Mbytes of unique memory exist, but it appears four times in the 16 Mbytes of address space anned by the set, repeating at every 4 Mbyte boundary. This "mirror" behavior can be detected by software by doing a store of one bit pattern to offset 0 of a set, followed by a store of another bit pattern to offset 4 Meg (0x00400000) of the set, followed by a load from offset 0. If the data at offset 0 was changed the store to offset 4 Meg, then only 4 Mbytes of memory is present and the rest is filled with mirrors.

The following decision table summarizes this behavior.

PA27	PA26	SIMM Set	PA23:22	Action
1	-	-	-	Timeout
0	1	-	-	Timeout
0	0	none	-	Hole
0	0	4 Mbit	-	Memory (16 Mbytes worth)
0	0	1 Mbit	00	Memory (4 Mbytes worth)
0	0	1 Mbit	01	Mirror
			10	
			11	

4.2. Type 1 Space

Type 1 space contains all of the I/O devices, including those that are associated with the Sbus. Bit PA27 is used to indicate an onboard device (PA27=0) or an Sbus device (PA27=1). For onboard devices, PA26:24 (and in some cases PA26:20) determine the particular device. For Sbus devices, PA26:25 select one of four Sbus slots. The (physical or logical) board plugged into the Sbus slot then has an address space of 25 bits, or 32 Mbytes, to divide up as it sees fit. Sbus addressing is further described in the Section "Sbus Devices", below. For compatibility with the Sun4 architecture conventions, the non-existent bits (PA31:28) are assumed to be all ones. The following table describes the layout of Type 1 space:

Address	SPARCstation-1 Use	Comments
0xF0000000	Keyboard/Mouse	Note 1
0xF1000000	Serial Ports	Note 1
0xF2000000	TOD Clock and NVRAM	Note 2
0xF3000000	Counter-Timer Registers	Note 3
0xF4000000	Memory Error Registers	Note 1
0xF5000000	Interrupt Register	Note 1
0xF6000000	EPROM	Note 3
0xF7000000	EPD "Private":	Note 4
0xF7100000	ECC registers	(HPD only)
0xF7200000	Floppy Controller	
0xF7201000	Audio/ISDN	
0xF7400003	Auxiliary Input/Output Register	
0xF7F00000	VME Control Register	(SunFed only)
0xF8000000	Sbus Slot 0 (25 bits)	Note 4
0xF9000000	"	"
0xFA000000	Sbus Slot 1 (25 bits)	Note 4
0xFB000000	"	"
0xFC000000	Sbus Slot 2 (25 bits)	Note 4
0xFD000000	"	"
0xFE000000	Sbus Slot 3 (25 bits)	Note 4
0xFF000000	"	"

ffe80000 → ffe9f000

Note 1. Same as Sun4 use.
Note 2. Sun4 has a different kind of TOD at this address. It also has an EEPROM at a different address.
Note 3. Sun4 has same function, but at a different address.
Note 4. Sun4 has no corresponding function.

Reference to a Type 1 address to which no device responds results in a timeout.

4.2.1. Onboard Devices

2.1.1. Keyboard/Mouse

The keyboard/mouse UART is a Z8530 chip (Zilog or AMD equivalent) accessed via byte loads and stores at the following addresses:

Address	Description
0xF0000000	Mouse Control Port
0xF0000002	Mouse Transmit (W)/Receive (R) Data Port
0xF0000004	Keyboard Control Port
0xF0000006	Keyboard Transmit (W)/Receive (R) Data Port

The Z8530 contains an array of read registers and write registers, accessed through the control port. Access to a register is done by writing the register index to the control port, and then reading or writing the register data to the control port. In addition, the UART transmit and receive data registers may be directly accessed by writing and reading, respectively, from the Transmit/Receive Data Port.

See the Z8530 data sheet for more information.

4.2.1.2. Serial Ports

The serial ports UART is also a Z8530 chip, identical to the one used for the keyboard/mouse. It is addressed as follows:

Address	Description
0xF1000000	Serial Port B Control Port
0xF1000002	Serial Port B Transmit (W)/Receive (R) Data Port
0xF1000004	Serial Port A Control Port
0xF1000006	Serial Port A Transmit (W)/Receive (R) Data Port

2.1.3. TOD Clock and NVRAM (EEPROM)

The Time of Day Clock is a Mostek MK48T12-15 Zeropower/Timekeeper RAM which includes 2K of RAM, the topmost 8 bytes of which are the clock. The Timekeeper contains its own battery backup, which has a worst-case storage life (oscillator off or power on) of 11 years at 70°C and a worst case consumption life (oscillator on and power off) of 2.8 years at 0°C. Unlike EEPROMs, there is no limitation on the number of times the CMOS RAM can be written, nor are special write timings required.

The Clock/NVRAM is accessed via byte, halfword, or fullword loads and stores at the following addresses:

Address	Description
0xF2000000 to 0xF20007d7	NVRAM
0xF20007d8 to 0xF20007f7	"IDPROM"
0xF20007f8	TOD Control
0xF20007f9	Seconds (00-59)
0xF20007fa	Minutes (00-59)
0xF20007fb	Hour (00-23)
0xF20007fc	Day (01-07)
0xF20007fd	Date (01-31)
0xF20007fe	Month (01-12)
0xF20007ff	Year (00-99)

Thirty-two bytes of NVRAM acts as the ID prom" of SPARCstation-1. The id_machine byte contains 0x51; 0x50 is the architecture code for Sun4C, and 0x51 indicates the SPARCstation-1 machine.

The TOD Control register should only be written with byte stores to prevent modifying the data to be read.

The time and date information is stored in 24 hour BCD format. For more information, including the protocol to be used to read, write, start, and stop the clock, see the MK48T12-15 data sheet.

2.1.4. Counter-Timer Registers

The Counter-Timer Registers are accessed via fullword loads and stores at the following addresses:

Address	Description
0xF3000000	Counter 0
0xF3000004	Limit 0
0xF3000008	Counter 1
0xF300000C	Limit 1

All registers have the following format:

```
31                                                  9              0
v--------------v--------------v--------------v--------------v
|L|                21-bit value              |0 0 0 0 0 0 0 0 0 0|
^--------------^--------------^--------------^--------------^
```

L Limit Reached

Each counter is incremented by one in bit position 10 at one microsecond intervals. When a counter reaches the value in its corresponding limit register, it is reset to "one microsecond," the limit-reached bit in both the counter and limit registers is set, and an interrupt is generated (if enabled) at level 10 for Counter 0 and level 14 for Counter 1.

The interrupt is cleared and the limit bits reset by reading the appropriate limit register. Reading the counter register does not change the state of the limit bits. Writing the limit register resets the counter register to a value equivalent to one microsecond. Except for testing purposes, the counter registers should not be written.

Setting a limit register to zero causes the corresponding counter to freerun. Interrupts will occur when the counter overflows back to zero, approximately every 2 seconds.

4.2.1.5. Memory Error Registers

SPARCstation-1 uses a single Parity Control Register. This is a fullword read/write register at location 0xF4000000 in Type 1 physical space. The format of this register is as follows:

```
31             23             15             7              0
v--------------v--------------v--------------v--------------v
|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|E|M|T|N|A|B|C|D|
^--------------^--------------^--------------^--------------^
```

E Parity Error. Set on any parity error.
M Multiple Errors. Set when a parity error occurs and E=1.
T Parity Test. When set, inverse parity is generated.
N Parity Check. Enables parity checking.
A Parity Error 24. Records parity error on data bits 31:24.
B Parity Error 16. Records parity error on data bits 23:16.
C Parity Error 08. Records parity error on data bits 15:8.
D Parity Error 00. Records parity error on data bits 7:0.

The bits that indicate errors (E, M, and A-D) are cleared when the register is read. All bits are cleared on reset.

Note that when a parity error occurs, the cache will have loaded itself with the data from memory anyway. This means that software must flush the cache after parity errors if it is to continue operation. On a single parity error (M=0), only the affected cache line (as determined from the old PC, the SEVAR, or the ASEVAR, as appropriate) need be flushed. On multiple parity errors (M=1), the entire cache must be flushed.

Also note that the address in the SEVAR or ASEVAR, as appropriate, may not be the address of the word with the parity error, if the error occurred during a cache-fill operation.

2.1.6. Interrupt Register

The Interrupt Register is a one-byte read/write register at location 0xF5000000 in Type 1 physical space. The format of this register is as followed:

```
 7              0
v---------------v
|A|0|C|D|E|F|G|H|
^---------------^
```

A Enable Level 14 Interrupts
C Enable Level 10 Interrupts
D Enable Level 8 Interrupts
E Software Interrupt Level 6
F Software Interrupt Level 4
G Software Interrupt Level 1
H Enable all Interrupts

clock counter 1
clock counter 0
video

!! B= enable level 12 interrupts. (what disheard!) (not necessary?)

Writing a zero to an Enable Level *N* Interrupt bit only masks out that interrupt, it does not clear the source. Writing a one to a software interrupt bit requests an interrupt on that level; the bit must be cleared to clear the request.

Writing a zero to the Enable All Interrupts bit will clear the Asynchronous Memory (level 15) Interrupt, as well as masking all interrupts. Of course, interrupts should be immediately re-enabled by writing a one.

On reset, all bits are cleared and all interrupts are reset.

4.2.1.7. EPROM

SPARCstation-1 has 128K bytes of EPROM containing the boot monitor beginning at location :F6000000 in Type 1 physical space. The EPROM is also referenced by all Supervisor Virtual addresses when the ENA_NOTBOOT bit in the System Enable Register is zero, for example at boot time. The boot code must initialize the MMU to at least map itself before setting the ENA_NOTBOOT bit to one.

Note that the EPROM does not obey the normal memory mapping rules. PA[16:0] into the EPROM always come from VA[16:0]. Although VA[29:12] are processed by the MMU to select a physical address, when bits PA[27:24] of that physical address select the EPROM then bits PA[23:12] from the MMU are ignored. This means that, for proper operation of the EPROM, it must be mapped one-for-one to contiguous virtual pages beginning on a 128K boundary.

4.2.1.8. Floppy Controller

The Floppy Disk Controller is an Intel 82072. It is accessed using byte loads and stores at the following addresses:

Address	Description
0xF7200000	Main Status (R)/Data Rate Select Register (W)
0xF7200001	FIFO Data Port (R/W)

For more information see the Intel 82072 data sheet. Note that the floppy must be selected as drive 1 (or 3, but 1 is preferred) in the command sequence sent to the controller. See also the Terminal Count and Floppy Eject bits in the "Auxiliary Input/Output Register" described below.

4.2.1.9. Audio/ISDN

The audio interface of the SPARCstation-1 is provided through the Main Audio Processor (MAP) of the AMD 79C30A Digital Subscriber Controller. The 79C30A is a highly integrated circuit which provides an ISDN 4-wire subscriber level interface, an audio processing circuit, a parallel microprocessor interface, and a serial interface. For SPARCstation-1 Audio use the microprocessor interface and the audio processing circuits are the only portions of the circuit which are used.

The interrupt from the 79C30 is attached to IRQ<13> of the MMU (which is interrupt level 13). The data bus is connected to the IO data bus. The circuit includes an oscillator circuit which uses an externally provided 12.288 MHz crystal with a tolerance of + or - 80 ppm. The oscillator is a parallel resonant circuit.

The 79C30 registers are located at a base address of 0xF7201000. The 79C30 is accessed using byte loads and stores at the following addresses:

Address	WR*	RD*	Register description
0xF7201000	0	1	Command Register (CR), write only
	1	0	Interrupt Register (IR), read only
0xF7201001	0	1	Data Register (DR), write
	1	0	Data Register (DR), read
0xF7201002	1	0	D-channel Status Register 1 (DSR1), read only
0xF7201003	1	0	D-channel Error Register (DER), read only
0xF7201004	0	1	D-channel Transmit Buffer (DCTB), write only (8-byte FIFO)
0xF7201004	1	0	D-channel Receive Buffer (DCRB), read only (8-byte FIFO)
0xF7201005	0	1	Bb channel Transmit Buffer (BBTB), write only
0xF7201005	1	0	Bb channel Receive Buffer (BBRB), read only
0xF7201006	0	1	Bc channel Transmit Buffer (BBTB), write only
0xF7201006	1	0	Bc channel Receive Buffer (BBRB), read only
0xF7201007	1	0	D-channel Status Register 2 (DSR2), read only

Note that the other registers in the 79C30, of which there are many, are indirectly accessed through the command register. Pages 2-71 through 2-77 of the 79C30A Data Sheet describe this indirect addressing.

Please refer to the 79C30A Data Sheet for full details on operation of this circuit.

4.2.1.10. Auxiliary Input/Output Register

The Auxiliary Input/Output Register is a one-byte, read-write register at location 0xF7400003 in Type 1 physical space. It has the following format:

```
 7 6 5 4 3 2 1 0
v---------------v
|1 1|D|C|S|T|E|L|
^---------------^
```

D	In	Density
C	In	Floppy Diskette Change (must be written as one)
S	Out	Floppy Drive Select
T	Out	TC (Floppy controller Terminal Count input)
E	Out	Floppy Eject
L	Out	LED (1=on, 0=off)

All bits are set to one on reset.

Bit 5 (Density) is a signal from the drive indicating the density of the diskette inserted. A 1 indicates high density, a 0 indicates low density. This signal is meaningful only if the floppy drive is capable of sensing the "density" hole in the diskette. The Sony drives do not generate this signal; for them, software must through trial and error determine the density of the inserted diskette. This can be done by initializing the controller with parameters for a given density and attempting to read the diskette; if the wrong parameters were chosen read errors will occur. Note that the density of an unformatted floppy cannot be determined through this method; the floppy format software must have a user option to set the density to be used. (If the user selects the wrong density, the floppy will be unusable, but the user will quickly discover this mistake.)

Bit 4 (Floppy Diskette Change) is an input bit that signifies when a diskette has been removed from the drive. This bit must always be written as one in order for it to work. It reads as one when the drive is selected and there is no diskette in the drive. It reads as zero if the drive is not selected or if a diskette is present in the drive. The Sony drives reset the bit when they receive a step pulse from the controller; i.e., when the software issues a "Seek" command. Other vendor drives require a separate Diskette Change

:eset signal; a bit will need to be provided for this function in the Auxiliary Input/Output Register if a non-Sony drive is chosen for SPARCstation. **When will this decision be made?**

Bit 3 (Floppy Drive Select) is connected to the floppy drive select pin. It is used in conjunction with all floppy operations, whether through the Floppy Disk Controller registers or the bits in the Auxiliary I/O Register. A one selects the floppy drive; a zero de-selects it.

Bit 2 (TC) is connected to the Terminal Count input pin of the floppy controller. It is used to signal the floppy controller (which is designed to be connected to a DMA controller, even though in SPARCstation-1 it is not) that all the data for a given operation has been transferred. This is done by writing a 1 to this bit, delaying for a specific amount of time, and then writing a 0 to it. (The specific amount of time depends upon the data rate and can be found in the Intel 82072 data sheet.)

Bit 1 (Floppy Eject) is connected to the floppy drive eject mechanism. To eject a floppy, set bit 3 (Floppy Drive Select), wait 2.0 microseconds, set bit 1, hold it set for at least 2.0 microseconds, then reset both it and bit 3 to zero.

Bit 0 (LED) controls the LED on the front panel.

Unused bit positions should be written with ones when writing to the register. This will allow them to be used for input signals if this becomes necessary.

4.2.2. Sbus Devices

Unlike previous busses, the Sbus is geographically addressed. PA26:25 select which of four Sbus "slots" is being referenced. A board plugged into an Sbus slot has PA24:0, or 25 bits or 32 Mbytes of address space addressability to divide up among the devices contained on that board. A Forth program beginning at offset 0 of the slot describes the devices on that board to the system. The details of the Forth specification are described in Sun Forth User's Guide.

Slot 0 is not a physical slot. Rather, it refers to the onboard DMA, SCSI, and Ethernet controllers :hich, for convenience, are viewed as being plugged into Slot 0.

Slots 1, 2, and 3 are physical slots into which the user may plug boards containing devices. Slots 1 and 2 have DVMA-master capability; slot 3 is a slave-only slot and does not support boards that operate as DVMA masters. The board containing the video subsystem (video control registers, RAMDAC, and frame buffer) is usually, but need not be, plugged into Slot 3.

If no device responds to a particular Sbus address, a bus timeout will occur.

The following table summarizes the devices:

PA26:25	Device
0 0	Onboard DMA, SCSI, and Ethernet controllers
0 1	Sbus Slot 1
1 0	Sbus Slot 2
1 1	Sbus Slot 3 (usually video subsystem)

4.2.2.1. DMA, SCSI, and Ethernet Devices

The following table describes the offsets to the onboard DMA, SCSI, and Ethernet devices, relative to the beginning of Sbus "Slot 0" (base physical address 0xF8000000 in Type 1 space).

Offset	Description
0x000000	ID (4 bytes, 0xFE810101)
0x400000	DMA Registers
0x800000	SCSI Registers
0xC00000	Ethernet Registers

4.2.2.1.1. DMA Registers

The DMA registers are accessed via fullword loads and stores to the following offsets (the addresses in this table do not include the slot base address, which must be added to the device offset):

Address	Description
0x400000	DMA Control/Status Register
0x400004	DMA Address Register
0x400008	DMA Byte Count
0x40000C	Diagnostic Register

The DMA registers are used when programming SCSI operations. Other than the ILACC bit in the DMA Control/Status Register, they are not used when programming Ethernet operations.

4.2.2.1.1.1. DMA Control/Status Register

The DMA Control/Status Register has the following format:

```
31        27                            15     12   10    8    6    4 3    1 0
v--------------v--------------v--------------v--------------v
|DEV_ID |   unused (read as zero)|L|T|C|ADR|P|N|W|R|D|F|I|I|PCK|E|J|
^--------------^--------------^--------------^--------------^
```

DEV_ID
DEV_ID. Device ID. Read-only. (0b1000 in this implementation.)

L L. ILACC. When 0, the Ethernet/DMA interface is configured to use the Lance Ethernet controller. When 1, the interface is configured to use ILACC, "the new Ethernet chip from AMDpq (Cliff Buckley).

T TC. Terminal Count. Read-only. Byte counter has expired. This bit is cleared by setting the Flush bit (bit 5).

C EN_CNT. Enable Count. Read/write. Enables the DMA Byte Count Register. (Not used in normal SPARCstation-1 operation.)

DR BYTE_ADDR. Read-only. Next byte number to be accessed.

P REQ_PEND. Request pending. Read-only. Set when the DMA interface is active. RESET and FLUSH must not be asserted if REQ_PEND is one.

N EN_DMA. Enable DMA. Read/write. Set to enable DMA activity, reset to disable.

W WRITE. Read/write. Set for DMA from device to memory (read), reset for DMA from memory to device (write).

R RESET. Read/write. When set, acts as a hardware reset. ERR_PEND, PACK_CNT, INT_EN, FLUSH, DRAIN, WRITE, EN_DMA, REQ_PEND, EN_CNT, and TC are all set to zero. RESET remains at 1, and must be set back to 0 by software to resume operation.

D DRAIN. Read/write. Set to force remaining pack register bytes to be drained to memory. Clears itself.

F FLUSH. Write-only. Set to force PACK_CNT and ERR_PEND to zero. Also clears TC and the interrupt TC=1 causes. Always reads as zero.

I INT_EN. Interrupt enable. Read/write. Set to enable interrupts.

PCK PACK_CNT. Pack Count. Read-only. Number of bytes in Pack Register.

E ERR_PEND. Error Pending. Read-only. Set when a memory exception occurs. Reset by setting FLUSH. DMA activity stops until reset.

J INT_PEND. Interrupt Pending. Read-only. Set when TC=1 or when external device raises an interrupt. Cleared when read (if TC=1 is the cause) or by servicing the external device (if that is the cause).

4.2.2.1.1.2. DMA Address Register

The DMA Address Register has the following format:

```
31            23                                          0
v-------------v-------------v-------------v-------------v
|VA31:24-latched|          VA23:0 - address               |
^-------------^-------------^-------------^-------------^
```

The high byte is latched by the hardware and indicates which 16 Mbyte region of Virtual Memory is accessed. (The MMU recognizes a DMA virtual address and forces Context 0 to be selected.) The low-order 3 bytes contain the address of the byte to be transferred. Rollover is only through the low-order 24 bits.

4.2.2.1.1.3. DMA Byte Count

The DMA Byte Count Register has the following format:

```
31            23                                          0
v-------------v-------------v-------------v-------------v
|0 0 0 0 0 0 0 0|          BCNT23:0 - counter             |
^-------------^-------------^-------------^-------------^
```

This register is only used when EN_CNT is on in the DMA Control/Status Register, and so is not used in normal SPARCstation-1 operation. The high byte is unused and will always read back as zero. The low order bytes contain the number of bytes to be transferred, and counts down to zero. When zero is reached, TC, and thus INT_PEND, are set to one. Further DMA transfers cannot take place until a new value is loaded into the Byte Count Regiser.

4.2.2.1.1.4. Diagnostic Register

The format of the Diagnostic Register is not available.

?.2.1.2. SCSI Registers

The SCSI registers are accessed via byte loads and stores to the following offsets (the addresses in this table do not include the slot base address, which must be added to the device offset):

Address	Description
0x800000	Transfer Count Low
0x800004	Transfer Count High
0x800008	FIFO Data
0x80000C	Command
0x800010	Status/Bus ID
0x800014	Interrupt/Status Timeout
0x800018	Sequential step/Synchronization transfer period
0x80001C	FIFO flags/Synchronization offset
0x800020	Configuration
0x800024	Clock Conversion Factor (write only)
0x800028	ESP TEST (chip test use only)
0x80002C	ESP II Configuration-2

Note that byte accesses must be performed even though the addresses are all fullword-aligned.

Since the SCSI controller uses the DMA controller to perform the actual transfer of data to and from memory, the two devices must be programmed together. One possible algorithm is as follows:

```
scsi_start()
{
        /* start an operation on the SCSI */
        lock data pages into contiguous virtual memory;
        DMA_address_register = starting virtual address;
        setup SCSI registers (except for "go");
        DMA_control_status_register = (EN_DMA | INT_EN | (other bits));
        start SCSI;
```

```
        /* The SCSI will interrupt us when it is done. */
}

scsi_interrupt()
{
            /* must drain DMA on a read from disk/write to memory */
            if (last operation == READ) {
                                DMA_control_status_register = (DRAIN);
            }
}
```

For a detailed description of the SCSI registers, see the NCR 53C90 Data Sheet.

4.2.2.1.3. Ethernet Registers

The Ethernet registers are accessed via halfword loads and stores to the following offsets (the addresses in this table do not include the slot base address, which must be added to the device offset):

Address	Description
0xC00000	Register Data Port (RDP)
0xC00002	Register Address Port (RAP)

For a detailed description of the Ethernet registers, see the AMD Am7990 Data Sheet.

4.2.2.2. Video Subsystem

The following table describes the offsets to the devices located on the Video Subsystem Board. This board is usually plugged into Sbus "Slot 3" (base physical address 0xFE000000 in Type 1 space).

Offset	Description
0x000000	ID (4 bytes, 0xFE010101)
0x400000	Video and DAC Registers
0x800000	Frame Buffer

4.2.2.2.1. Video and DAC Registers

The Video and DAC registers are accessed via byte loads and stores to the following offsets (the addresses in this table do not include the slot base address, which must be added to the device offset):

Address	Description
0x400000	Video Control Register
0x400001	Video Status Register
0x400002	HBS (Horizontal Blank Set)
0x400003	HBC (Horizontal Blank Clear)
0x400004	HSS (Horizontal Sync Set)
0x400005	HSC0 (Horizontal Sync Clear, !VS)
0x400006	HSC1 (Horizontal Sync Clear, VS)
0x400007	VBSH (Vertical Blank Set High Byte)
0x400008	VBSL (Vertical Blank Set Low Byte)
0x400009	VBC (Vertical Blank Clear)
0x40000A	VSS (Vertical Sync Start)
0x40000B	VSC (Vertical Sync Clear)
0x400010	DAC Address Register
0x400014	DAC Color Palette Register Port
0x400018	DAC Control Register Port
0x40001C	DAC Overlay Palette Register Port

See the S-4 Video data sheet for a detailed description of the Video Registers, and the Brooktree Bt458/451 data sheet for a detailed description of the DAC Registers. Note that setting incorrect values into the registers can damage the attached monitor.

Note that the DAC registers are 8-bits wide even though they are aligned on fullword boundaries. Fullword accesses can be used to quickly read or write one or more palette entries, by storing the index of the first palette to be accessed in the address register and then doing fullword accesses to the appropriate palette port. The data must be packed into bytes in the order "RGBRGBRGBRGB"; in other words, 3 fullwords will hold 4 palette entries. Palette entries are only stored when the Blue value is written; partial update of a palette is not possible.

4.2.2.2.2. Frame Buffer

The frame buffer is a megabyte of RAM occupying offsets from 0x800000 to 0x8FFFFF. Each byte corresponds to one pixel. Accesses may be by bytes, by halfwords, or by fullwords.

If the frame buffer is only half-populated, then only the lower four bits of each byte will be significant. As the upper four bits will be (weakly) pulled up with resistors, only the upper 16 color map entries (entries 240 through 255) in the DAC will be usable. Software can detect this case by writing, then reading, the frame buffer. If the upper four bits always read back as ones, independent of the data written, then the frame buffer is half-populated. **(This is grody — Ed.)**

5. Interrupt Levels

The following table describes the interrupt levels defined by the Sun4 Architecture and the SPARCstation-1 implementation.

Level	Sun4 Use	SPARCstation-1 Use
15	Memory Error	Asynchronous Memory Error
14	Clock	Counter 1
13	VMEbus level 7	Audio
12	Keyboard, Mouse, Serial Ports	Same
11	VMEbus level 6	Floppy
10	Clock	Counter 0
9	VMEbus level 5	Sbus IRQ7
8	Video	Sbus IRQ6
7	VMEbus level 4	Video, Sbus IRQ5
6	Ethernet, Software request 6	Software request 6
5	VMEbus level 3	Ethernet, Sbus IRQ4
4	SCSI, Software request 4	Software request 4
3	VMEbus level 2	SCSI, DMA, Sbus IRQ3
2	VMEbus level 1	Sbus IRQ2
1	Software request 1	Same, plus Sbus IRQ1

6. Resets

Although there is only one type of reset in SPARCstation-1 (a reset of the entire machine that causes system registers to be restored to a known state), there are three ways to effect a reset:

(1) Power-on. A power-on reset (POR) occurs when power is initially applied to SPARCstation-1.

(2) Watchdog. A watchdog reset occurs when the IU signals an error condition. This can occur, for example, if the IU attempts to take a trap when traps are disabled.

(3) Software. Software can initiate a reset by writing a one to the ENA_RESET bit of the System Enable Register.

The SE_WATCHDOG bit in the Synchronous Error Register is set to one on watchdog-initiated resets, and set to zero for all other resets.

7. Contexts, Caching, and the MMU

This section describes the interaction of the context register, the cache, and the MMU from the ~ogrammers perspective.

1. **Context Register** (ASI=2, A=0x30000000, byte access only)

The Context Register has the following format:

```
 7       3     0
v--------------v
|0 0 0 0|  CID |
^--------------^
```

Note that although the CID is four bits wide, only the low-order 3 bits (CID2:0) are actually used. CID3 is ignored.

The context register selects one of 8 contexts for translating User Mode addresses. It exists in both the Cache and the MMU.

Programming note: A byte store (STBA) into (ASI=2, A31:28=0x3) writes both the MMU and Cache Context Registers. A byte load (LDUBA, LDSBA) from (ASI=2, A31:28=0x3, A0=0) reads the MMU's Context Register, and a byte load from (ASI=2, A31:28=0x3, A0=1) reads the Cache's Context Register. The ability to read each register separately is provided for diagnostic purposes; they should always contain the same value and standard software will usually just read the MMU's Context Register.

7.2. MMU decoding of Virtual Addresses

From the MMU's standpoint, a virtual address has the following format:

```
31  29                   17         11                     0
v-------------v-------------v-------------v-------------v
|   |                     | page in  |                       |
|   | segment (12 bits)   | segment  |byte in page (12 bits) |
|   |                     | (6 bits) |                       |
^-------------^-------------^-------------^-------------^
```

ote: VA31:29 must all be the same (all 0 or all 1). An SE_INVALID error results otherwise.

CID2:0 is concatenated with VA29:18 to select one of 32K segment map entries. (One can view the segment map as consisting of 8 contexts, each context containing 4K segments.) The segment map entry is 8 bits wide, although only the lower 7 bits are used, and points to a Page Map Entry Group (PMEG):

```
 7 6           0
v--------------v
|0|   PMEG     |
^--------------^
```

PMEG6:0 is concatenated with VA17:12 to select one of 8K Page Map Entries (PME). (One can view the page map as consisting of 128 PMEGs, each PMEG containing 64 pages.) The PME is 32 bits wide, organized as follows:

```
31  29  27  25  23             15                          0
v-------------v-------------v-------------v-------------v
|V|W|S|X|TYP|A|M|0 0 0 0 0 0 0 0| physical page number (16 bits)|
^-------------^-------------^-------------^-------------^
```

V	1=entry is valid
W	1=write access allowed
S	1=Supervisor mode access only
X	1=don't cache this page
TYP	0=Main Memory; 1=Sbus and I/O space; 2,3=reserved for VMEbus
A	1=page has been accessed
M	1=page had been modified

PME15:0 is concatenated with VA11:0 to form a 28-bit physical address whose interpretation depends upon the type field.

Programming Notes:

) A page is 4K bytes. A segment is 64 pages or 256K bytes. A context contains 4K segments or 1G byte. This last is divided into two address ranges of 512M bytes each, from 0x00000000-0x1fffffff and from 0xe0000000-0xffffffff.

(2) Unlike architectures used by other vendors, in this architecture there is no way to explicitly mark a segment as invalid. However, the operating system can reserve one PMEG and mark all of its PMEs invalid, and then point invalid segments at this PMEG. SunOS has traditionally used the last PMEG for this purpose, but this may be subject to change.

(3) Because the cache ignores the context register when resolving accesses to supervisor-mode-only pages, the kernel segments should be identical in each context. This can be accomplished by repeating the same PMEG in the appropriate segment map entries.

(4) A context is selected by performing a byte store into the Context Register (ASI=2, A31:28=0x3).

A segment map is initialized by selecting a context, and then performing byte stores into (ASI=3, A29:18=0x0 to 0xfff). (Half and fullword stores will work but are not recommended.)

A PMEG is initialized by selecting a context, and then performing fullword stores into (ASI=4, A29:18=desired segment, A17:12=0x0 to 0x3f).

(5) The hardware does not insure consistency between the cache and the MMU. The operating system software must flush the cache appropriately before updating the MMU. Before changing the mapping of a context, a Flush Cache (Context) operation must be performed. Before changing the mapping of segment, a Flush Cache (Segment) operation must be performed. Before changing the mapping of a page, a Flush Cache (Page) operation must be performed. These operations are described in the Cache section, below. Also note that these are not the only circumstances when flushing the cache is necessary.

7.3. Cache decoding of Virtual Addresses

To improve performance, SPARCstation-1 contains a 64K byte virtual address cache, consisting of 4K lines of 16 bytes each. The cache is one-way set associative, with each virtual address mapping to one and only one possible cache line. There is a 4 byte cache tag associated with each data line.

From the Cache's standpoint, a virtual address has the following format:

```
31  29                    15                  3    0
v------------v------------v------------v----------v
|  |                      |                 |byte of|
|  | cache tag id (14 bits) | cache line (12 bits) | line  |
|  |                      |                 | (4 b.)|
^------------^------------^------------^----------^
```

Note: VA31:29 must all be the same (all 0 or all 1). An SE_INVALID error occurs otherwise.

VA15:4 selects one of 4K cache lines. If the cache tag id matches (and, for non-supervisor-mode-only pages, the context ID), then a cache hit occurs. VA3:2 selects the desired word from the cache line.

A cache tag has the following format:

```
31               24    21   18    15                        1 0
v-----------------v---------------v-------------v-----------v
|0 0 0 0 0 0 0 0| CID |W|S|V|0 0 0|  cache tag id (14 bits) |0 0|
^-----------------^---------------^-------------^-----------^
```

CID Cache Tag Context (copied from Cache Context Register when cache line is filled.) Note that only CID2:0 are present.

W 1=write access allowed (copied from MMU when cache line is filled.)

S 1=Supervisor mode access only (copied from MMU when cache line is filled.)

V 1=entry is valid

Programming Notes:

) The cache tags must be initialized by software before the cache is enabled, by clearing the valid bit in the cache tag of each cache line. It is sufficient to do fullword stores of zero into (ASI=2, A31:28=0x8, A15:4=0x0 to 0xfff).

(2) To flush all references to a context from the cache, a Flush Cache (Context) operation must be performed by selecting the appropriate context (by performing a byte store into the Context Register, (ASI=2, A31:28=0x3)) and doing fullword stores of zero into (ASI=0xe, A15:4=0x0 to 0xfff).

(3) To flush all references to a segment from the cache, a Flush Cache (Segment) operation must be performed by selecting the appropriate context and doing fullword stores of zero into (ASI=0xc, A29:18=desired segment, A15:A4=0x0 to 0xfff). A17:16 are ignored for this operation.

(4) To flush all references to a page from the cache, a Flush Cache (Page) operation must be performed by selecting the appropriate context and doing fullword stores of zero into (ASI=0xd, A29:12=desired page, A11:4=0x0 to 0xff).

7.4. Aliasing

Because the cache is bigger than a page, a physical page that is mapped by two (or more) distinct virtual addresses could result in data from the same physical address appearing in two (or more) cache lines:

```
31  29                    17           11    -              0
v-------------v-------------v-------------v-------------v
|   |  segment (12 bits)  |  page     | byte in page (12 bits)|
|   |  cache tag (14 bits)    | cache line (12 bits) | byte  |
^-------------^-------------^-------------^-------------^
31  29                       15                      3     0
```

This situation cannot be detected by the hardware and must be avoided by the software. There are two methods that may be used:

(1) All the virtual addresses for an aliased page must be identical in bits A15:12. That is, the virtual addresses must be congruent modulo 64K (the cache size). This will result in the same cache line being used for the different virtual addresses that map to the same physical address. This is the preferred method. (Note that the hardware doesn't know that the different virtual addresses map to the same physical address, and alternate use of the different virtual addresses will result in invalidating and then refilling the cache line from the same physical address. Also, the hardware automatically invalidates a cache line when a cache miss occurs on a write operation. This insures the consistency of the cache with memory when aliasing via this method occurs.)

(2) Each PME that points to the aliased physical page must have the "Don't Cache" bit (PME28) set. This method must be used if the previous method cannot.

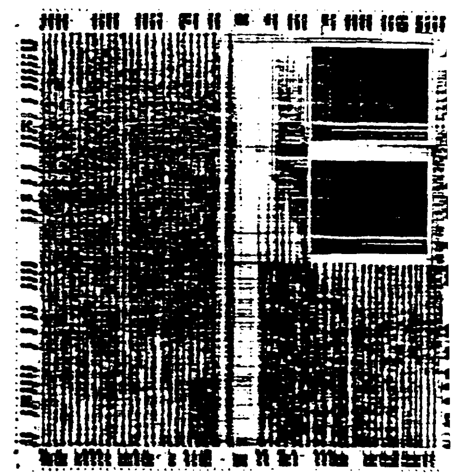
L64801
High Performance Open Architecture RISC Microprocessor
Preliminary

Description

The L64801 Integer Unit (IU) is a high performance CMOS implementation of the SPARC (Scalable Processor ARChitecture) 32-bit RISC microprocessor. SPARC is an open architecture which is being implemented in a variety of forms by various semiconductor manufacturers. This multiple sourcing allows designers to choose from a wide variety of price/performance options and provides a rich selection of peripherals, memory devices and proprietary ASIC extensions.

The L64801 features a large register file to optimize procedure calls, variable assignments and context switches. Execution speed improves significantly because this register-to-register architecture minimizes the number of external memory accesses. Most of the L64801 instructions execute in a single cycle due to its 4-stage pipeline that minimizes interlocks, a bus structure that allows single-cycle instruction/data accesses and an optimized branch handler.

The L64801 can sustain 15 VAX MIPS performance with peak performance of 25 MIPS, offering designers the speed and power of a super minicomputer.



L64801 Chip Photo

Features

- High performance operation
  Commercial
  L64801C-20 12 VAX MIPS
  L64801C-25 15 VAX MIPS
  Military
  L64801M-15 9 VAX MIPS
  L64801M-20 12 VAX MIPS
- Open architecture:
  - Multiple vendor sourced
  - Each vendor provides unique features and extensions
  - Variety of binary compatible price/performance options
- Optimized for operation under high-level languages such as C, FORTRAN, Pascal and Ada and the UNIX™ operating system
- External MMU, memory system and floating-point unit assure flexible interface for the largest range of applications and price/performance levels
- 32-bit virtual address bus
  - Supports up to 4 Gbytes of direct address spa
  - Allows a variety of memory management and caching schemes
- Simple instruction format with fast instruction cycle with a 4-stage pipeline
- Single cycle execution for the majority of instructions
- Large central register file divided into seven overlapping windows of 24 registers each
- All pipeline interlocks implemented directly in hardware
- High performance coprocessor interface for concurrent execution of floating-point or other coprocessor instructions
- Multitasking support with user/supervisor mode and privileged instructions
- Artificial intelligence support through use of tagg instructions
- Option to use as ASIC core
- 179-pin ceramic or plastic pin grid array packages

L64801
High Performance
Open Architecture
RISC Microprocessor
Preliminary

Pinout Diagram

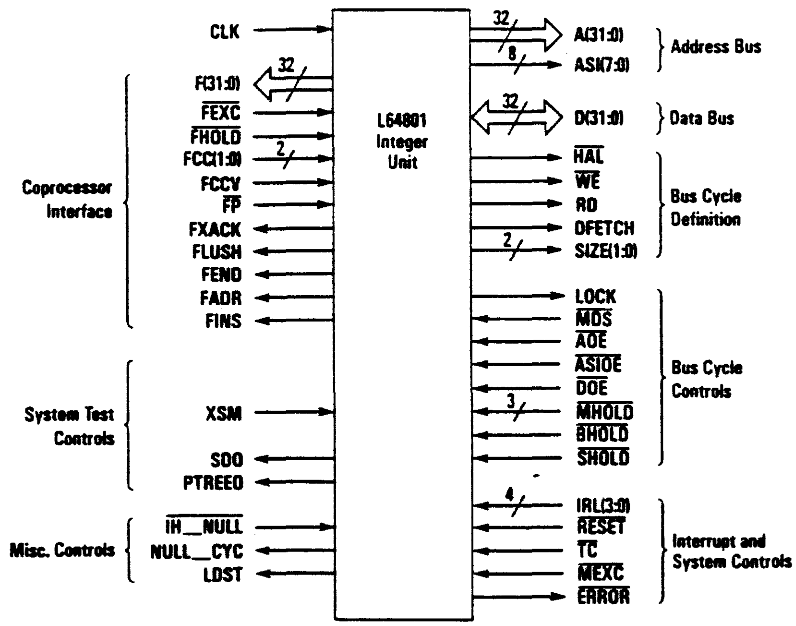


Figure 1. L64801 Pinout Diagram

Block Diagram

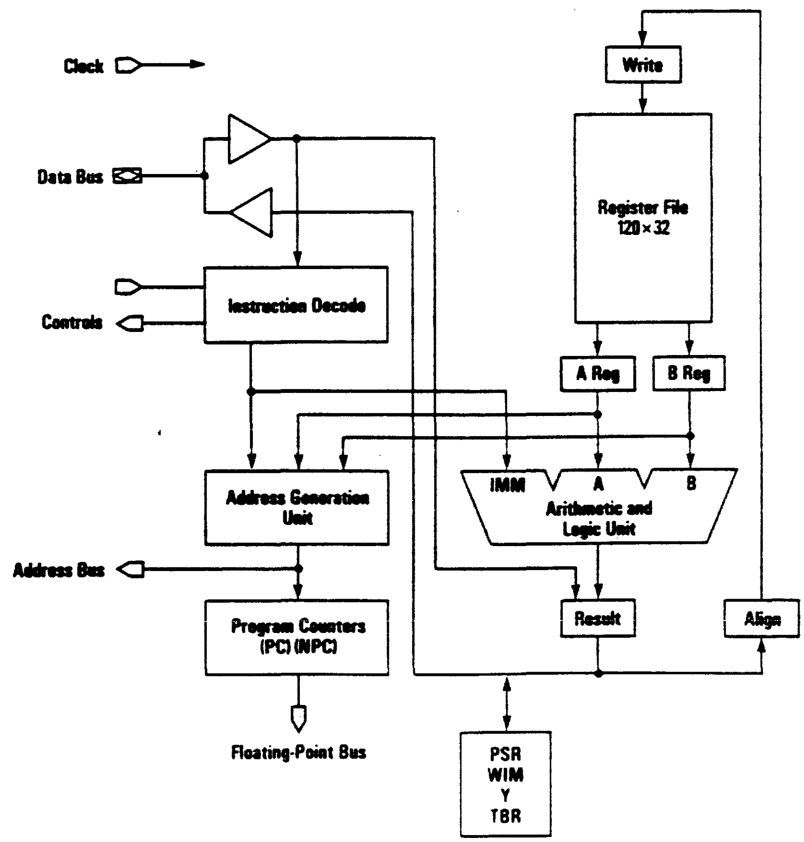


Figure 2. L64801 Functional Block Diagram

L64801
High Performance
Open Architecture
RISC Microprocessor
Preliminary

Introduction

The L64801 is the first processor in the LSI Logic family of SPARC (Scalable Processor ARChitecture) microprocessors. SPARC is an architecture defined by Sun Microsystems which is based on the principles of RISC (Reduced Instruction Set Computer) techniques. The key feature of SPARC is its use of a large central register file which is divided into several "register windows" for high performance during subroutine calls and context switching.

The SPARC family is supported by a full line of highly optimizing compilers, operating systems, development boards, development systems and development tools.

SPARC is an open architecture, built by a number of semiconductor suppliers, which will provide rapid enhancement of features for different markets and a wide range of price/performance options. LSI Logic has chosen to implement the L64801 using its own industry standard ASIC techniques. This allows rapid implementation of the L64801 design into new process technologies as well as the availability of the L64801 as a microprocessor core within a more complex ASIC.

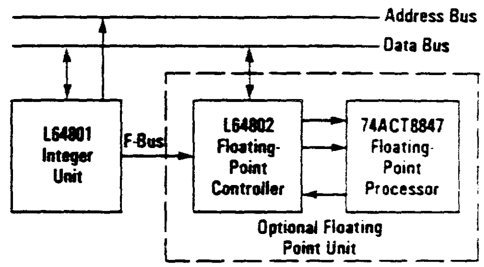
Architecture Overview

The L64801 SPARC chip set consists of a central integer unit (IU) which provides all the core functions of the SPARC instruction set as defined by the SPARC architecture manual. To increase performance of floating-point operations, there is an optional floating-point unit (FPU) and a separate interface chip called the floating-point controller (FPC).

The IU is the primary computing element. It performs all operations except floating-point operations (FPops) which are either performed in hardware through the FPC/FPU combination, or in software. The FPC/FPU provides execution of FPops concurrent with integer operations.

The IU features a large central register file partitioned as sets of working registers (r registers) which provide storage for processes. In addition, there are independent control registers which keep track of and control the state of the IU.

There are a total of 120 32-bit registers which are divided into seven separate register windows. Each window contains 24 working registers plus eight global registers.



Note: All lines shown are 32 bits wide.

Figure 3. L64801 Core Chip Set

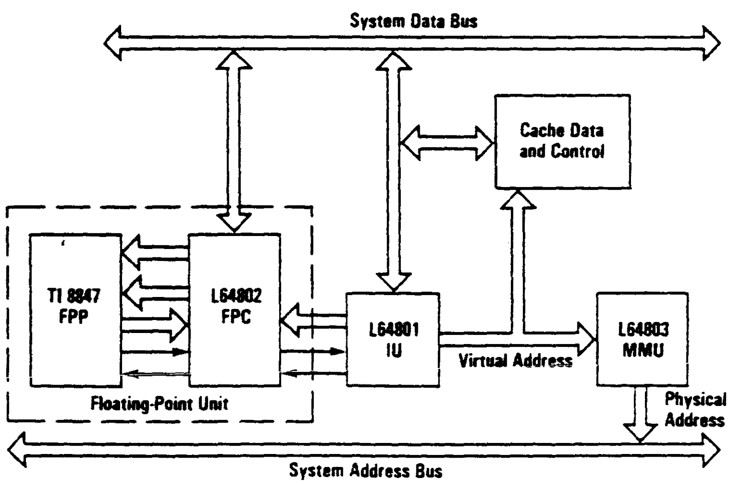


Figure 4. SPARC System-Level Diagram

Register Windows

Perhaps the most distinguishing characteristic of the SPARC architecture is the overlapping register windows. In order to optimize operations such as subroutine calls and context switching, the register file is divided into sets of register windows. There are a total of eight global registers which are available at all times and seven register windows of 24 registers each that are available at any point in time. These register windows overlap each other by eight registers on either side for parameter passing between processes. The register configuration at any point in time is as follows:

R0 thru R7	Global Registers
R8 thru R15	Output Parameters to Next Process
R16 thru R23	Local Registers to Current Process
R24 thru R31	Input Parameters from Previous Process

In the L64801 IU, there are a total of 120 registers divided into seven register windows. The current window pointer (CWP) field within the processor state register (PSR) keeps track of which window is currently active. The pointer is decremented when the processor executes a call to the next window and is incremented when a return is executed. The windows are joined in a circular stack where the output parameters of window 6 are coincident with the input parameters of window 0.

The register file is triple ported. This allows the fetching of two register operands and the writing of a destination register to occur simultaneously in a single clock cycle.

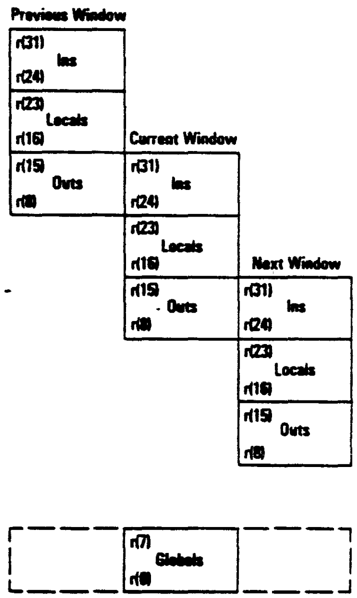
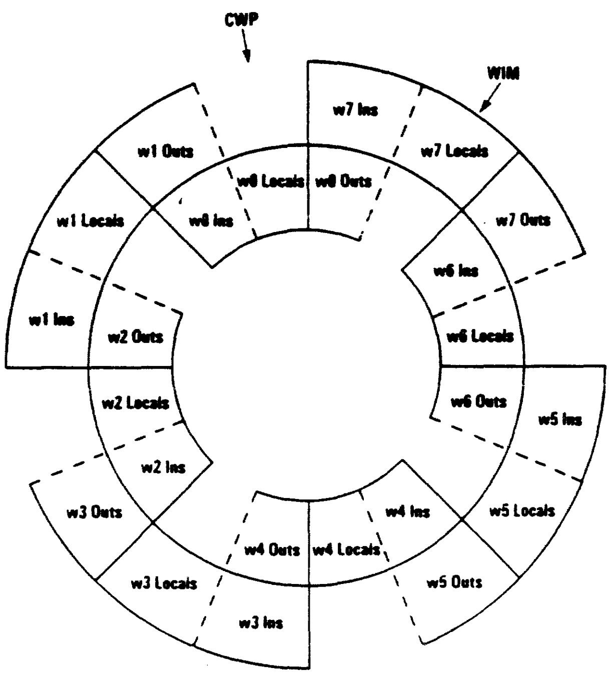


Figure 5. Example of Register Windows (3 Windows Shown)



In this figure, NWINDOWS = 8. It does not show the 8 globals. If the procedure corresponding to the window labeled w0 does a procedure call (executes a SAVE instruction), a window_overflow trap will occur. The overflow trap handler uses the locals of w7:

CWP = 0 — active window = 0
CWP + 1 = 1 — previous window = 1
CWP − 1 = 7 — next window = 7
WIM = 10000000_2 — trap window = 7

Note: In LR64801 implementation NWINDOWS actually equal 7, not 8.

Figure 6. Register Windows Implemented as a Circular Stack

Four-Stage Pipeline

The L64801 Integer Unit uses a 4-stage instruction pipeline comprised of; Fetch, Decode, Execute and Write stages. A basic single-cycle instruction enters the pipeline and completes four cycles later. Once the pipeline is filled, four separate instructions may be executing each of the following phases in an overlapping fashion.

Fetch (F)
An instruction is fetched from the bus interface and placed in the instruction register.

Decode (D)
The instruction is decoded and operands are read from the register file. Memory addresses are evaluated for loads, stores and control transfers.

Execute (E)
The operation specified by the instruction is executed and the results are saved in the processor's temporary registers.

Write (W)
The result of the executed operation is store the destination register (provided that no trap exceptions have occurred during execution).

The L64801 Integer Unit detects data depend cies and provides hardware interlocks in pipeli operation to properly resolve such dependenci without complex software intervention. Pipelir interlock occurs if an instruction fetch takes m than one clock cycle. Multi-cycle instructions c the pipeline long enough to complete their execution.

Bus Interface

The L64801 accesses instructions and data and performs system control functions through its high bandwidth bus interface. The bus interface has separate address and data lines and sets of control lines with protocols which support:

- Single and multiple-clock period reads and writes
- Full and partial-word (byte and halfword) writes
- Multimaster bus protocols
- Fifteen levels of external interrupt requests
- Memory exception traps

The IU acts as a bus slave: it has no bus grant o bus request circuits. It uses signals such as LOC to lock the bus and $\overline{BHOLD}$, $\overline{MHOLD}$, or $\overline{SHOLD}$ t be locked off the bus.

Memory/Cache Interface

The L64801 Integer Unit can be interfaced to a variety of memory subsystems: cached, non-cached, virtual, physical, static, dynamic, etc. The processor normally expects to receive a new instruction every cycle. If the memory is not fast enough to provide instructions at this rate, then wait states are inserted using the memory hold ($\overline{MHOLD}$) inputs. In systems with non-cached memory, every memory reference appears to the IU as a cache miss. In a fast memory (cached) system, the bus interface protocol maximizes the advantage of such memory by receiving or sending data during the same clock period in which the address is transmitted. Thus single-cycle reads and writes can be performed with sufficiently fast memory or peripheral devices.

Cached memory systems should use lower order address bits to address cache RAMs and higher order address bits to compare cache tags. There no strict definition of cache sizes or tag sizes. $\overline{HA}$ is used to synchronize an off-chip register known as the cache address register (CAR) with on-chip address registers. CAR operates as part of the IU pipeline and $\overline{HAL}$ inhibits the latch. For every cach access, the cache miss logic must send a hit or miss indication to the processor in the next cycle. If the cache hits, no wait state is inserted and the memory access completes in one cycle.

Coprocessor Interface

The integer unit is the basic processing engine which executes all of the instruction set except for floating-point operations. Software for non-floating-point intensive applications is supported. Where high performance floating-point is desirable, a floating-point controller (FPC) and IU operate concurrently. The FPC recognizes floating-point instructions and places them in a queue while the IU continues to execute non-floating-point instructions. If the FPC encounters an instruction which will not fit in its queue, the FPC holds the IU until the instruction can be stored. The FPC contains its own set of registers on which it operates. The contents of these registers are transferred to and from external memory under control of the IU via floating-point load/store instructions. Processor interlock hardware hides floating-point concurrency from the compiler or assembly language programmer. A program containing floating-point computation generates the same results as if instructions were executed sequentially.

Special Purpose Registers

The integer unit contains six 32-bit special purpose control/status registers which are used for general program control, setting modes of operation and showing processor status.

Processor status register (PSR) contains fields describing the state of the IU.

impl(31:28)	Implementation Number of the Processor
ver(27:24)	Version Number of the Processor
icc(23:20)	Integer Condition Codes (n, z, v, c)
reserved(19:14)	Reserved for Future Options
EC(13)	Enable Coprocessor
EF(12)	Enable Floating-Point Unit
PIL(11:8)	Processor Interrupt Level
S(7)	Supervisor Mode
PS(6)	Prior S-Bit (held at time of trap)
ET(5)	Enable Traps
CWP(4:0)	Current Window Pointer (marks current reg window)

Program Counter and Next Program Counter (PC and NPC)
PC contains the address of the instruction currently being executed by the IU. NPC holds the address of the next instruction to be executed (except when a trap occurs).

Window Invalid Mask Register (WIM)
WIM is used to determine whether a window overflow or underflow trap should be generated. Each bit of the WIM corresponds to a single register window. For the L64801 with seven register windows, only WIM(6:0) are used.

Trap Base Register (TBR)
TBR contains three fields that generate the address of the trap handler when a trap occurs.

TBA(31:12)	Trap Base Address (most significant 20 bits of trap table address)
tt(11:4)	Trap Type, provides offset into the trap table
zero(3:0)	Zero

Y Register
The Y register is used by the multiply step instruction to hold 32-bit results and create 64-bit products.

Control/status registers contain two types of fields, mode and status. Mode fields are set by the programmer and are designated through the use of an upper-case naming convention. Status fields are set by the processor and use a lower-case naming convention.

Exception Handling

The L64801 generates traps in response to both internal (synchronous) and external (asynchronous) events. These traps switch control from the instruction stream to an address in a trap table (except a reset trap which transfers control to virtual address 0). Synchronous traps occur immediately, not waiting for the current instruction to be completed. Asynchronous traps wait for the currently executing instruction to complete before they occur.

Each type of trap is assigned a priority; when multiple traps occur, the highest priority trap is taken and lower priority traps are ignored. To be taken, the request for the lower priority trap must either persist or be repeated.

Traps are vectored. The trap base address (TBA) register points to the trap table. Interrupts are given to the processor using four interrupt input signals. Any signal other than zero on these inputs is interpreted by the processor as an external interrupt request. This value is compared with the current processor interrupt level in the processor status register (PSR). The interrupt is taken if the external interrupt request level is greater than the processor interrupt level. The highest level interrupt (level 15) is nonmaskable. When a trap is detected, the processor takes the following actions:

1. The program counters corresponding to the trapped instruction and the instruction following the trapped instruction are saved in the register file.

2. The execution of the trapped instruction is aborted and all fetched but unfinished instructions are flushed out of the pipeline.

3. All traps are disabled. The processor mode is set to superuser and the CWP is set to point to the next window.

4. The trap address, based on the contents of the TBR and tt registers, is computed and loaded into the program counter.

5. Execution is restarted from the new trap address.

All external interrupts are ignored when traps are disabled. If a synchronous trap is detected while traps are disabled, the IU enters into an error mode and remains in that mode until the processor is reset externally. At reset, the processor enters into an initial state and starts execution from address 0.

Instruction Categories

The L64801 instructions fall into five basic categories:

Load and Store Instructions. (The only way to access memory). These instructions use two registers, or a register and a signed immediate value to generate the memory address. Integer load and store instructions support 8-, 16-, 32- and 64-bit accesses while floating-point instructions support 32- and 64-bit accesses.

- Load/Store Signed Byte
- Load/Store Signed Halfword
- Load/Store Unsigned Byte
- Load/Store Unsigned Halfword
- Load/Store Word
- Load/Store Double Word
- Load/Store Floating-Point Registers
- Load/Store Double Floating-Point Registers
- Load/Store Floating-Point State Register
- Store Double Floating-Point Queue

Arithmetic/Logical/Shift Instructions. These instructions compute a result that is a function of two source operands and then write the result back into a destination register. They perform arithmetic, tagged arithmetic, logical or shift operations. Tagged instructions are useful for implementing artificial intelligence languages such as LISP because tags provide interpreters with the type of arithmetic operands.

- Add (w/wo modifying condition codes)
- Add with Carry (w/wo modifying condition codes)
- Tagged Add (w/wo trap on overflow)
- Subtract (w/wo modifying condition codes)
- Subtract with Carry (w/wo modifying condition codes)
- Tagged Subtract (w/wo trap on overflow)
- Multiply Step (modify condition codes)
- AND (w/wo modifying condition codes)
- NAND (w/wo modifying condition codes)
- OR (w/wo modifying condition codes)
- NOR (w/wo modifying condition codes)
- Exclusive-OR (w/wo modifying condition codes)
- Exclusive-NOR (w/wo modifying condition codes)
- Shift Left Logical
- Shift Right Logical
- Shift Right Arithmetic
- Set High 22 Bits of Register

Coprocessor Operations. These include floating-point calculations, operations on floating-point registers and instructions involving the optional coprocessor. Floating-point operations execute concurrently with IU instructions and with other floating-point operations when necessary. This architectural concurrency hides floating-point operations from the applications programmer.

- Convert Integer to Single/Double/Extended Precision
- Convert Single/Double/Extended Precision to Integer (w/wo rounding)
- Convert Single Precision to Double/Extended Precision
- Convert Double Precision to Single/Extended Precision
- Move/Negate/Absolute Value
- Square Root Single/Double/Extended
- Add Single/Double/Extended
- Subtract Single/Double/Extended
- Multiply Single/Double/Extended
- Divide Single/Double/Extended
- Compare Single/Double/Extended (w/wo exception if unordered)

Control-Transfer Instructions. These include jumps, calls, traps and branches. Control transfers are usually delayed until after execution of the next instruction, so that the pipeline is not emptied every time a control transfer occurs. Thus, compilers can be optimized for delayed branching. Branch and call instructions use program counter relative displacements. A jump and link instruction uses a register indirect displacement computing its target address as either the sum of two registers, or the sum of a register and a 13-bit signed immediate value. The branch instruction provides a displacement of eight megabytes and the call instructions 30-bit displacement allows transfer to any address.

- Increment Current Window Pointer
- Decrement Current Window Pointer
- Branch on Integer Condition Codes
- Trap on Integer Condition Codes
- Branch on Floating-Point Condition Codes
- Call
- Jump and Link
- Return from Trap

Read/Write Control Register Instructions. These include instructions to read and write the contents of various control registers. Generally the source or destination is implied by the instruction.

- Read/Write Multiply Step Register
- Read/Write Processor State Register
- Read/Write Window Invalid Mask Register
- Read/Write Trap Base Register
- Flush Instruction Cache

ıstruction Categories :ontinued)

Instruction Execution Times. All instructions execute in a single cycle except the following instructions:

Instruction Type	Cycles
Load (word/halfword/byte)	2
Load (double)	3
Store (word/halfword/byte)	3
Store (double)	4
Atomic Load and Store	4
Floating-Point Ops	2+Cf
Jump and Rett	2
Branch (taken)	1
Branch (untaken)	2
All Other Instructions	1

ıstruction Set ·ummary

Opcode	Name
LDSB (LDSBA†)	Load Signed Byte (from Alternate Space)
LDSH (LDSHA†)	Load Signed Halfword (from Alternate Space)
LDUB (LDUBA†)	Load Unsigned Byte (from Alternate Space)
LDUH (LDUHA†)	Load Unsigned Halfword (from Alternate Space)
LD (LDA†)	Load Word (from Alternate Space)
LDD (LDDA)†	Load Doubleword (from Alternate Space)
LDF	Load Floating-Point
LDDF	Load Double Floating-Point
LDFSR	Load Floating-Point State Register
LDC*	Load Coprocessor
LDDC*	Load Double Coprocessor
LDCSR*	Load Coprocessor State Register
STB (STBA†)	Store Byte (into Alternate Space)
STH (STHA†)	Store Halfword (into Alternate Space)
ST (STA†)	Store Word (into Alternate Space)
STD (STDA†)	Store Doubleword (into Alternate Space)
STF	Store Floating-Point
STDF	Store Double Floating-Point
STFSR	Store Floating-Point State Register
STDFQ†	Store Double Floating-Point Queue
STC*	Store Coprocessor
STDC*	Store Double Coprocessor
STCSR*	Store Coprocessor State Register
STDCQ†*	Store Double Coprocessor Queue
LDSTUB (LDSTUBA†)	Atomic Load-Store Unsigned Byte (in Alternate Space)
SWAP (SWAPA†)	Swap r Register with Memory (in Alternate Space)
ADD (ADDcc)	Add (and Modify icc)
ADDX (ADDXcc)	Add with Carry (and Modify icc)
TADDcc (TADDccTV)	Tagged Add and Modify icc (and Trap on Overflow)
SUB (SUBcc)	Subtract (and Modify icc)
SUBX (SUBXcc)	Subtract with Carry (and Modify icc)
TSUBcc (TSUBccTV)	Tagged Subtract and Modify icc (and Trap on Overflow)

Opcode	Name
MULScc	Multiply Step and Modify icc
AND (ANDcc)	And (and Modify icc)
ANDN (ANDNcc)	And Not (and Modify icc)
OR (ORcc)	Inclusive-Or (and Modify icc)
ORN (ORNcc)	Inclusive-Or Not (and Modify icc)
XOR (XORcc)	Exclusive-Or (and Modify icc)
XNOR (XNORcc)	Exclusive-Nor (and Modify icc)
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
SETHI	Set High 22 bits of r register
SAVE	Save Caller's Window
RESTORE	Restore Caller's Window
Bicc	Branch on Integer Condition Codes
FBfcc	Branch on Floating-Point Condition Codes
CBccc	Branch on Coprocessor Condition Codes
CALL	Call
JMPL	Jump and Link
RETT†	Return from Trap
Ticc	Trap on Integer Condition Codes
RDY	Read Y Register
RDPSR†	Read Processor State Register
RDWIM†	Read Window Invalid Mask Register
RDTBR†	Read Trap Base Register
WRY	Write Y Register
WRPSR†	Write Processor State Register
WRWIM†	Write Window Invalid Mask Register
WRTBR†	Write Trap Base Register
UNIMP	Unimplemented Instruction
IFLUSH	Instruction Cache Flush
FPop	Floating-Point Operate: FiTO(s, d, x), F(s, d, x)TOi FsTOd, FsTOx, FdTOs, FdTOx, FxTOs, FxTOd, FMOVs, FNEGs, FABSs, FSQRT(s, d, x), FADD(s, d, x), FSUB(s, d, x), FMUL(s, d, x), FDIV(s, d, x), FCMP(s, d, x), FCMPE(s, d, x)
CPop	Coprocessor Operate

*Unimplemented Instruction
†Privileged Instruction

Instruction Formats (Summary)

Format 1: CALL

op	disp30
31	29 … 0

Format 2: SETHI and Branches (Bicc, FBfcc, CBcc)

op	a	cond	op2	imm22 / disp22
op	rd		op2	imm22
op	a	cond	op2	disp22
31	29	28	24	21 … 0

Format 3: Remaining Instructions

op	rd	op3	rs1	i	asi	rs2
op	rd	op3	rs1	i	simm13	
op	rd	op3	rs1	opf		rs2
31	29	24	18	13	12	4 … 0

Instruction Format Field Definitions

op
This field places the instruction into one of the three major formats:

Use of op Field

Format	op Value	Instruction
1	1	Call
2	0	Bicc, FBfcc, CBccc, SETHI
3	2 or 3	Other

op2
This field comprises bits 24 through 22 of format 2 instructions. It selects the instruction as follows:

Use of op2 Field

op2 Value	Instruction
0	UNIMP
2	Bicc
4	SETHI
6	FBfcc
7	CBccc

rd
For store instructions, this register selects an r register (or an f register pair), or an f register (or an f register pair) to be the source. For all other instructions, this field selects an r register (or an f register pair), or an f register (or an f register pair) to be the destination.

Note: Reading r[0] produces the result 0, and writing it causes the result to be discarded.

a
The "a" bit means "annul" in format 2 instructions. This bit changes the behavior of the instruction encountered immediately after a control transfer.

cond
This field selects the condition code for format 2 instructions.

imm22
This field is a 22-bit constant value used by the SETHI instruction.

disp22 and disp30
These fields are 30-bit and 22-bit sign-extended word displacements, for PC-relative calls and branches, respectively.

op3
The op3 field selects one of the format 3 opcodes.

i
The i bit selects the type of the second ALU operand for non-FPop instructions. If i = 0, the second operand is r[rs2]. If i = 1, the second operand is sign-extended simm13.

asi
This 8-bit field is the address space identifier generated by load/store alternate instructions.

Instruction Format Field Definitions (Continued)

rs1
This 5-bit field selects the first source operand from either the r registers or the f registers.

rs2
This 5-bit field selects the second source operand from either the r registers or the f registers.

simm13
This field is a sign-extended 13-bit immediate value used as the second ALU operand when i = 1.

opf
This 9-bit field identifies a floating-point operate (FPop) instruction or a coprocessor operate (CPop) instruction.

Pin Descriptions

The signals on the L64801 are divided into three main categories: memory subsystem interface signals, floating-point unit interface signals and miscellaneous I/O signals. Signals which are asserted LOW are indicated by an overscore.

Memory Subsystem Interface Signals

A[31:0]
Address Bus
The address bus is output directly from an on-chip memory address register and is valid every cycle. During an instruction fetch cycle, the bus carries an instruction address, and during a load or store data cycle, it carries a data address. The address bus remains valid during all data cycles of loads, stores, load doubles and atomic load/stores. In systems with cache, the low bits of the address are used to read the cache RAMs and cache TAGs, and the high bits of the address are used to compare the TAGs.

ASI[7:0]
Address Space Identifier
These bits identify the address space during instruction or data accesses. The value of these signals at any given cycle represents the address space containing the memory address specified by A[31:0] during that cycle. ASI[7:0] remains valid on the bus during all data cycles of loads, stores, load doubles, and atomic load/stores. ASI[7:0] pins are 3-stated if $\overline{AOE}$ is disasserted. The following ASI values are currently assigned:

ASI	Address Space
00001000	User Instruction
00001010	User Data
00001001	Supervisor Instruction
00001011	Supervisor Data

During the data cycles of alternate load and store instructions, ASI[7:0] carries the space identifier specified by the instruction opcode.

D[31:0]
Data Bus
The bidirectional data bus to and from the IU. It is driven by the IU only during the execution of integer store instructions or during the store cycle of atomic load/store instructions. It is driven by the FPC only during the execution of floating-point store instructions. The alignment for load and store instruction is done inside the IU, which always expects instructions to be fetched from 32-bit wide memory.

$\overline{MEXC}$ (Asserted LOW)
Memory Exception Input
The memory or cache controller asserts this signal to signal an instruction-access-exception, or a data-access-exception. It is latched in the IU and used during the following cycle. If $\overline{MEXC}$ is asserted during an instruction fetch cycle, the IU generated an instruction access exception. If $\overline{MEXC}$ is asserted during a data fetch cycle, the IU generates a data access exception trap.

$\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, $\overline{SHOLD}$ (Asserted LOW)
Hold From Memory
These signals freeze the processor pipeline as long as any of them are asserted. They are used to freeze the clock to the IU and FPU during a cache miss (for system with cache), or when accessing a slow memory. The IU hardware uses the logical OR of $\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, and $\overline{SHOLD}$ to generate a final $\overline{MHOLD}$ for freezing the processor pipeline.

$\overline{BHOLD}$ (Asserted LOW)
Hold From I/O System
The I/O controller asserts this signal when an external bus master needs the data bus. This signal freezes the processor pipeline. External logic should guarantee that the data on the inputs to the IU is the same after $\overline{BHOLD}$ is disasserted as it was before $\overline{BHOLD}$ was asserted.

$\overline{DOE}$ (Asserted LOW)
Data Bus Output Enable
This signal turns on the output drivers to the D[31:0] bus. It is connected directly to the drivers and therefore must normally be asserted. It may be disasserted only when the bus is to be used by another bus master. This should only occur when $\overline{BHOLD}$, $\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, or $\overline{SHOLD}$ is asserted.

Pin Descriptions
(Continued)

AOE (Asserted LOW)
Address Bus Output Enable
This signal enables the A[31:0] outputs. It is normally asserted except when the bus is to be used by another bus master.

ASIOE (Asserted LOW)
Address Space Indentifier Output Enable
ASIOE enables the ASI outputs. It is normally asserted except when the bus is to be used by another bus master.

MDS (Asserted LOW)
Memory Data Input Strobe During Hold
This signal enables the clock input to the on-chip instruction register (during an instruction fetch), or to the load result register (during a data fetch). It is used in systems with cache or with slow memory, to signal the processor when data is ready on the bus. It should only be asserted when the processor pipeline is frozen (MHOLDA, MHOLDB, MHOLDC, or SHOLD is asserted).

TC (Asserted LOW)
Trap Condition
The state of this signal controls the behavior of the IFLUSH instruction. If TC is HIGH, IFLUSH executes like NOP with no side effects. If TC is LOW, IFLUSH causes an unimplemented instruction trap.

SIZE [1:0]
Data Bus Transfer Size
SIZE represents the data size of the memory address currently on A[31:0]. They remain valid on the bus during all data cycles of loads, stores, load doubles, store doubles, and atomic load/stores. They are encoded as follows:

Size 1:0	Data Size
00	Byte
01	Halfword
10	Word
11	Word for LDDF, STDF, and STDFQ

LDST
Load/Store Cycle
This signal is asserted during all data cycles of atomic load/store instructions. LDST is 3-stated if AOE is disasserted

RD
Read Cycle
This signal is set LOW during data cycles of store instructions (including the store cycles of atomic load/store instructions). In conjunction with SIZE[1:0], ASI[7:0], and LDST, it can be used to determine the type of a bus transaction, and to check read/write access rights. RD may also be used to turn off the output drivers of data RAMs during a store operation. For atomic load/store instructions, RD is HIGH during the first data (read) cycle, and LOW during the second and third data (write) cycles. RD is 3-stated if AOE is disasserted.

WE (Asserted LOW)
Write Cycle
This signal is asserted only during 1) the second data cycle of store instructions, 2) the second and third data cycles of store double instructions, or 3) the third data cycle of atomic load/store instructions. This signal is 3-stated when not asserted.

NULL__CYC
Null Cycle
This signal indicates that the current memory address (whose address is held in the external memory address register) is nullified by the IU. It is used to disable cache miss in systems with cache, and for memory exception handling during the current memory access.

IH__NULL (Asserted LOW)
Null Cycle Reset
When active, this signal resets NULL__CYC to LOW.

LOCK
Bus Lock Request
LOCK is set HIGH when the IU needs the bus for multiple-cycle transactions. The bus may not be granted to another bus master as long as LOCK is active.

HAL (Asserted LOW)
Hold Address Latch
HAL freezes the clock to the external memory address register. It is asserted during the execution of some multiple-cycle instructions, internal interlocks and whenever at least one of the hold signals (MHOLDA, MHOLDB, SHOLD, BHOLD, or FHOLD) is asserted.

DFETCH
Data Fetch Cycle
DFETCH marks the beginning of a data cycle. When DFETCH is HIGH, it indicates a data cycle and when DFETCH is LOW, it indicates an instruction cycle. The IU can nullify an instruction or data cycle by asserting NULL__CYC.

Pin Descriptions
(Continued)

Floating-Point Unit Interface Signals

The floating-point unit interface is a dedicated group of connections between the IU and the FPC and no external circuits are required. The interface consists of the following signals:

FP (Asserted LOW)
Floating-Point Unit Is Present
When FP is LOW, it indicates that an FPU exists in the system. FP is tied to VDD by an internal resistor and is pulled to ground only when the FPU is present. The IU generates an fp__disabled trap if FP is HIGH during the execution of a floating-point instruction, a floating-point load or store, or an FBfcc.

FCC[1:0]
Condition Code Inputs
The floating-point condition codes are valid only if FCCV is HIGH. An FBfcc instruction uses these bits to compute the next instruction address, and then waits if FCCV is LOW.

FCCV
Condition Codes Valid
The FPU asserts FCCV to indicate that FCC[1:0] are valid. The FPU must guarantee that FCCV is LOW (disasserted) if floating-point compare instructions are pending in the floating-point queue.

FHOLD (Asserted LOW)
Hold Input
The FPU asserts FHOLD when it cannot continue executing instructions. When it receives an instruction, the FPU checks for dependencies, and if any are discovered, it asserts FHOLD during the same cycle or during the cycle that follows. FHOLD is latched into the IU, where it freezes the instruction pipeline in the following cycle. The FPU must disassert FHOLD to unfreeze the IU's instruction pipeline.

FEXC (Asserted LOW)
Exception Input
The FPC asserts FEXC to indicate that a floating-point exception has occurred. It must remain asserted until the IU takes the trap and acknowledges by asserting FXACK. Floating-point exceptions are only taken during execution of floating-point instructions.

F[31:00]
Floating-Point Bus
This dedicated 32-bit bus sends floating-point instructions and addresses to the FPU chip. Each floating-point instruction uses this bus for two cycles; the first cycle carries the instruction and the second cycle carries the address.

FINS
Floating-Point Instruction
The IU asserts FINS during the cycle in which F[31:00] carries a valid floating-point instruction. The FPU uses this signal to latch the instruction into its instruction register.

FADR
Floating-Point Address
The IU asserts FADR during the cycle in which F[31:00] carries a valid floating-point instruction address. The FPU uses this signal to latch the address into its address register.

FEND
End Floating-Point Instruction
The IU generates FEND, which the FPU uses to synchronize the instruction/address in its execution pipeline with the IU's pipeline. The IU asserts FEND during the last cycle of a floating-point instruction in the IU's pipeline.

FLUSH
Flush Floating-Point Instruction
The IU asserts FLUSH to cause the FPU to flush the instruction in its instruction register. This may happen when the IU takes a trap. FLUSH has no effect on instructions in the floating-point queue.

FXACK
Exception Acknowledge
The IU asserts FXACK to indicate to the FPU that the current FEXC trap has been taken. The FPU must disassert FEXC after it receives FXACK so that the next floating-point instruction does not cause a repeated floating-point exception trap.

Pin Descriptions
(Continued)

Miscellaneous I/O Signal Descriptions

These signals are used by the IU to control external events or to receive input from external events.

$\overline{\text{RESET}}$ (Asserted LOW)
Reset Input
Assertion of this pin will reset the Integer Unit. The $\overline{\text{RESET}}$ signal must be asserted for a minimum of eight processor clock cycles. After a $\overline{\text{RESET}}$, the Integer Unit will start fetching from address 0.

IRL[3:0]
Interrupt Request Level
The value on IRL defines the external interrupt request level. When IRL[3:0] = 0000, no interrupts are pending. External interrupts must be latched and prioritized by external logic before they are passed to the IU and held until they are acknowledged by the IU. External interrupts must be acknowledged by software.

$\overline{\text{ERROR}}$ (Asserted LOW)
Processor In Error State
When the IU detects a trap while the ET bit in the PSR is 0, the processor saves the PC and NPC, sets the tt value in the TBR, enters into an error state, asserts $\overline{\text{ERROR}}$ and halts. To restart the processor from this state, external logic should send a $\overline{\text{RESET}}$ to the chip.

CLK
Clock Input
The rising edge of CLK defines the beginning of each pipeline stage in the IU chip. CLK can have any duty cycle ranging from 30% to 70%.

XSM
Scan Mode Input
During test and debug, this signal disables the normal clocks and activates the scan clocks for scan operations. XSM must be set HIGH during normal operation.

SDO
Scan Data Output
SDO is the serial data output for the IU's scan path.

PTREEO
Parametric Tree Output
This signal is the output of an internal test string, which test parametric input levels during test. PTREEO is 3-stated when XSM is set HIGH. It need not be connected for normal operation.

L64801 High Performance Open Architecture RISC Microprocessor

Preliminary

LSI LOGIC

Pin Description Summary Table

Pin Name	Description	Input/Output	Active
A (31:0)	Address	3-State Output	
ASI (7:0)	Address Space Identifier	3-State Output	
D (31:0)	Data	3-State Bidirectional	
HAL	Hold Address Latch	Output	LOW
WE	Write Enable	Output	LOW
RD	Read	Output	HIGH
DFETCH	Data Fetch Cycle	Output	HIGH
SIZE (1:0)	Bus Transaction Size	3-State Output	
LOCK	Multi-Cycle Bus Lock	3-State Output	
MDS	Memory Data Strobe	Input	LOW
AOE	Address Output Enable	Input	LOW
ASIOE	ASI Output Enable	Input	LOW
DOE	Data Output Enable	Input	HIGH
MHOLDA	Memory Hold A	Input	LOW
MHOLDB	Memory Hold B	Input	LOW
MHOLDC	Memory Hold C	Input	LOW
BHOLD	Bus Hold	Input	LOW
SHOLD	System Hold	Input	LOW
IRL (3:0)	Interrupt Request Level	Input	
RESET	Reset	Input	LOW
TC	Trap Condition	Input	LOW
MEXC	Memory Exception	Input	LOW
ERROR	IU Error Mode	Output	LOW
LDST	Load/Store Operation	3-State Output	HIGH
NULL_CYC	Null Cycle	3-State Output	HIGH
IH_NULL	Null Cycle Reset	Input	LOW
PTREEO	Parametric Tree Output	Output	
TSTO	Test Output	Output	
XSM	Scan Mode Input	Input	HIGH
FINS	Floating-Point Instruction	3-State Output	HIGH
FADR	Floating-Point Address	3-State Output	HIGH
FEND	End Floating-Point Instruction	3-State Output	HIGH
FLUSH	Flush Floating-Point Instruction	3-State Output	HIGH
FXACK	Floating-Point Exception Acknowledge	3-State Output	HIGH
FP	Floating-Point Unit Present	Input w/Pullup	LOW
FCCV	FPU Condition Codes Valid	Input	HIGH
FCC (1:0)	FPU Condition Codes	Input	
FHOLD	FPU Hold	Input	LOW
FEXC	FPU Exception	Input	LOW
F (31:0)	Floating-Point Bus	3-State Output	
CLK	System Clock	Input	
VDD	Input Circuit Power	Power	
GND	Input Circuit Ground	Ground	

High Performance Open Architecture RISC Microprocessor

Preliminary

Operating Characteristics

Absolute Maximum Ratings (Referenced to VSS)

Parameter	Symbol	Limits	Unit
DC Supply Voltage	VDD	-0.3 to +7	V
Input Voltage	VIN	-0.3 to VDD +0.3	V
DC Input Current	IIN	±10	mA
Storage Temperature Range (Ceramic)	TSTG	-65 to +150	°C
Storage Temperature Range (Plastic)	TSTG	-40 to +125	°C

Recommended Operating Conditions

Parameter	Symbol	Limits	Uni
DC Supply Voltage	VDD	+3 to +6	V
Operating Ambient Temperature Range Military	TA	-55 to +125	°C
Industrial Range	TA	-40 to +85	°C
Commercial Range	TA	0 to +70	°C

DC Characteristics: Specified at VDD = 5 V ±5% ambient temperature over the specified temperature range[1].

Symbol	Parameter	Condition			Min	Typ	Max	Unit
VIL	Voltage Input LOW							
	TTL Inputs						0.8	V
	CMOS Levels						1.5	V
VIH	Voltage Input HIGH							
	TTL Inputs, Commercial Temperature Range				2.0			V
	TTL Inputs, Military and Industrial Temperature Range				2.25			V
	CMOS Levels				3.5			V
VT+	Schmitt-Trigger, Positive-going Threshold					3.0	4.0	V
VT-	Schmitt-Trigger, Negative-going Threshold				1.0	1.5		V
⎍	Hysteresis, Schmitt Trigger	VIL to VIH VIH to VIL			1.0	1.5		V
IIN	Input Current, CMOS, TTL Inputs	VIN = VDD or VSS			-10	±1	10	µA
	Inputs with Pulldown Resistors	VIN = VDD			10	35	120	µA
	TTL Inputs & Inputs with Pullup Resistors	VIN = VSS			-8	-30	-100	µA
VOH	Voltage Output HIGH		Comm	Mil				
	Type B1	IOH =	-1 mA	-0.8 mA				
	Type B2	IOH =	-2 mA	-1.6 mA	2.4	4.5		V
	Type B4	IOH =	-4 mA	-3.2 mA				
	Type B6	IOH =	-6 mA	-4.8 mA				
	Type B8	IOH =	-8 mA	-6.4 mA				
	Type B12[2]	IOH =	-12 mA	-9.6 mA				
VOL	Voltage Output LOW		Comm	Mil				
	Type B1	IOL =	1 mA	0.8 mA				
	Type B2	IOL =	2 mA	1.6 mA		0.2	0.4	V
	Type B4	IOL =	4 mA	3.2 mA				
	Type B6	IOL =	6 mA	4.8 mA				
	Type B8	IOL =	8 mA	6.4 mA				
	Type B12[2]	IOL =	12 mA	9.6 mA				
IOZ	3-State Output Leakage Current	VOH = VSS or VDD			-10	±1	10	µA
IOS	Output Short Circuit Current[3]	VDD = Max, VO = VDD			15	50	130	mA
		VDD = Max, VO = 0 V			-5	-25	-100	mA
IDD	Quiescent Supply Current	VIN = VDD or VSS			User-Design Dependent			
CIN	Input Capacitance	Any Input[4]				2		pF
COUT	Output Capacitance	Any Output[5]				4		pF

Notes:
1. Military temperature range is -55°C to +125°C, ±10% power supply (ceramic packages only); industrial temperature range is -40°C to +85°C, ±5% power supply; commercial temperature range is 0°C to 70°C, ±5% power supply.
2. Requires two output pads.
3. Type B4 output. Output short circuit current for other outputs will scale. Not more than one output may be shorted at a time for a maximum duration of one second
4. Not applicable to assigned bidirectional buffer (excluding package).
5. Output using single buffer structure (excluding package).

System Interface Timing

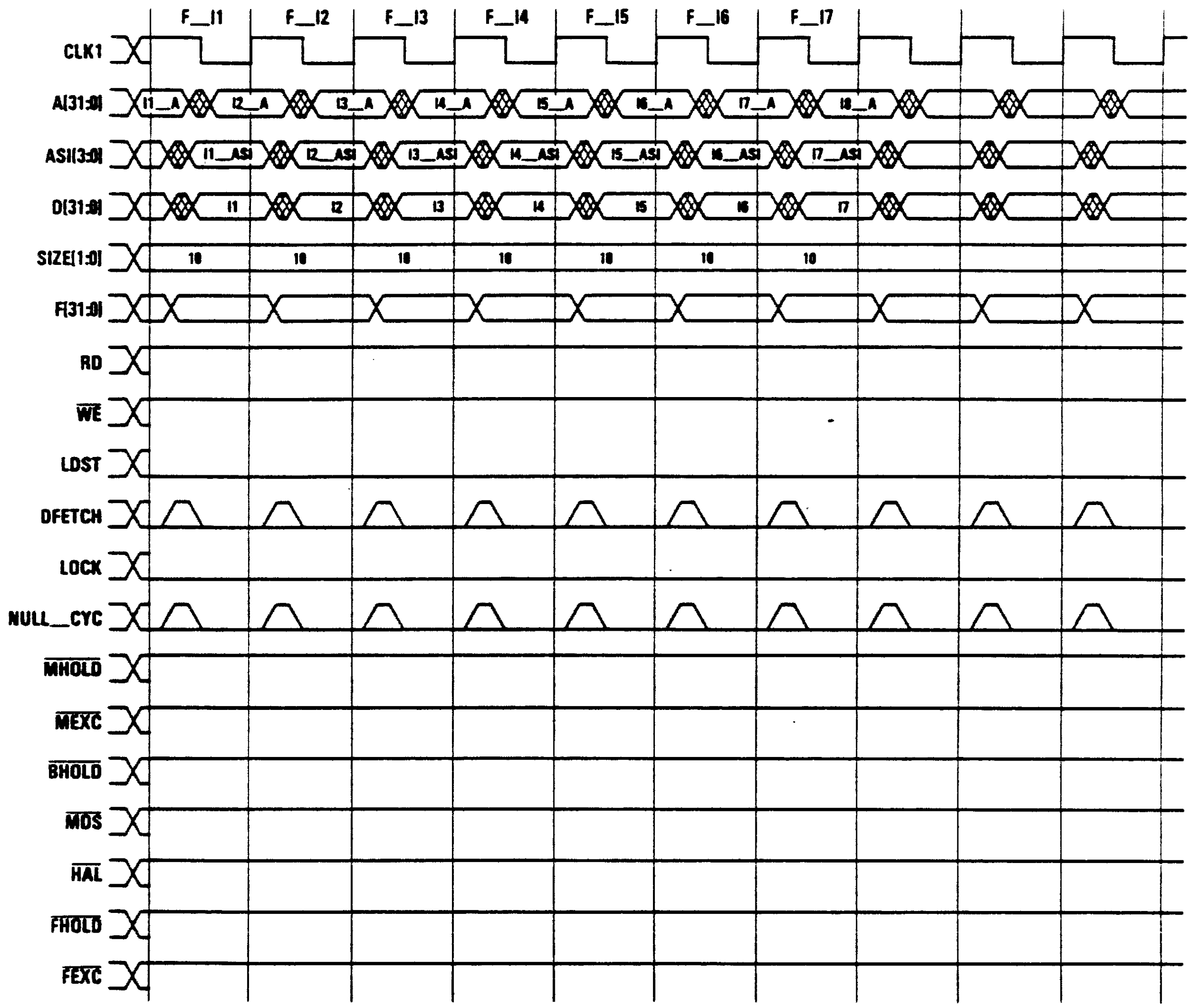


Figure 7. Instruction Fetch Timing

Load Transactions

Figure 8 shows the timing for a load integer instruction. This instruction causes a one-cycle delay; during T4, the bus contains the datum to be loaded and the processor cannot use it to fetch I4. Because of this delay, I4 is fetched during T5.

The delay also gives the IU time to deal with any trap caused by I1.

Figure 9 shows the timing for a load double integer. This works similarly to the load integer, except that it uses the bus during T4 to load the first half and during T5 to load the second. Note that the address of the second load is equal to the address of the first load +4 and that the size bits = 1,1 during T4 and T5. The processor fetches I4 during T6.

Figure 10 shows the timing for a load floating-point instruction. It works like the load integer except that is also generates floating-point control signals in T3, T4 and T5.

Figure 11 shows the timing for a load double floating-point instruction. It works like the double integer instruction except that it generates additional floating-point signals T3, T4 and T5.

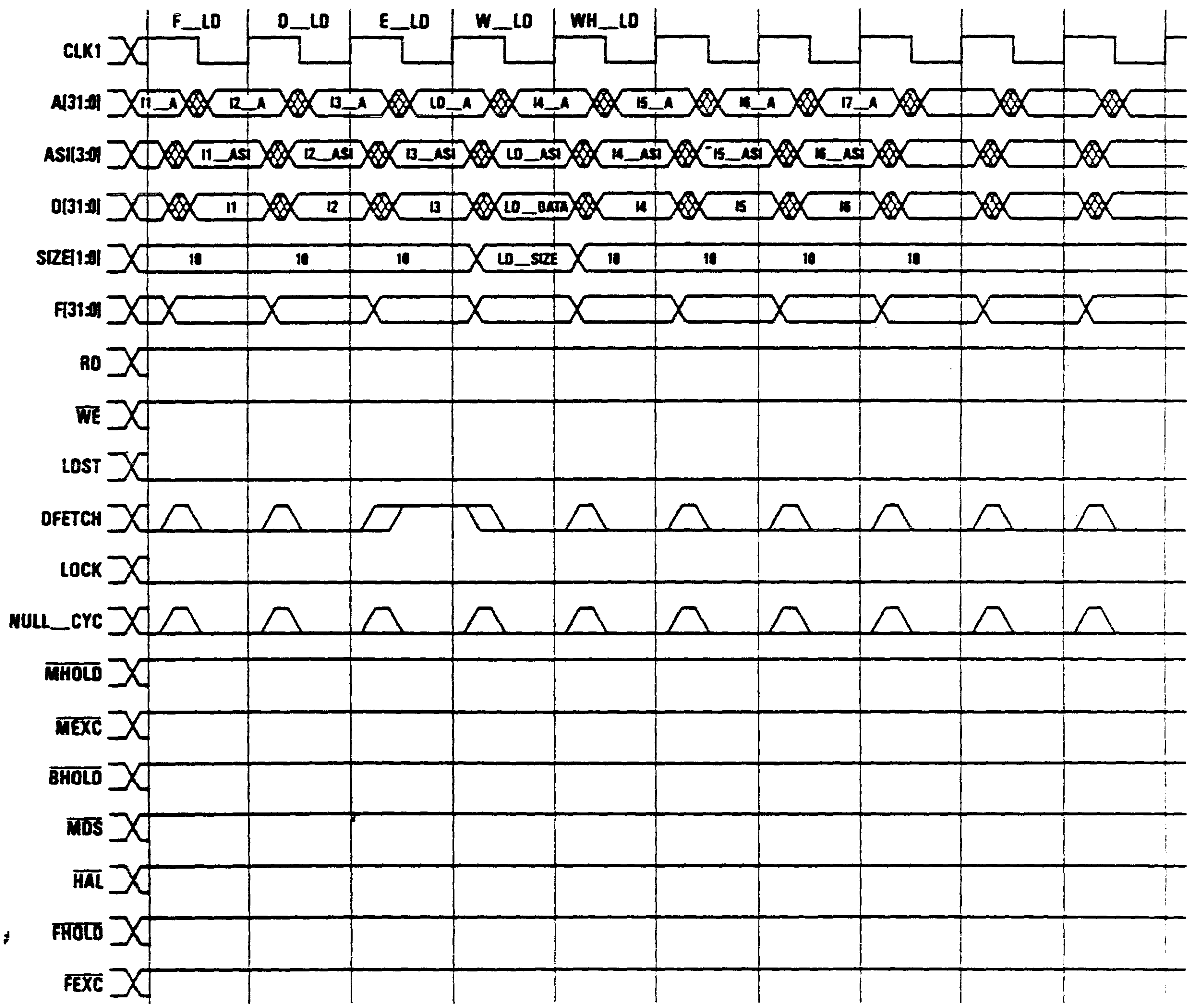


Figure 8. Load Integer Timing

LSI LOGIC

Load Transactions
(Continued)

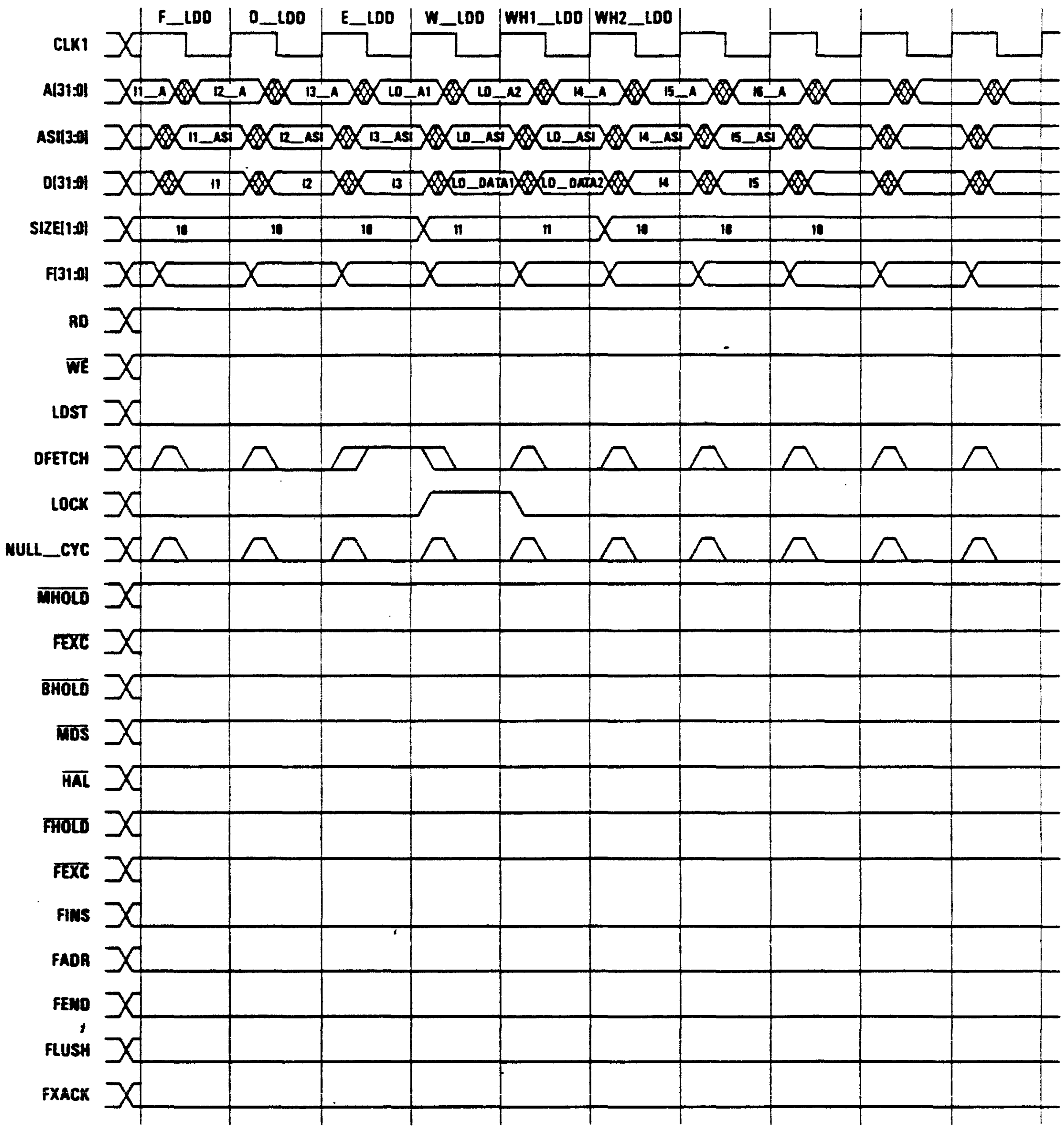


Figure 9. Load Double Integer Timing

Load Transactions
(Continued)

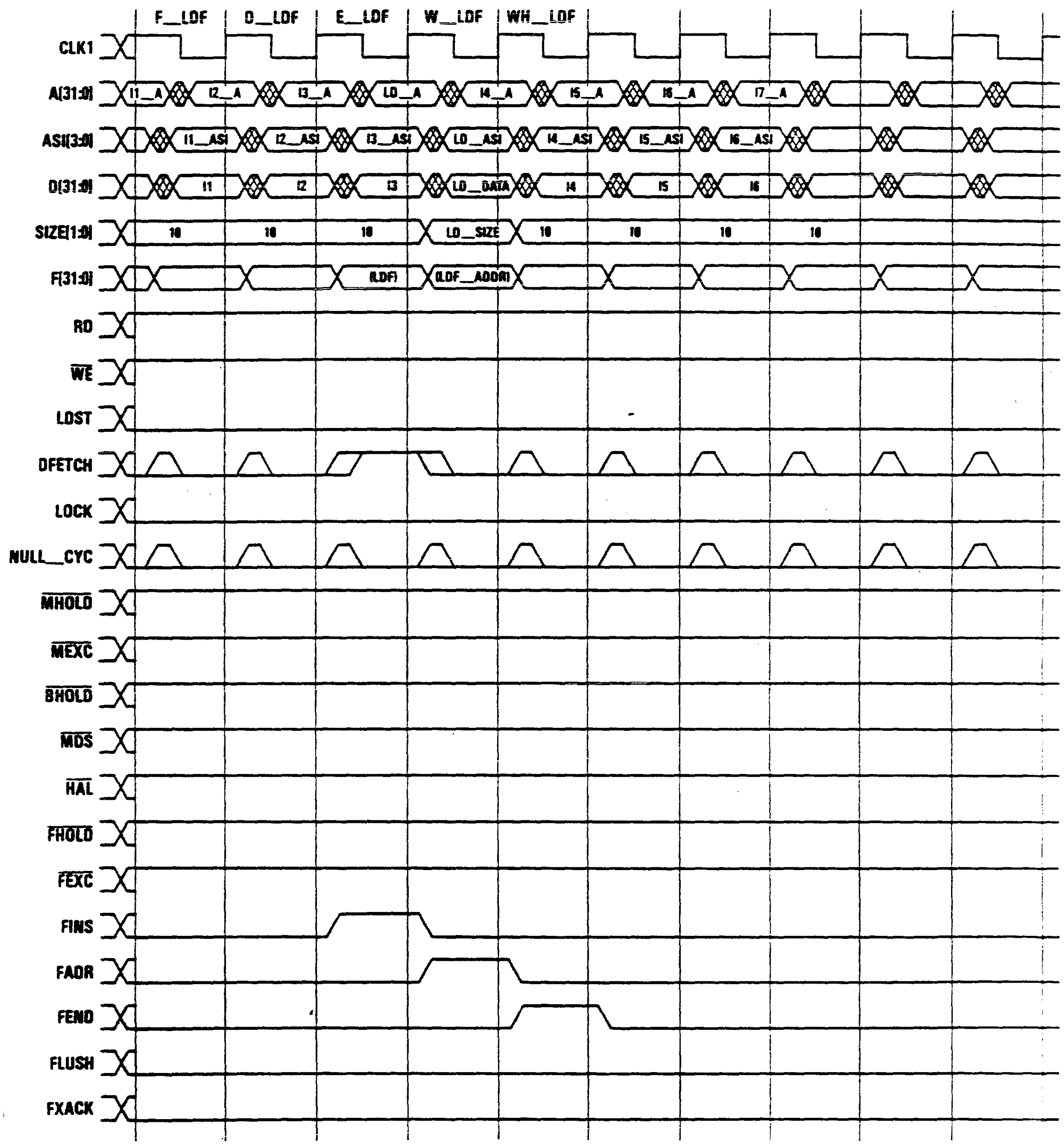


Figure 10. Load Floating-Point Timing

Load Transactions
(Continued)

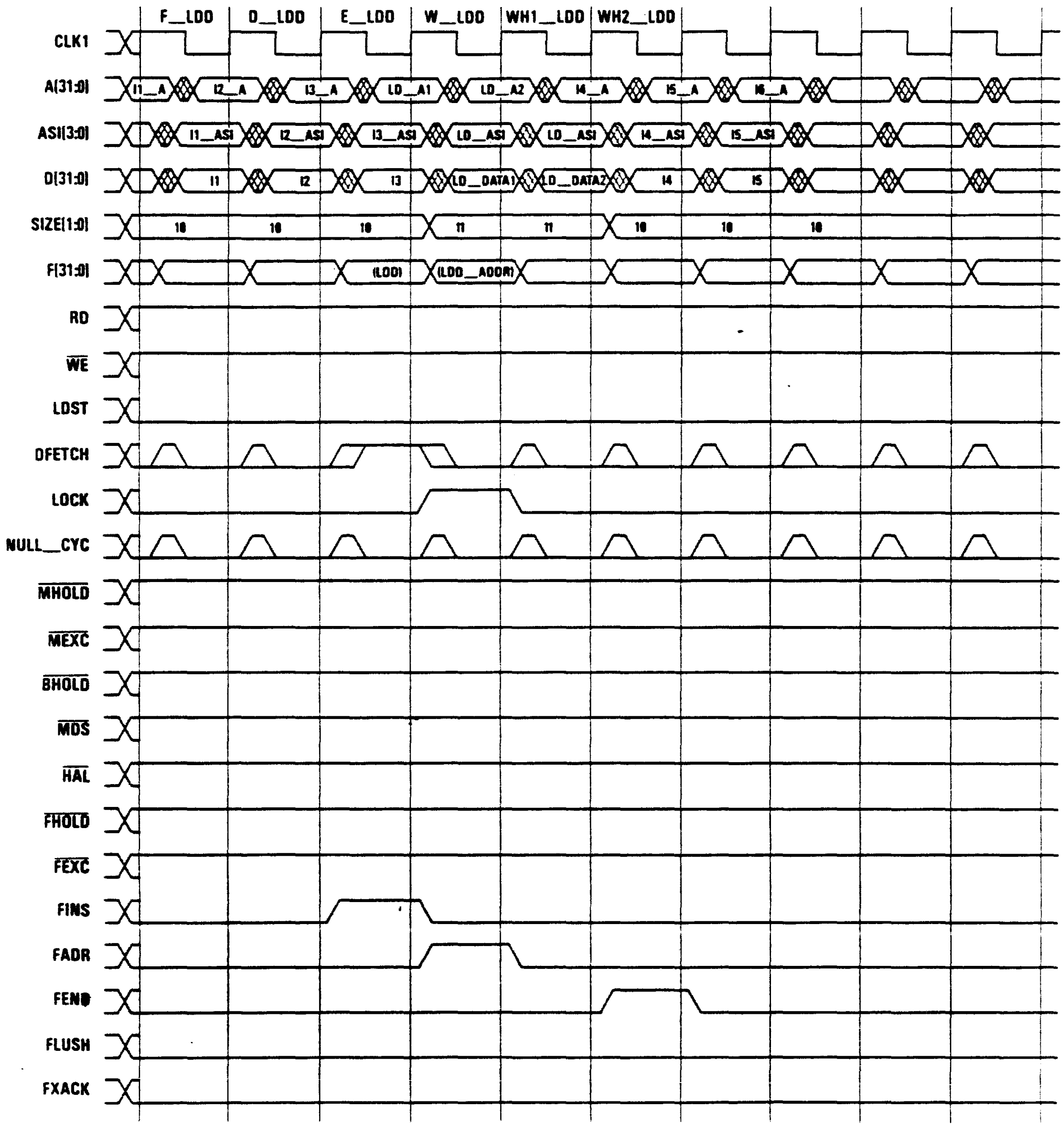


Figure 11. Load Double Floating-Point Timing

Store Transactions

Figure 12 shows a store integer instruction; these take two extra cycles. During T4, the address of the store goes on the bus; during T5, the address remains on the bus and store data goes on the bus as well. This requires two extra cycles because the processor cannot send both the address and the data out simultaneously, and because the processor has to wait to see if the store is going to generate an exception or a cache miss. It fetches I4 during T6.

Figure 13 shows the timing for a store double integer instruction. It works like the store integer timing except that the processor must delay an extra cycle to repeat the store operation for the second word. Note that the address of the second store equal to the first address +4, and that the size b are set to 1,1 to indicate a double operand.

Figure 14 shows the timing for a floating-point store. This works similarly to the integer store, except that it generates the additional floating-point signals, FINS, FADR, and FEND during T3, and T6.

Figure 15 shows the timing for a store double flo ing-point instruction. It works just like the store floating-point instruction except that it requires a extra cycle to store the second half of the floatin point operand.

Figure 12. Integer Store Timing

Store Transactions
(Continued)

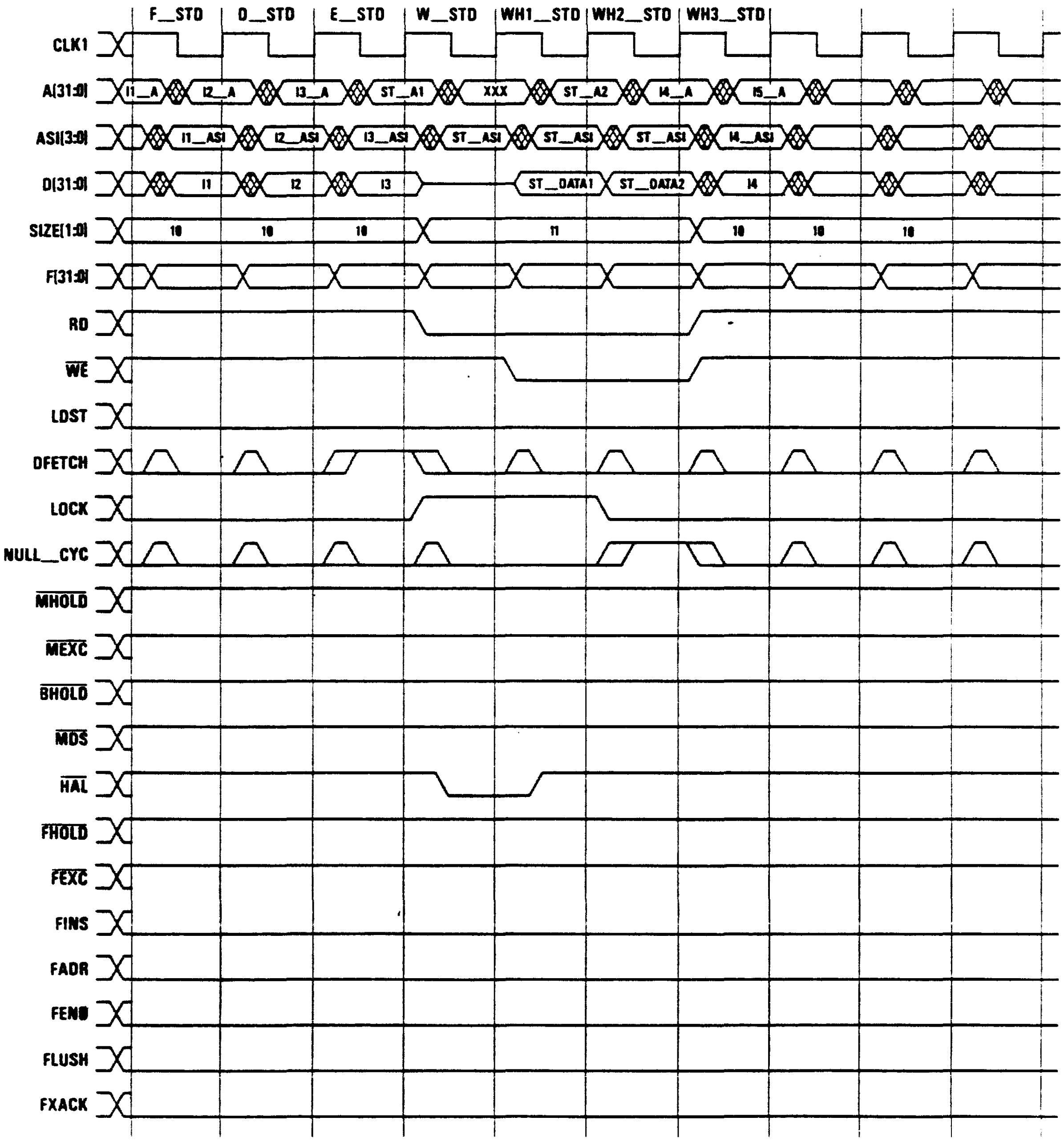


Figure 13. Store Double Integer Timing

Store Transactions
(Continued)

Figure 14. Store Floating-Point Timing

Store Transactions
(Continued)

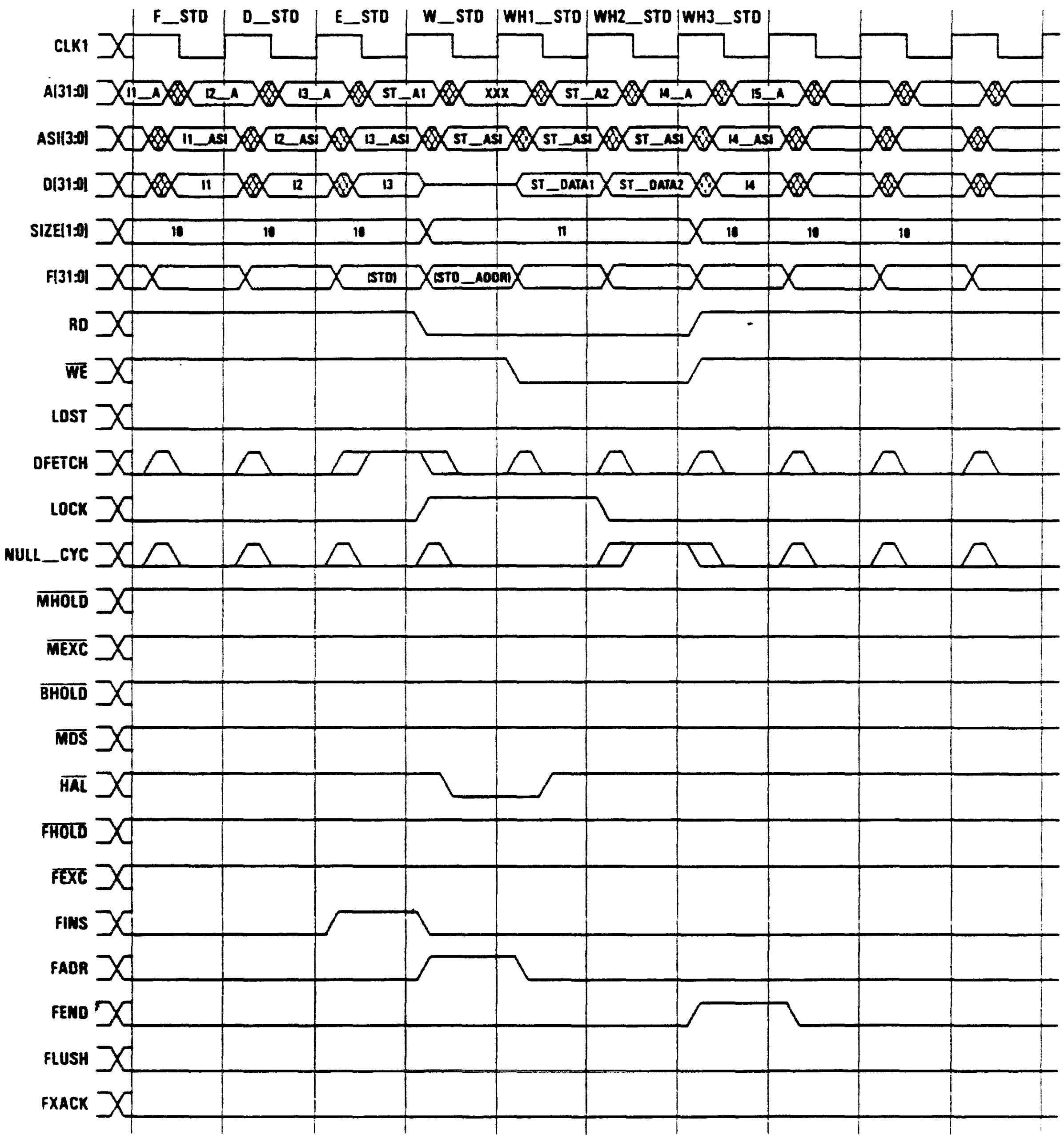


Figure 15. Store Double Floating-Point Timing

Atomic Transactions

Atomic transactions consist of two or more steps which are indivisible; once the sequence is started, it cannot be interrupted. To ensure that it has the bus for the second transaction, the IU asserts LOCK for as long as necessary.

The atomic load and store unsigned byte is the only atomic transaction currently supported. It takes seven cycles and is described in the The SPARC Architecture Manual.

Figure 16 shows an atomic load and store unsigned byte.

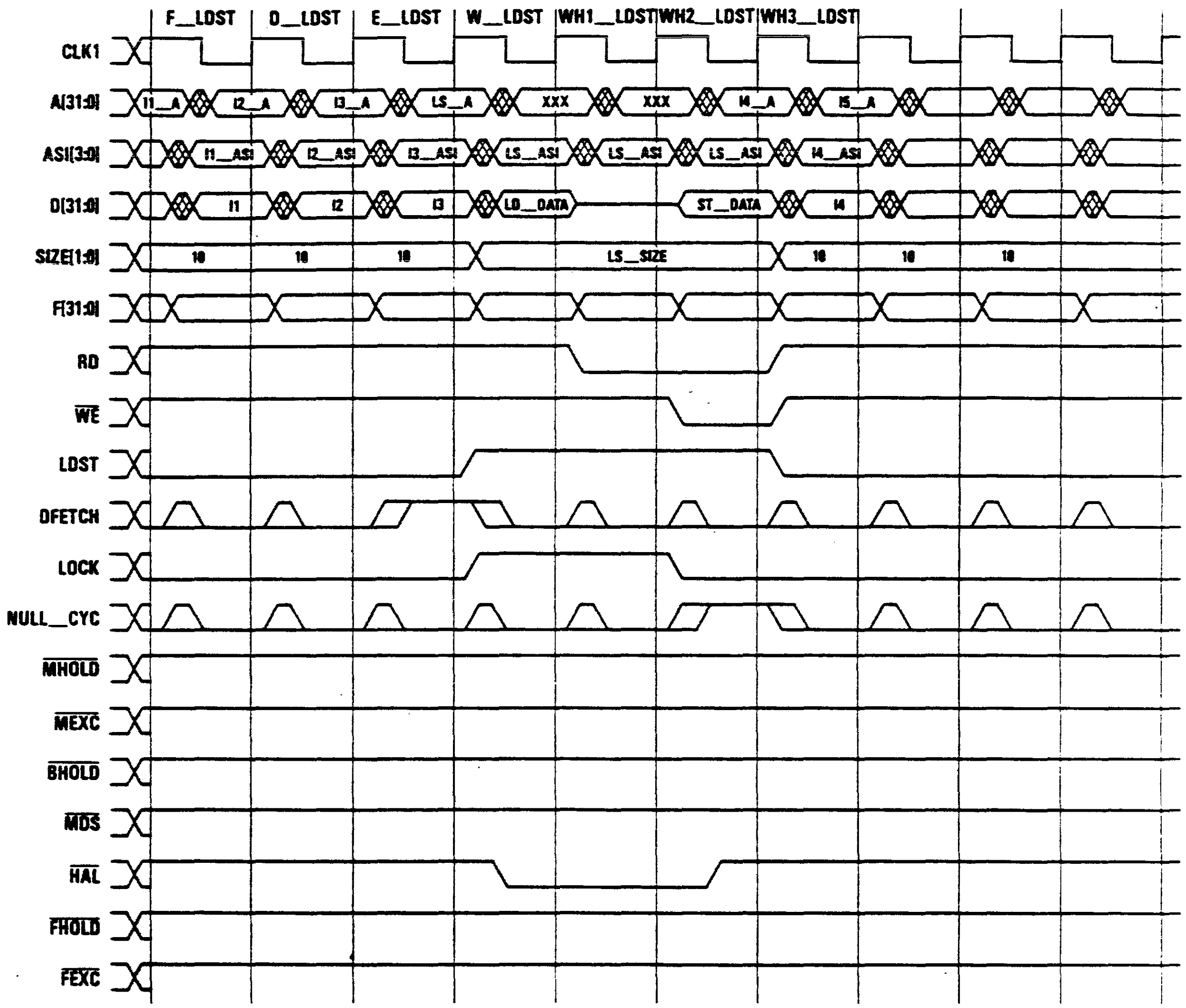


Figure 16. Atomic Load-Store Unsigned Byte Timing

Floating-Point Operations

The IU fetches and decodes FPops, then broadcasts them to the FPU controller over the floating-point bus (F[31:0]). It also provides control signals to inform the FPU controller when an FPop is decoded. During an FPop, the IU puts the instruction on the floating-point bus during the execute cycle and puts the instruction address on the floating-point bus during the write cycle.

The FPU controller stops the IU by asserting FHOLD if it detects a condition that requires it to delay executing the current floating-point instruction. This can happen under the following conditions:

- When a store FSR instruction starts execution and FPops are pending in the floating-point queue. In this case, the FPU controller detects the condition and asserts FHOLD. The store FSR instruction must wait until all pending FPops complete execution.

LSI LOGIC

Floating-Point Operations
(Continued)

- When FPop is issued and there is either a resource or an operand dependency between the present FPop and one or more of the previously fetched instructions.
- When a branch on floating-point condition (FBfcc) starts executing while the floating-point conditions are not ready. This occurs when one of the previously fetched instructions is a floating-point compare (FCMP) that the FPU has not yet completed.

Figure 17 shows the timing for a floating-point operation.

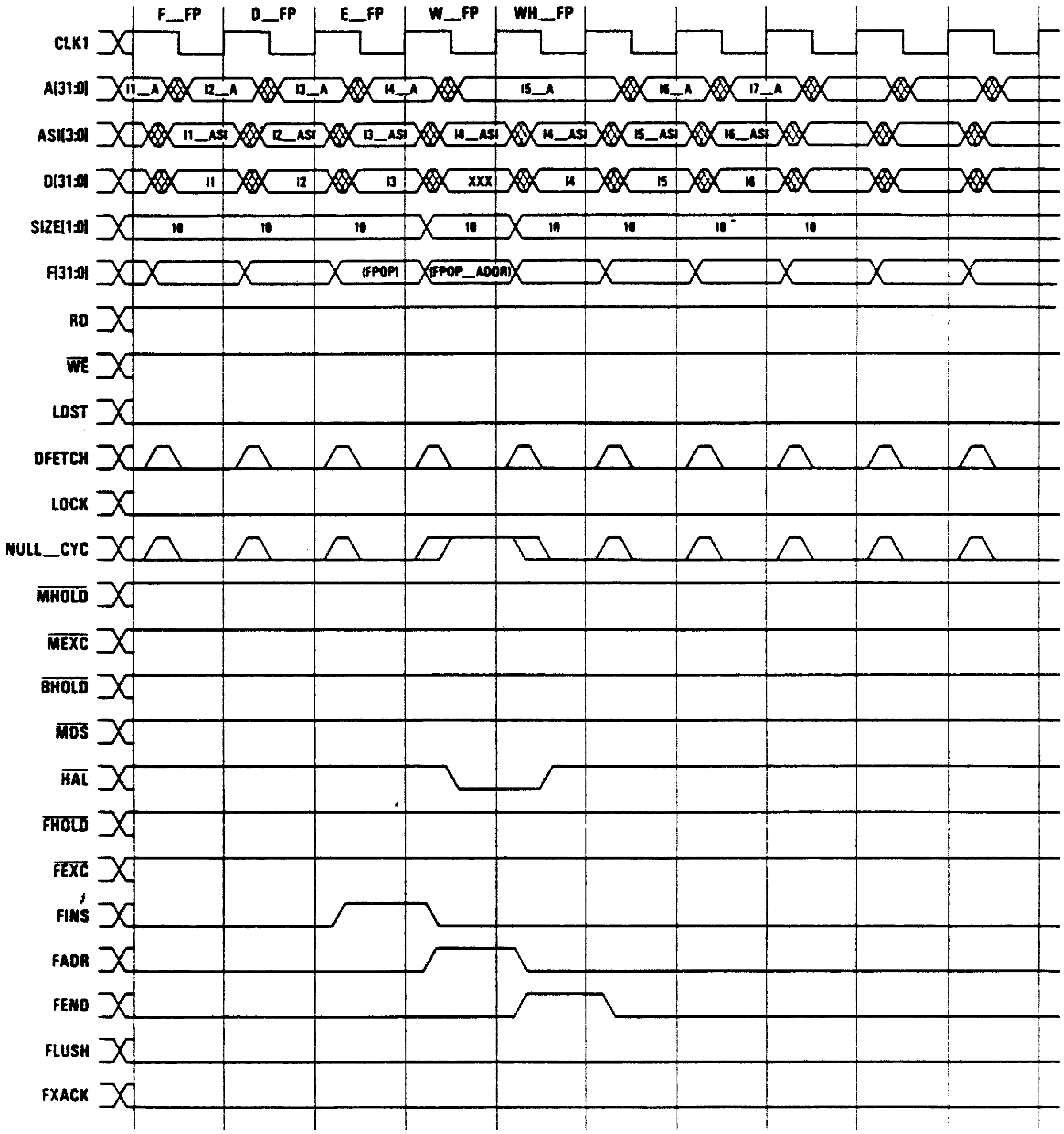


Figure 17. Floating-Point Operations Timing

Bus Arbitration

Because the L64801 chip set is a bus slave, bus arbitration must be performed externally using the BHOLD and LOCK pins. The L64801 IU asserts LOCK when it needs to retain the bus. External hardware should assert BHOLD when it needs to keep the L64801 from using the bus.

When BHOLD is asserted, it stops the processor's pipeline until it is disasserted. The signals DOE and AOE can be used to turn off the output drivers of the data bus, the address bus and the other control signals. This allows these to be driven by external hardware. Figure 18 shows the bus arbitration timing.

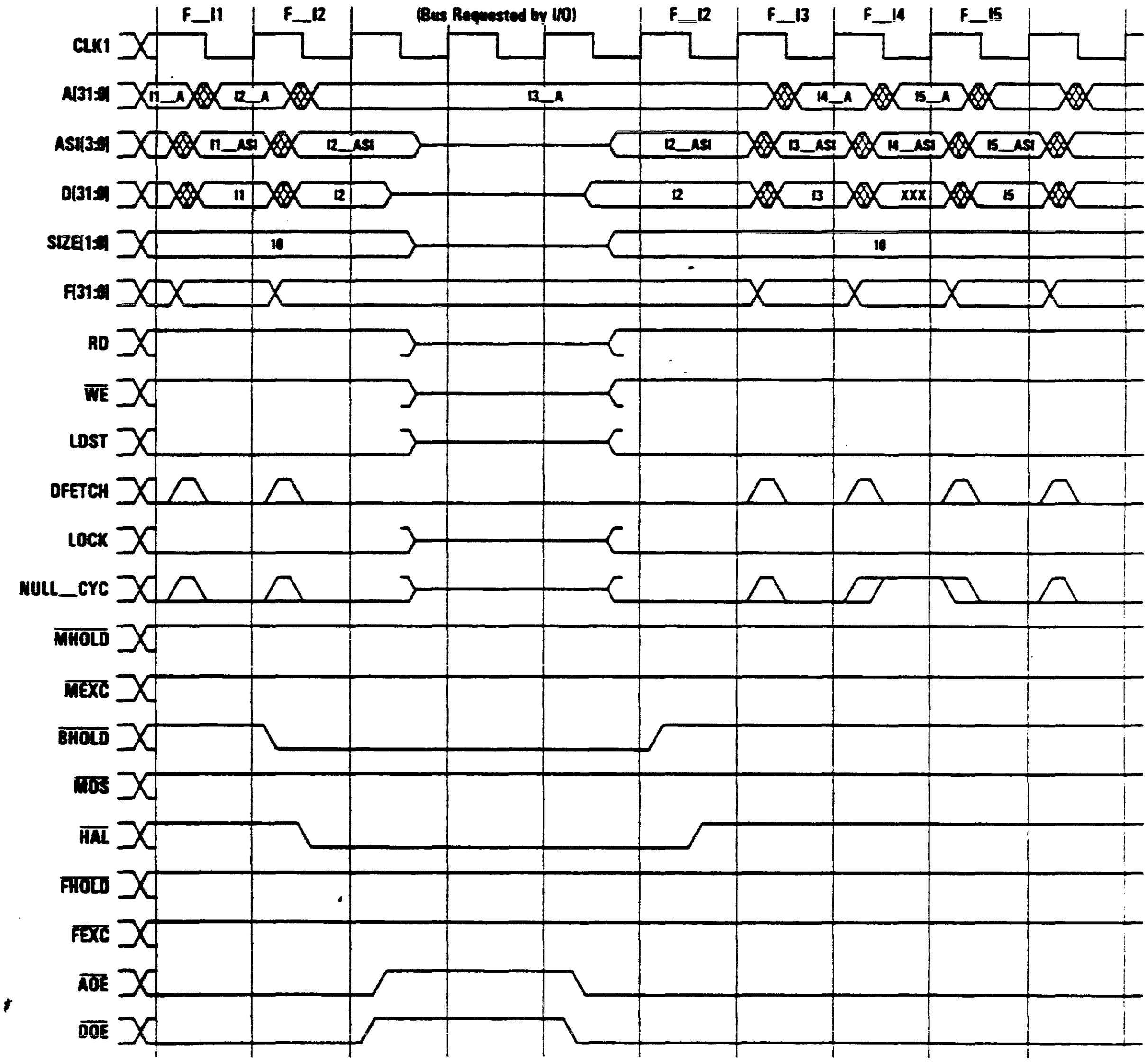
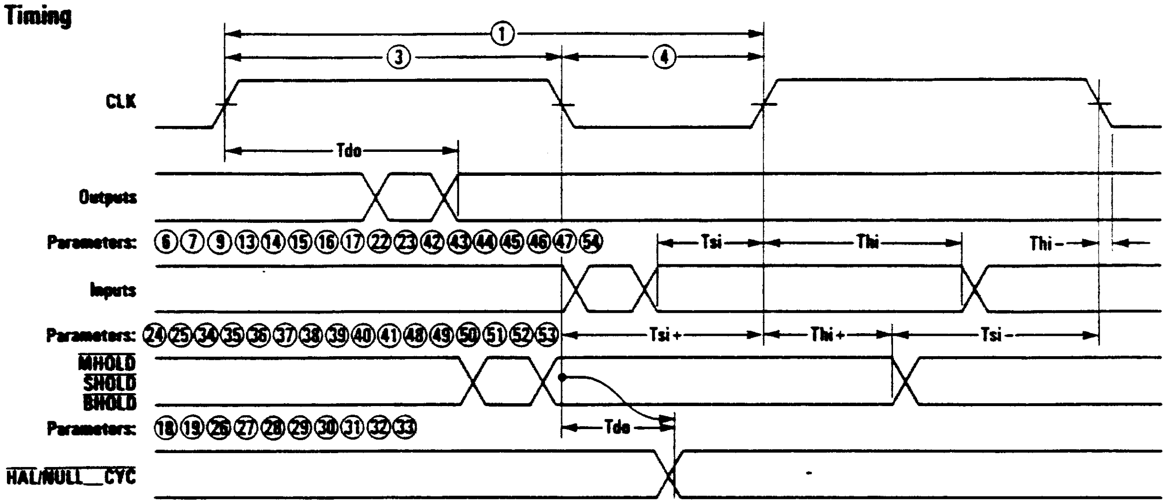


Figure 18. Bus Arbitration Timing

LSI LOGIC

L64801 AC Parameter Timing



Tdo = Output delay
Tsi = Input setup time
Thi = Input hold time
- = Falling edge of clock
+ = Rising edge of clock

AC Characteristics: VDD = 4.75 V to 5.25 V, TA = 0°C to 70°C, all output capacitances are 50 pF.

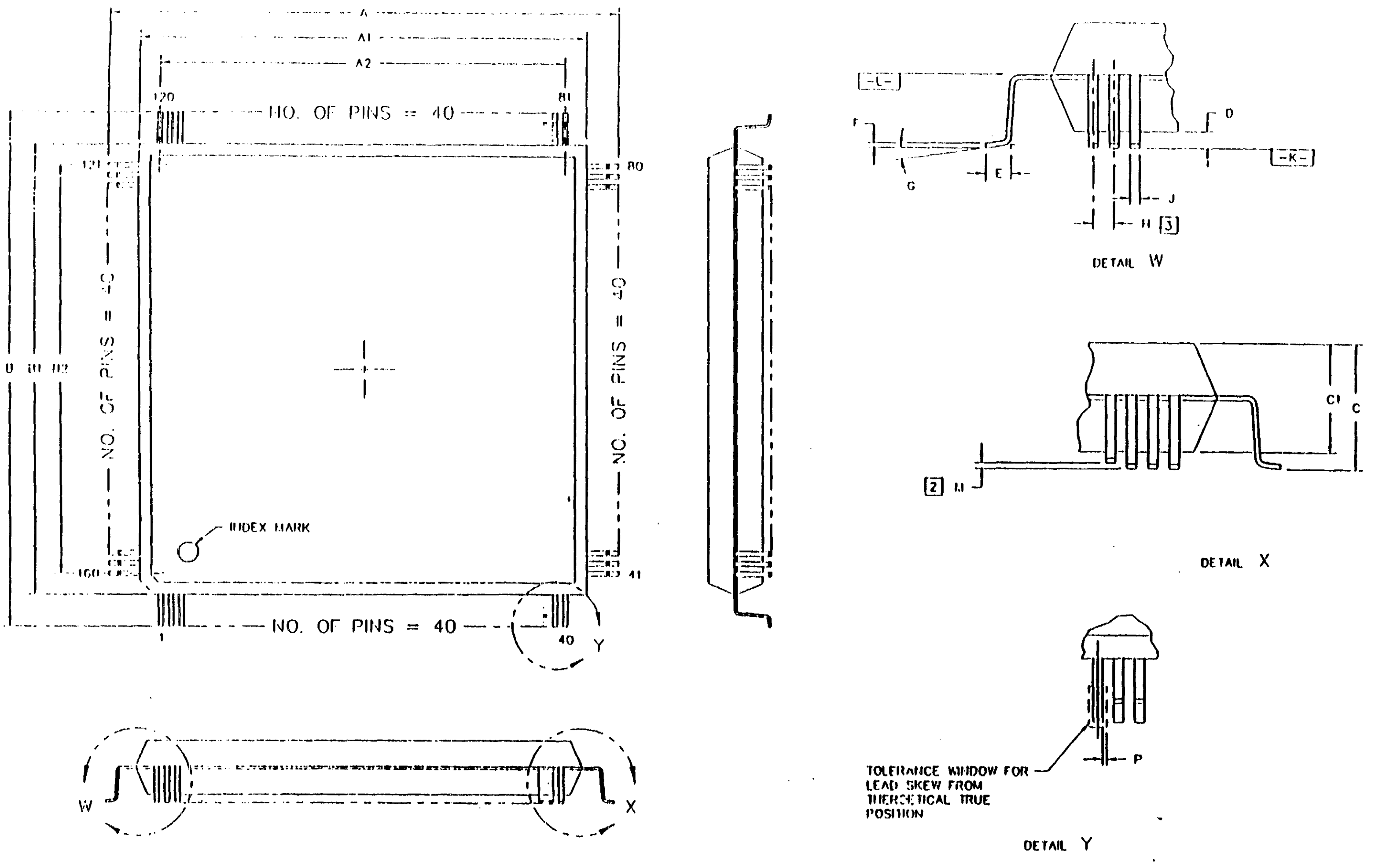
Number	Characteristic	20 MHz		25 MHz		Units	Notes
		Min	Max	Min	Max		
1	System Clock Cycle Time	50		40		ns	
2	System Clock Rise/Fall Times		3		3	ns	
3	System Clock High Duration	20		17		ns	
4	System Clock Low Duration	15		13		ns	
5	RESET Active Time	10		10		T	
6	Address Valid Delay from CLK Rising	5	44	4	37	ns	
7	ASI Valid Delay from CLK Rising	5	32	4	27	ns	
8	Read Data Setup before CLK Rising	5		4		ns	
9	Write Data Valid from CLK Rising	5	32	4	27	ns	
10	Write Data Turn Off from CLK	5		4		ns	
11	AOE, Enable/Disable	4	19	3	16	ns	
12	DOE, Enable/Disable	4	25	3	21	ns	
13	Size Valid Delay from CLK Rising	5	20	4	17	ns	
14	RD Valid Delay from CLK Rising	5	20	4	17	ns	
15	WE Valid Delay from CLK Rising	5	21	4	18	ns	
16	LDST Valid Delay from CLK Rising	5	20	4	17	ns	
17	NULL__CYC Valid Delay from CLK Rising	5	41	4	34	ns	
18	MHOLD (A/B/C) Valid to NULL__CYC	5	22	4	19	ns	
19	IH__NULL Valid to NULL__CYC	5	14	4	12	ns	
20	HAL Valid Delay from CLK Rising	5	36	4	30	ns	
21	MHOLD (A/B/C) Valid to HAL	4	20	3	17	ns	
22	LOCK Valid Delay from CLK Rising	5	21	3	18	ns	
23	DFETCH Valid Delay from CLK Rising	5	32	3	27	ns	

AC Characteristics (Continued): VDD = 4.75 V to 5.25 V, TA = 0°C to 70°C, all output capacitances are 50 pF.

Number	Characteristic	20 MHz		25 MHz		Units	Notes
		Min	Max	Min	Max		
24	$\overline{\text{MDS}}$ Setup before CLK Falling	27		23		ns	
25	$\overline{\text{MDS}}$ Hold after CLK Rising	0		0		ns	
26	$\overline{\text{MHOLD}}$ (A/B/C) Setup before CLK Rising	27		23		ns	
27	$\overline{\text{MHOLD}}$ (A/B/C) Hold after CLK Rising	0		0		ns	
28	$\overline{\text{MHOLD}}$ (A/B/C) Setup before CLK Falling	9		7		ns	
29	$\overline{\text{MHOLD}}$ (A/B/C) Hold after CLK Falling	0		0		ns	
30	$\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Setup before CLK Rising	27		23		ns	
31	$\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Hold after CLK Rising	0		0		ns	
32	$\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Setup before CLK Falling	9		8		ns	
33	$\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Hold after CLK Falling	0		0		ns	
34	FCC Setup before CLK Rising	5		4		ns	
35	FCC Hold after CLK Rising	0		0		ns	
36	FCCV Setup before CLK Rising	3		3		ns	
37	FCCV Hold after CLK Rising	0		0		ns	
38	FHOLD Setup before CLK Rising	2		2		ns	
39	FHOLD Hold after CLK Rising	1		1		ns	
40	$\overline{\text{FEXC}}$ Setup before CLK Rising	2		2		ns	
41	$\overline{\text{FEXC}}$ Hold after CLK Rising	2		2		ns	
42	F Valid Delay after CLK Rising	5	43	4	36	ns	
43	FINS Valid Delay after CLK Rising	5	30	4	23	ns	
44	FADR Valid Delay after CLK Rising	5	29	4	23	ns	
45	FEND Valid Delay after CLK Rising	5	29	4	23	ns	
46	FLUSH Valid Delay after CLK Rising	5	25	4	21	ns	
47	FXACK Valid Delay after CLK Rising	5	29	4	23	ns	
48	TC Setup before CLK Rising	12		9		ns	
49	TC Hold after CLK Rising	0		0		ns	
50	IRL Setup before CLK Rising	18		14		ns	
51	IRL Hold after CLK Rising	2		2		ns	
52	RESET Setup before CLK Rising	2		2		ns	
53	RESET Hold after CLK Rising	2		2		ns	
54	ERROR Valid Delay after CLK Rising	5	23	4	18	ns	
55	Address Drivers Off/On after $\overline{\text{AOE}}$	4	19	3	15	ns	
56	ASI Drivers Off/On after $\overline{\text{ASIOE}}$	4	16	3	13	ns	
57	$\overline{\text{WE}}$ Driver Off/On after ASIOE	4	16	3	13	ns	
58	RD Driver Off/On after $\overline{\text{ASIOE}}$	4	16	3	13	ns	
59	LDST Driver Off/On after $\overline{\text{ASIOE}}$	4	16	3	13	ns	
60	Data Bus Drivers Off/On after $\overline{\text{DOE}}$	4	25	3	19	ns	
61	Data Bus Drivers Off/On after XSM	4	31	3	24	ns	

L64801 160 QFP Signal Definition

Pin No.	Signal	Pin No.	Signal	Pin No.	Signal	Pin No.	Signal
1	A.4	41	A.0	81	ASI.7	121	F.22
2	F.14	42	IRL.1	82	FEXC_	122	F.18
3	A.15	43	FADR	83	ASIOE_	123	F.26
4	A.10	44	IRL.2	84	D.27	124	D.14
5	A.2	45	ASI.1	85	D.15	125	D.10
6	A.9	46	ASI.0	86	VSS	126	F.10
7	A.5	47	RD	87	D.7	127	F.2
8	A.13	48	FLUSH	88	D.3	128	F.6
9	VSS	49	VSS	89	SDO	129	F.11
10	A.26	50	MHOLDA_	90	D.11	130	F.27
11	XSM	51	SIZE.0	91	D.30	131	F.23
12	A.27	52	WEN	92	D.26	132	F.19
13	A.24	53	IRL.3	93	ERROR_	133	F.15
14	A.30	54	VDD	94	D.23	134	FCC.1
15	VDD	55	ASI.2	95	D.1	135	VDD
16	A.29	56	HAL_	96	VDD	136	F.7
17	A..25	57	LOCK	97	D.6	137	F.3
18	FCCV	58	D.2	98	D.22	138	F.16
19	A.20	59	VSS	99	D.29	139	F.28
20	A.28	60	SIZE.1	100	D.25	140	CLK
21	VSS	61	LDST	101	VSS	141	VSS
22	A.18	62	PTREEO	102	D.21	142	F.20
23	A.21	63	MEXC_	103	D.17	143	F.0
24	A.19	64	ASI.3	104	D.19	144	F.24
25	A.31	65	VDD	105	FP_	145	FCC.0
26	VDD	66	ASI.4	106	D.13	146	F.12
27	A.22	67	ASI.5	107	VDD	147	VDD
28	A.23	68	ASI.6	108	D.9	148	F.8
29	A.17	69	MDS_	109	D.5	149	F.4
30	A.6	70	VSS	110	D.18	150	F.29
31	VSS	71	RESET_	111	D.28	151	F.21
32	A.7	72	FHOLD_	112	D.24	152	F.31
33	FXACK	73	FINS	113	D.20	153	F.17
34	A.11	74	AOE_	114	D.16	154	F.25
35	A.3	75	D.0	115	D.31	155	F.13
36	A.14	76	DFETCH	116	D.8	156	VSS
37	A.16	77	DOE_	117	D.4	157	F.9
38	VSS	78	IRL.0	118	VSS	158	F.5
39	A.8	79	NULL_CYC	119	D.12	159	F.1
40	A.12	80	FEND	120	F.30	160	A.1



NOTES:

SEE SHEET 2 OF 2

DRAFT	CHECK	APPR	DESCRIPTION		REV
			INITIAL RELEASE		0

UNLESS OTHERWISE SPECIFIED
DIMENSIONAL TOLERANCES
.X +/- .1
.XX +/- .01
.XXX +/- .005

D. VILLEGAS 07-01-88

LSI LOGIC CORPORATION
48580 KATO ROAD FREMONT CA 94539

TITLE
160 LD QUAD FLAT PACKAGE

CODE PF
DWG NO. J201-000162-00 REV. 0
NOT TO SCALE SHEET 1 of 2

WEITEK

ABACUS [redacted] 3172 FLOATING-POINT COPROCESSOR FOR SPARC

PRELIMINARY DATA
August 1989

RECEIVED
NOV 29 1989
NE/THW

WEITEK
TIME: 2 Pm DATE: 8/1
PRELIMINARY
TRANSMITTED BY:

Nov. 29, 1989
Tony F

The Abacus 3172 is a single-chip floating-point coprocessor for the Fujitsu S-20[redacted] implementation of the SPARC architecture. It incorporates a floating-point datapath and a floating-point controller. The Abacus 3172 provides direct interface to the integer unit and memory. It is available in speed grades of 20 [redacted] MHz.

Related product: The Abacus 3171 single-chip floating-point coprocessor for Cypress 7C601 implementation of SPARC architecture.

Contents

ABACUS [redacted] 3172 FLOATING-POINT COPROCESSOR FOR SPARC

PRELIMINARY DATA
August 1989

Features

SINGLE-CHIP 64-BIT FLOATING-POINT DATA PATH AND CONTROLLER

64-bit multiplier and divide/square root unit
64-bit ALU
16×64 or 32×32 three-port register file with an independent load/store port

DIRECT INTERFACE TO FUJITSU S-20[redacted] AND LSI LOGIC L64801 SPARC PROCESSORS

DIRECT INTERFACE TO MEMORY

20 [redacted] OPERATION

FULL COMPLIANCE WITH ANSI/IEEE-754 STANDARD FOR BINARY FLOATING-POINT ARITHMETIC

143-PIN PGA PACKAGE

LOW-POWER CMOS

Description

The Abacus 3172 is a high-performance, single-chip floating-point coprocessor for the Fujitsu S-20 and [redacted]/LSI Logic L64801 implementation of the SPARC architecture. It incorporates a floating-point datapath and a floating-point controller. The Abacus 3172 provides direct interface to the integer unit and memory. It is available in speed grades of 20 and [redacted]z.

The floating-point datapath circuitry contains a 64-bit multiplier, a 64-bit ALU, a 64-bit divide/square root unit, and a 16-word by 64-bit (or 32-word by 32-bit) three-port register file.

The floating-point controller circuitry handles IEEE exceptions and the interface between the floating-point datapath and the integer unit, as well as between the datapath and memory.

CONFORMANCE TO SPARC ARCHITECTURE

The Abacus 3172 processes instructions within the specifications of the SPARC architecture as described in the *SPARC Architecture Manual*, by Sun Microsystems.

DATA TYPES

The SPARC architecture specifies four data types that can be used in conjunction with the floating-point unit (FPU):

- 32-bit two's complement integer
- single-precision floating-point
- double-precision floating-point
- extended-precision floating-point

The Abacus 3172 supports all of these data types except extended-precision. Any operation specifying extended-precision data types will be trapped to system software, with unimplemented instruction trap type.

INSTRUCTION PROCESSING

When the integer unit (IU) decodes a floating-point operate (FPop) or a floating-point load/store (FPLd/St) instruction, it sends the instruction to the FPU over the F bus during the Execute stage of the IU pipeline.

During the Write stage of the IU pipeline, the IU sends the FPop address over the F bus to the FPU so that it will be available for floating-point exception handling. Also during this cycle, the FPU will assert FHOLD– if a dependency exists. FHOLD– will remain asserted until the dependency has been resolved.

CONFORMANCE TO ANSI/IEEE-754 SPECIFICATION FOR BINARY FLOATING-POINT ARITHMETIC

The Abacus 3172 conforms to the requirements of the ANSI/IEEE-754 specification.

FLOATING-POINT STATE REGISTER (FSR)

The *SPARC Architecture Manual* contains detailed information about the Floating-Point State Register (FSR). Bits 19:17 of the FSR comprise the version field. The version field specifies the particular floating-point unit/controller implementation. In the case of the [redacted], FSR (19:17) = 011_2. 3172

Description, continued

IMPLEMENTED INSTRUCTIONS

Operations involving NaNs and denormalized numbers
require system software assistance or intervention.
They terminate with trap type unfinished.

Mnemonic(s)		Operation
ldf		Load floating-point register
lddf		Load double floating-point register
ldfsr		Load floating-point status register
stf		Store floating-point register
stdf		Store double floating-point register
stfsr		Store floating-point status register
stdfq		Store double floating-point queue
fitos	fitod	convert integer to floating-point (rounded as per *fsr.rd*) (single/double)
fstoi	fdtoi	convert floating-point to integer (rounded toward zero) (single/double)
fstod	fdtos	convert single to double/double to single floating-point
fmovs		register to register move
fnegs		register to register move with sign bit inverted
fabss		register to register move with sign bit set to 0
fsqrts	fsqrtd	floating-point square root (single/double)
fadds	faddd	floating-point add (single/double)
fsubs	fsubd	floating-point subtract (single/double)
fmuls	fmuld	floating-point multiply (single/double)
fdivs	fdivd	floating-point divide (single/double)
fcmps	fcmpd	floating-point compare (single/double)
fcmpes	fcmped	floating-point compare and exception if unordered (single/double)

Figure 1. Implemented instructions

UNIMPLEMENTED INSTRUCTIONS

Mnemonic(s)		Operation
fitox		convert integer to extended floating-point (rounded as per *fsr.rd*)
fxtoi		convert extended floating-point to integer (rounded toward zero)
fxtos	fxtod	convert extended floating-point to single/double floating-point
fstox	fdtox	convert single/double floating-point to extended floating-point
fsqrtx		floating-point square root (extended-precision)
faddx		floating-point add (extended-precision)
fsubx		floating-point subtract (extended-precision)
fmulx		floating-point multiply (extended-precision)
fdivx		floating-point divide (extended-precision)
fcmpx		floating-point compare (extended-precision)
fcmpex		floating-point compare and exception if unordered (extended-precision
fsmuld		single product to double
fdmulx		double product to extended

Figure 2. Unimplemented instructions

Description, continued

DEVICE DESCRIPTION

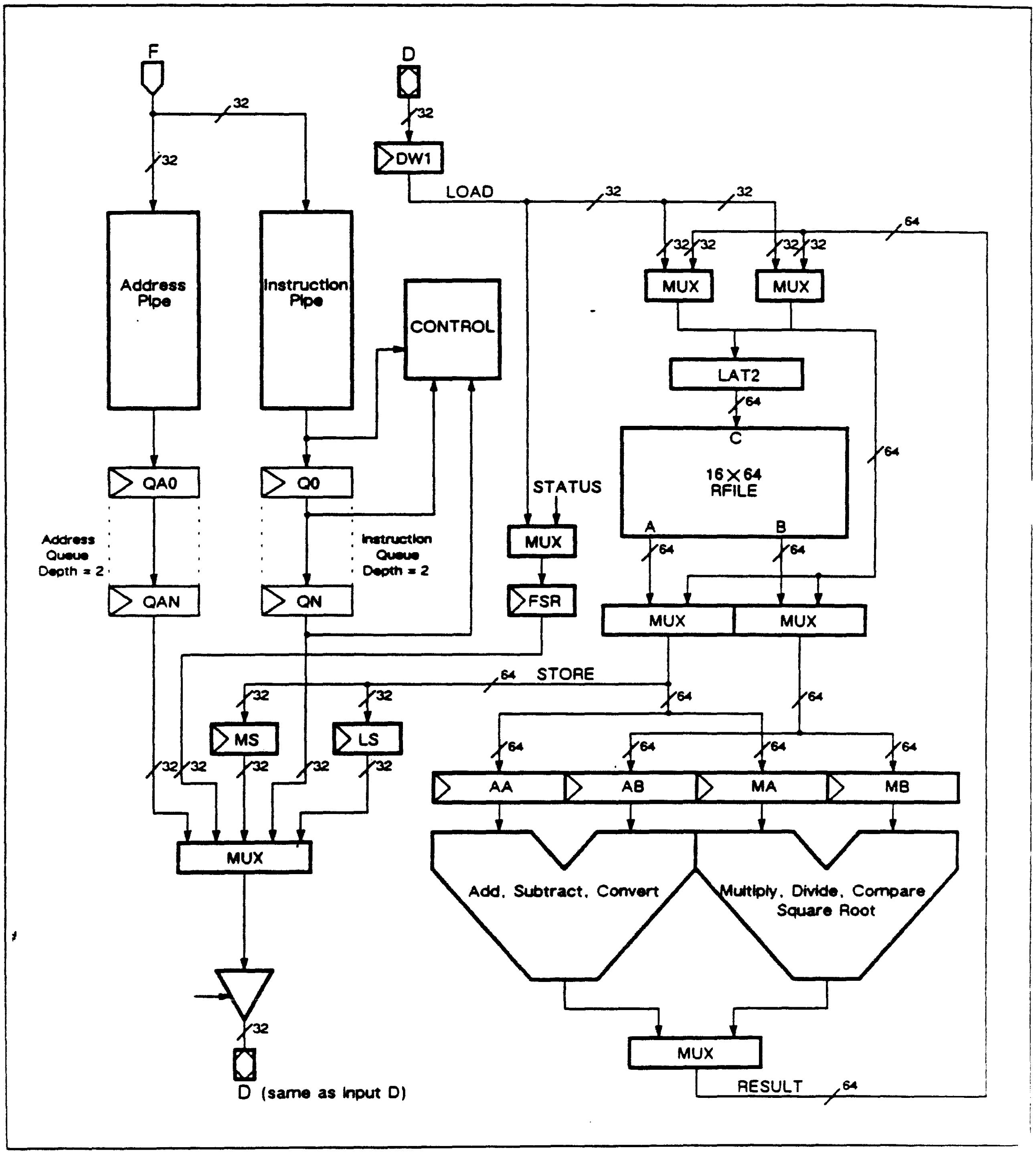


Figure 3. Conceptual block diagram

Description, continued

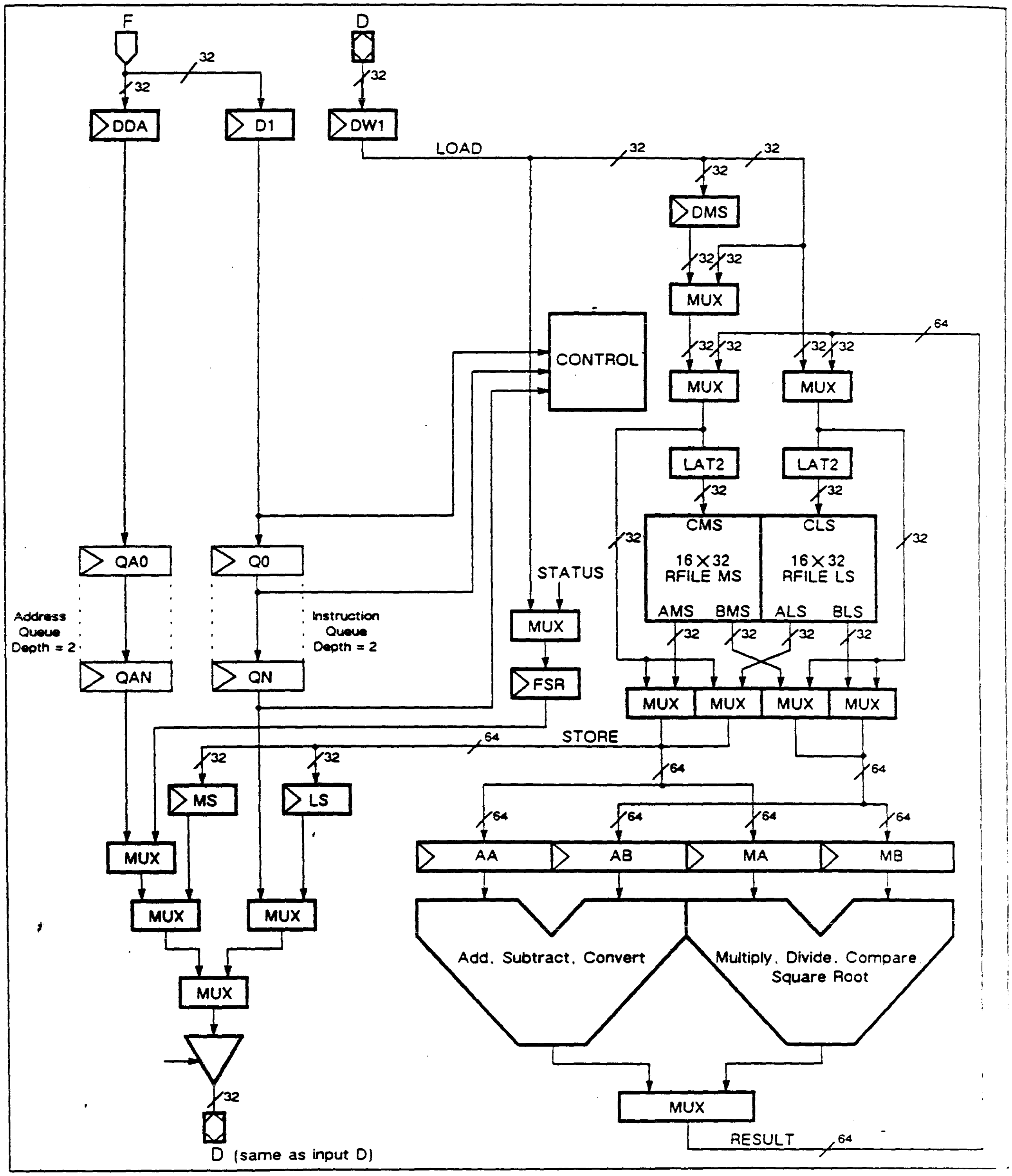


Figure 4. Simplified block diagram

Description, continued

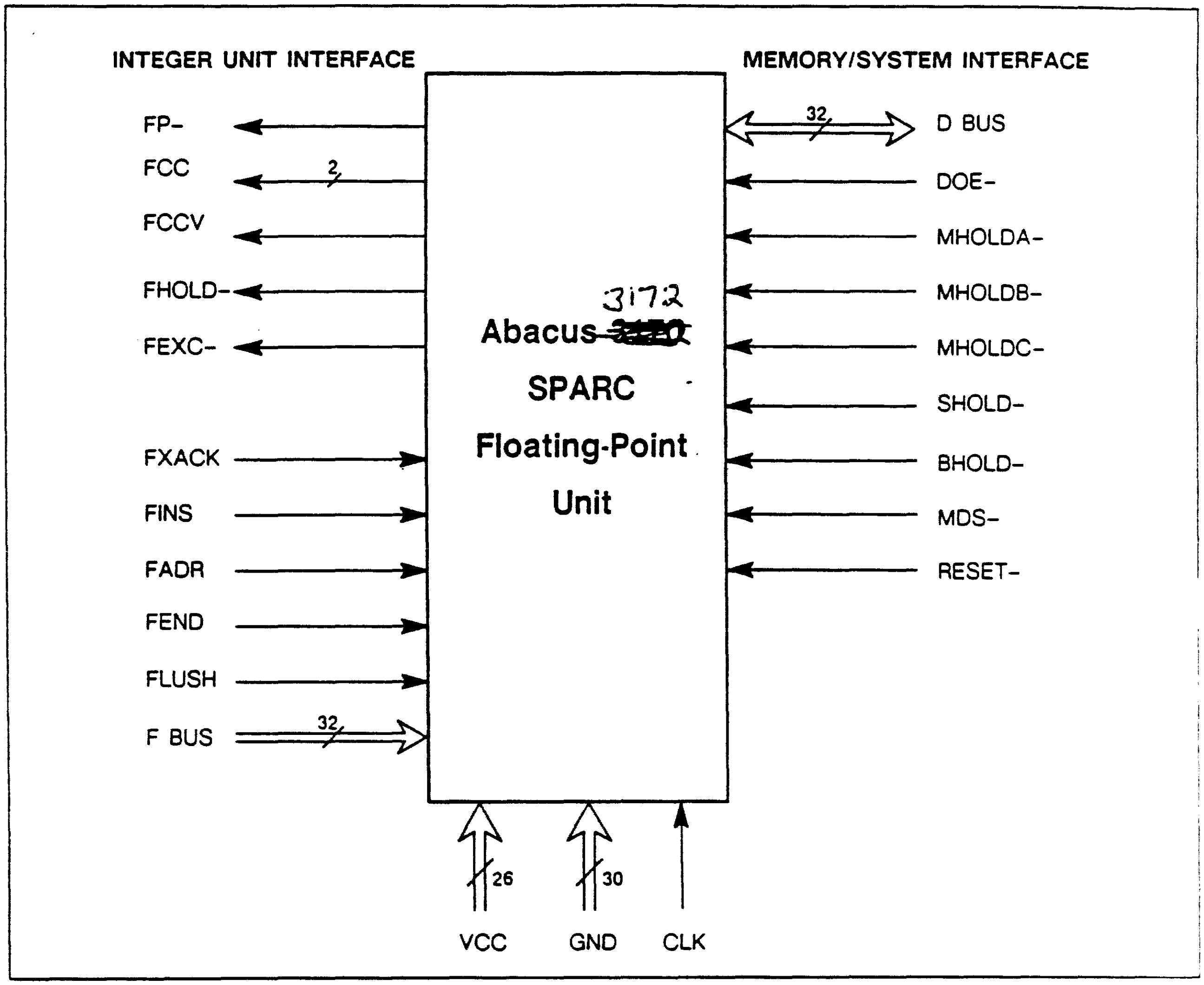


Figure 5. Abacus [redacted]0 signals
3172

Description, continued

SIGNAL DESCRIPTION

Signals marked with a minus sign (-) after their names are active low; all other signals are active high.

INTEGER UNIT INTERFACE SIGNALS

FP- OUTPUT

Floating-point unit present. The FP- signal indicates whether a floating-point unit FPU is present in the system. In the absence of an FPU the FP- signal is pulled up to VCC by a resistor. When an FPU is present the FP- signal is grounded.

FCC OUTPUT

Floating-point condition code. The $FCC_{1..0}$ bits represent the current condition code of the FPU. They are valid only if FCCV is asserted.

FBfcc instructions use these bits during the execute cycle if they are valid, and delay the execute cycle if they are not valid. The condition codes are shown below.

FCC (1)	FCC (0)	CONDITION
0	0	Equal
0	1	Op1 < Op2
1	0	Op1 > Op2
1	1	Unordered

Figure 6.

FCCV OUTPUT

Floating-point condition code valid. The FPU asserts the FCCV signal when FCC bits represent a valid condition. The FPU deasserts FCCV if pending floating-point compare instructions exist in the floating-point queue. FCCV is reasserted when the compare instruction is completed and FCC bits are valid.

FHOLD- OUTPUT

Floating-point hold. The FHOLD- signal is asserted by the FPU if it cannot continue execution due to a resource or operand dependency. The FPU checks for all dependencies in the write stage and, if necessary, asserts FHOLD- in the same cycle. The FHOLD- signal is used by the IU to freeze its pipeline in the next cycle. The FPU must eventually deassert FHOLD- to release the IU's pipeline.

FEXC- OUTPUT

Floating-point exception. The FEXC- signal is asserted if a floating-point exception has occured. It remains asserted until the IU acknowledges that it has taken a trap by asserting FXACK. Floating-point exceptions are taken only during the execution of a floating-point instruction, FBfcc instruction, or floating-point load or store instructions. When the FPU receives an asserted level of the FXACK signal it deasserts FEXC-.

FXACK INPUT

Floating-point exception acknowledge. The FXACK signal is asserted by the IU to acknowledge to the FPU that the current FEXC- trap is taken.

FINS INPUT

Floating-point instruction. The IU asserts FINS during the cycle in which $F_{31..0}$ carries a valid floating-point instruction. The FPU uses this signal to latch the instruction into its instruction register.

FADR INPUT

Floating-point address. The IU asserts FADR during the cycle in which $F_{31..0}$ carries a valid floating-point instruction address. The FPU uses this signal to latch the instruction into its address register.

FEND INPUT

End floating-point instruction. The IU asserts FEND during the last cycle of a floating-point instruction in the IU pipeline. The FPU uses FEND to synchronize the instruction/address in it execution pipeline with the IU pipeline.

FLUSH INPUT

Floating-point instruction flush. The FLUSH signal is asserted by the IU to signal to the FPU to flush the instructions in its instruction registers. This may happen when a trap is taken by the IU. The IU will restart the flushed instructions after returning from the trap. FLUSH has no effect on instructions in the floating-point queue.

F BUS INPUT

Floating-point bus. $F_{31..0}$ is a dedicated 32-bit bus that receives floating-point instructions and addresses from the IU. Each floating-point instruction must use this bus for two cycles. The first cycle carries the instruction and the second its address.

Description, continued

SYSTEM/MEMORY INTERFACE SIGNALS

D BUS INPUT/OUTPUT

Data bus. The $D_{31..0}$ bus is driven by the FPU only during the execution of floating-point store instructions. The alignment for load and store instructions is done in the FPU. A double word is aligned on an 8-byte boundary, a word is aligned on a 4-byte boundary.

DOE– INPUT

Data output enable. The DOE– signal is connected directly to the data output drivers and must be asserted during normal operation. Deassertion of this signal tri-states all output drivers on the data bus. This signal should be deasserted only when the bus is granted to another bus master, i.e., when either BHOLD–, MHOLDA–, or MHOLDB–, MHOLDC– or SHOLD– is asserted.

MHOLDA–, MHOLDB–, MHOLDC–, SHOLD– INPUTS

Memory hold. Asserting either MHOLDA–, MHOLDB–, MHOLDC–, or SHOLD– freezes the FPU pipeline.

BHOLD– INPUT

Bus hold. The BHOLD– signal is asserted by the system's I/O controller when an external bus master requests the data bus. Assertion of this signal will freeze the FPU pipeline.

MDS– INPUT

Memory data strobe. The MDS– signal is used to load data into the FPU when the internal FPU clock is stopped while on hold.

RESET– INPUT

Reset. Asserting the RESET– signal resets the pipeline and sets the writable fields of the floating-point status register (FSR) to zero. The RESET– signal must remain asserted for a minimum of eight cycles. After a reset, the IU will start fetching from address 0.

CLK INPUT

Clock. CLK is used for clocking the FPU. It is high during the first half of the processor cycle and low during the second half. The rising edge of CLK defines the beginning of each pipeline stage in the FPU.

VCC

Power supply. All VCC pins must be connected to 5.0 volt power supply.

GND

System ground. All GND pins must be connected to system ground.

NC

No connection. All no-connect pins must remain unconnected.

SYSTEM CONSIDERATIONS

INSTRUCTION CYCLE COUNTS

The 3170 has the following datapath instruction cycle counts. In order to arrive at register-to-register cycle counts, one cycle must be added to each number below.

Mnemonic(s)	Operation	Clock Cycles
fmovs	move	4
fnegs	negate	4
fabss	absolute value	4
fadds, fsubs	add/subtract single	8
faddd, fsubd	add/subtract double	8
fmuls	multiply single	8
fmuld	multiply double	14
fcmps	compare single	4
fcmpd	compare double	4
fcmpes	compare single and exception if unordered	4
fcmped	compare double and exception if unordered	4
fitos	convert integer to single	16
fitod	convert integer to double	8
fstod	convert single to double	8
fdtos	convert double to single	8
fdivs	divide single	40
fdivd	divide double	65
fsqrts	square root single	62
fsqrtd	square root double	120

Figure 7. Implemented instructions

LINPACK BENCHMARK ESTIMATE

The code shown below represents the inner loop of the SAXPY subroutine of the LINPACK benchmark. This loop requires 60 cycles on the Abacus 3170. At 25 MHz, this translates into a peak performance of 3.33 MFLOPS.

```
loop_top:
        ldd      [dx+0],dx0
        fmuld    dx0,da,dx0
        ldd      [dy+0],dy0
        ldd      [dx+8],dx1
        addcc    n,-4,n
        faddd    dx0,dy0,dy0
        fmuld    dx1,da,dx1
        ldd      [dy+8],dy1
        ldd      [dy+16],dx2
        add      dx,32,dx
        faddd    dx1,dy1,dy1
        fmuld    dx2,da,dx2
        ldd      [dy+16],dy2
        ldd      [dx-8],dx3
        add      dy,32,dy
        faddd    dx2,dy2,dy2
        fmuld    dx3,da,dx3
        ldd      [dy-8],dy3
        std      dy0,[dy-32]
        std      dy1,[dy-24]
        faddd    dx3,dy3,dy3
        std      dy2,[dy-16]
        bg       loop_top
        std      dy3,[dy-8]
```

Figure 8. LINPACK benchmark code

System Considerations

INTERFACE TO IU AND MEMORY

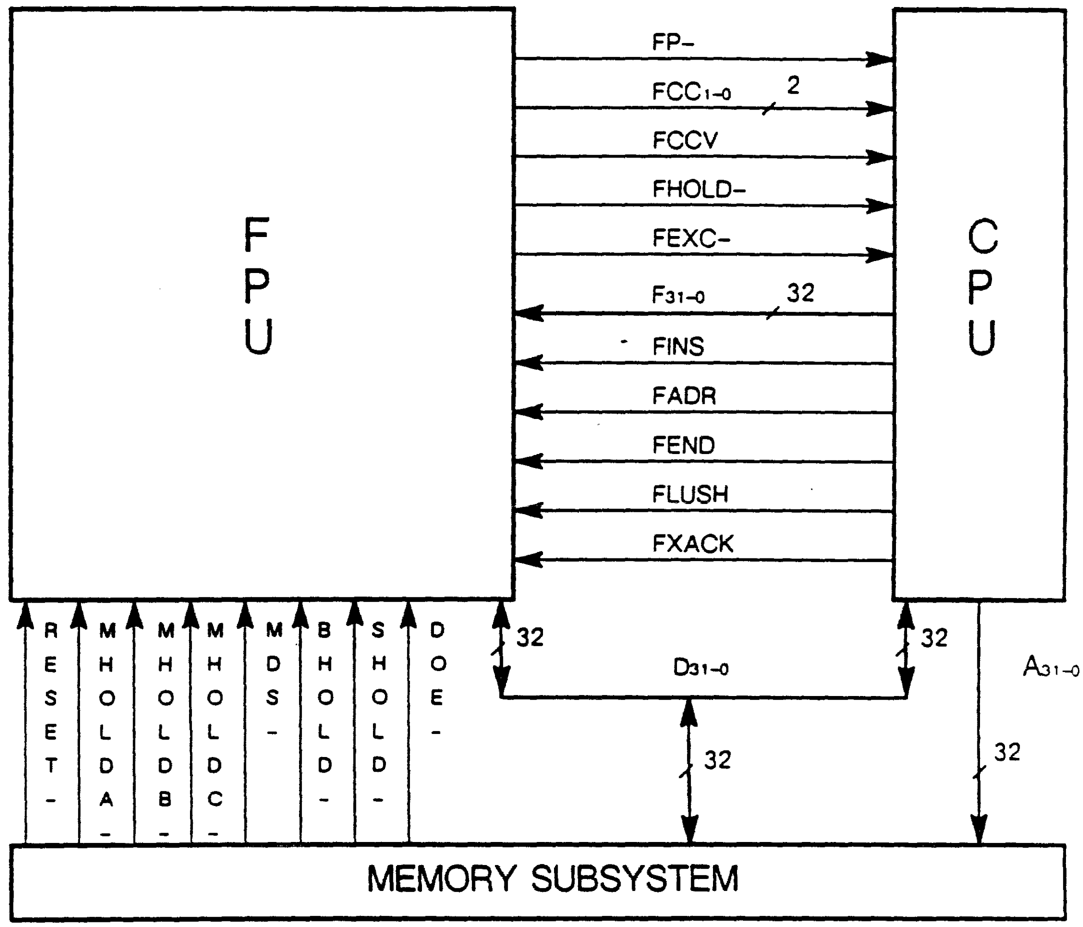


Figure 9. Interface to integer unit and memory

System Considerations, continued

INSTRUCTION OPERATION

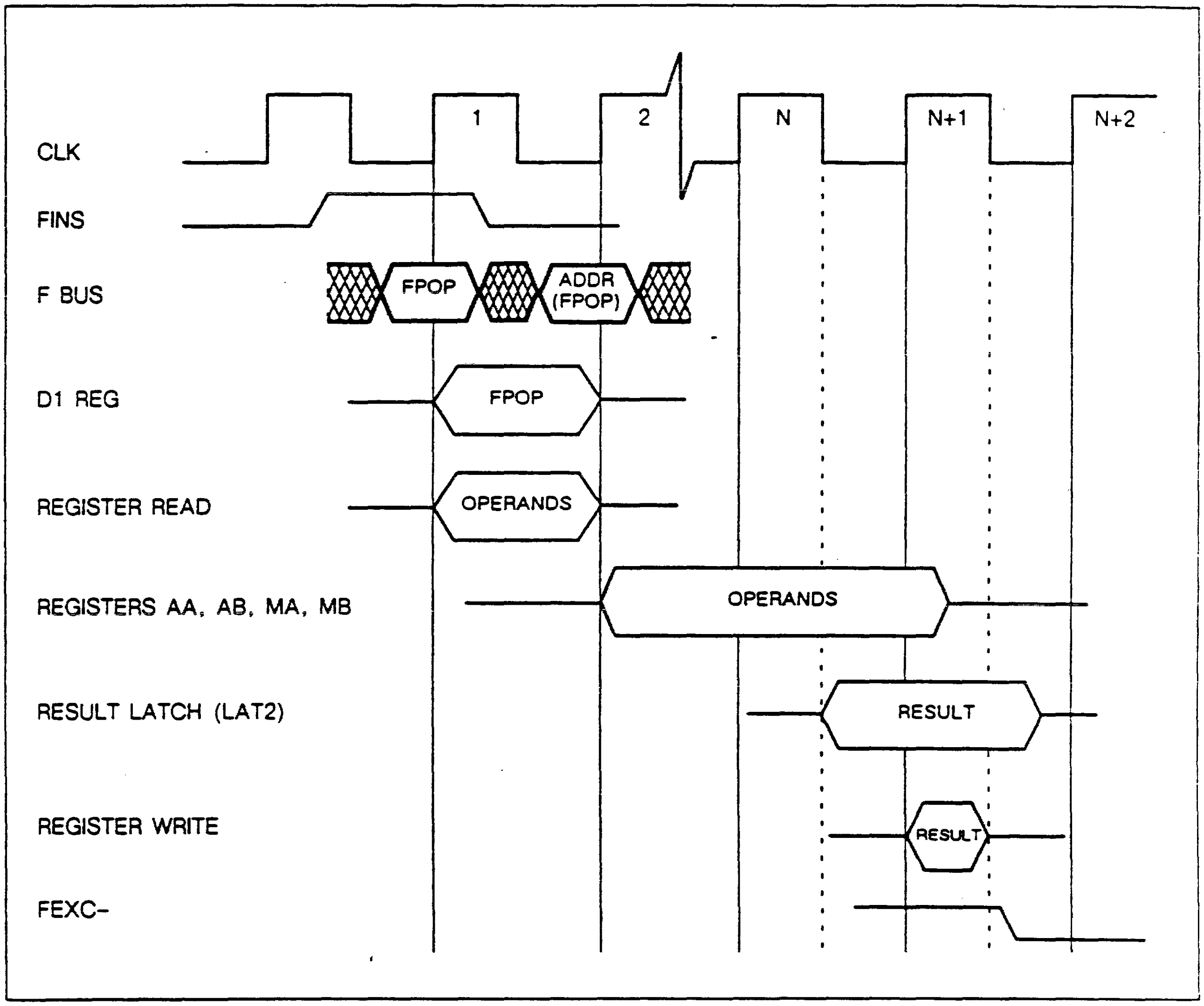


Figure 10. Instruction operation

Specifications

ABSOLUTE MAXIMUM RATINGS

Supply voltage	-0.5 to 7.0 V
Input voltage	-0.5V to VCC
Output voltage	-0.5V to VCC
Operating temperature range (T_{CASE})	0° to 85° C
Storage temperature range	-65° C to 150° C
Lead temperature (10 seconds)	300° C
Junction temperature	155° C

Figure 11.

OPERATING CONDITIONS

PARAMETER	MIN	MAX	UNIT
V_{CC} Supply voltage	4.75	5.25	V
I_{OH} High-level output current		-1.0	mA
I_{OL} Low-level output current		4.0	mA
T_{CASE} Operating case temperature	0	85	°C

Figure 12.

DC SPECIFICATIONS

PARAMETER	TEST CONDITIONS	MIN	MAX	UNIT
V_{IH} High-level input voltage	V_{CC} = MIN	2.1		V
V_{IL} Low-level input voltage	V_{CC} = MIN		0.8	V
V_{IHC} High-level input voltage	V_{CC} = MIN	2.4		V
V_{ILC} Low-level input voltage	V_{CC} = MIN		0.8	V
V_{OH} High-level output voltage	V_{CC} = MIN, I_{OH} = MAX	2.4		V
V_{OL} Low-level output voltage	V_{CC} = MIN, I_{OL} = MAX		0.4	V
I_{LI} Input leakage current	V_{CC} = MAX, V_{IN} = 0 to V_{CC}		±10	µA
I_{LO} Output leakage current (output disabled)	V_{CC} = MAX, V_{IN} = 0 or V_{CC}		±10	µA
C_{IN} Input capacitance*	V_{CC} = MAX, V_{IN} = 0 to Vcc		15	pF
C_{OUT} Output capacitance*	V_{CC} = MAX, V_{OUT} = 0 to Vcc		20	pF
C_{CLK} Clock Input capacitance*	Vcc = MAX, V_{IN} = 0 to Vcc		25	pF
C_{DOE-} DOE- Input capacitance*	Vcc = MAX, V_{IN} = 0 to Vcc		30	pF
I_{CC} Supply current	V_{CC} = MAX, T_{CY} = MIN; TTL inputs			mA
* Guaranteed, but not tested				

Figure 13. DC specifications

Specifications, continued

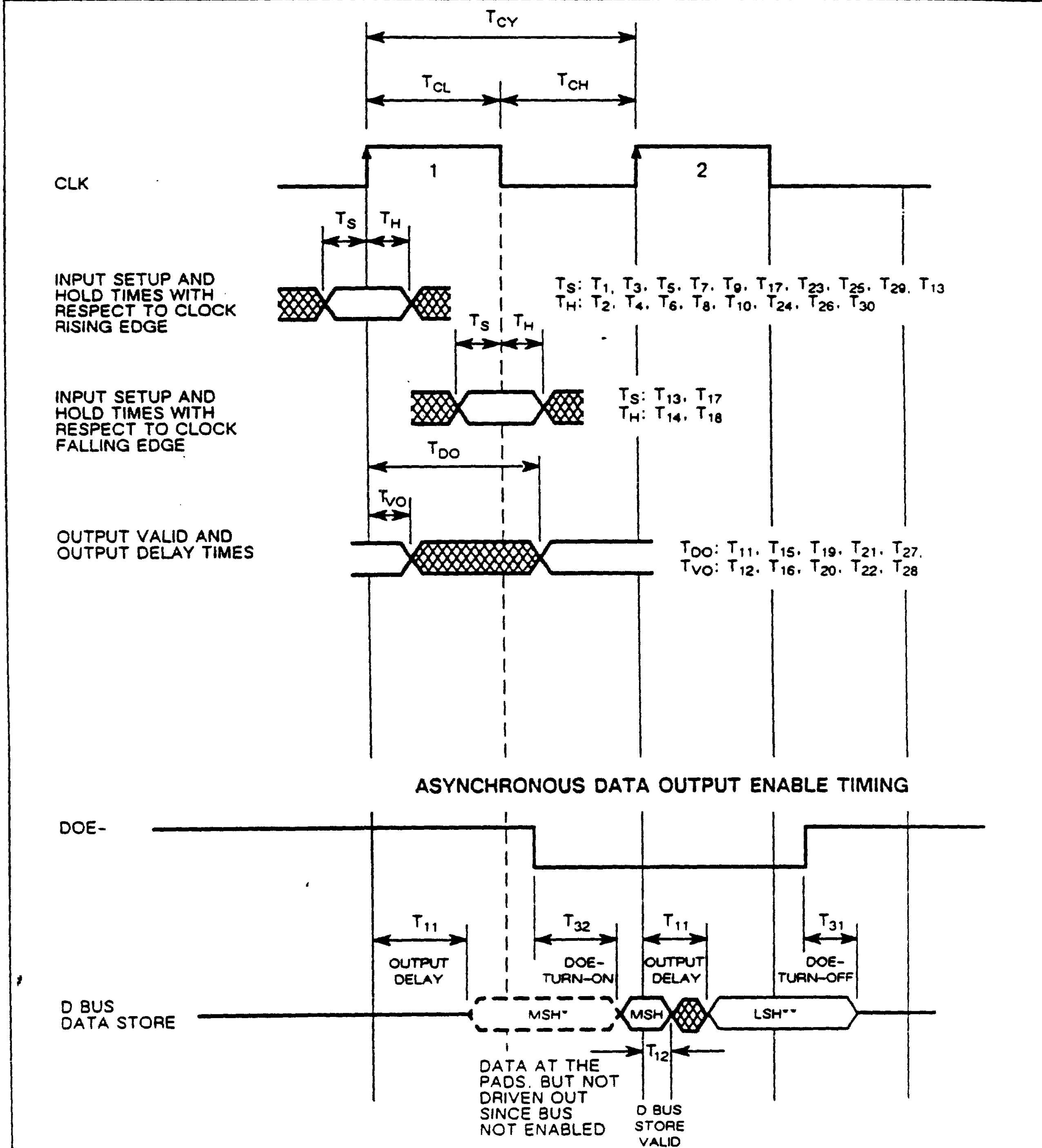
AC SPECIFICATIONS AND TIMING DIAGRAMS

SYMBOL	DESCRIPTION	Min/Max	Reference	20 MHz	25 MHz
TCY	Clock Cycle Time	MIN		50	40
TCH	Clock High time	MIN		15	12
TCL	Clock Low Time	MIN		15	12
TR	CLK Rise time	MIN		3	2
TF	CLK Fall time	MIN		3	2
T1	FINS Setup Time	MIN	CLK+	16	12
T2	FINS Hold Time	MIN	CLK+	4	3
T3	F bus (Abus) Instruction Setup Time	MIN	CLK+	6	3
T4	F bus (Abus) Instruction Hold Time	MIN	CLK+	6	5
T5	FADR Setup Time	MIN	CLK+	16	12
T6	FADR Hold Time	MIN	CLK+	4	3
T7	D bus Data Load Setup Time	MIN	CLK+	5	4
T8	D bus Data Load Hold Time	MIN	CLK+	5	5
T9	FEND Setup Time	MIN	CLK+	16	12
T10	FEND Hold Time	MIN	CLK+	4	3
T11	D bus Data Store Output Delay Time	MAX	CLK+	33	27
T12	D bus Data Store Output Valid Time	MIN	CLK+	6	5
T13	MHOLDA- Setup Time*	MIN	CLK-/+	6/25	6/20
T14	MHOLDA- Hold Time*	MIN	CLK-	6	6
T15	FHOLD- Output Delay Time	MAX	CLK+	44	35
T16	FHOLD- Output Valid Time	MIN	CLK+	8	7
T17	MDS- Setup Time	MIN	CLK-/+	6/25	6/20
T18	MDS- Hold Time	MIN	CLK-	6	6
T19	FCCV Output Delay Time	MAX	CLK+	44	34
T20	FCCV Output Valid Time	MIN	CLK+	8	7
T21	$FCC_{1..0}$ Output Delay Time	MAX	CLK+	44	34
T22	$FCC_{1..0}$ Output Valid Time	MIN	CLK+	8	7
T23	FLUSH Setup Time	MIN	CLK+	22	16
T24	FLUSH Hold Time	MIN	CLK+	4	3
T25	FXACK Setup Time	MIN	CLK+	16	12
T26	FXACK Hold Time	MIN	CLK+	4	3
T27	FEXC- Output Delay Time	MAX	CLK+	30	24
T28	FEXC- Output Valid Time	MIN	CLK+	7	7
T29	RESET- Setup Time	MIN	CLK+	12	9
T30	RESET- Hold Time	MIN	CLK+	5	4
T31 **	D Bus Turn-off Time	MIN/MAX	DOE-	6/33	5/25
T32 **	D Bus Turn-on Time	MIN/MAX	DOE-	6/33	5/25

* Specifications for MHOLDB-, MHOLDC-, SHOLD-, and BHOLD- are the same.
** Guaranteed, but not tested

Figure 14. AC specifications

Specifications, continued



* MSH = Most significant half of a 64-bit word
**LSH = Least significant half of a 64-bit word

Figure 15. Timing diagrams

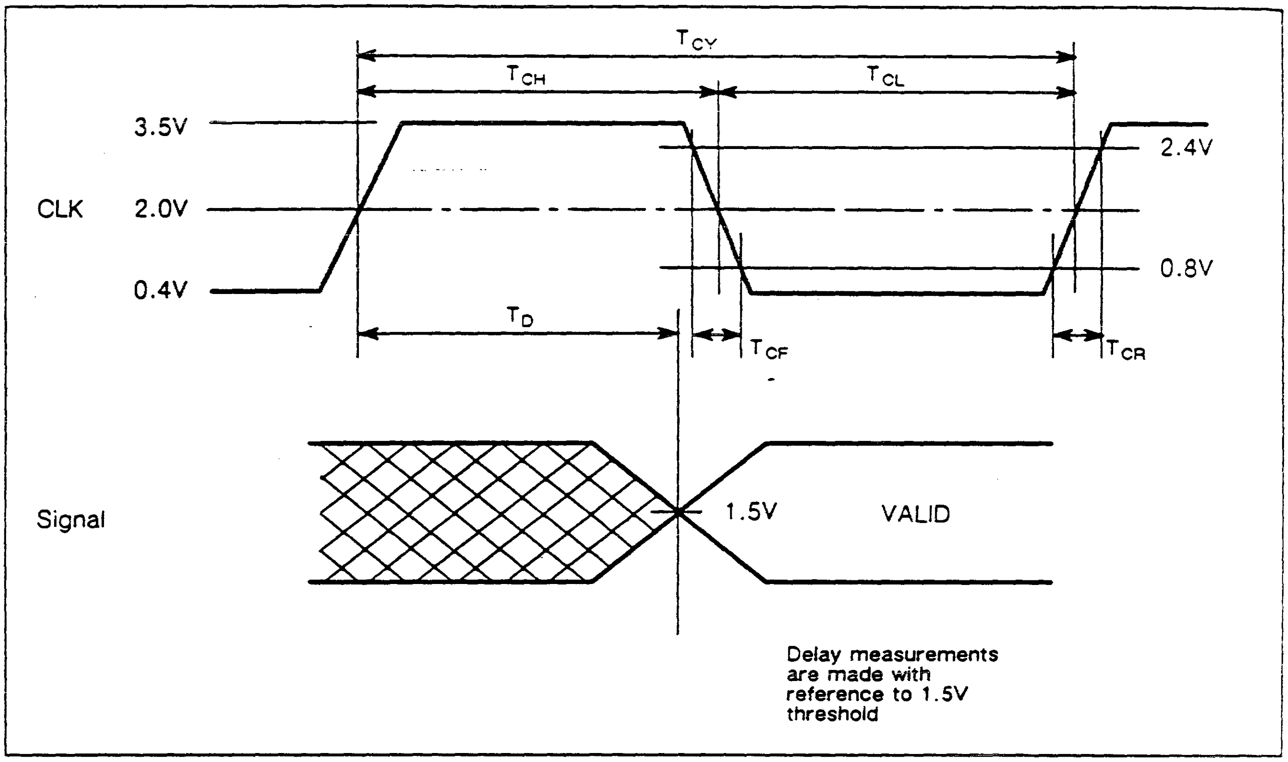


Figure 16. Reference levels in delay measurements

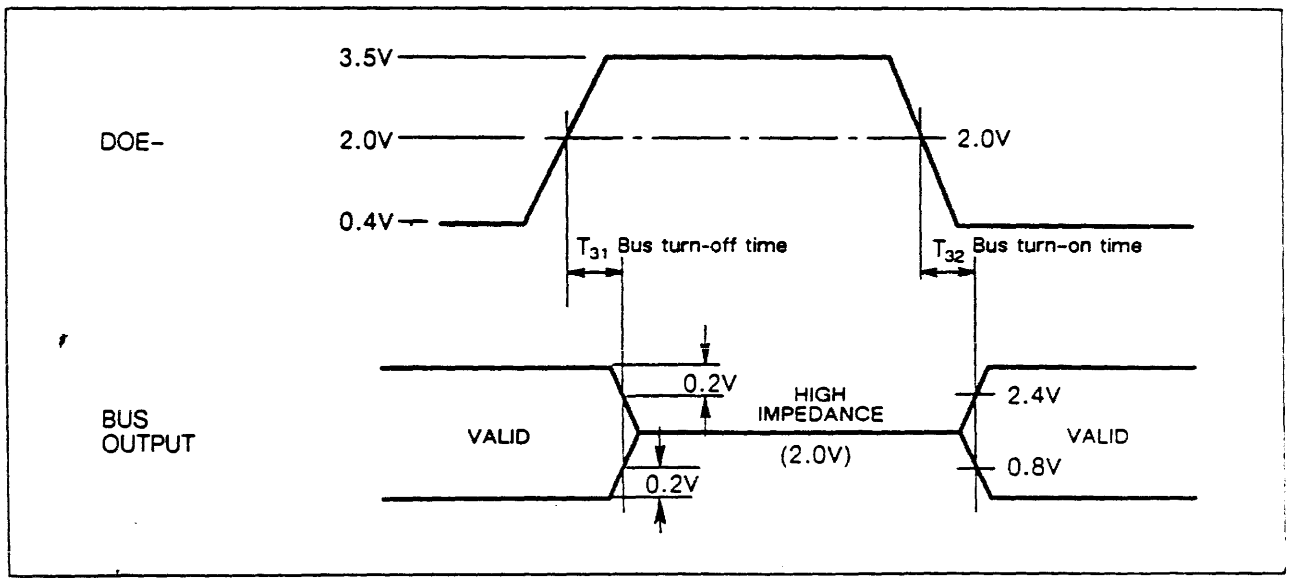


Figure 17. Tri-state timing

Specifications, continued

I/O CHARACTERISTICS

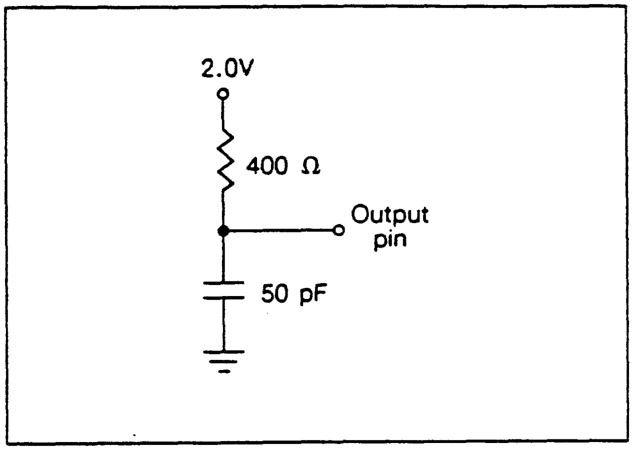


Figure 18. AC test load

Pin Configuration

	Pin A1 Identifier 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
A	X	D22	F22	D24	F24	F25	D26	F26	F27	F28	F29	F30	F31	D31	NC	A
B	D21	VCC	VCC	F23	D23	VCC	D25	VCC	D27	D28	D29	D30	VCC	VCC	VCC	B
C	D20	F21	GND	GND	VCC	GND	GND	VCC	GND	GND	GND	GND	GND	VCC	FCCV	C
D	D19	VCC	GND										GND	Vcc [redacted]	FCC1	D
E	F18	F19	F20										VCC	FCC0	FXACK	E
F	F16	D17	D18										RESET-	GND	FEXC-	F
G	D16	F17	GND										CLK	GND	NC	G
H	F0	F1	D0										GND	VCC	FHOLD-	H
J	D1	DOE-	GND										VCC	MHLDA-	BHOLD-	J
K	D2	VCC	GND										GND	MDS-	MHLD8-	K
L	F2	D3	GND										FLUSH	MHLDC-	SHOLD-	L
M	F3	VCC	D5										GND	FADR	FINS	M
N	D4	VCC	GND	GND	GND	D8	GND	D10	GND	GND	D14	GND	GND	VCC	FEND	N
P	F4	VCC	Vcc [redacted]	F6	VCC	F8	VCC	F11	D12	VCC	VCC	VCC	D15	VCC	NC	P
R	F5	VCC	D6	F7	D7	F9	D9	F10	D11	F12	F13	D13	F14	F15	FP-	R
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

15×15 143-PIN PGA

TOP VIEW
CAVITY DOWN

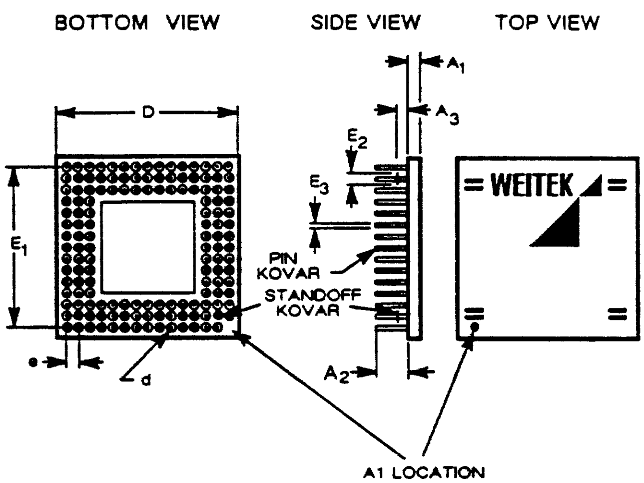
3172

Note: NC = not connected; pins so marked must be left unconnected.
There is no pin at A1. A1 is a locator hole.

Figure 19.

Physical Dimensions

143-PIN GRID ARRAY



Symbol	DIMENSIONS	
	INCHES	MM
A1	0.100 ± 0.010	2.54 ± .20
A2	0.180 typ.	4.57 typ.
A3	0.050 typ.	1.27 typ.
D	1.575 sq. ± 0.016	40.0 ± 0.41
E1	1.400 sq. ± 0.012	35.56 ± 0.30
E2	0.050 dia. typ.	1.27 dia. typ.
E3	0.018 ± 0.002	.46 ± 0.05
d	0.065 dia. typ.	1.65 dia. typ.
e	0.100 typ.	2.54 typ.

Ordering Information

Package Type	Frequency	Case Temperature Range	Order Number
143-pin PGA	20 MHz	0-85°C	3172-020-GCO ~~3170-020-GCD~~
~~143-pin PGA~~	~~25 MHz~~	~~0-85°C~~	~~3170-025-GCD~~

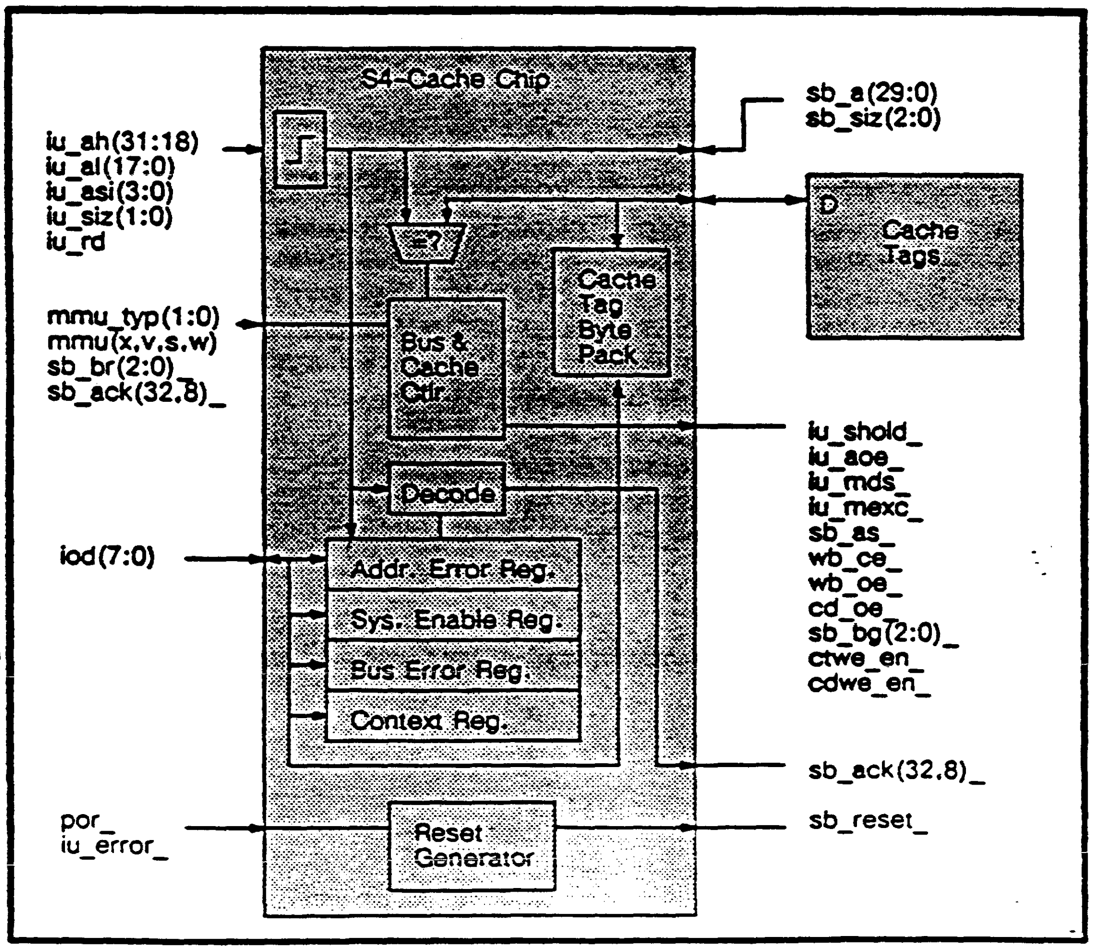
Revision Summary

The following changes have been made in this data sheet relative to the previous edition (May 1989).

Change	Page
Instruction Cycle Counts section added	8
FEXC- Output valid time changed from 8 to 7 ns for 20 MHz	12

Features

- Implements 64-256 KByte write-through Instruction/Data cache with 16-byte line size
- Performs cache tag comparison
- Controls SBus reads and writes
- Automatically fills cache on cache misses
- Controls mastership of SBus for DMA
- Performs buffered writes with external write buffer
- Replaces cache tag read/write buffers
- Performs cache flush comparisons
- Controls system-wide byte packing
- Contains Sun-4 Virtual Address Error Latches
- Maintains copy of 4-bit Sun-4 context register
- Contains Sun-4 System Enable Register
- Contains Sun-4 Bus Error Registers
- Monitors bus for unacknowledged transfers
- Generates system reset



Cache Interface	25	
ct_a(29:16)	BD4TU	Cache Tag Address bits
ct_c(3:0)	BD4TU	Cache Tag Context bits
ct_s	BD4TU	Cache Tag Supervisor
ct_v	BD4TU	Cache Taq Valid
ct_wa	BD4TU	Cache Tag Write Allowed
ctwe_en_	BT4	Cache Tag Write Enable Enable. Goes to S4–Clock.
cdwe_en_	BT4	Cache Data Write Enable Enable. Goes to S4–Clock.
cd_oe_	BT4	Cache Data Output Enable.
car_en_	BT4	Cache Address Register Clock Enable.
Miscellaneous	4	
wb_oe_	BT4	Write Buffer Output Enable
wb_ce_	BT4	Write Buffer Clock Enable.
s4c_oe_	TLCHTNU	S4–Cache chip output enable.
s4c_test_	IBUFNU	S4–Cache chip Test mode.

Signals:	144	
Device Type:	LMA9284	(IO:158 VDD:4 VSS:6)
Package Type:	PFP160	(PADS:160 VDD:7 VSS:9)

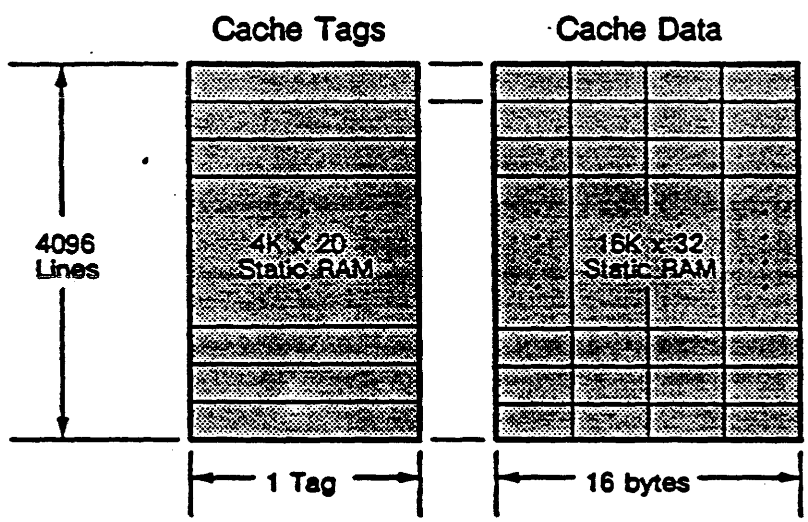
Input/Output Buffer Definitions

DRVC8	Input clock driver
IBUFNU	Input buffer, CMOS level, inverting, internal pullup
TLCHT	Input buffer, TTL level, non–inverting
TLCHTU	Input buffer, TTL level, non–inverting, internal pullup
TLCHTNU	Input buffer, TTL level, inverting
BD#TU	Bidirectional buffer, TTL input levels, internal pullup, # indicates output drive
BD#TRU	Bidirectional buffer, TTL input levels, internal pullup, slew–rate controlled output, # indicates output drive
BT#	Tri–statable Output buffer, CMOS, # indicates output drive current.

Functional Description

Cache Overview

The cache implemented with the aid of the S4–Cache chip is a write-through mixed instruction/data cache with a 16–byte line size. A typical implementation is shown in the following diagram:

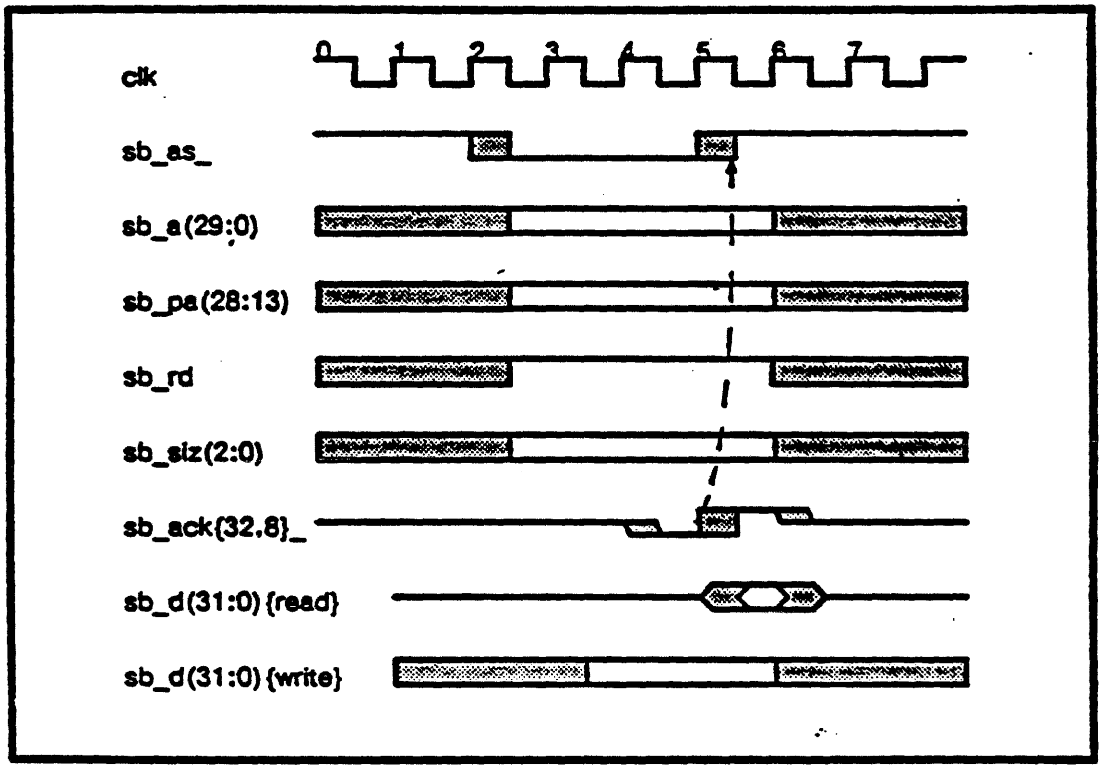


The cache tag and cache data memories are built using external generic static RAM chips. Although the programmer's model of the cache data RAM is 4096 lines of 16 bytes, it is currently implemented with eight 16K x 4 static RAMs.

The size of the cache may vary from 4096 lines deep to 16,384 lines deep. Larger implementations of the cache will connect the unused cache tag pins to the appropriate address bits latched in the cache address register.

SBus Overview

The SBus fundamental operation is shown in the diagram below. The SB_AS_ signal indicates the validity of SB_PA(28:00), SB_RD, SB_SIZ(2:0) and the signals derived combinatorially from these signals. On the rising clock edge at which AS_ is sampled true, these signals will also be valid with the setup specified. The cycle will continue until an acknowledge is received from the accessed device. Wait states will be inserted on the SBus until the acknowledge is received.

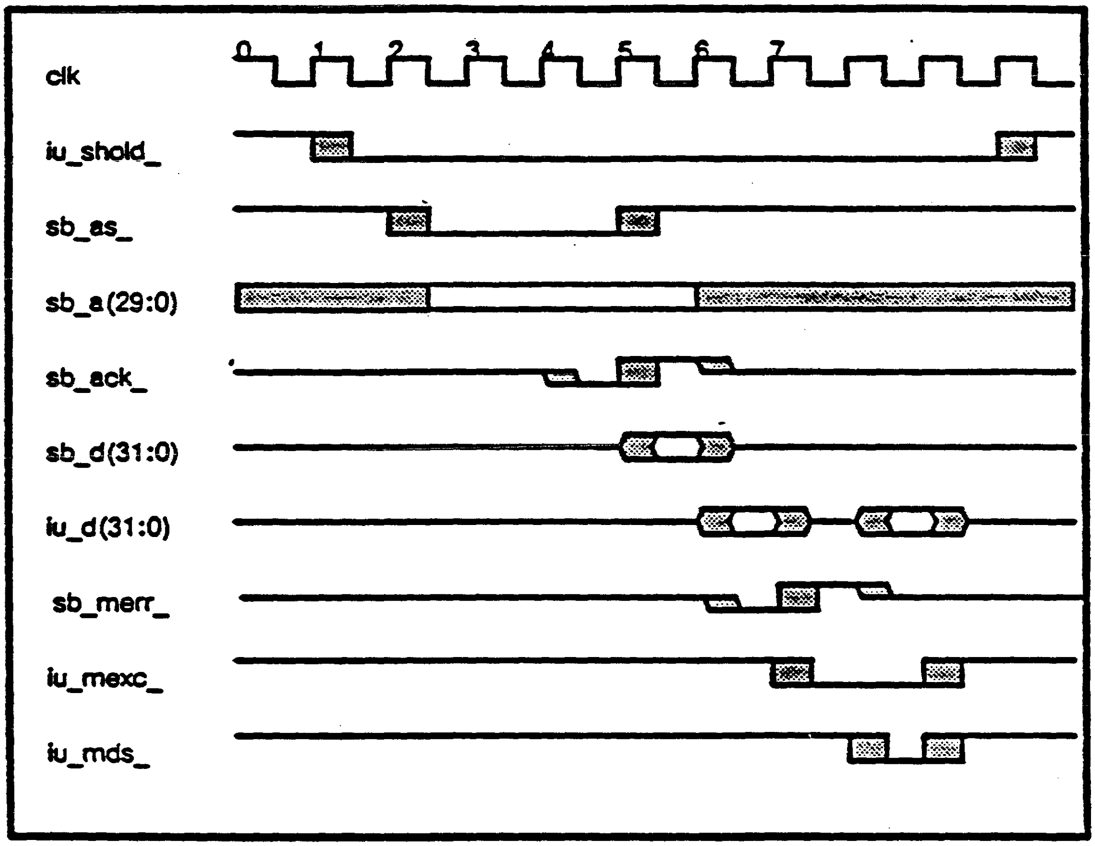


The addresses, read, and size signals will be held valid until the clock edge after the one on which the acknowledge is sampled true. See the tables below for acknowledge and size encoding.

Shared control signals SB_ACK32_, SB_ACK8_, SB_ERR_, and SB_MERR_ must follow a special protocol, which requires that the signal is taken out of tri-state mode, driven low for the desired number of clocks, then driven high for one clock before being tri-stated again. See the SBus specification for further details.

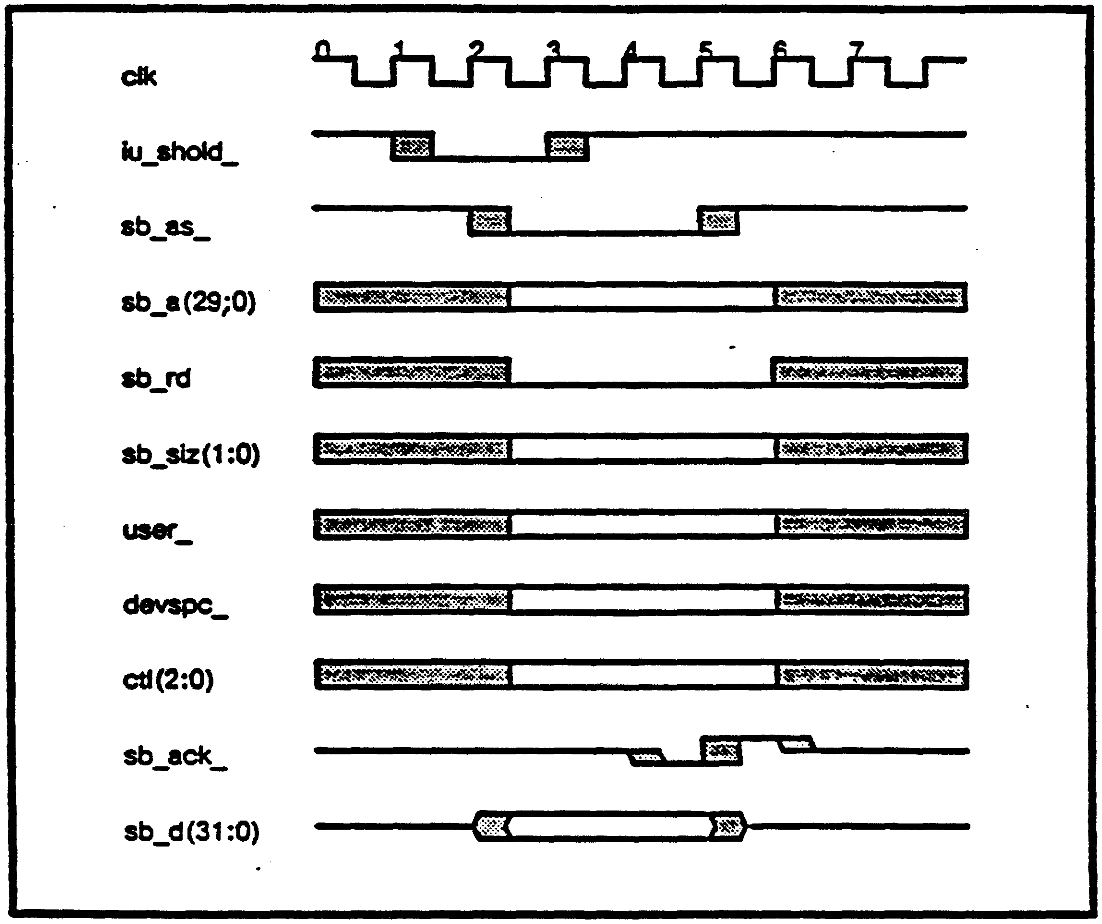
Parity Errors

Parity errors are reported by the S4-Buffer chip to the S4-Cache chip via SB_MERR_. The S4-Cache chip reports parity errors to the IU on IU cycles by asserting IU_MEXC_ as shown in the following diagram.



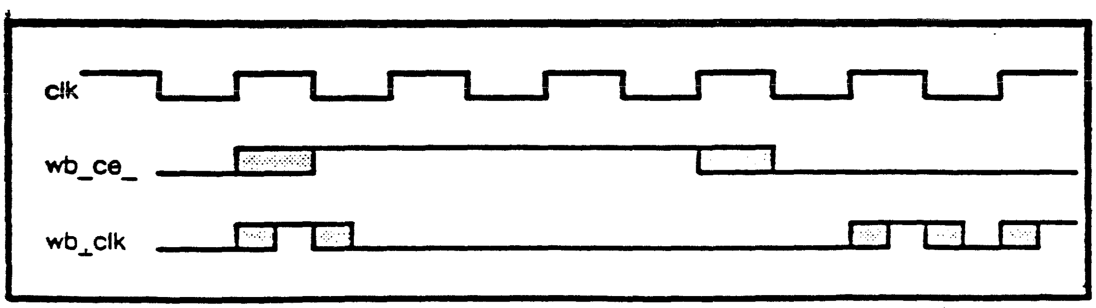
SBus Buffered Writes

The S4–Cache chip performs buffered writes to Type 0 and Type 1 Spaces using the write buffer in the S4–Buffer chip. The IU is held starting when the miss is detected and ending when the MMU has been checked. This occurs invisibly to the SBus, where the buffered write is indistinguishable from a standard write. Write data is available on the SBus on the rising edge at which AS_ is sampled true, and on the IOD bus one clock later.



WB_CE_ Function

The WB_CE_ signal goes to the S4–Buffer chip, where it is used to generate the clock to the write buffer as shown in the following diagram:



Dynamic Bus Sizing

Byte Packing

To execute code contained in 8-bit devices on either the SBus or the IOD bus, the S4_Cache chip must pack the bytes up to fit the word length of the SPARC chip, as instruction fetches assume this data width. The S4-Cache chip transforms the SPARC data bus into a dynamically-sized bus somewhat like that of the Motorola 68020. The number of bytes involved in the first cycle is encoded on the three SB_SIZ signals. The current slave device responds with its port width encoded on the two SB_ACK signals. An IU word-length access will be converted into the appropriate number of shorter accesses if the accessed device indicates its port width is less than 32 bits.

Transfer Size Encoding

sb_siz2	sb_siz1	sb_siz0	Transfer Size
0	0	0	4 Bytes
0	0	1	1 Byte
0	1	0	2 Bytes
0	1	1	Not Used
1	0	0	16-Byte Burst
1	0	1	Not Used
1	1	0	Not Used
1	1	1	Not Used

Although the SBus specification allows 3-byte operations, none will be generated by the S4-Cache chip because all SPARC transfers are aligned.

DMA Cycles

Bus Arbitration

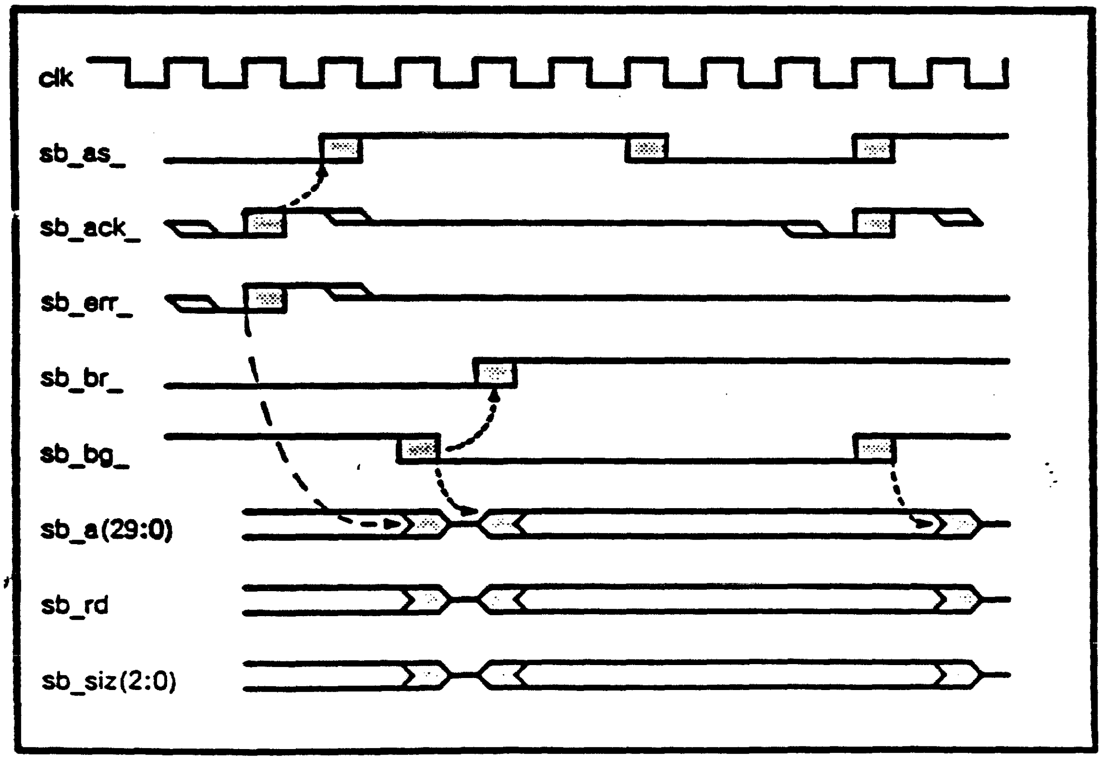
The S4–Cache chip receives three levels of DMA bus request {SB_BR(2:0)–} and generates three corresponding levels of bus grants {SB_BG(2:0)–}. In case more than one bus request is received simultaneously, the bus request priorities are as follows:

IU Write Hits	Highest Priority
SB_BR0_	
SB_BR1_	
SB_BR2_	
IU Misses	Lowest Priority

If a bus request is pending at the end of a DMA cycle, the bus arbiter will use a round–robin bus grant scheme so that all DMA masters can share equal bus bandwidth.

Rerun Cycles

The S4–Cache chip implements a rerun protocol that causes the current SBus cycle to be aborted and restarted later. This allows resolution of deadlocks between the IU and DMA, and allows SBus slaves to have long read latency without locking out DMA.



Deadlocks can occur when a single functional module is capable of being both a SBus slave and a DMA master. Such a module typically selects either its master or slave mode.

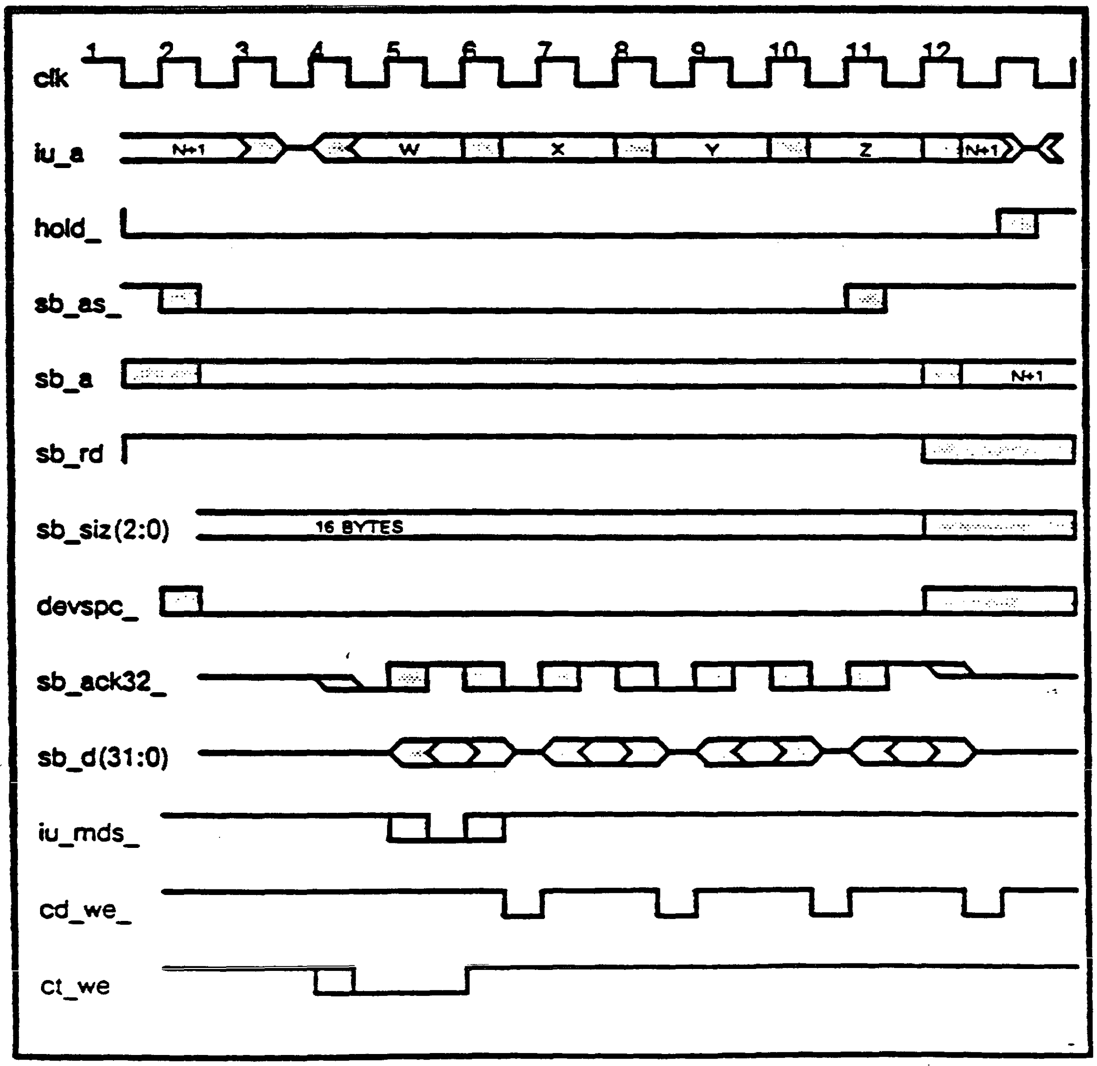
Cache Fills

The cache is filled under the following conditions:

Read cycle &
Device space &
Page is marked cacheable (IMMU_X) &
EN_CACHE bit in System Enable Register is set &
No protection error is detected.

A cache fill cycle consists of four 32-bit reads of main memory. As the cache controller is capable of accepting an acknowledge on every clock, the four reads will typically be done using a high-speed burst mode access of the main RAMs. After the first acknowledge the bus controller will strobe the data into the IU, making the assumption that the memory provides the requested word first rather than providing the first word in the line.

Cache Fill with Non-Continuous ACKs

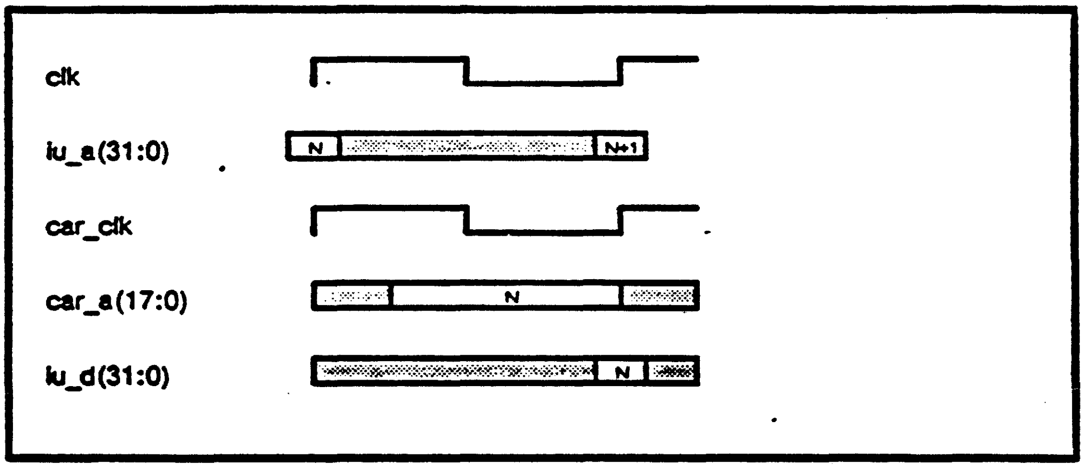


Cache Hits

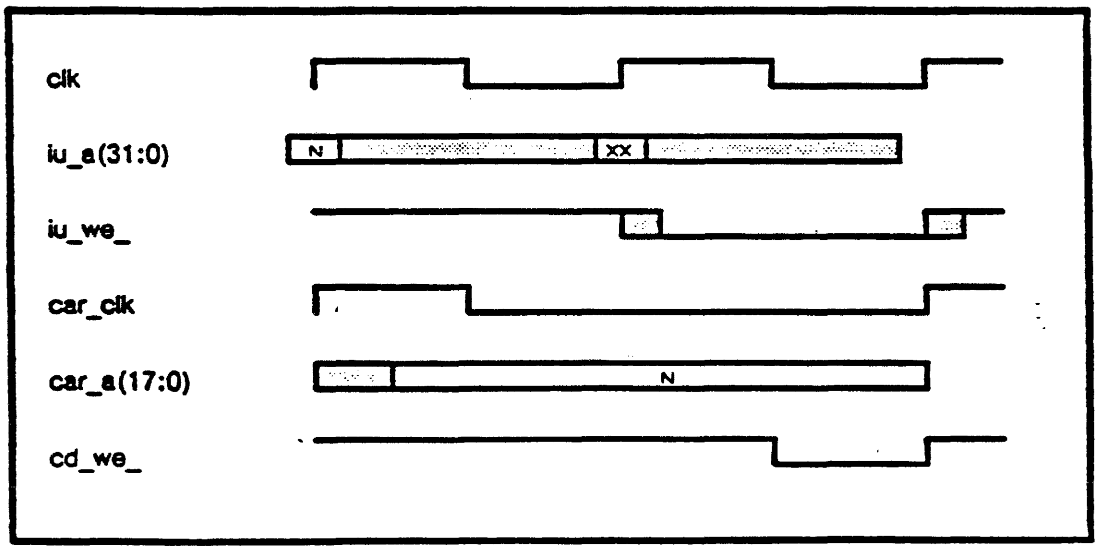
A cache hit occurs under the following conditions:

Device space &
CT_V high (cache tag is valid) &
CT_A(29:16) == latched IU_A(29:16) &
IU_A(31) == IU_A(30) == IU_A(29) &
{CT_S & Supervisor cycle} OR {!CT_S & CT_C(3:0) == CID(3:0)} &
{IU_RD OR (CT_WA & !Store double & SBus Idle)}

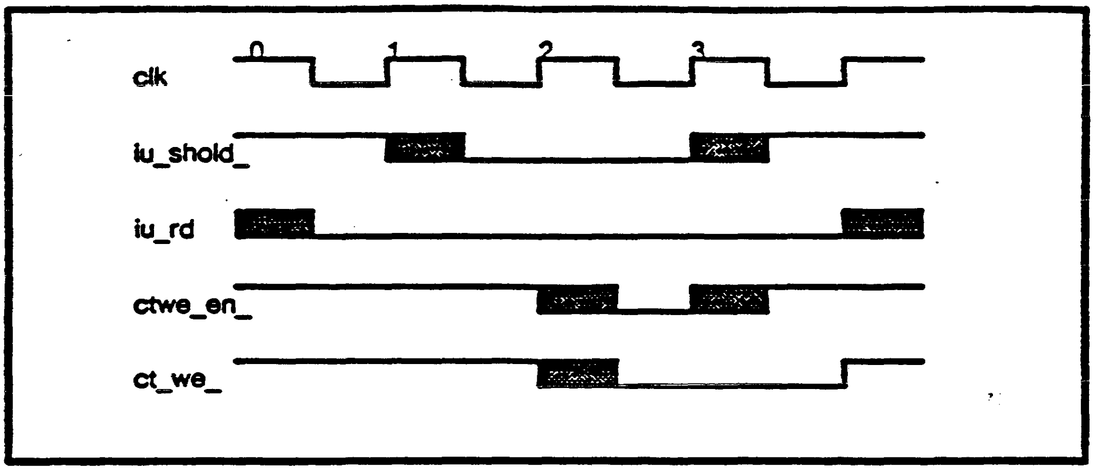
Cache Read Hit



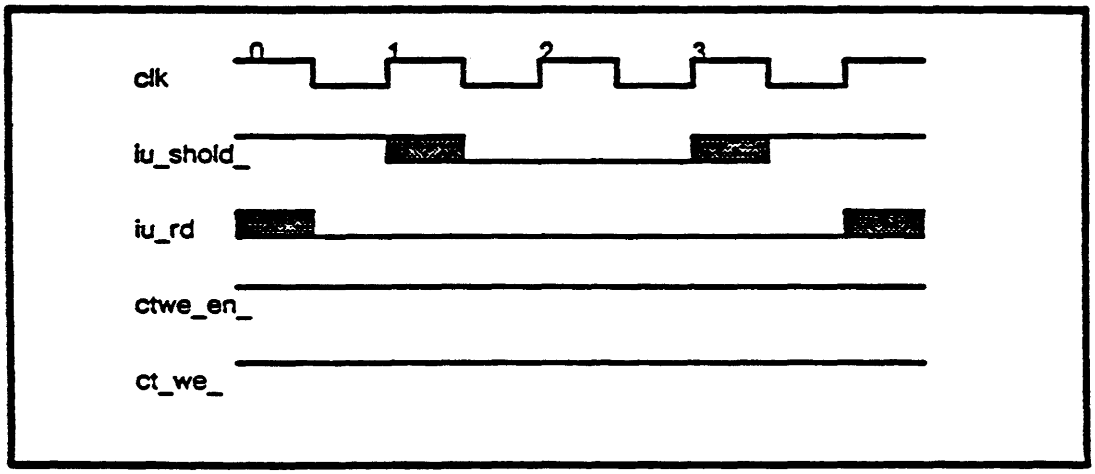
Cache Write Hit



Cache Flush Satisfying Match Criteria



Cache Flush Not Satisfying Match Criteria



Address Map

Device Space and Control Space

The SPARC address space identifiers are divided into two "spaces" according to the following table: The signal DEVSPC_ chooses between device space and control space address maps. Device space devices are accessed with physical addresses provided by the MMU, while control space devices are accessed with virtual addresses on the SBus.

ASI	Function	Space
0–1	Reserved	Control
2	IU Extensions	Control
3	Segment Map	Control
4	Page Map	Control
5–7	Reserved	Control
8	User Instruction	Device
9	Supervisor Instr.	Device
A	User Data	Device
B	Supervisor Data	Device
C	Segment Flush	Control
D	Page Flush	Control
E	Context Flush	Control
F	Reserved	Control

Registers

Shadow Context Register

The Shadow Context Register maintains a copy of the Context Register that is found in the S4–MMU chip. It is used internally to the S4–Cache chip in the cache hit comparator, the cache flush comparator, and the cache tag write data. It is cleared on SB_RESET_ and written simulteneously with the Context Register in the S4_MMU chip. It can be read only with 8–bit operations on an odd–byte location. The bits are assigned as follows:

Write:	D(31:28)	Unused	Read back as zeroes
	D(27:24)	CID(3:0)	Write Only
Read	D(23:20)	Unused	Read back as zeroes
	D(19:16)	CID(3:0)	Read Only

System Enable Register

The System Enable Register enables various system functions and allows booting. This register can be read and written under software control, but can only be accessed with 8–bit operations. All bits are initialized to zero by SB_RESET_. Bits are assigned as follows:

D(31)	EN_BOOT_	Enable Boot State
D(30)	Unused	
D(29)	EN.DVMA	Enable Direct Virtual Memory Access
D(28)	EN_CACHE	Enable Cache Fills & Hits
D(27)	Reserved	
D(26)	SWRESET	Software Reset.
D(25)	Reserved	
D(24)	Reserved	Reads back as zero. Write has no effect.

EN_BOOT_. Boot state (active low) forces all supervisor program fetches to the EPROM device independent of the setting of the memory management. All other types of references are unaffected and will be mapped as during normal operation of the processor.
EN_DVMA. This bit enables all DVMA, including on–board and off–board.
EN_CACHE. When this bit is cleared, no cache fills will be performed and all IU reads will miss.
SWRESET. A low–to–high transition on this bit will generate a SB_RESET_.

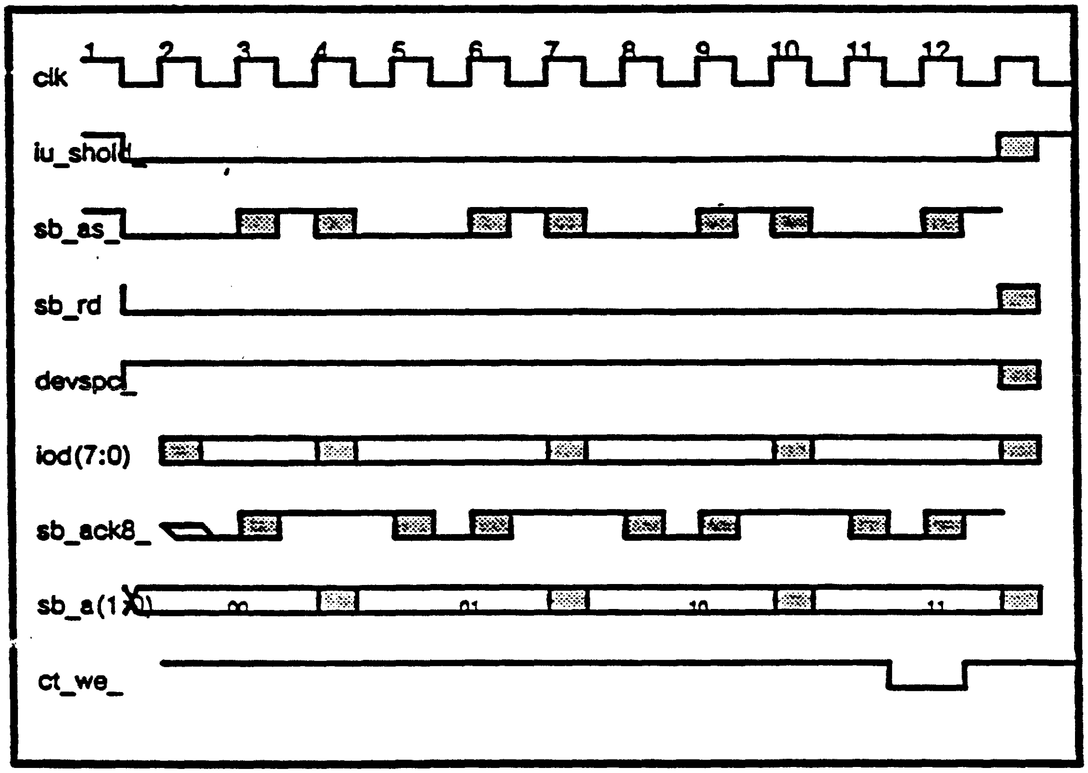
BUS ERROR REGISTERS

Four bus error registers are contained in the S4–Cache chip, located at the following addresses:

0x6000 0000	Synchronous Error Register
0x6000 0004	Synhronous Error Virtual Address Register
0x6000 0008	Asynchronous Error Register
0x6000 000C	Asynchronous Error Virtual Address Register

Cache Tags

The cache tags are directly readable and writable in control space. Write cycles must be performed with 32-bit accesses only. Other widths during writes will cause a Size Error Memory Exception because the S4-Cache chip includes a byte packing register that demultiplexes the 8-bit IOD bus up to the 32-bit cache tag bus, and it can only operate four bytes at at time. The cache tags are not initialized in hardware, and so zeroes must be written to all CT_V bits before the cache is enabled. Cache tag direct reads make use of the standard byte-packing feature of the S4 chip set described earlier. The following diagram shows the operation of the cache tag byte packing register in the S4-Cache chip on a cache tag direct write:



The format of the cache tags is as follows:

D(31:26)		Unused
D(25:22)	CT_C(3:0)	Cache Tag Context bits
D(21)	CT_WA	Write Allowed
D(20)	CT_S	Supervisor-only access protection bit
D(19)	CT_V	Cache Tag Valid
D(18:16)		Unused
D(15:2)	CT_A(29:16)	Virtual address bits A(29:16)

Timing Specifications

Output Delays

Conditions: VCC=4.75 to 5.25V, TA=0 to +70C, Output Load=15 pF

Symbol	From	To	min	max	unit
t1	clk high	clk high	50	—	ns
t2	clk	iu_a!		34.4	ns
t3	clk	iu_aoe_		17.8	ns
t4	clk	iu_mds_		17.1	ns
t5	clk	iu_mexc_		17.4	ns
t6	clk	iu_mhold_		16.3	ns
t7	clk	sb_a (untransl.)		41.8	ns
t8	clk	sb_a (seg. map)		30.3	ns
t9	clk	sb_ack32_		24.4	ns
t10	clk	sb_ack8_		24.4	ns
t11	clk	sb_as_		27.7	ns
t12	clk	sb_bg_		19.8	ns
t13	clk	sb_err_		26.1	ns
t14	clk	sb_merr_		24.7	ns
t15	clk	sb_rd		29.1	ns
t16	clk	sb_reset_		23.0	ns
t17	clk	sb_siz		36.9	ns
t18	clk	car_en_		16.7	ns
t19	clk	cd_oe_		22.0	ns
t20	sb_rd_	cd_oe_		16.1	ns
t21	iu_rd_	cd_oe_		11.8	ns
t22	clk	cdwe_en_		15.6	ns
t23	clk	ct_a		33.7	ns
t24	clk	ct_c		24.9	ns
t25	clk	ct_s		24.7	ns
t26	clk	ct_v		24.8	ns
t27	sb_rd	ct_v		12.5	ns
t28	clk	ct_wa		24.7	ns
t29	clk	ctl		16.0	ns
t30	clk	ctwe_en_		18.2	ns
t31	clk	devspc_		16.8	ns
t32	clk	io_d		74.5	ns
t33	sb_a	io_d		41.0	ns
t34	clk	user_		17.1	ns
t35	clk	wb_ce_		16.0	ns
t36	clk	wb_oe_		16.4	ns

Change History

2/1/88

Sunray support—	Removed.
Hardware Cache Consistency—	Removed.
SB_ACK @ State 4—	Removed restriction of no ACKs before state 5.
Cache filling—	Removed restriction to Type 0 Space. Added requirement of IMMU_X.
Cache Hit definition—	Added term for write hits.
Context Flush criteria—	Fixed bug in CT_S polarity.
Table of Contents—	Added.
Timing Specifications—	Added a few.

7/18/88

Cleaned up errors everywhere.....

Timing—	Added many new timing specs. Used post-route timings.
Reruns—	SB_AS_ is negated one clock later than prev. spec.
Cache Hits—	Changed definition of cache hit on page 19.
Cache Flushing—	Removed notes about flushes before changing MMU. Modified timing diagrams; IU_SHOLD_ for 2 clocks.
Bus Error Registers—	Added SER, SEVAR, AER, AEVAR definitions.
Cache Data—	Added restriction: no write after control space read.

Errata

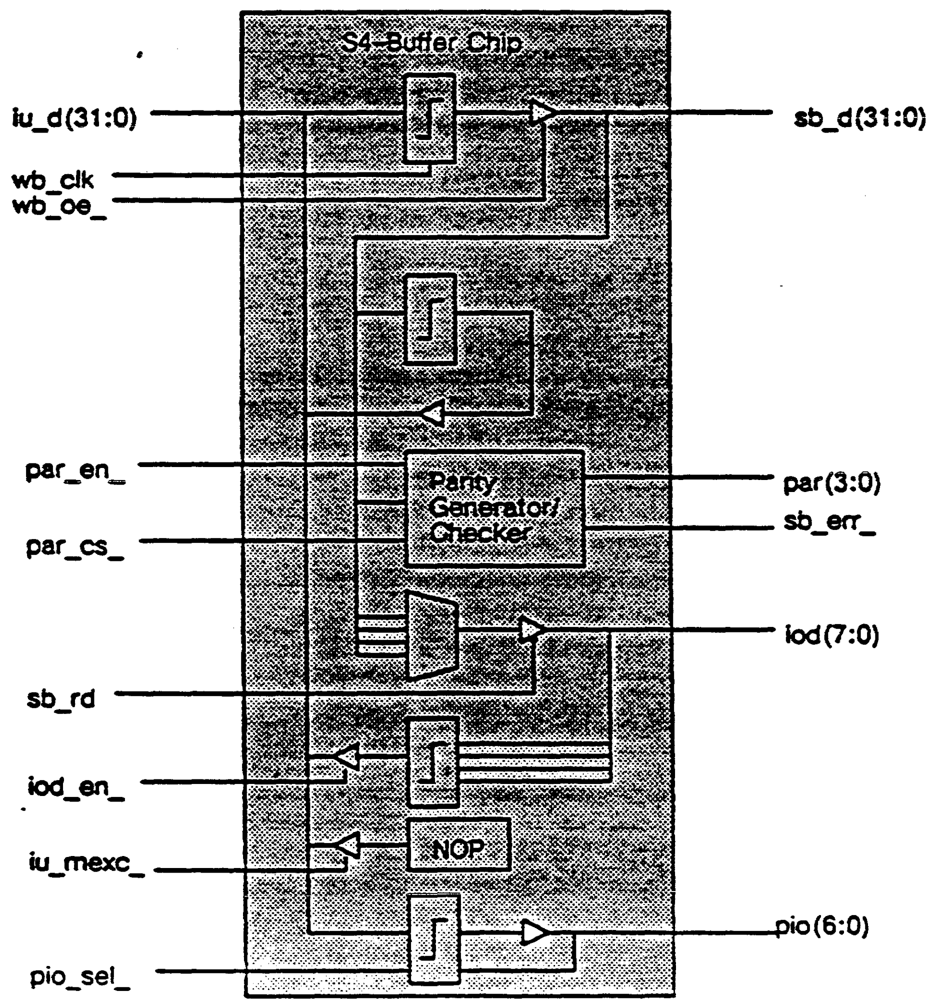
7/18/88

DMA Timeouts:

Timeouts that terminate DMA cycles will cause the TO_ERR bit in the Synchronous Error Register will be set incorrectly.

Features

* Generates and checks parity on main memory accesses
* Performs buffered write cycles in conjunction with the S4–Cache chip
* Multiplexes 32–bit IU data bus down to 8–bit IO data bus on write cycles
* Demultiplexes and latches 8–bit IO data bus up to 32–bit IU data bus on read cycles
* Contains byte–packing registers for dynamically sized reads from SchoolBus data bus
* Contains Sun–4 Parity Control Register
* Contains 7–bit open–drain general purpose I/O register (PIO)
* Forces No Op on memory exceptions



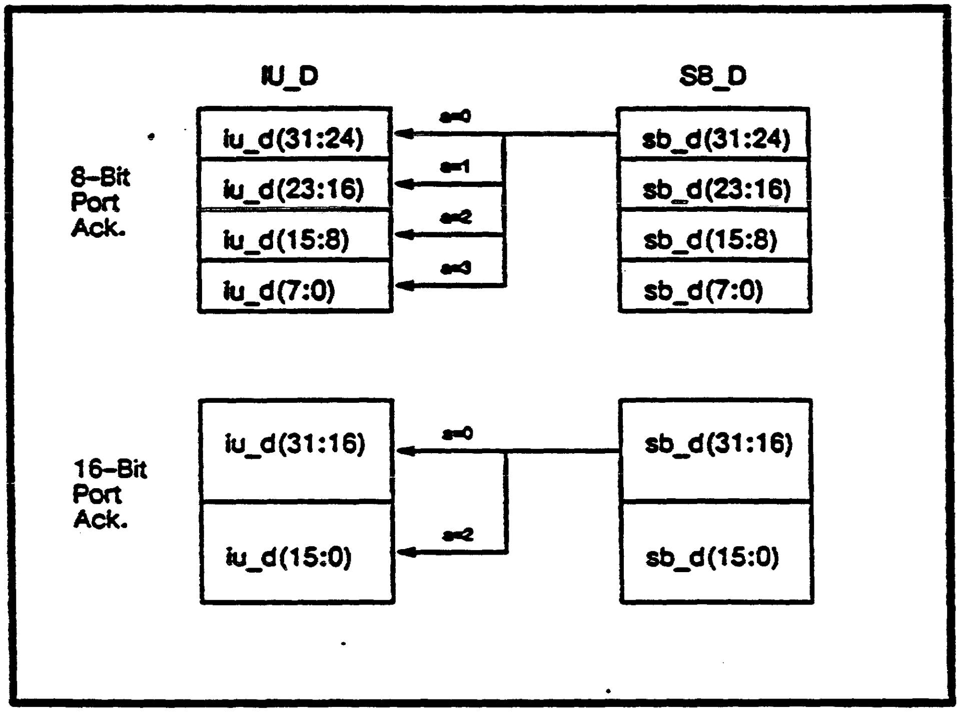
IBUFN	Input buffer, CMOS, inverting
IBUFNU	Input buffer, CMOS, inverting, internal pullup
TLCHT	Input buffer, TTL, non-inverting
TLCHTN	Input buffer, TTL, inverting
BD#TRU	Bidirectional buffer, TTL input levels, # indicates output drive, internal pullup
BT#	Tri-statable output buffer, CMOS, # indicates output drive current.
BD4TOD	Open drain buffer, TTL, non-inverting.

Port Location

The location of 8, 16 and 32–bit ports on the 32–bit SchoolBus data bus is defined as follows:

sb_d(31:24)	sb_d(23:16)	sb_d(15:8)	sb_d(7:0)
8–bit port			
16–bit port			
32–bit port			

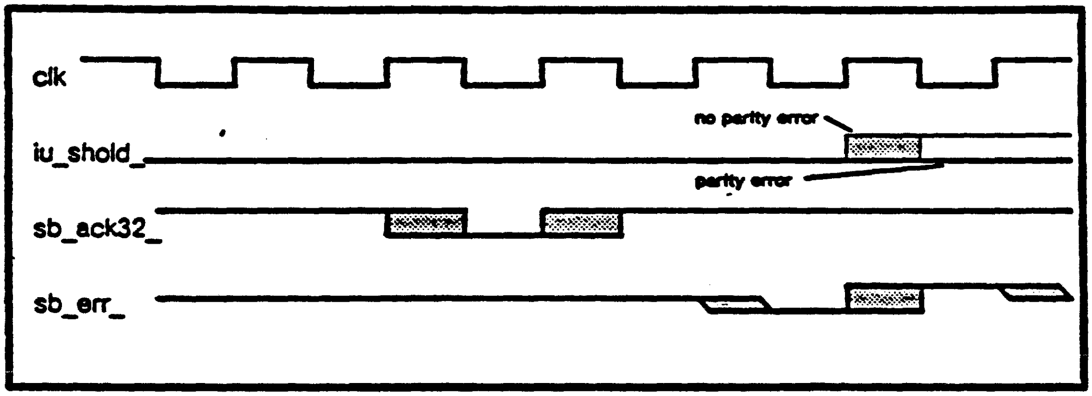
SB_D Read Data Latching



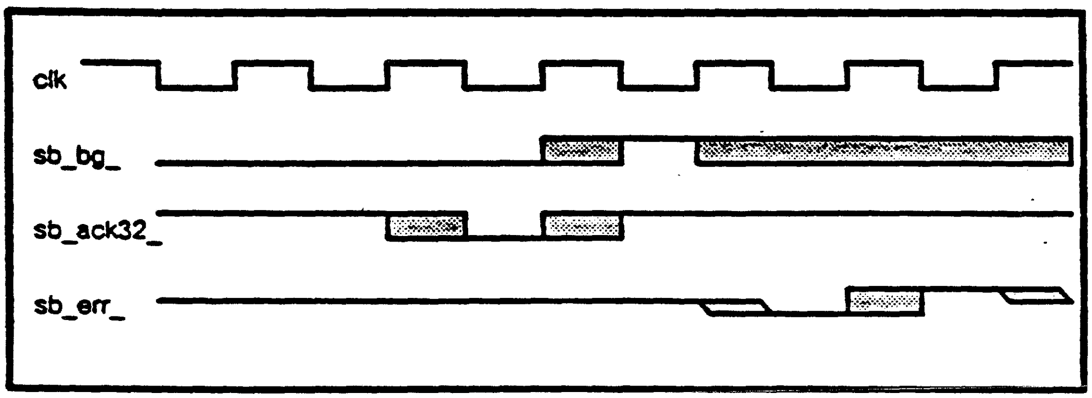
Parity Checking

Parity is checked on read cycles during which PAR_EN_ is active and the Parity Check bit is set in the Parity Control Register (See below for a description of the Parity Control Register). Parity errors are reported by asserting SB_ERR_ for one clock period, and setting the bits in the parity control register corresponding to the bytes in which parity errors were detected. SB_ERR_ will cause the S4-Cache chip to assert IU_MEXC_, causing the IU to take a memory exception trap. Parity checking is even, meaning a byte of ones requires a zero parity bit, so that a data and parity bus floating high will cause a parity error.

Parity errors are reported on IU cycles by a one-clock low pulse on the SB_ERR_ signal, two clocks after SB_ACK32_, as shown in the following diagram:



Parity errors are reported on DVMA cycles by a one-clock low pulse on the SB_ERR_ signal, one clock after SB_ACK32_, as shown in the following diagram. Note that on DVMA cycles, this SB_ERR_ signal could occur after SB_BG_ has been asserted to another device, so that device must take care not to react.



The system bus controller implements dynamic bus sizing for CPU cycles. This function is performed through the joint efforts of the S4-Cache and the S4-Buffer.Taking the desired transfer width and the port size into account, the bus controller packs data from narrower ports up to the desired width by performing several bus cycles. This byte packing is performed only for CPU cycles, not for DMA cycles. The cycles appear as separate cycles indistinguishable from cycles that don't involve byte packing.

Transfer Size	Port Size	Controller Response
1-Byte	Any	Single BYTE cycle
2-Byte	8-bit	Two BYTE cycles
"	16-bit	One HALF cycle
"	32-bit	One HALF cycle
4-Byte	8-bit	Four BYTE cycles
"	16-bit	Two HALF cycles
"	32-bit	One WORD cycle

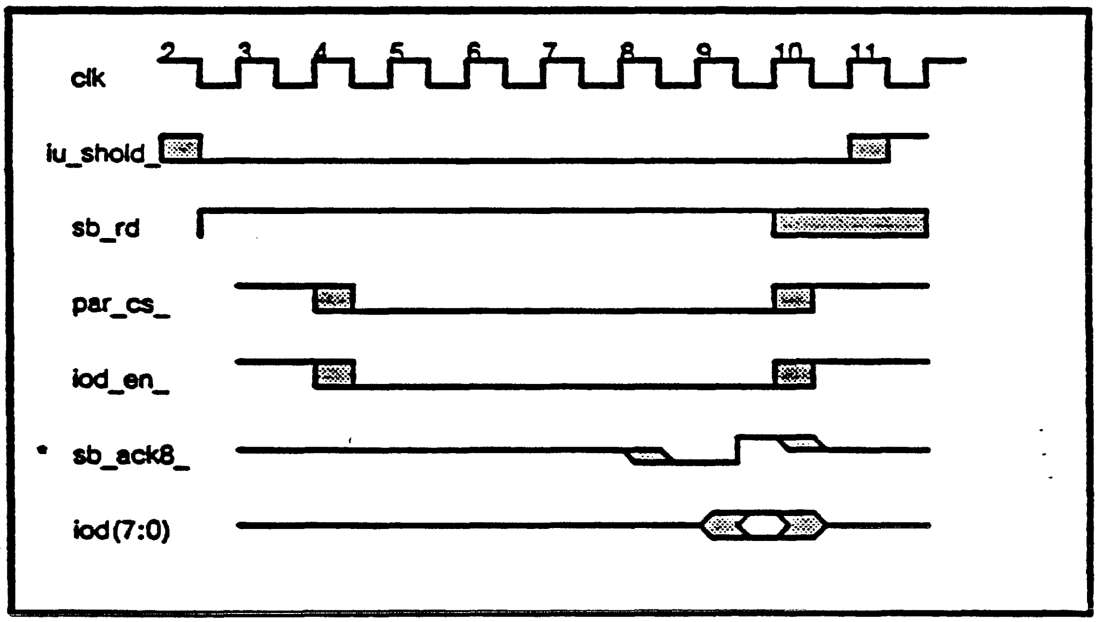
Parity Control Register

The Parity Control Register provides facilities for enabling and reporting parity errors and for testing the parity generation and checking logic. It is a 32-bit read/write register, cleared on SB_RESET_, accessible 8 bits at a time over the IOD bus. It has the following fields:

D(31:8)	Reserved	Read as zero
D(7)	Parity Error	Set on any parity error
D(6)	Second Error	Set if D(7) is set and new error occurs
D(5)	Parity Test	Set to write parity with the inverse polarity to test the operation of the parity error circuitry. With Parity Test off, correct parity is generated on all memory write cycles.
D(4)	Parity Check	Enables parity checking
D(3)	Parity Error 24	Records parity error on data bits 31:24
D(2)	Parity Error 16	Records parity error on data bits 23:16
D(1)	Parity Error 08	Records parity error on data bits 15:8
D(0)	Parity Error 00	Records parity error on data bits 7:0

Note that the Error Bits D(7, 6, 3:0) are not writable. They are set by errors and reset automatically when read back.

Parity Control Register Read



* sb_ack8_ is generated by the MMU on Parity Control Register accesses.

Timing Specifications

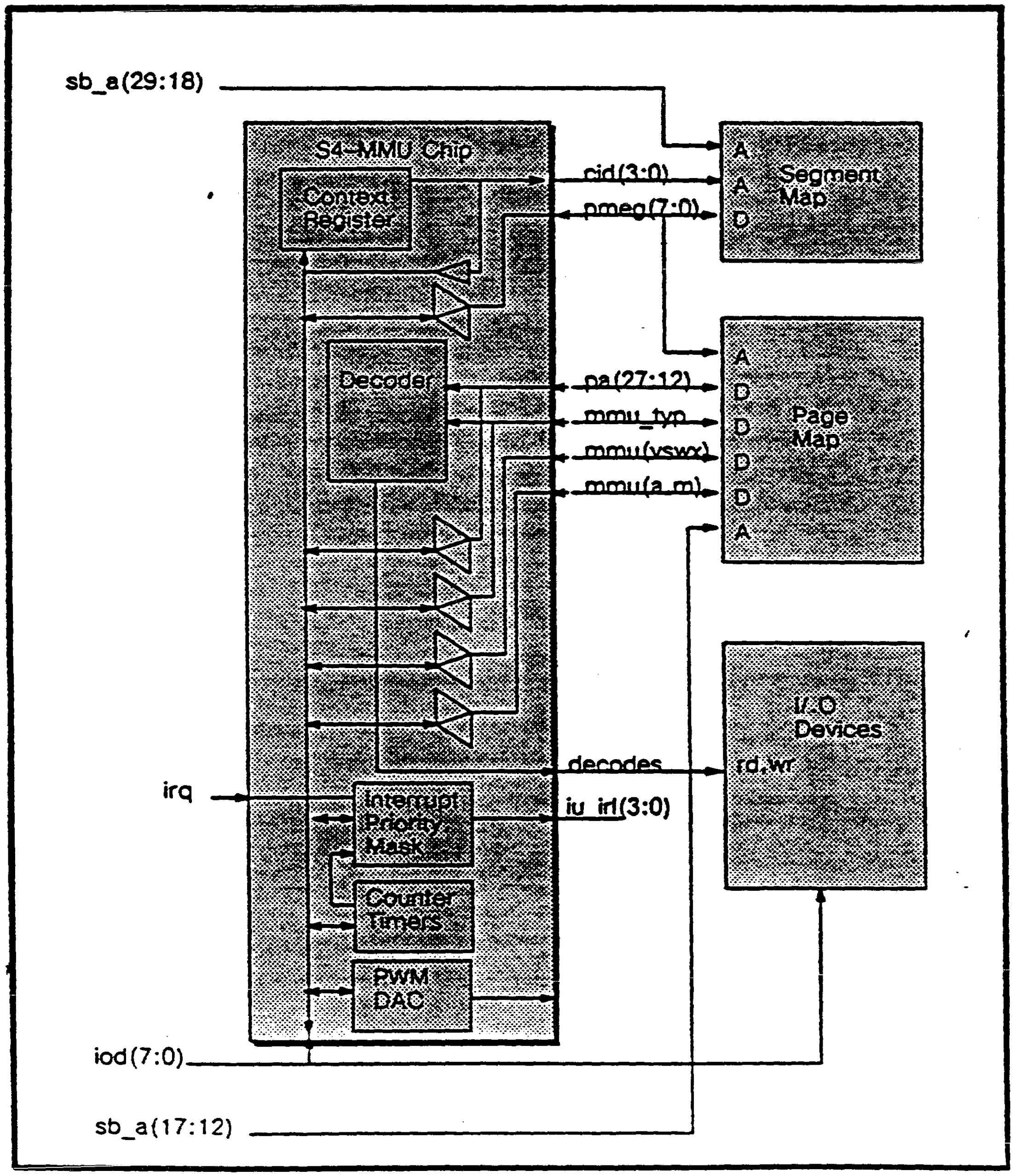
Conditions: VCC=4.75 to 5.25V, TA=0 to +70C, Output Load=100 pF

Symbol	From	To	min	max	unit
t	xclk high	xclk high	40	—	ns
t	clk	iu_d	10.5	23.5	ns
t	clk	sb_d	18	27	ns
t	clk	par	23	34.5	ns
t	clk	pio	12	21.5	ns
t	clk	iod	8	31.5	ns
t	clk	sb_merr_	16	23.5	ns
t	iu_mexc	iu_d	6.5	12	ns
t	wb_oe_	sb_d	5.5	21	ns

Setup time for all signals is 15 ns. Hold time for all signals is 3 ns.

Features

- Provides decodes and timing strobes for all Sun-4 Type 1 devices
- Replaces all MMU read/write buffers
- Automatically updates MMU statistics bits during bus cycles
- Prioritizes 15 levels of Interrupts
- Sun-4 Interrupt register provides software interrupts, interrupt enable
- 4-bit context register provides switchable MMU contexts
- Two counters generate high-resolution periodic interrupts



Functional Description

Device Space and Control Space

The SPARC address space identifiers are divided into two "spaces" according to the following table:

ASI	Function	Space
0–1	Reserved	
2	IU Extensions	Control
3	Segment Map	Control
4	Page Map	Control
5–7	Reserved	
8	User Instruction	Device
9	Supervisor Instr.	Device
A	User Data	Device
B	Supervisor Data	Device
C–F	Reserved	

The signal DEVSPC_ chooses between device space and control space address maps. Device space devices are accessed with physical addresses provided by the MMU, while control space devices are accessed with virtual addresses provided by the SPARC processor.

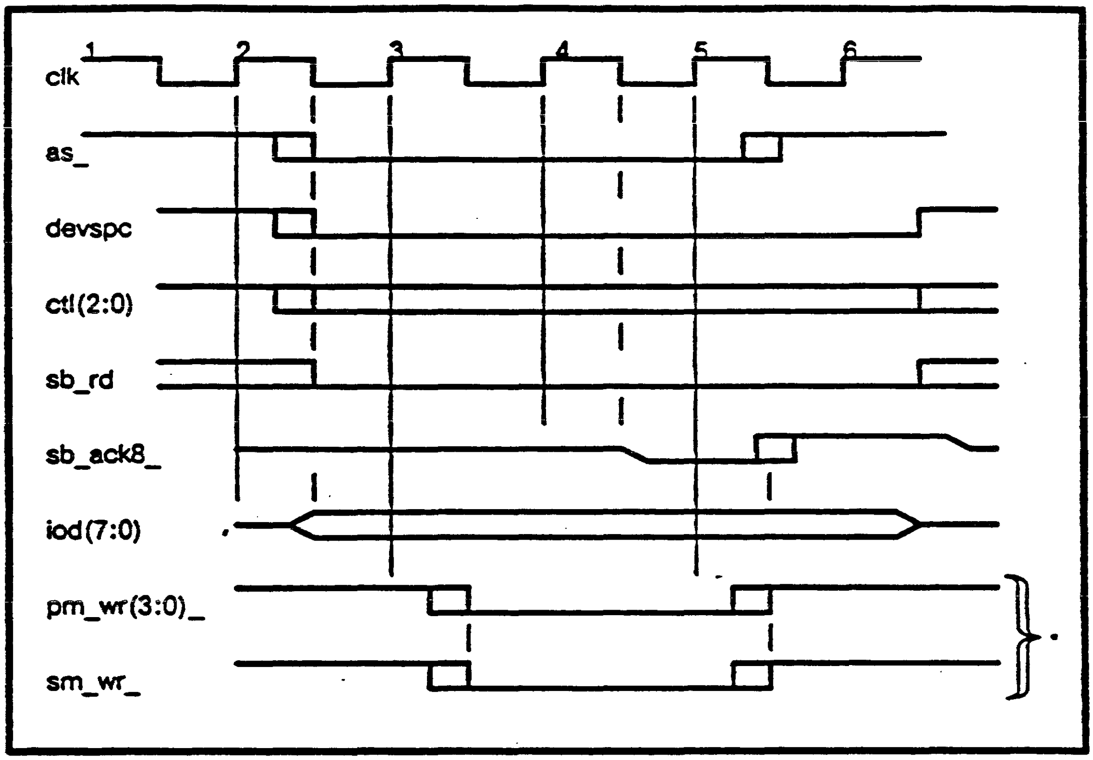
Control Space

CTL(2:0) Encoding (Control Space Address Map)

ctl(2:0)	Device
0	Device on S4–Cache Chip
1	Reserved for VME IACK
2	Context Register *
3	Diagnostic Register (unused)
4	Serial Controller Chip (MMU Bypass)
5	Segment Map
6	Page Map
7	EPROM (Boot Cycle, Supv. Instr. Fetch)

* – Context reg access requires A0 low.

In Device Space (DEVSPC_ low) the ctl(0) input is used as an invalidation input for any cycle from the cache chip. It is used when the cache chip determines an illegal virtual address (a(31:28) not all ones or all zeroes) which the MMU cannot detect, to inhibit though the PMEG(7:0) bus. The following diagram shows a Segment Map write cycle:



* Note that only one of these signals is asserted at a time.

Page Map

The page map is the second level of the two-level MMU, and contains 8k or 16k page map entries each mapping an 4 Kbyte page. It is indexed by the 7/8-bit PMEG provided by the segment map concatenated with virtual address bits SB_A(17:12). The page map bit definition is as follows:

Bit	Type	Description
31	V	valid bit, implies read access
30	W	write allowed protection bit
29	S	supervisor only protection bit
28	X	don't cache bit
27:26	MMU_TYP(1:0)	0 => main memory
		1 => input/output space
		2,3 => reserved for VMEbus
25	A	accessed (statistic bit)
24	M	modified (statistic bit)
23:16	none	reserved
15:0	page	physical page number

Device Space Address Map

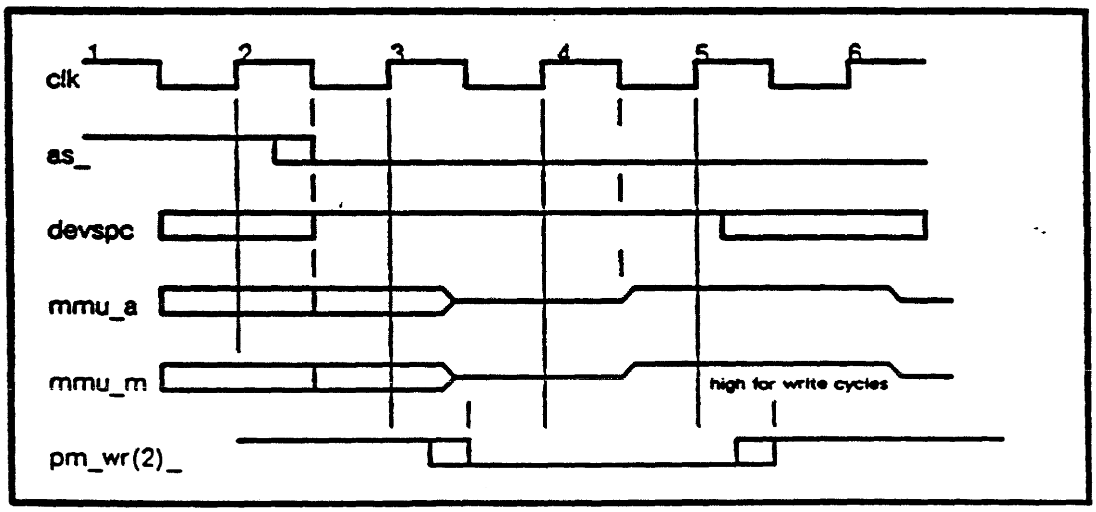
mmu_typ(1:0)	pa	device
0	[28:26] 0 X X	RAMSEL (main RAM)
1	[31:20] * F 0 X	Keyboard/Mouse
	F 1 X	Serial Controller Chip
	F 2 X	TOD Clk, NVRAM
	F 3 X	Counter Registers
	F 4 X	Parity Ctrl/Aux
	F 5 X	Interrupt Register
	F 6 X	EPROM
	F 7 2	Floppy Controller
	F 7 3	Audio DAC
	F 7 4	Aux Out Register
	F 8 X	SchoolBus Onboard
	F 9 X	
	F A X	Video Onboard
	F B X	
	F C X	SchoolBus Slot 1
	F D X	
	F E X	SchoolBus Slot 2
	F F X	
2	all	Unused
3	all	Unused

Video Onboard

* PA[31:28] are not actually decoded, but assumed to be 1's on Type 1 accesses and 0's on Type 0 accesses.

Main RAM— Statistics Update Cycles

The operating system requires certain information about the read/write history of each page mapped into main memory. The S4-MMU chip maintains this information in the MMU_A and MMU_M bits, automatically updating them on any reads or writes of main memory. A statistics update cycle is shown below:



Because the PM_WR_ signals will be asserted in Cycle 3 and negated in Cycle 5, addresses *must* remain stable to the MMU RAMs throughout Cycle 5; the earliest they may change is Cycle 6. Statistics bits are tri-stated in Cycle 6 No data collision occurs because the addresses do not change; we are reading the data we wrote.

Interrupt Register

The Interrupt Register provides for software generation of interrupts and allows the CPU to disable all interrupts or only certain ones. It is cleared on S8_RESET_, and has the following fields:

31	30	29	28	27	26	25	24
Enable Level 14 Interrupts	Reserved	Enable Level 10 Interrupts	Enable Level 8 Interrupts	Software Interrupt Level 6	Software Interrupt Level 4	Software Interrupt Level 1	Enable Interrupts Clears Level 15 when 0

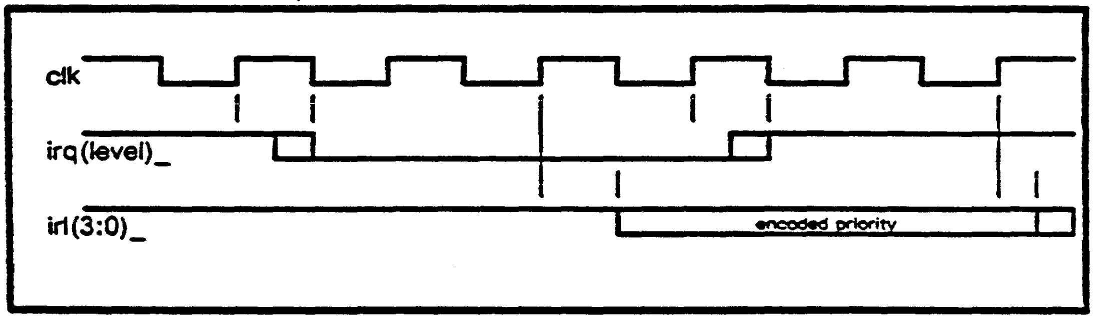
All IRQ(13:1)_ signals may be asynchronous to the system clock.

Software interrupts may be generated on levels 6, 4, and 1 by writing a 1 into bits 27, 26, or 25 when interrupts are enabled (bit 24 high).

Level 15 Interrupt requests are captured on a clock edge and held asserted to the CPU until bit 0 of the Interrupt Register is cleared.

Note that writing a zero to the Enable bits in the Interrupt Register only masks out that level's interrupt *It does not clear the source* (with the exception of Level 15 requests). This is different from the Sun-4 Architecture, in that the periodic interrupts at Levels 10 and 14 must be cleared by accessing their respective Limit registers.

Level-Sensitive Interrupts:



Interrupting Devices (assumed system configuration):

Int Level	Device
15	Buffered Write Timeout Error
14	Clock Interrupt 14 from Counter 1
13	Bus IRQ13
12	Keyboard/Mouse Serial Ports
11	Bus IRQ11 Floppy
10	Clock Interrupt 10 from Counter 0
9	Bus IRQ9
8	Video
7	Bus IRQ7
6	SWIRQ6 Ethernet
5	Bus IRQ5
4	SWIRQ SCSI DMA
3	Bus IRQ3
2	Unused
1	SWIRQ1 Bus IRQ1

EPROM

Both Counters are separately writeable for testing purposes. They should not be written in normal operation. Because of the 8-bit interface unpredictable carrys could occur.

Auxiliary Output Registers

An additional read/write strobe has been added for a set of Auxiliary Output Registers located in Type 1 Device space beginning at F7400000.

DAC Write and Transfer strobes

The DAC_WR_ and DAC_XFER_ signals are somewhat overloaded. In the power-up mode, they are used to access an external double-buffered DAC. The DAC_WR_ signal is asserted when the cpu attempts to write to the audio DAC address range. It is a slow device, inserting 7 waitstates, like the SCC's. The DAC_XFER_ signal is asserted when counter 1 hits its limit register value, transferring the holding register data into the DAC internal register. It is asserted for 6 clocks or until the interrupt source (Limit 1) is removed, whichever comes first.

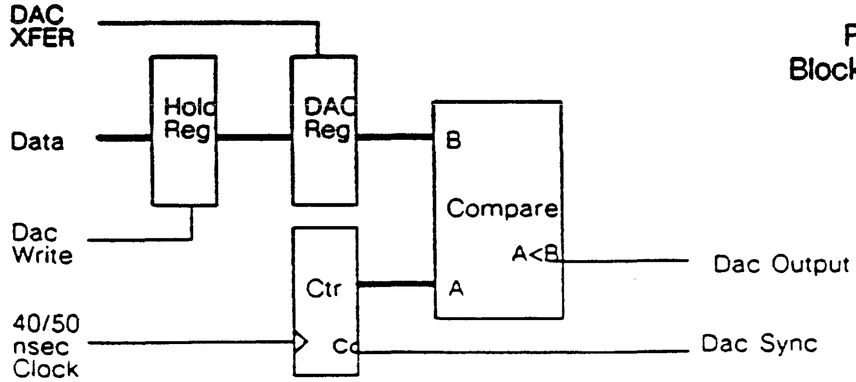
When the Internal DAC is enabled (see below) the DAC_WR_ pin becomes the DAC2 output. The DAC_XFER_ pin becomes the PWM output, varying in duty-cycle between 0-511 CLKs out of 512.

In addition, the DAC_WR_ signal is asserted for both reads and writes at location 0xF7FXXXXX. This is used as an S4-VME chip select signal.

Internal PWM DAC

Two 8-bit Pulse-Width Modulation DACs are implemented, operating off of the 40/50 nSec CLK input. When enabled, this DAC outputs replace the DAC_WR_ and DAC_XFER_ output pins. It responds to the same address space as the external DAC, only faster: Type 1 Device Space, $F7300000.

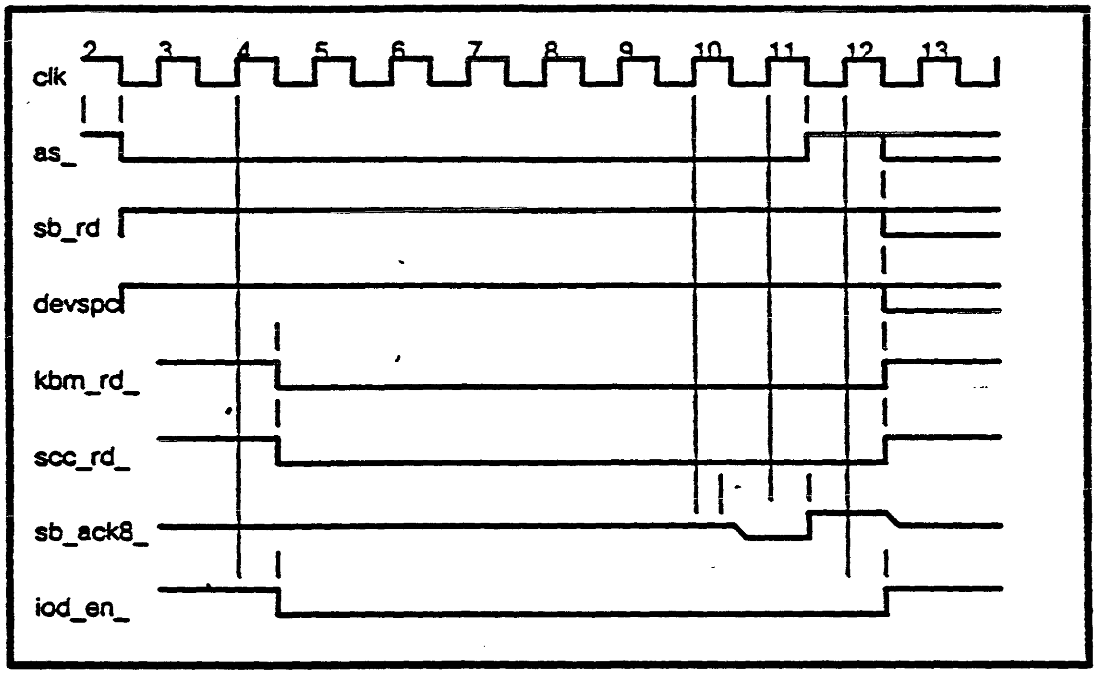
The output of the PWM DAC is a square wave with a duty cycle between 0 and just under 100%. When the DAC data register is programmed with 0's, the output is never high. When it is programmed with $0080 (least-significant bit of 9-bit DAC set), the output is high for one clock every 512. When it is programmed with $FF80 the output is high 511 out of every 512 clocks.



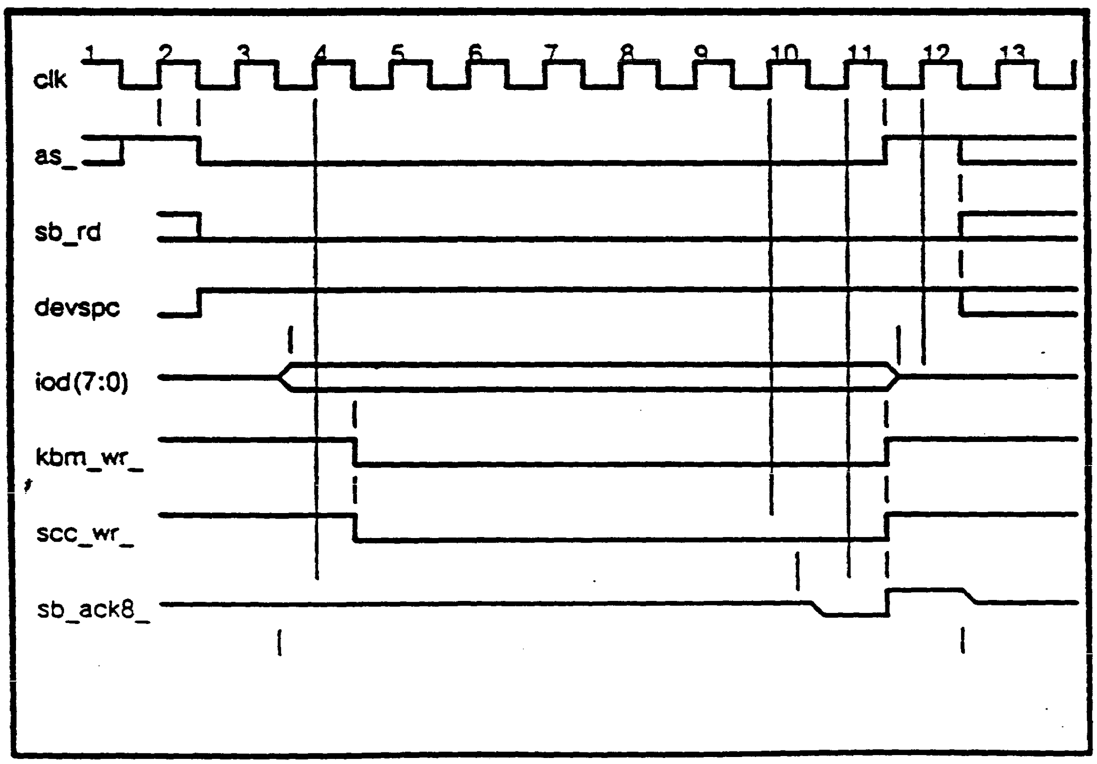
PWM DAC
Block Diagram

Functional Timing Diagrams

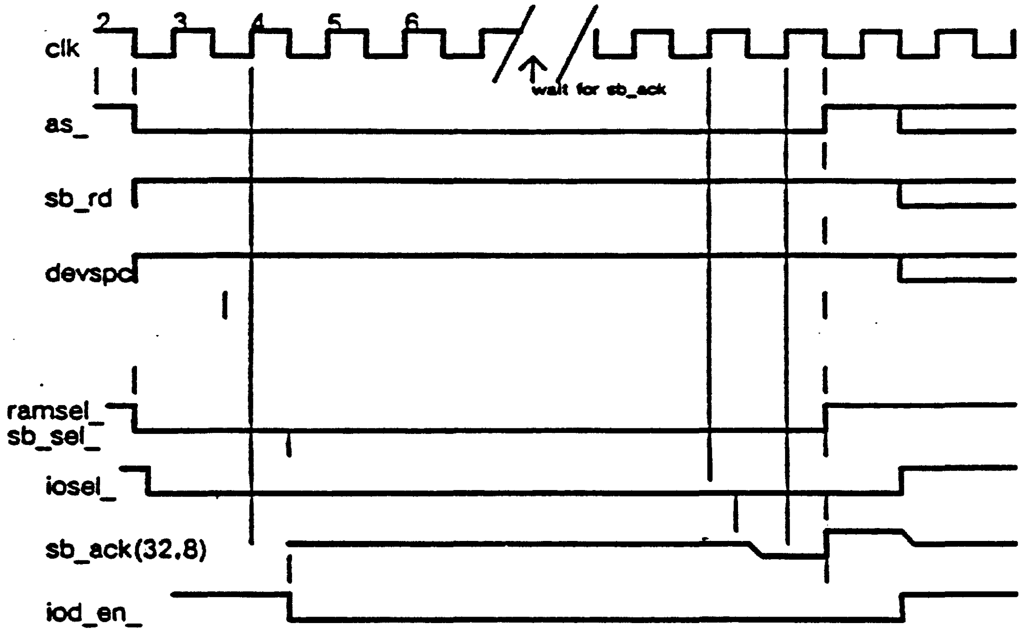
Keyboard/Mouse or SCC Read



Keyboard/Mouse, or SCC Write

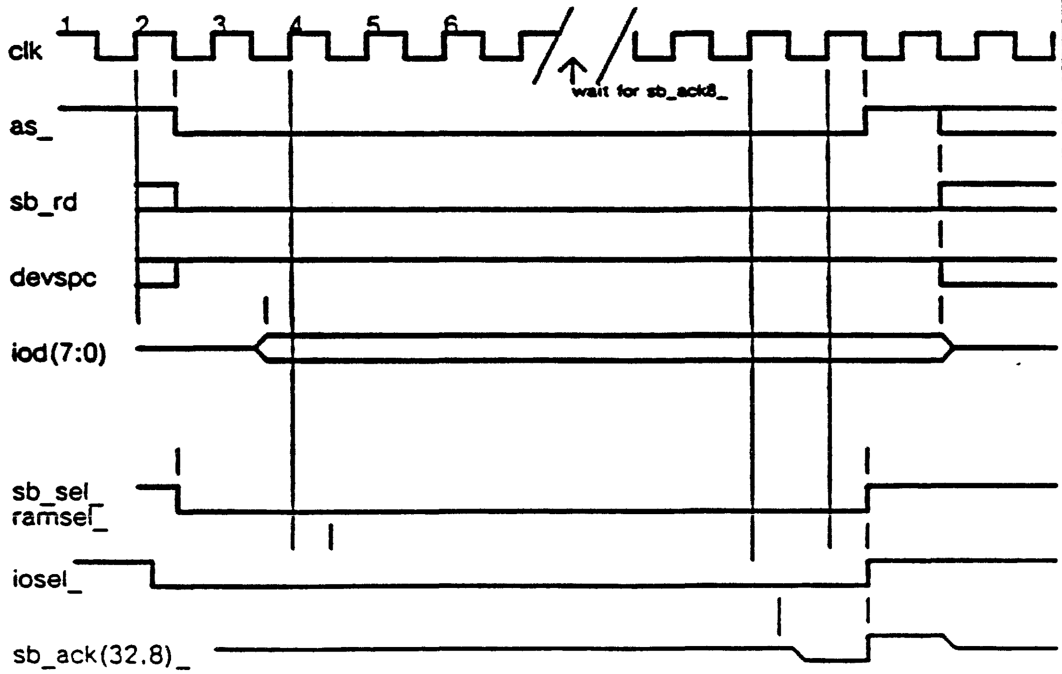


SBus, RAM, or IOSEL Read



iosel_, ramsel_ and sb_sel_ are NOT referenced to any clk edge.
Note: SB_ACK8_ is generated by these slave devices, not by the S4-MMU chip.

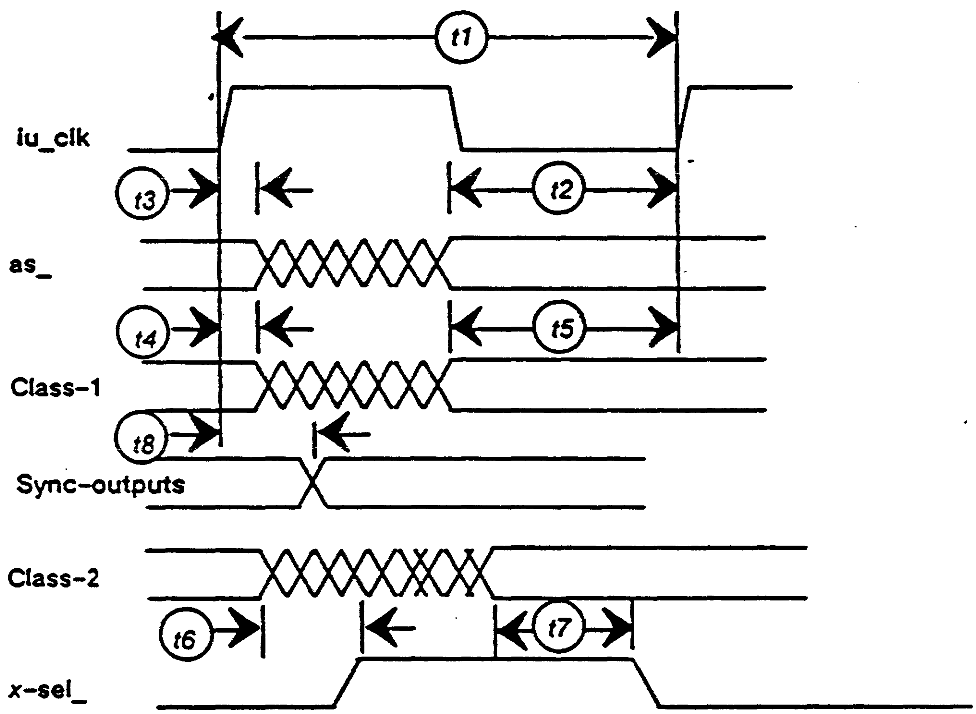
SBus, RAM, or IOSEL Write



Note that sb_ack[8,32]_ or sb_err_ sampled by the mmu terminates the select.
Note: SB_ACK8_ is generated by these slave devices, not by the S4-MMU chip.

Timing Specifications and Diagrams

Tn	Description	min	max
t1	iu_clk cycle time	40	
t2	Setup time, as_ signals before clk	3	
t3	Hold time, as_ signals after clk	15	
t4	Hold time, Class-1 signals after clk	0	
t5	Setup time, Class-1 signals before clk	15	
t6	Delay Class-2 to *x*-sel_ negated		22
t7	Delay Class-2 to *x*-sel_ asserted		23
t8	Synchronous output delay		22



Class-1 signals are: io_a[3:0], ctl[2:0], devspc_, sb_rd, user_, pmeg[7:0] (in), pa[27:12] (in), mmu_[vwsxam] (in), mmu_typ[1:0] (in). *These signals are used synchronously in this case.*

Class-2 signals are: pa[27:12] (in), mmu_[vwsxam] (in), mmu_typ[1:0] (in), ctl[1:0], devspc_, sb_rd, user_. *These signals are used asynchronously in this case, affecting outputs sb_sel[3:0]_ and ramsel_.*

t1	clk cycle	40	—	ns	
t2	as_ setup to clk	15			
t3	as_ hold from clk	2			1

Notes:

1. This timinig specification does not meet the ideal requirements for 25 MHz system operation.

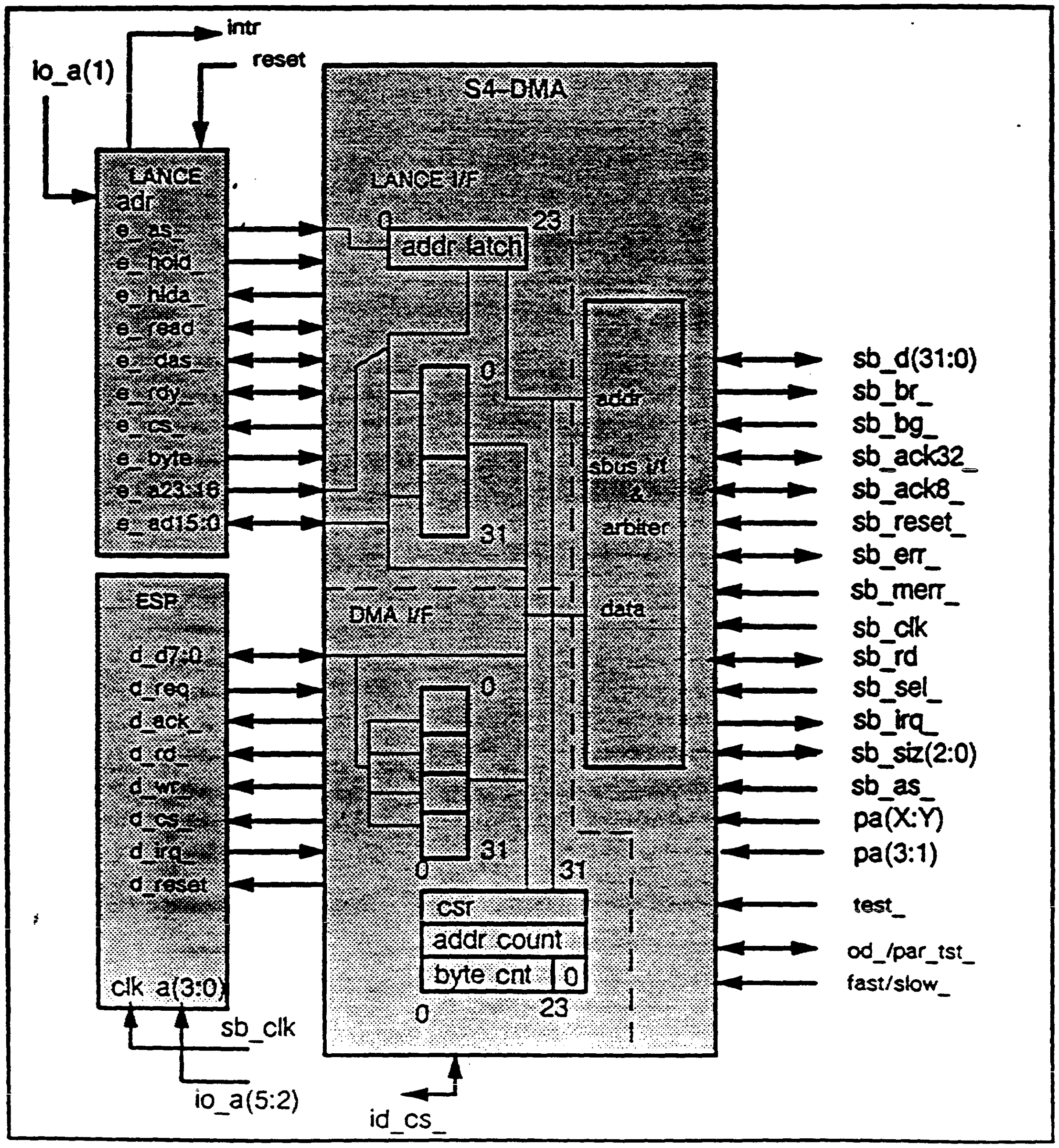
- NOTE: IO_DEN_ is asserted only on READS. It is assumed that all write cycles drive the iod bus.

Change History

12/15 tw	Config register is gone.
	Counter/Timer is 30 bits. Added Interrupt_Occurred bit.
	Diag register and bit added.
12/17 tw	Added sb_ack32_, made sb_ack8 and sb_err BD4's.
	Statistics updates tristate in Cycle 6.
12/18 tw	Modified Counter/Timer to freerun on reset.
	Moved DACWR, Ctr, Limit, Floppy to E01–4.
12/18 tw	Two Counter/Limit register sets, dedicated at Int levels 10 and 14.
	Deleted IRQ inputs 10 and 14.
	Deleted PARA output, multiplexed with od_ input in test mode.
	Diag is now a BT8.
	Added one more SB_SEL_ signal, deleted vctl_cs and vramsel.
	Deleted DMA Starvation timeout, deleted SB_BG pins.
	Added A2 and 3, gathered Counters and Limit registers in one page.
12/21 tw	DAC_WR Gone. It's now in the Video chip.
	Counter starts at 1.
12/22 tw	ramsel, vramsel (sbsel1) are now combinatorial.
	All inputs are ttl levels.
12/29 tw	PAR_EN_ signal removed. S4-Buffer will use RAMSEL_ instead.
1/5/88 tw	RAMSEL is now *all* of Type1 Device Space.
1/14/88 tw	DIAG changed to AUX_WR_. IOSEL changed slightly.
1/21 tw	Added Limit bit to Counter, Moved Counter to EF. Moved SB Slots to Type 0 Space.
1/27 tw	TOD is now just a CS_. Added DAC_XFER to allow for double-buffered DAC.
1/29 tw	SB_SELn_ are now all asynchronous.
2/8 tw	Removed SB_ACK32_ and SB_ERR_.
2/23 tw	Added DMA_ pin and description.
2/29 tw	Fixed mmu ram write pulse in Pg 4 diagram.
3/8 tw	Added internal pwm dac. IODEN_ documented.
3/15 tw	4k pages. Changed memory map.
4/7 tw	iosel_ asynchronous. VME select address removed due to lack of use.
4/18 tw	Level 15 interrupts captured and held. Cleared by turning off all interrupts.
4/19 tw	Changed Device Address Map to remove reference to onboard video.
4/26 tw	Video is back. Ignore previous change.

Features

* Single chip interface between Ethernet (LANCE), SCSI (ESP) and Sbus
* Handles 32 bit packing and unpacking
* Generic support for 8 bit peripherals
* Supports externally programmable Sbus ID
* Low cost 120PFP package



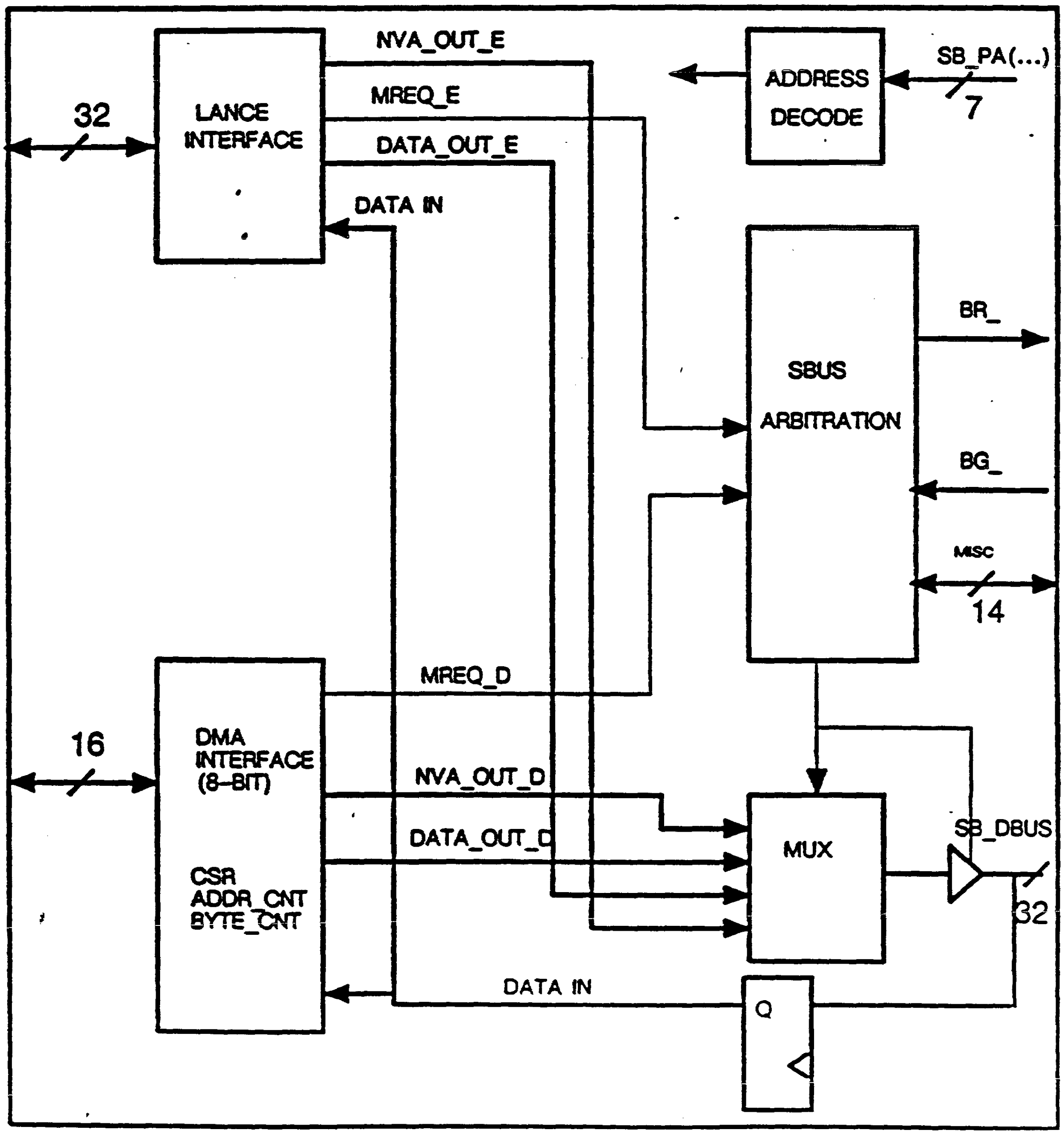
1.0 Pin Description

Name	Type	Description
Bus Interface	**51**	
sb_d(31:0)	BD4TU	Sbus Data Bus
sb_br_	BT4	Sbus Bus Request
sb_bg_	TLCHTU	Sbus Bus Grant
sb_ack32_	BD4TNU	Sbus 32bit Acknowledge
sb_ack8_	BD4TNU	Sbus 8bit Acknowledge
sb_reset_	TLCHTU	Sbus Reset
sb_err_	BD4TNU	Sbus Error
sb_merr_	TLCHTU	Sbus Memory Error (INT15)
sb_clk	DRVC16	Sbus Clock input
sb_rd	BD4TU	Sbus Read/Write_
sb_sel_	TLCHTU	Sbus Select
sb_irq_	BD4TOD	Interrupt Request (open-drain)
sb_siz(2:0)	BD4TU	Sbus transfer Size
sb_as_	TLCHTU	Address strobe (addr is valid)
pa(X:Y)	TLCHTU	Physical Address lines (for slave decodes)
pa(3:1)	TLCHTU	Physical Address bits
Ethernet Interface	**32**	
e_as_	TLCHTD	Ethernet Address Strobe
e_hold_	TLCHTU	Ethernet Hold
e_hlda_	BT4	Ethernet Hold Acknowledge
e_read	BD4TU	Ethernet Read
e_das_	BD4TU	Ethernet Data Strobe
e_rdy_	BD4TU	Ethernet Ready
e_cs_	BT4	Ethernet Chip Select
e_byte	TLCHTU	Ethernet Byte marker
e_a23:16	TLCHTD	Ethernet High Order Address
e_ad15:0	BD4TU	Ethernet Address / Data Bus
DMA Interface	**16**	
d_d7:0	BD4TD	DMA Data Bus
d_req	TLCHT	DMA Request
d_ack_	BT4	DMA Acknowledge
d_rd_	BT4	DMA Read Strobe. (reg read or dma to memory).
d_wr_	BT4	DMA Write Strobe. (reg write or dma from memory).
d_cs_	BT4	DMA Chip Select for slave register access.
d_irq_	TLCHTU	DMA Interrupt Request
d_reset	BT4	DMA Reset

1.1 BLOCK DIAGRAM

The S4–DMA gatearray provides three independent functions;

1. Sbus Identification
2. Ethernet Interface to the Sbus
3. Sbus DMA Channel



During Slave Cycles the S4–DMA takes control of the sb_err, sb_ack8_ and sb_ack32_, signals. The combination of responses are as follows;

sb_ack8_	sb_ack32_	sb_err_	Definition	
1	1	1	insert wait states	**
1	1	0	Error	
1	0	1	32–bit port ack	**
1	0	0	Error	
0	1	0	Rerun	**
0	0	1	16–bit port ack	
0	1	1	8–bit port ack	
0	0	0	Reserved	

This table represents all possible SBus responses. The S4–DMA gate–array can, however, only generate those responses marked with a **.

3.0 Sbus Identification

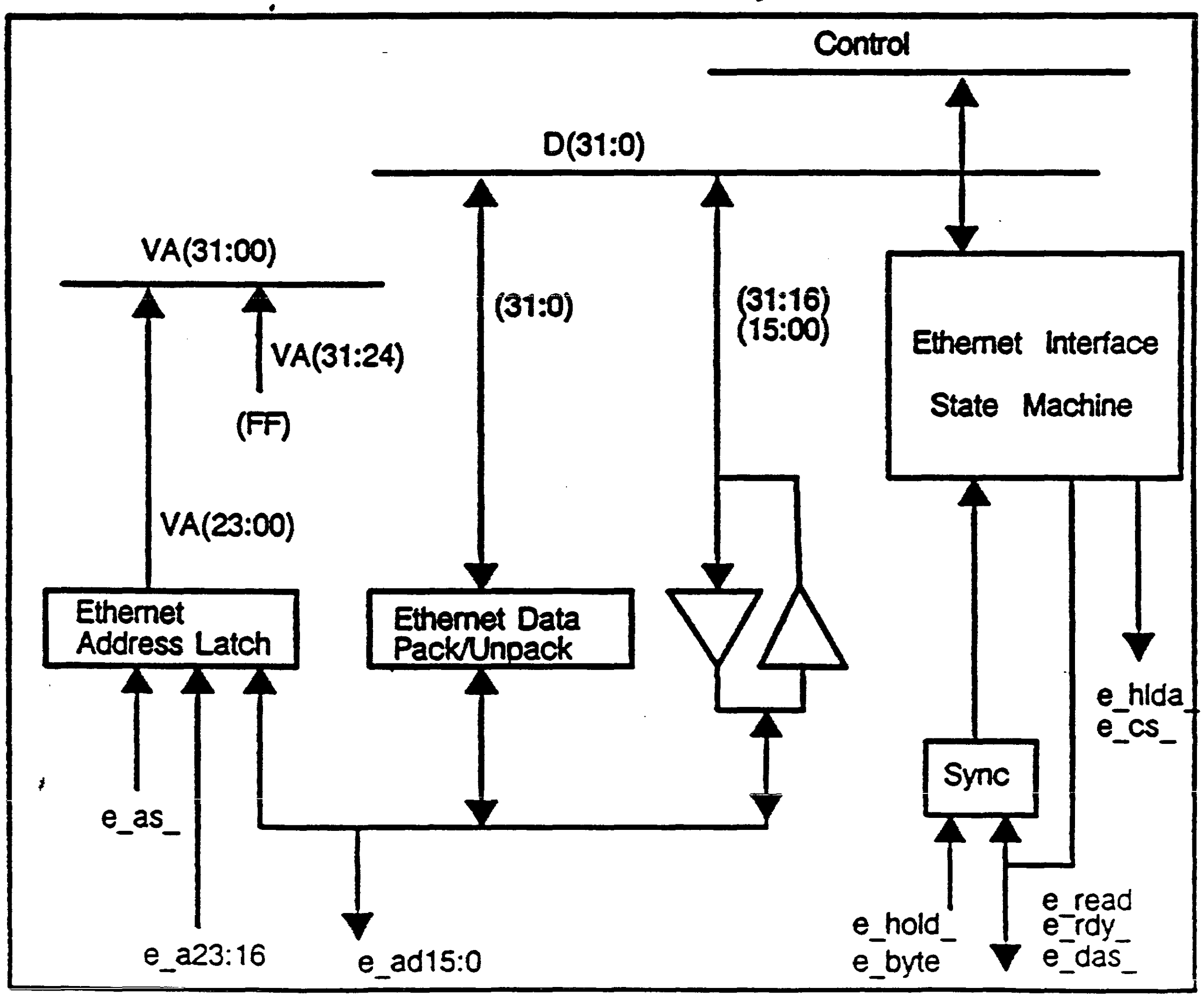
This is a mechanism which allows software to uniquely identify each Sbus device, since each device can have a unique ID.

Unique ID's will be provided by Sun. The onboard id is hardwired to the 32–bit value fe810101. This value will be returned when the ID field is accessed by the IU (and the –id_cs_ pin is tied low). If the id_cs_ pin is pulled high then access to the ID field will cause an external access using the id_cs_ pin as a external chip select. Refer to S4 Software Architecture Specification for further details.

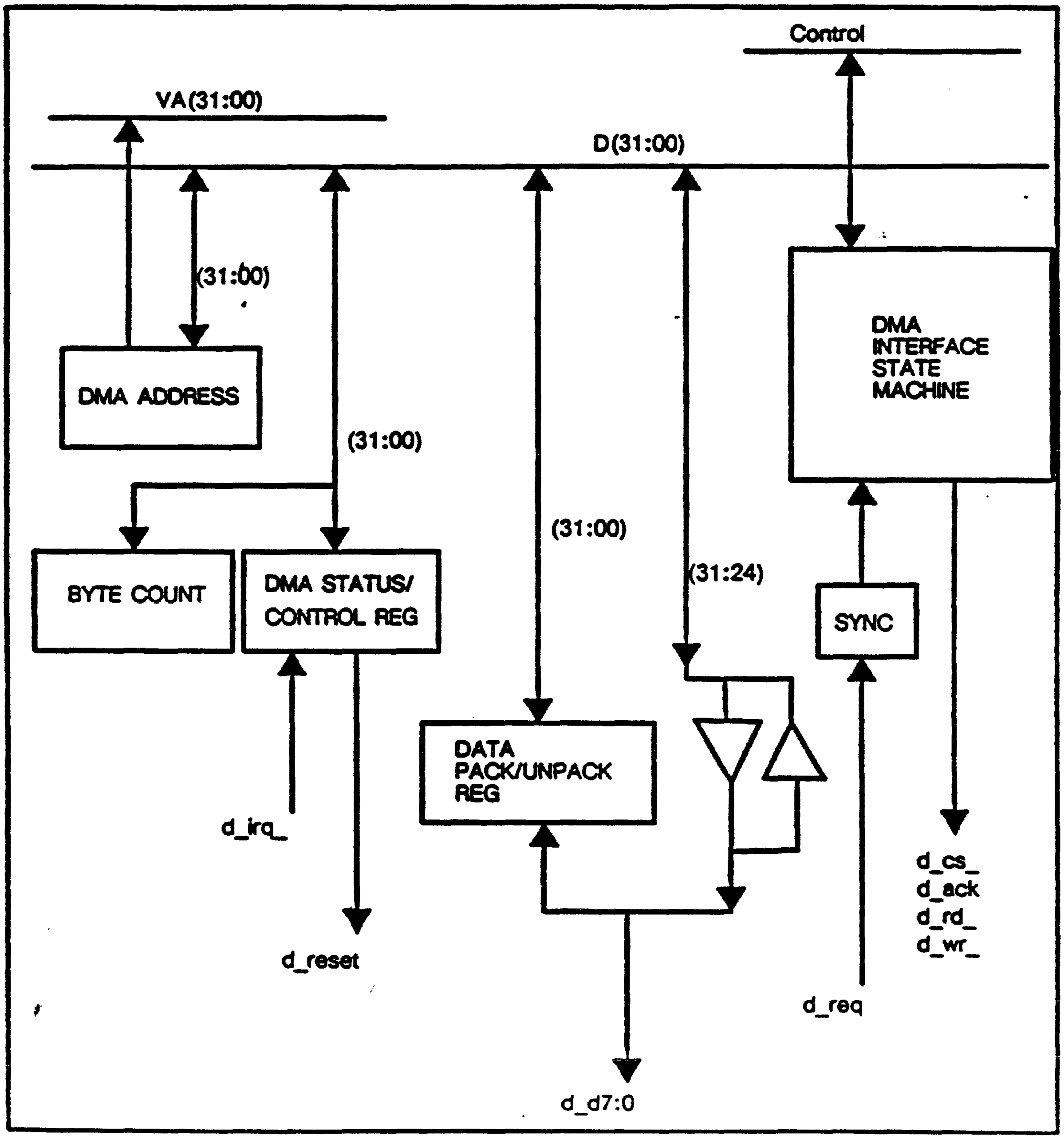
sb_sel_ & sb_as_	pa(X:Y)	io_a(1)	Register accessed	Size	Type
0	11	0	Register Data Port (RDP)	16–bit	R/W
0	11	1	Register Address Port (RAP)	16–bit	R/W

Once the S4–DMA has granted access of it's local bus to the LANCE, the CPU cannot access the LANCE until the pending cycles are completed. In order to remove the potential deadlock condition which results, the S4–DMA will cause a rerun according to the table on page 7.

4.1 Ethernet Interface Block Diagram



5.1 DMA Interface Block Diagram



5.3 DMA Control/Status Register Assignments *(DMA_CSR)*

Bit	Mnemonic	Description	Type
0	INT_PEND	Set when d_irq_ or TC asserted. Reset when not	R
1	ERR_PEND	Set when mem. exc occurred DMA stopped Reset on FLUSH command	R
3:2	PACK_CNT	Number of bytes in Pack Register	R
4	INT_EN	When set enables d_irq_ state onto sb_irq_	R/W
5	FLUSH	When set causes PACK_CNT, ERR_PEND and TC to be reset. Reads as 0	W
6	DRAIN	When set causes remaining pack register bits to be drained to memory. PACK_CNT = 00 Clears itself	R/W
7	RESET *	When set acts as a hardware reset.	R/W
8	WRITE	DMA direction; 1= to memory 0 = from memory	R/W
9	EN_DMA	When set allows the device to respond to DMA device requests	R/W
10	REQ_PEND	When set the DMA i/f is active. DO NOT assert RESET or FLUSH	R
12:11	BYTE_ADDR	Next byte number to be accessed.	R
13	EN_CNT	When set enables the internal byte counter. (not used with the ESP SCSI chip)	R/W
14	TC * *	Terminal Count. Byte counter has expired	R
15	ILACC* * *	When set this bit instructs the ethernet interface to act slightly differently — see note below	R/W
27:14	———	Reserved (all unused bits to read as 0)	R
31:28	DEV_ID	Device ID (for this implementation = 1000)	R

* RESET

POWER_ON RESET or RESET from bit 7 will leave the device in the following state:
ERR_PEND = PACK_CNT = INT_EN = FLUSH = DRAIN = WRITE = EN_DMA = REQ_PEND = EN_CNT = TC = 0, RESET = 1, and BYTE_ADDR = 00. All interface state-machines will revert to their idle states

5.6 Programming Notes

The address counter always points at the next memory location to be accessed. When the direction of transfer is to memory the counter is incremented by the size of the write (1 or 4) upon completion of the transfer. When the direction of transfer is from memory the address is always incremented by 4, but the lower 2 bits are driven low such that all reads are word sized and word aligned. Byte alignment is done inside the gate–array.

There is a 2–bit byte counter BYTE_ADDR that always points to the next byte location that the DMA device will access. This counter is incremented by 1 each time a byte is transferred between the external device and the gate–array. Note the byte counter is controlled by the DMA interface whereas the address counter is controlled by the memory interface, hence the two may disagree. This byte counter is loaded at the same time the address is loaded and receives the two least significant bits of the address.

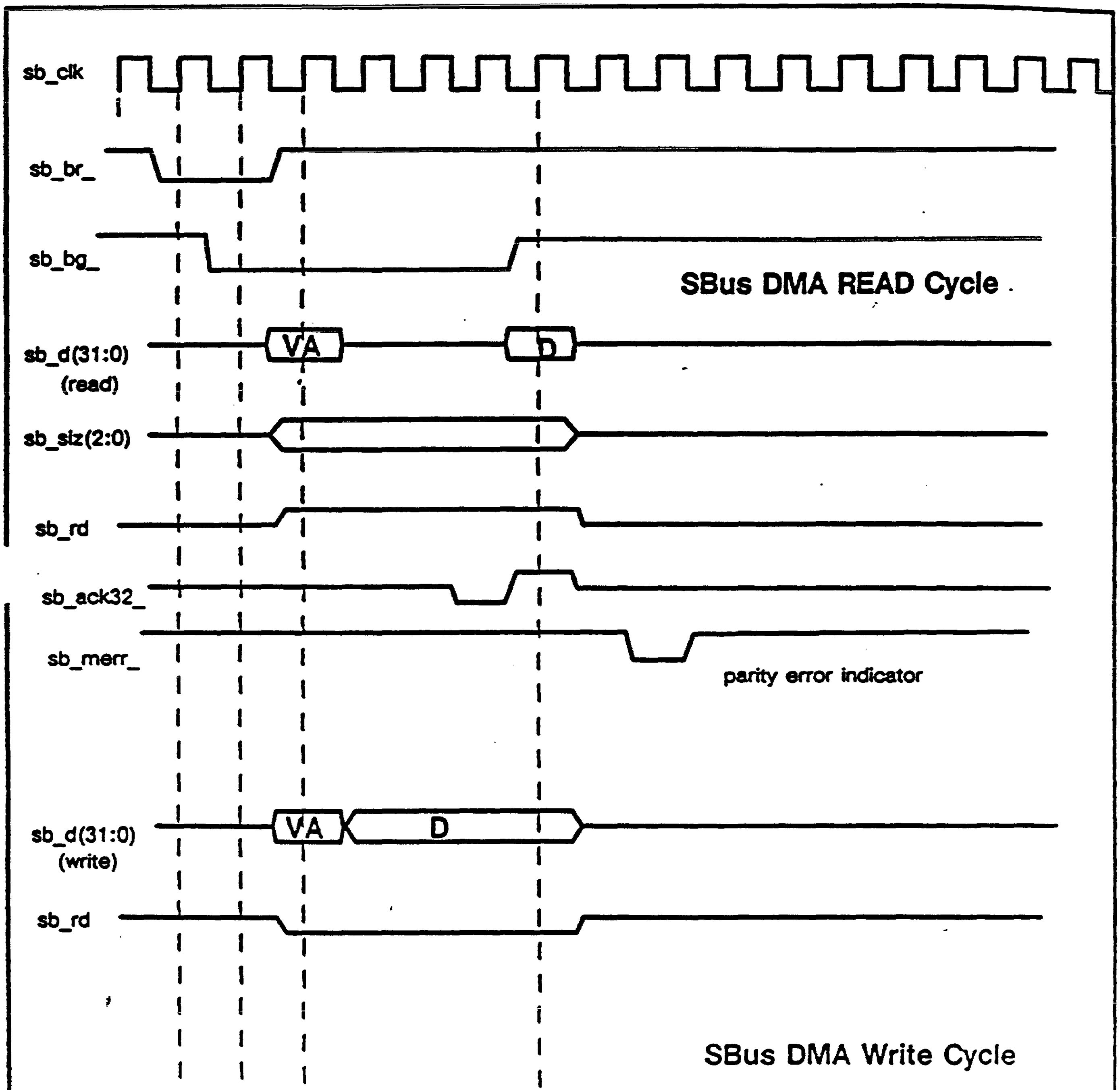
Another 2–bit counter PACK_CNT keeps track of how many bytes are stored in the internal PACK register. Note this pack count is only valid for transfers to memory. Whenever the PACK_CNT= 3 and another byte is accepted, a word write is scheduled with the memory interface. If a DMA transfer completes leaving a non–word fragment in the PACK register, then this counter is used by the hardware to determine how many bytes to write to memory when the DRAIN command is received. Both PACK_CNT and BYTE_ADDR can be read in the Control and Status Register (*DMA_CSR*).

If the driver desires to terminate a transfer, two control bits in the *DMA_CSR* can be used. The EN_DMA bit can be used to ignore new transfer requests from the DMA device when it is cleared. Memory accesses by the memory interface are unaffected by this bit. Th EN_DMA bit can be set or cleared at any time without affecting the state of a transfer currently in progress. The FLUSH bit is provided to clear the PACK_CNT if the driver wishes to clean up the state of a transfer, without draining the packed data to memory. It is also used to clear the ERR_PEND indicator, allowing an error condition, which subsequently halts the DMA interface state machine, to be cleared cleanly.

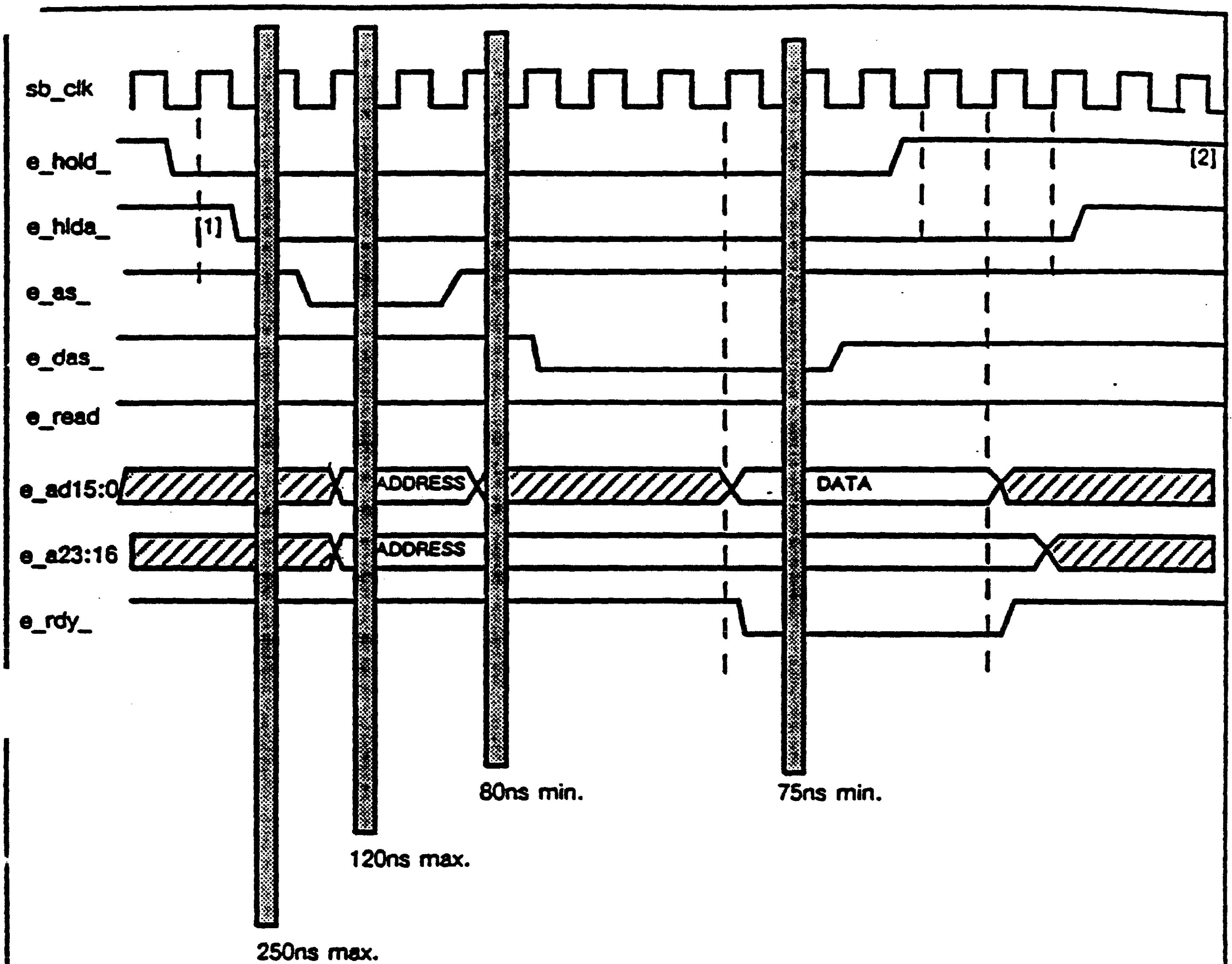
Th DRAIN bit will cause all packed data to be sent to memory. This is intended for use when a transfer completes and the data for transfer to memory does not fill the 32 bit word. It can also be used to leave a transfer in a clean state if a transfer is stopped via the EN_DMA bit, which may be restarted later. A DRAIN sequence will leave the address counter pointing to the byte address beyond the last byte or word written. *Hence the address counter must be reloaded before the next transfer to properly set the BYTE_ADDR.*

The *DMA_CSR* also contains a RESET bit which will generate an external reset signal and reset all DMA interface logic (state machines). It is vital the RESET and/or FLUSH bits are not set if any memory activity is still pending: a REQ_PEND bit is provided in the *DMA_CSR* to show the driver if the memory interface is active. If REQ_PEND is asserted the driver should poll it until it is deasserted. Similarly writing to the Address Counter, changing the WRITE bit in the *DMA_CSR*, or writing the Byte Counter

6.0 Timing Diagrams



For further possible SBus cycles see the SBus specification

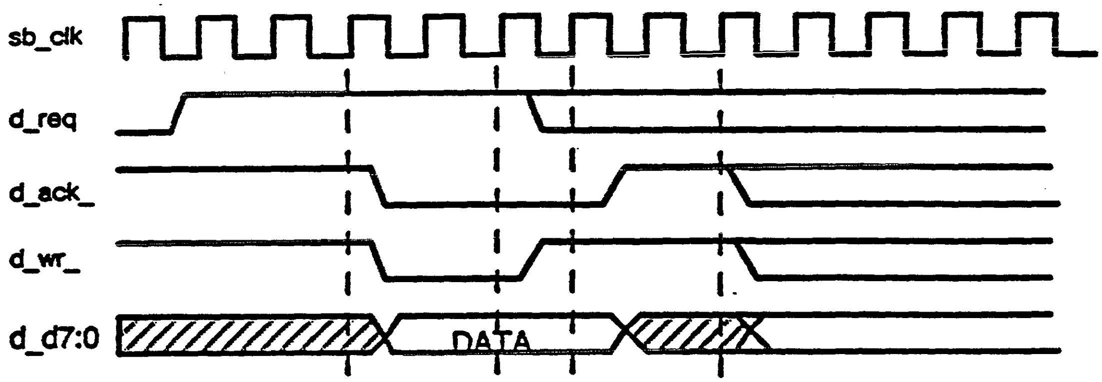


[1] e_hlda_ is only asserted when the interface is not busy

[2] e_hold_ will stay asserted for burst mode accesses e_hlda_ must follow it

Ethernet: LANCE DMA read cycle

(DATA available in PACK Reg)

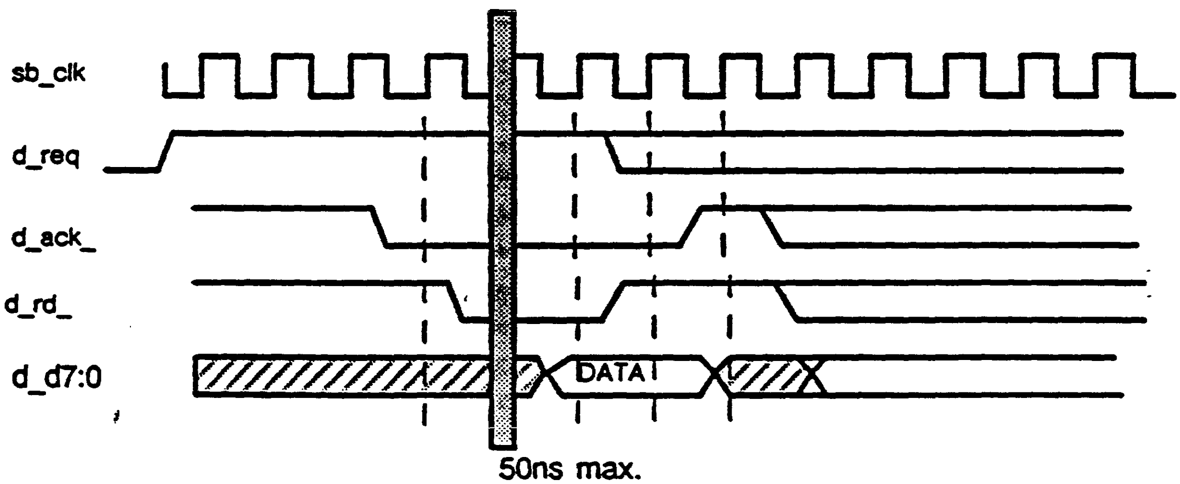


DMA Read Cycle [fast cycle]

(DATA available in UNPACK Register)

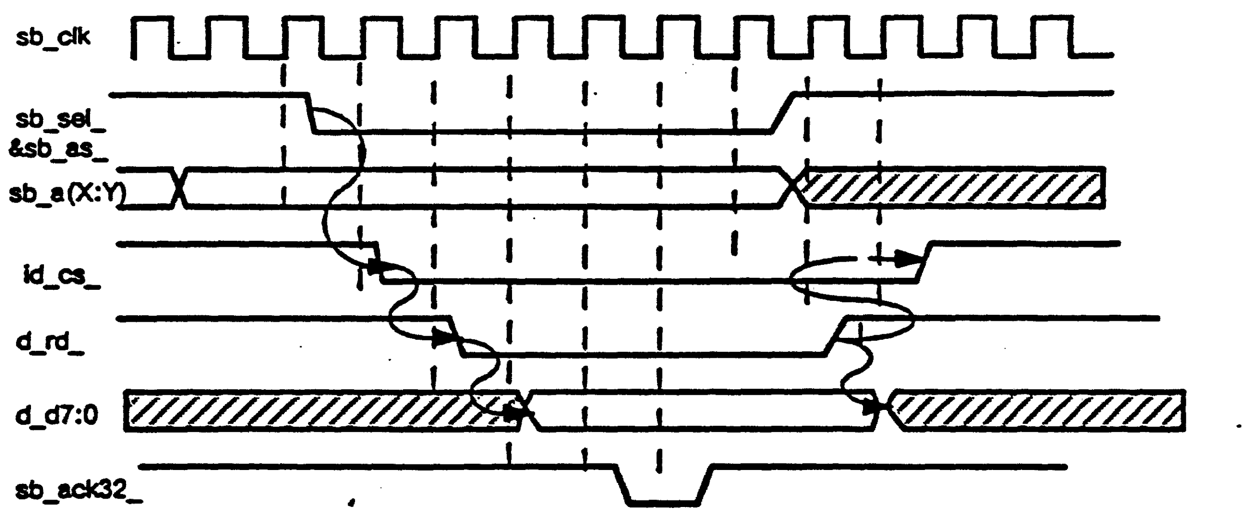
'READ' indicates transfer from memory to DMA device

When UNPACK reg is empty a memory read must occur which will subsequently lengthen this operation (see SBus READ Cycle)

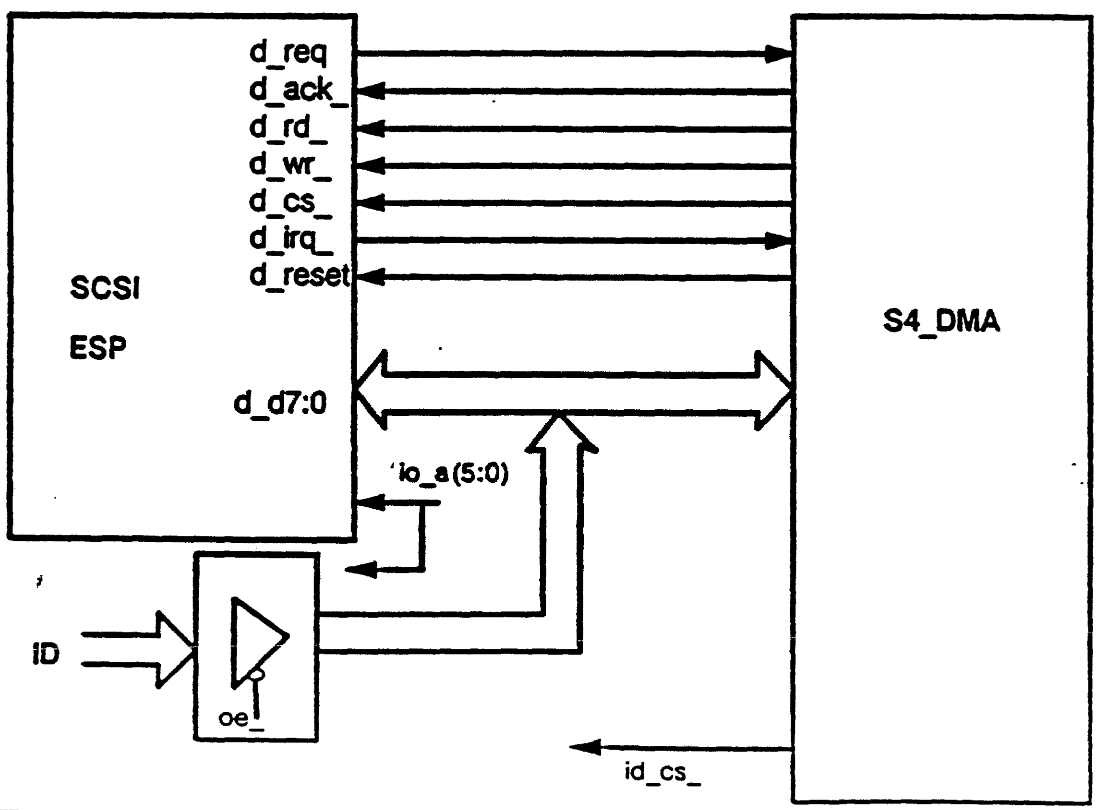


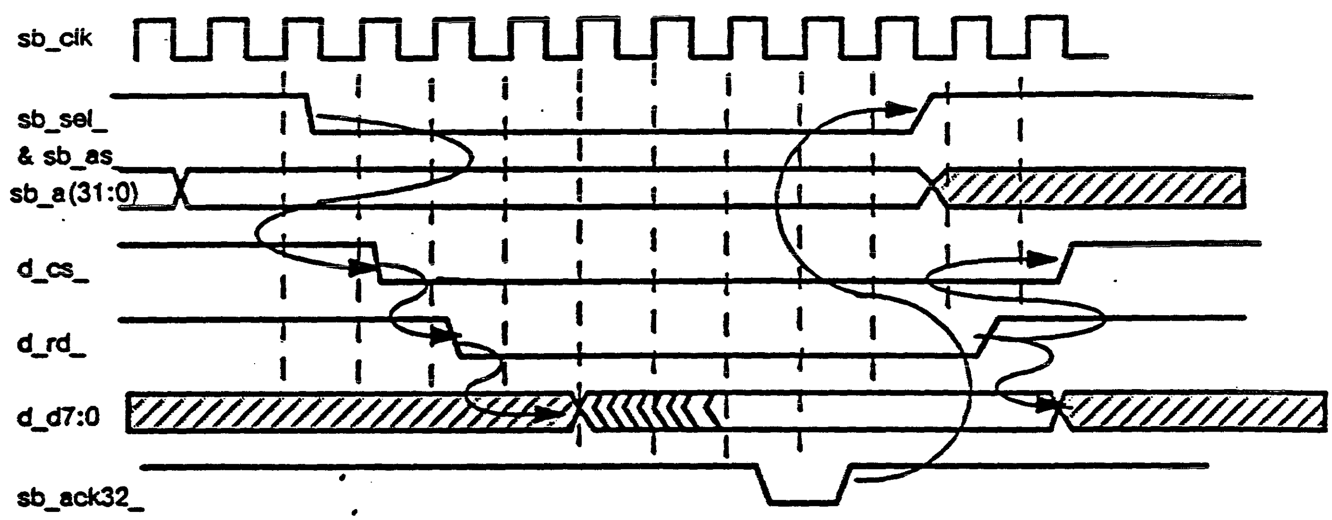
DMA Write Cycle [fast cycle]

'Write ' indicates transfer is from DMA device to memory

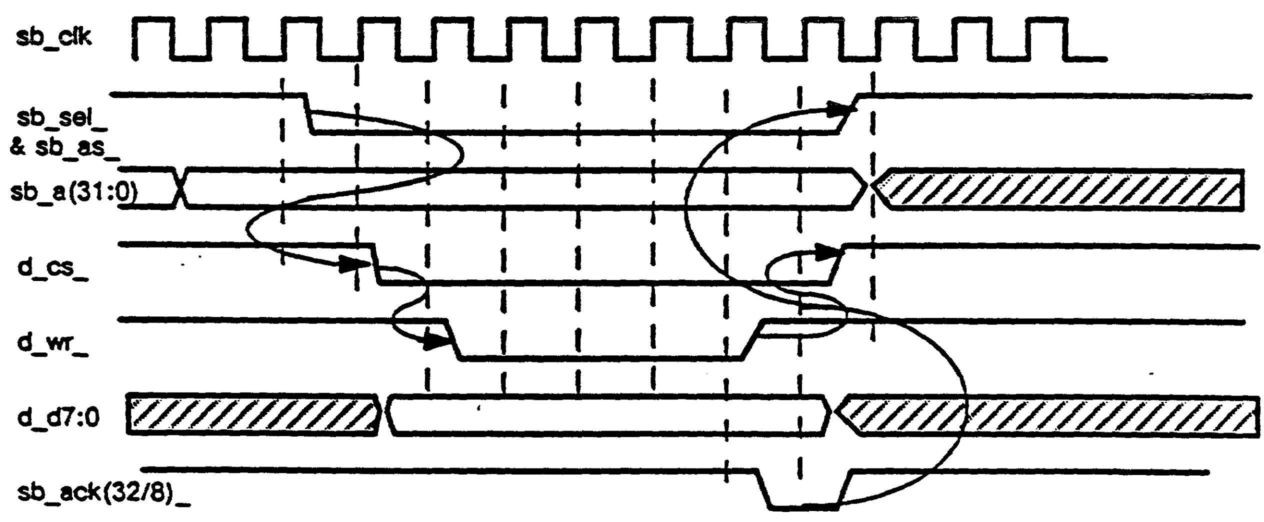


Offboard ID read cycle [fast cycle]





Extended DMA device register read cycle [slow cycle]



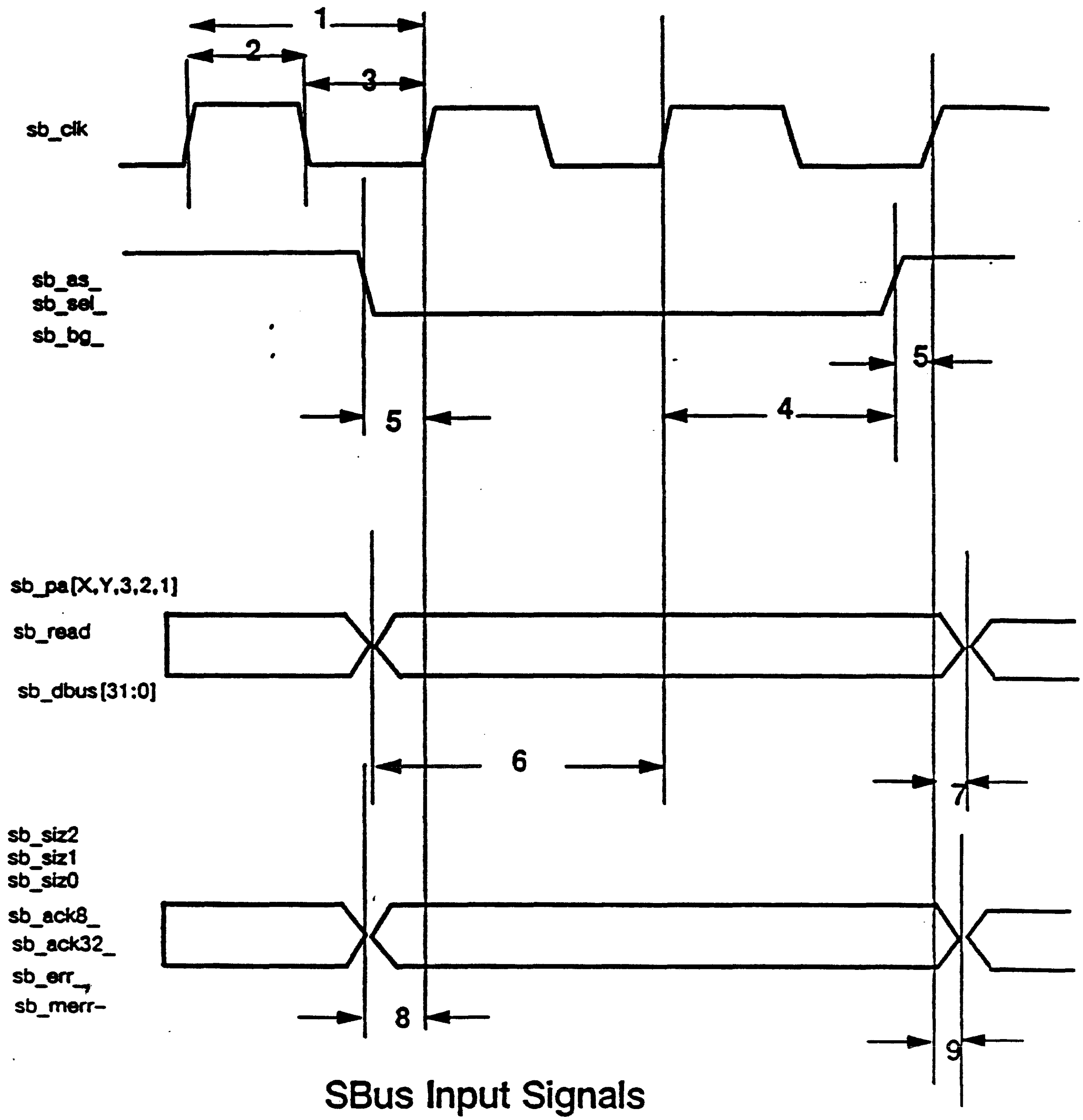
Extended DMA device register write cycle [slow cycle]

Switching Characteristics

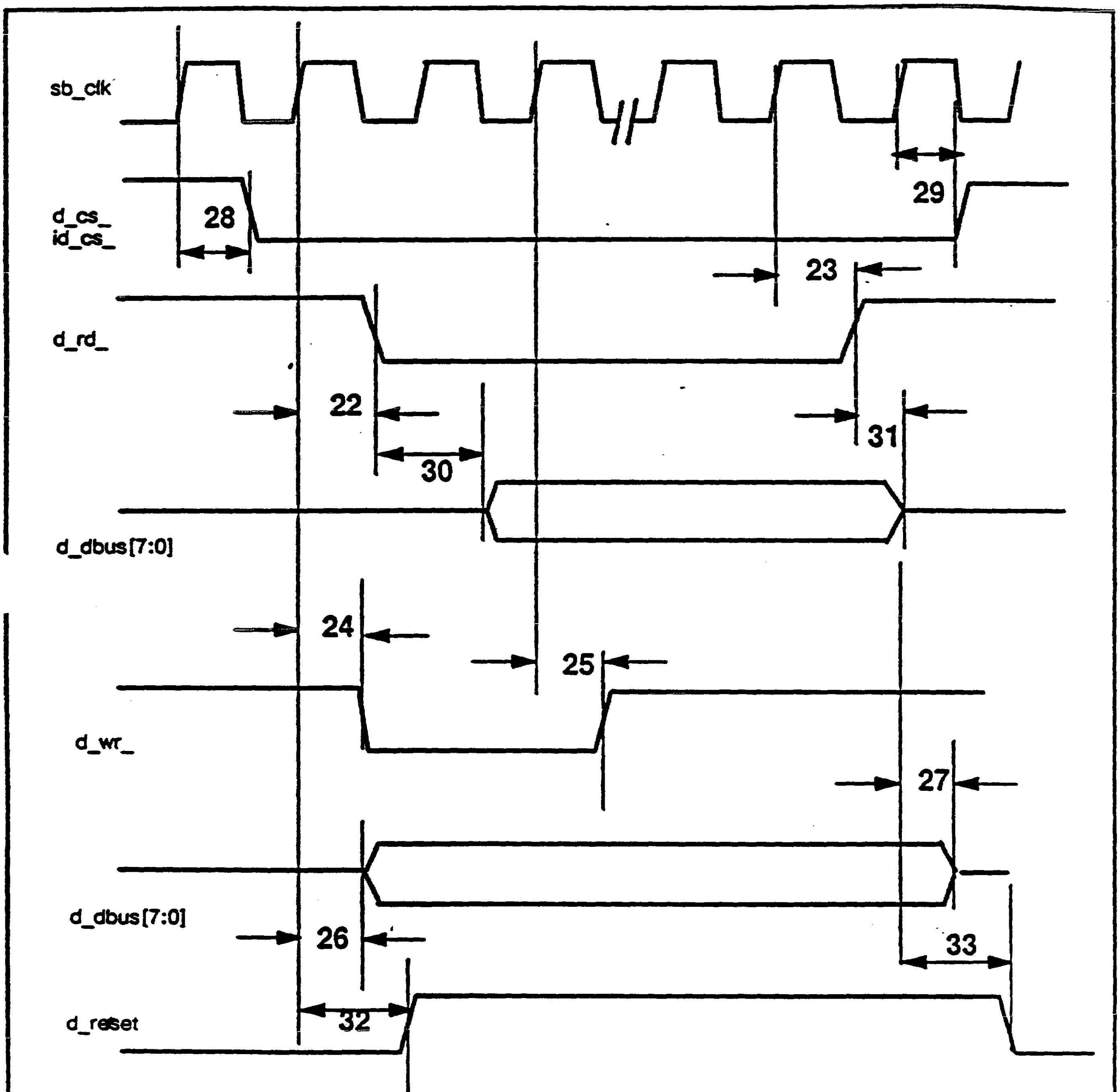
No.	SIGNAL	DESCRIPTION	CONDITIONS	min	max	units
1	CLK	clock period		30		ns
2		clock high				ns
3		clock low				ns
4	Note 1	hold wrt clk ^		0		ns
5	Note 1	setup to clk ^		14.0		ns
6	Note 1	setup to clk ^		23.0		ns
7	Note 1	hold wrt clk ^		5.0		ns
8	Note 1	setup to clk ^		13.5		ns
9	Note 1	hold wrt clk ^		0		ns
10	Note 1	clk ^ to output valid	Load = 100pf		30.4	ns
11	Note 1	clk ^ to output invalid	Load = 100pf		22.0	ns
12	Note 1	clk ^ to output valid	Load = 130pf		31.4	ns
13	Note 1	clk ^ to output invalid	Load = 130pf		19.7	ns
14	Note 1	clk ^ to output low	Load = 100pf		24	ns
15	Note 1	clk ^ to output high	Load = 100pf		18.5	ns
						ns
						ns

No.	SIGNAL	DESCRIPTION	CONDITIONS	min	max	units
36	e_ad[15:0]	settup to clk ^ Note4		1.0		ns
37	e_ad[15:0]	hold wrt to clk ^ Note4		4.0		ns
38	e_ad[15:0]	clk ^ to output valid	80pf		36.0	ns
39	e_ad[15:0]	clk ^ to output invalid	80pf		25	ns
40	e_hlda_	clk ^ to output high	80pf		18.0	ns
41	e_hlda_	clk ^ to output low	80pf		21.5	ns
42	e_read	clk ^ to output valid	80pf		15.5	ns
43	e_read	clk ^ to output invalid	80pf		12	ns
44	e_das_	clk ^ to output valid	80pf		23.0	ns
45	e_das_	clk ^ to output invalid	80pf		18.5	ns
46	e_rdy_	clk ^ to output valid	80pf		23.0	ns
47	e_rdy_	clk ^ to output invalid	80pf		17.5	ns
48	e_cs_	clk ^ to output high	80pf		15.5	ns
49	e_cs_	clk ^ to output low	80pf		20.0	ns
50	e_rdy_	settup to clk ^			0	ns
51	e_rdy_	hold wrt to clk ^			2.8	ns
52	e_ad[15:0]	ADDR settup to e_as_ low			15.0	ns
53	e_ad[15:0]	ADDR hold wrt e_as_ high			0	ns
54	e_hold_	settup to clk ^			0	ns
55	e_hold_	hold wrt to clk ^			4.0	ns

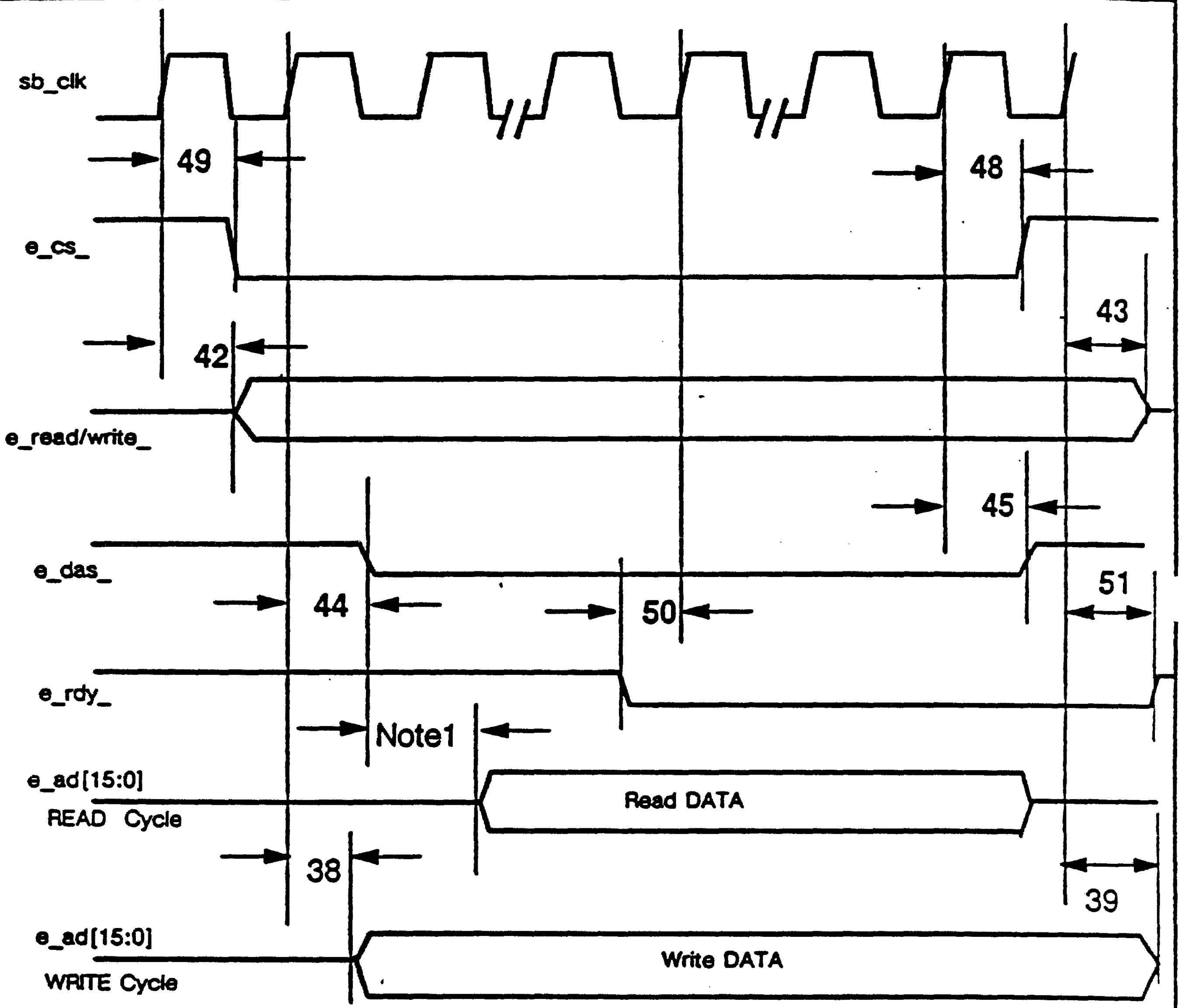
Timing Diagrams



SBus Input Signals



D_Bus Read/Write Cycle Timing (slow = high)



Note1 Refer to LANCE timing specs

LANCE READ and WRITE Cycle Timing

Note1: These values represent the timing characteristics of groups of signals. By referring to the Timing Diagrams it can be seen that one mnemonic value can represent many different signal paths.

Note2: These timing parameters are true for both the signals d_cs_ and id_cs_.

Note3: The documented values represent the timing of an external device (in this case the ESP SCSI chip), to which this gate–array is matched by design.

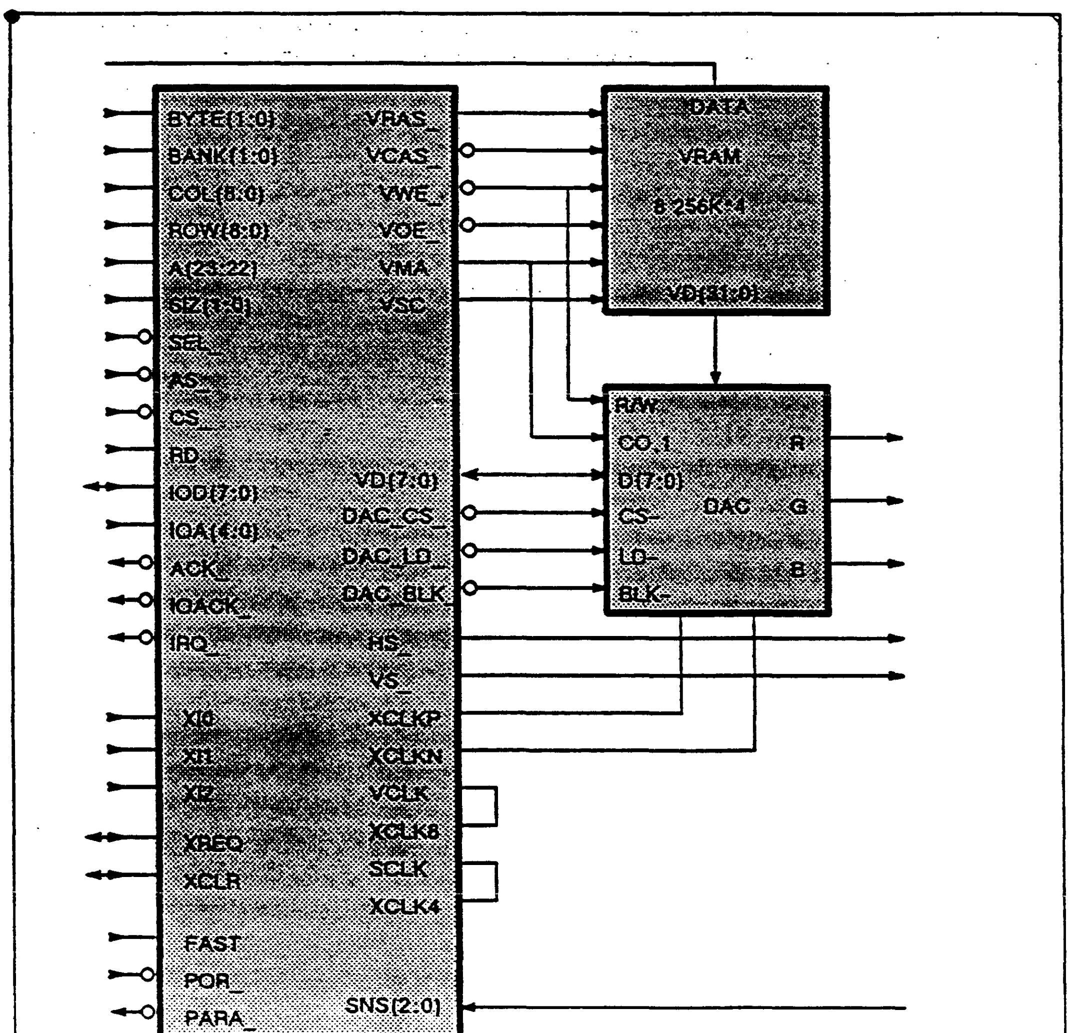
Note4: The settup and hold times refer to the timing diagram on which they are shown, and in particular to the clock edges shown. The e_ad bus is designed to be that of the LANCE Ethernet controller. Internal to this chip the e_ad bus is not latched for at least 2 clock cycles to alleviate any potential timing problems. Hence the 0ns timing requirements shown are true only if the cycle by cycle handshaking specified by the LANCE is maintained.

8.0 Revision History

12/22/87	First Release.
2/11/88	Remove sb_address bus and multiplex addr/data on sb_d bus. Add SBus Identification information.
3/28/88	Revised pinout. Corrected errors in register addressing. Added more info on programming. Updated timing diagrams. Revised block diagrams. Added register in MSbyte of ADDR_CNT. Revised operation of Terminal Count bit.
6/21/88	Added timing specs.
7/26/88	Included post_route timings

Features

- Single–chip video subsystem
- Directly interfaces to Sbus Interface
- Supports 256*4, 128K*8, and 64K*4 Video RAM
- Supports 1–bit, 8–bit, and 24–bit per pixel frame buffers
- Fully programmable video timing and resolution
- Supports up to 4 video clocks (software selectable)
- Supports Sun Video Monitor sense lines (for auto configuration)
- Directly interfaces to VRAM and RAMDAC (no external components required)
- Built–in Video Shifter for 1–bit frame buffers (maximum pixel clock 100 MHz)
- Low–cost 120PFP package



S4–Video 8–bit Frame Buffer Application

VRAM Interface	*20*	
vma(8:0)	BT4	Video Multiplexed Address
vcas(3:0)_	BT4	Video Cas Enable (Byte select)
vras(3:0)_	BT4	Video Ras Enable (Bank select)
voe_	BT4	Video Output Enable
vwe_	BT4	Video Write Enable
vsc	BT4	Video Shift Clock
Misc Pins	*8*	
mode(3:0)	TLCHTD	memory mode and configuration
type	TLCHTD	VRAM type: 0:256K, 1:1Mbit
por_	TLCHNU	Power On Reset. Clears control register
para_	BD1TU	Parametric Test Output and Output Disable

Device Number:	LMA9141	(PAD: 118, VDD:6 VSS:8, IO:104)
Package Type:	PFP120	

Address Space Decoding

This section explains the S4–Video registers and decoding. On Reset, the master control register is initialized to 0. All other registers are not initialized.

Master Decoding (A23:22)

When the S4–Video chip is selected, address A(23:22) define the master mode as follows:

0x000000–0x3FFFFF	Sbus ID
0x400000–0x7FFFFF	Video Registers
0x800000–0xBFFFFF	Video RAM
0xC00000–0xFFFFFF	Reserved

Sbus ID

The Sbus ID is either internal in the S4–Video chip or provided externally, as determined by the status of X_CS_ at the end of POR_. If X_CS_ is grounded externally, the ID will be provided internally and read as 0xFE01010y, where (y) is MODE(3:0). If X_CS_ is not grounded externally then the Sbus ID will be provided by an external PROM that is selected by X_CS_. The PROM can have a size up to 4 MBytes.

Video Registers

The video registers start at the four megabyte (0x400000) boundary and extend up to the 8 megabyte boundary. There are a total of 16 registers, including the external DAC, which are decoded with IOA(4:0).

Video RAM

The frame buffer is decoded at the eight megabyte boundary (0x800000) up to the twelve megabyte boundary (0xC00000). It is up to software to map in only memory that is physically present on the frame buffer.

Reserved

Accessing this area will return an ACK but cause no actions on the chip. This area can be used to provide "dummy pages" for software.

Interrupt

When enabled, Interrupt is asserted at the beginning of vertical blank. The interrupt is cleared by writing to the (read-only) status register.

Video Register Description

DAC Select

Accesses to these addresses are passed through to the external DAC for reading and writing of the DAC registers.

Master Control Register

Bit	Function
7	Enables interupts. When enabled, the S4–Video chip will generate an interupt when the end of the end of the frame is reached (start of VBLANK). The interupt is cleared by reading the status register.
6	Video Enable. When set to 0, the blank output is constantly asserted independent of the internal counters. When set to 1, the VBLANK output follows what is programmed into the timing registers.
5	Timing Enable/Slave Mode. When set to 0, the internal video timers are disabled and the internal state–machine that controls the transfer cycles is triggered from the external inputs XREQ and XCLR. When set to a 1, the video chip generates timing based on the values programmed and drives the XREQ and XCLR pins as outputs.
4	Cursor Enable Register. When set to a 1, accesses to the frame buffer will cause a buss error if the address is within the range of the two address values programmed into the Cursor Start Address and the Cursor End Address Registers located at 0x400012 and 0x400013.
2:3	Oscillator Select. Selects one of the three XI inputs as the source for the video timing. Selecting input 4 (2:3 = 0x11) causes the video logic to stop.
0:1	Divider. Selects a divide by 1, 2, 3 or 4 of the selected XI input.

Status Register

Bit	Function
7	Interupt Pending. An interupt was generated by the chip.
4:6	Monitor Sense. These three bits come directly from the three SNS inputs to the S4–Video chip. Usefull for determining the type of monitor connected then frame buffer.
0:3	Memory Mode. These four pins come directly from the MODE inputs the S4–Video chip. Usefull for determining what type of memory the frame buffer uses.

VBSH

The VBSH register contains the high order bits of the line number to start vertical blanking on. The vertical counter is a 12 bit counter which requires two registers to program. The four least significant bits of VBSH are used with VBSL to form the 12 bit line count. The four high order bits of VBSH are don't cares, and are read as zero.

VBSL

The VBSL register contains the low order bits of the line number to start vertical blanking on. The vertical counter is a 12 bit counter which requires two registers to program. The four least significant bits of VBSH are used with VBSL to form the 12 bit line count. The VBS registers are programmed in multiples of lines.

VBC

The VBC register contains the vertical blank end value. It is programmed in multiples of lines. When the vertical counter reaches this value, the composite blank (DAC_BLK_) goes active. The value for VBC must be programmed to be less than VBSH + VBSL.

VSS

The VSS register contains the vertical sync start value. It is programmed in multiples of lines. When the vertical counter reaches VSS, the vertical sync output (VS_) goes active. VSS must be programmed to be less than VBSH + VBSL, and should be less than VBC.

VSC

The VSC register contains the vertical sync end value. it is programmed in multiples of lines. When the vertical counter reaches VSC, the vertical sync output (VS_) goes inactive. VSC must be programmed to be less VBSH + VBSL. It must also be programmed to be greater than VSS and should be less than VBC. A basic vertical sweep with respect to the vertical counter should look something like:

0....VSS....VSC....VBC............................VBSH + VBSL

XCS

The XCS register contains the transfer hold off start value. It is programmed in multiples of 8 pixels. The S4–Video chip generates transfer cycles as necessary by counting shift clocks. The shift clock is inactive during horizontal blanking however. If an access to the frame buffer or any of the internal registers were attempted during a horizontal blank which occurred during a transfer cycle, The S4–VIDEO chip would not be able to respond until after the blanking and transfer were completed which in computer time could be a very long time degrading performance. The XCS and XCC registers allow for a window to be programmed around relative to the horizontal blank window (defined by HBS and HBC) which will prevent a transfer cycle from starting until late in the horizontal blank period, thus allowing other accesses to the video chip in the mean time.

The values for XCS anc XCC will be less than HBS and HBC respectively and will depend greatly on the relationship between the system clock and the video clock. The most important timing occurs before XCS (and HBS) then the request is processed. If the request occurs after XCS but before HBS, then the S4-video chip suspends the transfer cycle until XCC. This will allow the IU to continue with other accesses during the blanking period.

Memory Controller Interface

The video controller interfaces to the memory controller via two signals: XREQ and XCLR.

XREQ (Transfer Cycle Request) forces a video RAM reload cycle using the address of the transfer counter. Asserting XREQ causes the memory controller to begin a video reload cycle as soon as current cycles are complete. The memory controller will wait until XREQ drops, asynchronously deassert DT/OE and then complete the video reload cycle. This allows on-the-fly video reload cycles.

XCLR (Transfer Clear) clears the reload counter and the transfer counter and forces a minimum length, reload cycle. XCLR is asserted in the state following (VBC & HBS).

When the S4-video chip is in the master mode (control register bit 5 = 1) the XREQ and XCLR signal pins are driven as outputs mirroring the internally generated transfer requset and transfer clear signals. These two signal pins can then be connected to a parallel S4-Video chip which is configured in the slave mode (control register bit 5 = 0) to synchronize the two chips. When in the slave mode, the XREQ and XCLR signal pins are treated as inputs to the internal state-machines for synchronization purposes.

Video RAM Interface

The S4–Video controller generates its video memory outputs as follows:

```
VRAS(x)     = RAS * (VIDEO + CPU * (BANK(1:0) == x))

VCAS(x)     = CAS * (VIDEO + CPU * (SIZ==1) * (BYTE == x)
            + (SIZ==2) * ((BYTE ==x)+(BYTE==x+1)
            + (SIZ==3) * ((BYTE==x)+(BYTE==x+1)+(BYTE==x+2)
            +(SIZ==4) * ((BYTE==x)+(BYTE==x+1)+(BYTE==x+2)+(BYTE==x+3)))

VMA(8:0)    = MUX * (VIDEO * X(8:0) + CPU * ROW(8:0)
            + !MUX * (VIDEO * 0 + CPU * COL(8:0)
```

Cycle	CPU		Video/Refresh	
	row	col	row	col
vma0	row0	col0	x0	0
vma1	row1	col1	x1	0
vma2	row2	col2	x2	0
vma3	row3	col3	x3	0
vma4	row4	col4	x4	0
vma5	row5	col5	x5	0
vma6	row6	col6	x6	0
vma7	row7	col7	x7	0
vma8	row8	col8	x8	0

Size sb_siz(1:0)	Address sb_pa(1:0)	Byte(0) CAS(0)	Byte(1) CAS(1)	Byte(2) CAS(2)	Byte(3) CAS(3)
0,0	0,0	X	X	X	X
	0,1		X	X	X
	1,0			X	X
	1,1				X
0,1	0,0	X			
	0,1		X		
	1,0			X	
	1,1				X
1,0	0,0	X	X		
	0,1		X	X	
	1,0			X	X
	1,1				X
1,1	0,0	X	X	X	
	0,1		X	X	X
	1,0			X	X
	1,1				X

Timing Diagrams

The S4–Video controller supports four basic types of cycles: Refresh cycle, Video Cycle, CPU cycle, and Burst Cycle. For both the refresh cycle and the CPU cycle, the S4 RAM controller supports two VRAM speeds via the speed input: *fast* and *slow*. The *fast* mode supports a minimum cycle of 4 states for CPU cycles and 5 states for Refresh. In *slow* mode, RAS is extended by one additional state. This allows to use slower RAMs at the cost of an increased cycle time. There is no separate *fast* and *slow* mode for burst cycles.

Cycle Overview. The S4–RAM controller stays in the idle state (S0) until either activated by a refresh request, causing a refresh cycle, a video request, causing a video cycle, or by a CPU select, causing a CPU cycle. In case of simultaneous video, refresh, and CPU request the video request is the highest priority and the refresh request second.

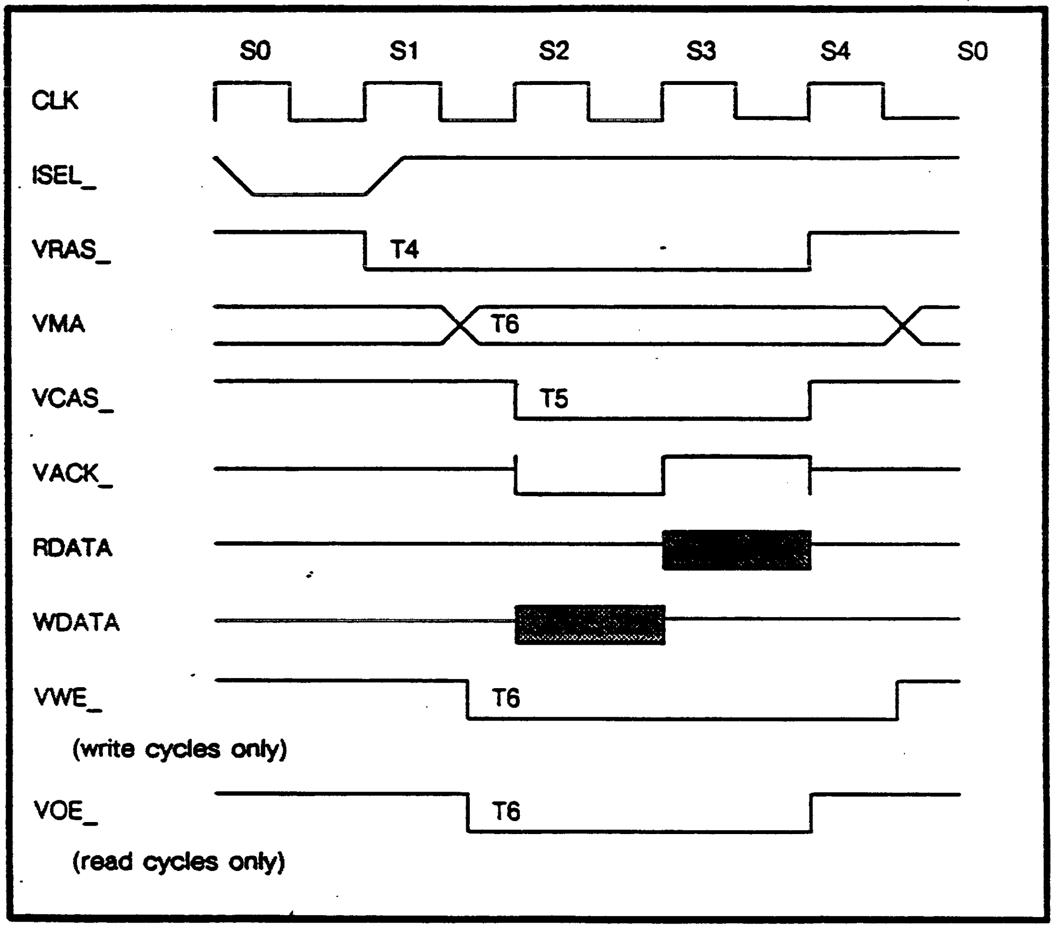
CPU cycles are initiated when a select RAM signal is received in conjunction with a matching set of addresses (see address decoding table). In response to the CPU request, the RAM controller activates RAS for the bank of memory decoded by the addresses.

Refresh cycles are generated internally by a refresh request which occurs every 320 system clocks. For a 20 MHz system clock, this is one refresh cycle every 16 usec.

Video cycles are initiated by a transition on input XREQ, which is generated by the video controller whenever a video transfer is necessary.

CPU Cycle

In response to a select, the RAM controller enters state S1 and asserts RAS for the bank of memory decoded by the addresses. The row/column addresses are multiplexed on the half-state following RAS. In state S2, the RAM controller asserts CAS and acknowledge signal VACK. Following S2, in fast mode the RAM controller finishes up with S10 which deasserts RAS and VACK while keeping CAS asserted. In slow mode, the RAM controller extends RAS in state S9, and then deasserts all control signals in state S10. In both cases, write data (WDATA) must be valid at beginning of CAS and read data (RDATA) is valid at the end of CAS.



Refresh Cycle

Refresh is implemented with a "CAS–before–RAS" cycle. Once a refresh request is recognized, all CAS outputs are asserted during state R0 followed by all RAS outputs asserted at state R1. REF, RAS, and CAS stay asserted during R2 and are deasserted in state S10. In "slow" mode, a state S9 is inserted that extends all control signals for one extra state. Refresh request takes priority over CPU cycles that arrive at the same time. Pending CPU cycles have to wait until they are recognized in S0.

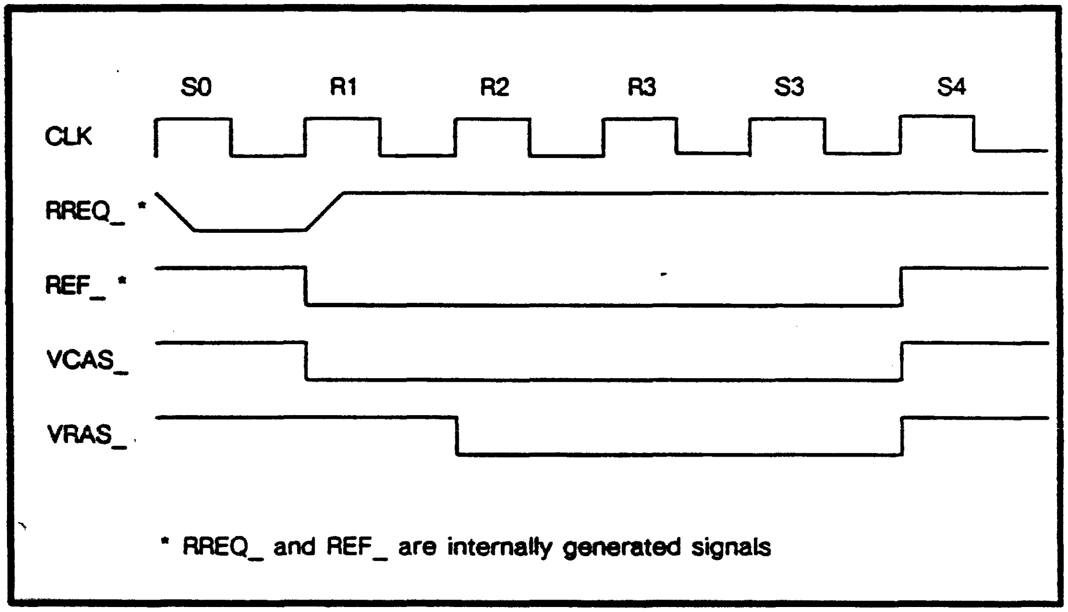


Table of Monitor Timings

Type	5 Sun 1600.1280	4 Sun 1280.1024	3 Sun 1152.900	2 Sun 1024.768	1 Apple 1152.870	0 Apple 640.480	Unit
HRes	1600	1280	1152	1024	1152	640	Pixel
VRes	1280	1024	900	768	870	480	Pixel
PClock	200.00	135.00	92.9405	70.400	100.00	30.000	MHz
HClock	89.00		61.80	53.66	68.700	35.000	kHz
HTime			16.182	18.64	14.56	28.5714	usec
VClock	67.00		65.96	66	75	66.666	Hz
VTime			15.163	15.15	13.32	16.000	msec

Register	Value	Value	Value	Value	Value	Unit	Conversion
HBS			1504	1312	1456	(856)	(X/8)-1
HBC			352	288	304	(216)	(X/8)-1
HSS			16	1312	32	(32)	(X/8)-1
HSC			144	160	160	(48)	(X/8)-1
VBS			937	813	915	525	(X)-1
VBC			37	45	45	45	(X)-1
VSS			2	(2)	3	(2)	(X)-1
VSC			6	6	6	(8)	(X)-1

Note1: HSC0 = HSC, HSC1 = HBS - HSC
Note2: VBSH = (X DIV 256), VBSL = (X MOD 256)-1
Note3: Values in parenthesis are estimates at this time.

Rivision History

Date	Change	By
7/18/88	First Release.	AVB
7/20/88	Defined XCS and XCC registers for use in delaying transfer cycles.	MWI
7/22/88	Added cursor registers, corrected timing diagram labels, corrected pin counts, removed diagnostic register, fixed address mappings for 64K x 4 VRAMs.	MWI
10/4/88	Added XREQ and XCLR, removed xi(3) and FAST, removed all timing diagrams related to FAST mode, added words about synchrnous operation of two chips, added words about cursor start and end address registers.	MWI
10/6/88	Removed confusing wording about cursor address registers in in description of control register	MWI