Firefly Programmer's Reference Manual

Sol Garcia
Sri Sriram
CSEM

Roy Levin

How would no-write-allocate affect - performance?
- consistency?
- complexity?

Last Modified On Fri Sep 14 16:15:19 1984 By levin

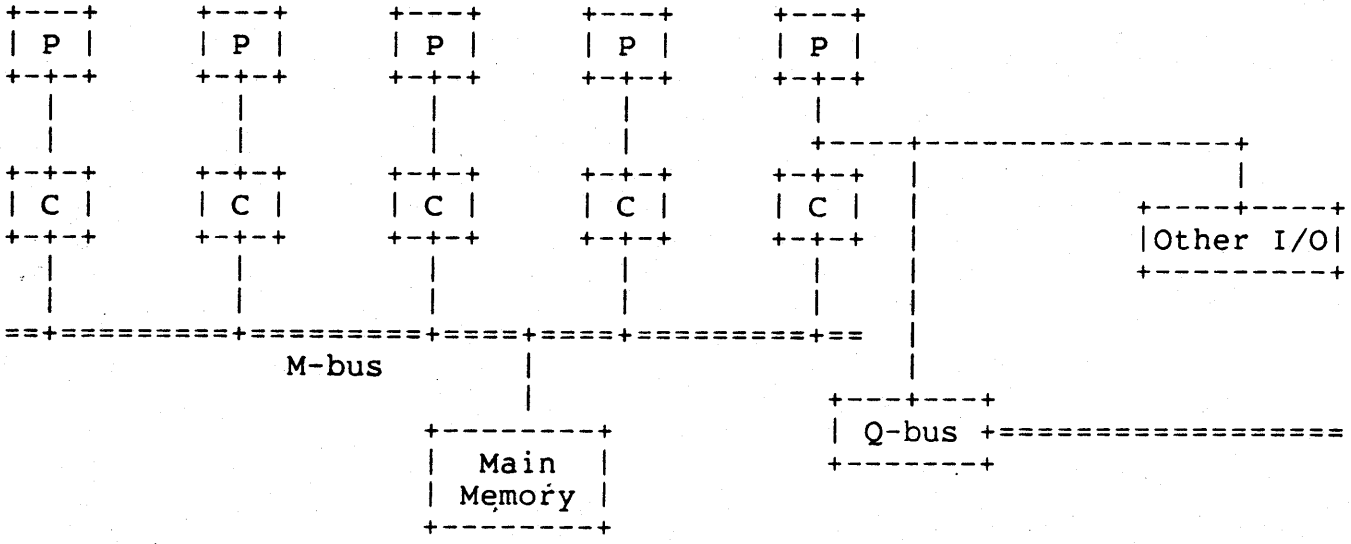
This document describes the details necessary for programming the Firefly hardware. The reader should (is assumed to) be familiar with the details of the 68010 architecture as described in

> M68000 16/32-Bit Microprocessor Programmer's Reference Manual
> Fourth Edition, 1984
> Prentice-Hall, Englewood Cliffs, NJ 07632

The information in this manual is largely derived from conversations with the hardware implementors, Chuck Thacker and Phil Petit.

0. Overview of the Firefly

The Firefly is a closely-coupled multiprocessor with a central main memory shared by five processor-cache subsystems. The processors are all the same; in the current implementation, they are Motorola 68010's. Each processor has its own cache. The caches communicate with each other over a common bus, the M-bus, to which the 8Mbyte main memory is also connected. One of the processors has access to all the I/O devices.



Section 1 of this document describes the operation of the cache subsystem, which provides virtual memory mapping and data caching facilities, and the use of the M-bus for memory accesses. Section 2 describes the inter-processor (actually, inter-cache) communication on the M-bus that doesn't involve main memory. Section 3 describes the I/O devices and related miscellaneous topics.

Throughout this document, addresses are expressed in hexadecimal, and all other quantities are expressed in decimal notation, unless

otherwise indicated.

1. The Cache/Virtual Memory Subsystem

1.1 Overview

The cache/virtual memory subsystem consists of four memories and related control. Two of these memories, A and D, comprise the cache section; the other two, AS and T, implement a simple memory mapping facility. Conceptually, the processor makes a memory reference using a virtual address, which is translated by the mapping mechanism to a real address. The real address is then passed to the cache, which interacts with the main memory and other caches (if necessary) to satisfy the reference. (This is a slightly simplified description; the truth will emerge though the remainder of this section.)

To understand the operation of the cache and virtual memory, we must recall several characteristics of the 68010 memory interface. The 68010 emits 24 bits of byte address and 3 bits of function code. Although software running on the 68010 can cause any value of the function code to be emitted, only five values are defined by the architecture: supervisor program, supervisor data, user program, user data, and CPU space (sometimes called "interrupt acknowledge"). The 68010 also classifies a memory reference as read, write, or read-modify-write. The 68010 expects that the memory system will either perform the indicated memory reference or will request that a bus error trap occur. In the latter case, it is the responsibility of the software trap handler to determine the cause of the bus error, eliminate it (if possible), and restart the failed memory reference (if appropriate). For more details, consult chapter 4 of the M68000 Programmer's Reference Manual.

1.1.1 Virtual Memory Mapping

The virtual memory portion of the cache subsystem consists of two memories, AS and T, and related control. Together they support a virtual memory system that can implement 128 address spaces of 2^24 bytes each. Each address space is subdivided into 4096 pages of 4096 bytes each.

The AS memory comprises 16 bytes and is divided into two independent 8-byte parts, called the "CPU" and the "DMA" parts. The former is used by processor-generated memory references, the latter by Q-bus-generated DMA references. Only the CPU part participates in the virtual memory mapping.

When the processor makes a memory reference, the mapping mechanism uses the 3-bit function code supplied by the 68010 to index the CPU part of the AS memory. The 8-bit entry contains a 1-bit flag and a 7-bit address space number. If the flag is set, the reference is said to be "transparent" (i.e., no mapping is to occur), in which case the address space number is ignored and the unmodified address is passed to the cache, which interprets it as a real address. Otherwise, the address is virtual. (There is also a mode in which all addresses are treated as transparent; see section 2.1.)

The T (for "translation") memory maps virtual page numbers to real page numbers. However, it doesn't hold the complete map. Rather, the T memory serves as the equivalent of a "translation look-aside

buffer" in other virtual memory implementations. If an address translation cannot be performed by the T memory, the memory reference is aborted and a bus error trap occurs in the processor. It is the responsibility of software to analyze the offending address, alter the T memory appropriately, and restart the memory reference. Thus, the T memory is a cache of recently-used memory map entries, and cache replacement occurs completely under software control. Consequently, the mapping hardware knows nothing about the data structures used to maintain the virtual-to-real page mapping.

Address translation occurs as follows. The T memory consists of 4096 entries, each of which contains

* a 7-bit address space number,
* a "not-present" flag,
* a "write-enabled" flag,
* a "trap-on-user-reference" flag, and
* 9-bits of real address.

The high-order 12 bits of a virtual address (the page number) are "hashed" with the address space number (from the AS memory entry) to form a 12-bit index into the T memory. The address space number from the entry is compared with the one from the AS memory entry; if they differ, the memory reference is aborted and a bus error trap ensues. If the address space numbers are equal, the three flag bits are inspected, and a bus error trap occurs if

* the "not-present" flag is set, or
* the memory reference is a write (or the write portion of a read-modify-write) and the "write-enabled" flag is not set, or
* the function code is "user program" or "user data" and the "trap-on-user-reference" flag is set.

If none of these conditions holds, the virtual memory reference is acceptable (however, see section 1.5) and a real address is formed by taking the low-order 14 bits of the original virtual address and appending them to the 9 bits in the selected T memory entry. This 23-bit real address is then passed to the cache section.

Note that the low-order bits of the real address include the two low-order bits of the virtual page number. This unusual arrangement results from an interaction between the mapping mechanism and the cache, and will become clear after the details of the cache have been presented.

1.1.2 The Cache

The cache provides the interface between the processor (and I/O devices) and the main memory bus (the M-bus). The collection of caches on the M-bus cooperate to present a consistent view of main storage to their associated processors. Data transferred on the M-bus is 32 bits wide; addresses on the M-bus always refer to 4-byte, aligned quantities. The main memory can hold 2^23 bytes, thus, a real (byte) address is 23 bits wide (although only 21 bits actually appear on the M-bus).

Physically, the cache consists of the A memory, which holds 4096 11-bit entries, and the D memory, which holds 4096 32-bit entries. Logically, they form a single memory of 4096 "lines", each of which has

* 9 bits of address residue,
* a "dirty" bit,
* a "shared" bit, and
* a 32-bit word of data.

The "dirty" and "shared" bits are used in the cache consistency algorithm, which is described in section 1.4.

The cache employs a direct-lookup scheme (sometimes called "1-way associative") to map an incoming real address to a cache line, as follows. Bits 13-2 of the real address select the cache line and bits 22-14 are compared with the address residue bits in the cache line. If these quantities are equal, a "hit" has occurred, otherwise, a "miss" has occurred. If a memory read was requested, a cache hit causes the data to be supplied from the selected cache line in the D memory without any traffic occurring on the M-bus. A miss causes the selected cache entry to be replaced with the data from the requested main memory location, which is obtained by an M-bus read cycle. (If the cache line was "dirty", it will first be written back to main memory; see section 1.4.) If a memory write was requested, a cache hit causes the appropriate portion of the selected line in the D memory to be overwritten, with write-through occurring only if needed by the consistency algorithm (section 1.4). A cache miss causes an implicit read of the requested location (which triggers cache replacement as described above), followed by a retry of the write (which will then experience a cache hit).

We can now explain the peculiar construction of a real address described at the end of section 1.1.1. Conceptually, the mapping mechanism performs its job, then passes the resulting real address to the cache. However, for speed, we would like to perform lookups in the T and A memories in parallel. If the bits used to index the cache didn't depend on the address bits involved in the mapping mechanism, we could do parallel lookups without any difficulty. Unfortunately, bits 13-12 of the virtual address are used in both mappings. To permit parallel lookups, therefore, we require that the software ensure that bits 13-12 of a virtual address equal bits 13-12 of the real address to which it maps. The hardware assumes this requirement is satisfied and therefore stores only real address bits 22-14 in each T memory entry.

1.2 The AS Memory

The AS memory is read by the mapping hardware as described in section 1.1.1. It may be written (but not read) by software through certain locations in CPU space (i.e., function code 7) using the 68010's MOVEC and MOVES.W privileged instructions. Even though the AS memory is only one byte wide, the MOVES instruction must specify a halfword (16-bit) operand. The format of this halfword is different for the CPU and DMA parts of the AS memory, as described below.

To write location i of the AS memory, software accesses location (i lsh 4)+4 in CPU space. Locations 0-7 of AS are the DMA part, locations 8-F are the CPU part.

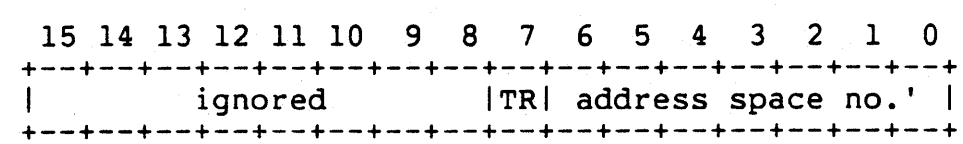
At power-up time, the contents of the AS memory are undefined.

1.2.1 The CPU Part

Locations 8-F are used for processor references and are indexed by the 68000's 3-bit function code. The following table restates the correspondance between CPU space addresses and AS locations:

Function Code	Name	AS Location	CPU Space Address
0	(Reserved)	8	84
1	User Data	9	94
2	User Program	A	A4
3	(Reserved)	B	B4
4	(Reserved)	C	C4
5	Supervisor Data	D	D4
6	Supervisor Program	E	E4
7	CPU Space	F	F4

The 16-bit data value written to the AS memory location is



where TR is the transparent mode flag (0 = address is virtual, 1 = address is real), and the low-order seven bits are the complement of the address space number (ignored if TR = 1).

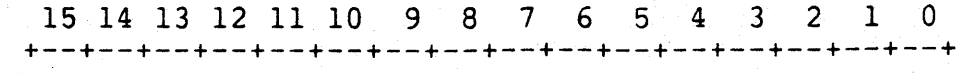
1.2.2 The DMA Part

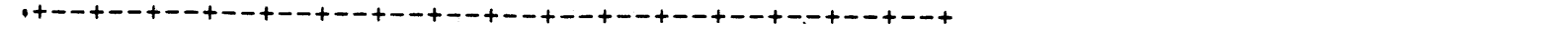
The DMA part of the AS memory is logically unrelated to the CPU part; it has nothing to do with address spaces. The two parts are packaged together for implementation convenience.

Unlike CPU references, all DMA references made by devices refer to real addresses. Q-bus DMA devices can only supply 22 address bits for their memory references. Since the real memory has 23 bits of address, some form of mapping is necessary to permit Q-bus DMA devices to access all of real memory. The DMA part of the AS memory is used for this purpose. During a DMA transfer, the 3 high-order bits of the Q-bus address are used to index the DMA part of the AS memory. Four bits are taken from the selected AS location and prepended to the remaining 19 bits of Q-bus address, forming a 23-bit real address. Note that, while this mapping mechanism makes all of real memory addressible by Q-bus devices, only half of it is accessible at any given instant. Note also that DMA transfers across 2^19 byte boundaries require multiple entries in the DMA part of the AS memory to be properly set up. The following table summarizes DMA addressing:

Q-bus DMA Address	AS Location	CPU Space Address
000000-07FFFF	0	04
080000-0FFFFF	1	14
100000-17FFFF	2	24
180000-1FFFFF	3	34
200000-27FFFF	4	44
280000-2FFFFF	5	54
300000-37FFFF	6	64
380000-3FFFFF	7	74

The 16-bit data value written to the AS memory location is





where ADDRHIGH' is the complement of the high-order 4 bits of real address supplied to the shared memory (i.e., real address bits 22..19).

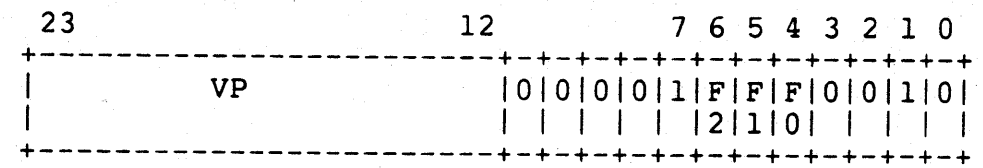
1.3 The T Memory

The T memory consists of 4K entries. Recall from section 1.1.1 that the T memory is addressed by the result of a hash function that combines the high-order 12 bits of the virtual address and the 7-bit address space number from the appropriate entry in the CPU part of the AS memory. More precisely,

```
HV = VP xor ((AS[8+FC] and 7) lsh 9)
```

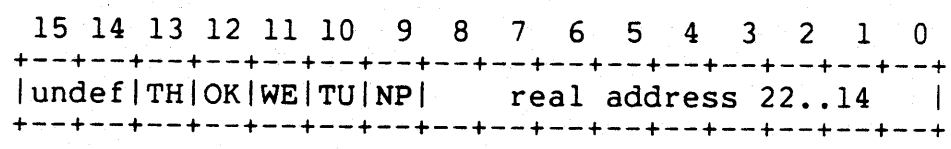
where HV is the hash value, VP is the virtual page number (bits 23-12 of the virtual address), AS is the AS memory, and FC is the function code.

Software can read or write an entry in T by making a halfword (16-bit) reference in CPU space to location:



The hardware will compute HV by the formula above, using F2..F0 to index the AS memory and extract a (complemented) address space number. (The observant reader will have noticed that bits 7-4 of this address are the AS memory address.) It will then use HV as the T memory address. Since any explicit read or write of T implicitly accesses an entry in AS, software must ensure that the selected location in AS has the proper value.

A read of the T memory returns the following halfword of data:

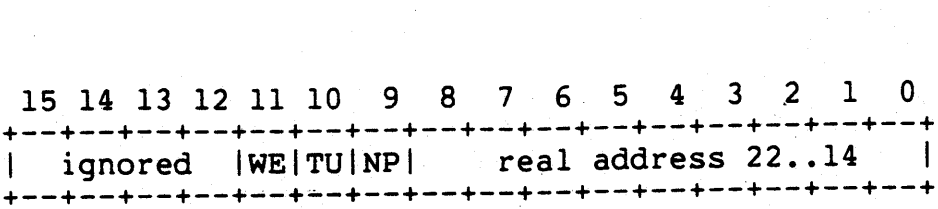


where

- TH is 1 if the address space number stored in the selected T entry matches the address space number in the selected AS entry,
- OK is 1 if the read actually worked (see below),
- WE is 1 if writes to the selected real page are permitted,
- TU is 1 if user-mode references to the selected page are prohibited, and
- NP is 1 if the selected page is not present.

Note that, because of various internal races, the read may "fail", in which case OK = 0. Thus, software must repeatedly read the desired T memory location until OK comes back as 1.

To write an entry in T, software constructs the CPU space address as described above and supplies the following data



where WE, TU, and NP are as described above. The hardware will fill in the address space number in the T entry from the AS memory location specified in bits 7-4 of the CPU space address.

At power-up time, the T memory has undefined contents. After the AS memory has been initialized, software should write each location of the T memory with a well-defined value (presumably with NP = 1).

1.4 The Cache Consistency Algorithm

The material in this section is not essential for programming the Firefly; it describes the algorithms employed by the caches to ensure that main memory appears consistent to all processors.

Each cache is connected to a processor and to the M-bus, and monitors memory requests on each. Thus, each processor communicates with precisely one cache, but all caches communicate with each other (and with the main memory) over the M-bus. Processor requests to read and write data are called PFetch and PStore, respectively; M-bus cycles are called MRead and MWrite. PFetch and PStore transfer 16-bits of data, while MRead and MWrite transfer 32 bits. All addresses involved are real addresses. Each cache action involves precisely one cache line, which is selected by bits 13-2 of the real address (see section 1.1.2).

When a cache performs an MRead cycle, each other cache inspects the address and, if it "hits", the cache puts the data for the addressed location on the M-bus and asserts MShared. (MShared is a signal line on the M-bus whose role in the consistency algorithm will become clear shortly.) If no cache supplies data, then the main memory does. All caches that assert MShared also set the "shared" bit in the cache line corresponding to the addressed location, as does the cache that performed the MRead cycle.

When a cache performs an MWrite cycle, each other cache inspects the address and, if it "hits", the cache takes the data from the M-bus, writes it into the cache line corresponding to the selected location, and asserts MShared. The main memory also writes the data into the addressed location. All caches, including the one that originated the MWrite, clear the "dirty" bit in the cache line corresponding to the addressed location. The originating cache may or may not pay attention to MShared, as shown in the table below.

The "shared" and "dirty" bits in each cache line are used to control the consistency algorithm. If the "shared" flag is set in a cache entry, the cache believes that one or more other caches have a copy of the same line. If the "dirty" flag is set, then the data in the cache line is newer than the data in the corresponding main memory location. More precisely, the following invariants hold (between M-bus cycles):

[I1] If the same memory location appears in more than one cache, then every cache that holds the location has the same data and has the "shared" flag set.

[I2] A location can be flagged "dirty" in at most one cache, and will be flagged dirty if and only if the value in the cache is newer than the value stored in main memory.

Let us now examine the detailed operation of a cache. On every memory reference (PFetch, PStore, MRead or MWrite), the cache interprets the real address, selecting a cache line, determines "hit" or "miss", then behaves as follows:

(1) If "hit", the cache manipulates the selected line according to the state transition diagram below.
(2) If "miss" then:
  (2a) If MRead or MWrite, do nothing.
  (2b) If PFetch or PStore then:
    (2b1) If the selected line is dirty (i.e., has its "dirty" flag set), the cache performs an MWrite cycle, writing the data in the selected line back to main memory. It then clears the dirty flag. The cache ignores the MShared signal on write-backs.
    (2b2) The cache performs an MRead to the desired real address and stores the address residue and data from the M-bus in the selected line. The cache sets the "shared" flag in the selected entry from the MShared signal on the M-bus, as described above.
    (2b3) The cache completes the PFetch or PStore by manipulating the selected line according to the state transition diagram below.

read before write + write allocate

	Operation			
Initial State	PFetch	PStore	MRead [A]	MWrite [B]
0 (D',S')	0	2	1	[D]
1 (D',S)	1	0 or 1 [C]	1	1
2 (D,S')	2	2	3	[D]
3 (D,S)	3	0 or 1 [C]	3	1 [E]

D = dirty; S = shared; ' = "not"

Notes:
  [A] A cache that hits asserts MShared and puts the data from the selected line on the M-bus, as described earlier.
  [B] A cache that hits asserts MShared and replaces the data in the selected line with the data on the M-bus, as described earlier.
  [C] A write-through occurs. The cache performs an MWrite cycle, and sets the "shared" bit of the selected line from the MShared line on the M-bus. Thus, if another cache asserts MShared, the selected line ends up in state 1, otherwise, it ends up in state 0.
  [D] This situation cannot occur, because a PStore that produces a write-through is always preceded by an MRead (see case 2b2, above), which will cause "shared" to be set.
  [E] This transition reflects the fact that main memory is updated when write-through occurs.

1.5 A Note on Read-Modify-Write

The previous section ignored the atomicity requirements of the 68010's test-and-set (TAS) instruction. When a cache receives a PFetch request, it doesn't know whether or not it is the read portion of a read-modify-write cycle (which is only generated by the TAS instruction). It discovers the read-modify-write only when the subsequent PStore request occurs; a flag (generated by the cache control logic) accompanies the PStore signalling the read-modify-write. If an MWrite cycle "hit" the selected cache location between the PFetch and the PStore, the atomicity of the read-modify-write would be compromised. To prevent this from happening, the cache watches for the sequence

```
PFetch  A
<an M-bus cycle that hits in the cache>
PStore  A (with both read-modify-write flag and "shared" set)
```

If this sequence occurs, the cache aborts the PStore, causing a bus error trap. Thus, software must also be prepared to take a bus error trap on a TAS instruction.

Note that this is a conservative implementation, since the PStore needs to be blocked only if the intervening M-bus cycle is an MWrite that hit location A in the cache. However, the scheme above requires less hardware to implement.

1.6 A Note on Cache Subsystem Performance

For a memory reference that "hits" in both the T and A memories and doesn't require write-through, the cache subsystem can complete the reference with no wait states in the 68010. That is, the timing numbers that appear in Appendix F of the Programmer's Reference Manual accurately describe the performance in this case. For each memory reference required, 3 cycles of delay are introduced, which must be added to the timings in Appendix F.

2. M-bus Communication

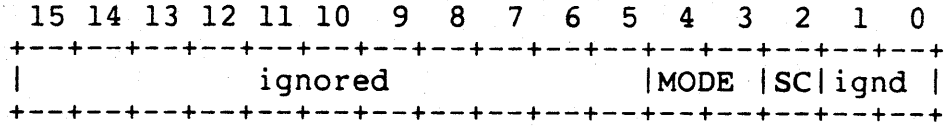
For the purposes of this section, participants in conversations on the M-bus are called "hosts". They are the several processor-cache subsystems and the main memory subsystem. Each processor-cache subsystem has a 3-bit host number:

0	Processor-cache subsystem on the I/O processor
1-4	Other processor-cache subsystems
5-7	(Reserved)

The processor-cache subsystems use these addresses to communicate with each other, as described in section 2.1. The main memory subsystem is "addressed" specially, as described in section 2.2.

2.1 Interprocessor Communication and Control

Hosts communicate by writing locations in the last page of the real address space. To cause some action to occur on target host number 'i', software on a host (possibly the same one) writes (real) location 7FF000+(i lsh 2) with



where

SC is 1 to set or 0 to clear the mode indicated by MODE,

MODE selects a mode on the target host, which is to be set or cleared as determined by SC.

MODE = 0 selects the interrupt mode. In this mode, the target host's cache subsystem requests an interrupt through the 68000 autovector for priority 1. It will continue to request interrupts until the interrupt mode is explicitly cleared. (For convenience, it is also possible for a processor to clear this mode in its own cache by writing arbitrary data to location 6 in CPU space.)

MODE = 1 selects the processor reset mode. In this mode, the processor is held in a reset state until the mode is cleared, at which time the processor loads its reset interrupt vector as described in the 68000 manual. (Exception: the reset mode doesn't work on the host number 0, the boot processor.) At power-up time, processor reset mode is enabled on all processors (except processor 0).

MODE = 2 selects cache miss mode. In this mode, all memory references will miss in the cache, and all cache entries are treated as not-dirty. Thus, reads will get data from main memory without causing a write-back. Cache miss mode is intended to be used to initialize the cache. At power-up time, cache miss mode is enabled on all processors. On each processor, software initializes the cache by performing a 32-bit read of location (i lsh 2), for all i in [0..4095]. (See section 1.1.2.) This suffices to initialize the address residue and the "shared" and "dirty" flags in each cache entry. After completing these reads,

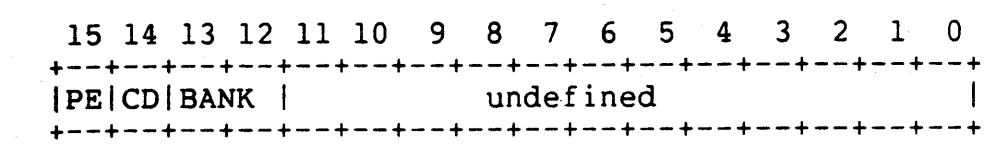
software clears cache miss mode.

MODE = 3 selects transparent mode. In this mode, all addresses presented to the cache subsystem by the processor are treated as real addresses, and the mapping provided by the AS and T memories is bypassed. At power-up time, transparent mode is enabled on all processors.

It is important to note that, when a write reference is made to any address in the last page of real memory, the cache forces a write-through (by asserting the "shared" signal). This ensures that inter-cache operations will actually cause an M-bus cycle.

2.2 Main Memory Control

The main memory maintains a parity bit for each 32-bit word. This bit checks the word only while it is in the memory; the M-bus is unchecked. The memory control logic contains a register holding parity error information, which can be accessed by a read cycle to any location in the last page of the physical address space, e.g., 7FF000. (By convention, software accesses the memory's error register at location 7FFFDC. In addition, bit 5 of the address has other significance; see below.) Reading the error register produces the following:



where
- PE is 0 if a parity error has occurred, and is 1 otherwise,
- CD if PE = 0, CD is the card on which the parity error occurred,
- BANK if PE = 0, BANK is the memory bank on card CD in which the error occurred.

Once an error has occurred, the contents of the error register are "frozen". Reading the error register clears an error condition (i.e., it sets PE to 1).

The memory will compute and check either even or odd parity under software control. Software encodes its choice in the address it uses to read the error register; bit 5 of the address selects even parity if it is zero, odd parity if it is one. Thus, reading location 7FF03C returns a 16-bit quantity of the above form, resets any parity error, and selects odd parity generation and checking for subsequent memory cycles. Reading location 7FF01C behaves identically except that it selects even parity. At power-up time, the main memory (including parity bits) and control register have undefined contents. Software should initialize the memory and control register (including even/odd parity selection) as follows. Following power-up and initialization of the cache on the boot processor (see the description of cache miss mode in section 2.1), software should read the parity error register, thereby selecting even or odd parity, as desired. Software then must write every location in real memory to set the parity bits correctly. This is slightly tricky, since the cache won't do write-throughs, only write-backs. The simplest way to ensure that every main memory location gets written is to sweep memory from lowest to highest address, then write the first 16K bytes

again. Finally, software reads the error register again (using the same address) to clear the parity error flag.

Note that a parity error does not trigger an interrupt and that potentially incorrect data will be supplied to the requesting cache without any error indication. Software must periodically check the error register and respond to parity error indications as best it can.

The cache takes no special actions on a read of the last page in real memory. (Recall from section 2.1 that the cache will force a write-through on every write to the last page of real memory.) In particular, if the read hits in the cache, no M-bus cycle will occur. Thus to ensure that the error register in the main memory is actually read, software must force a cache miss. That is, software must first read or write a memory address that maps to the same cache line as the address used to access the error register. When the error register is subsequently read, the cache will miss and the necessary M-bus cycle will occur.

3. I/O and Miscellaneous Functions

On the Q-bus processor, supervisor program space locations 000000-00FFFF and supervisor data space locations 010000-01FFFF are not interpreted by the cache and memory system. Instead, they access the following hardware:

Supervisor Program Addresses	Supervisor Data Addresses	Device
000000-007FFF	010000-017FFF	Boot ROM
008000-0087FF	018000-0187FF	Interval counter
008800-008FFF	018800-018FFF	Serial I/O (MC68701)
009000-0097FF	019000-0197FF	Control register
009800-009FFF	019800-019FFF	Encryption (AMD9518)
00A000-00AFFF	01A000-01AFFF	Clock chip (MC146818)
00B000-00BFFF	01B000-01BFFF	Q-bus interrupt vector
00C000-00FFFF	01C000-01FFFF	Q-bus address space

In the subsequent description, some 16-bit registers are described as "non-byte-swapped". This is intended to emphasize that, unlike Q-bus data transfers (see section 3.11), the 16-bit values read from or written to these registers appear in normal, 68000 order.

All interrupts from I/O devices use the 68000's autovectoring mechanism, according to the following table:

Priority	Vector Offset	Used by	Section
1	64	M-bus communication	2.2
2	68	Clock chip	3.6
3	6C	Serial I/O	3.3
4	70	Q-bus devices	3.11
5	74	Clock chip	3.6
6	78	Power failure	3.7

3.1 Boot ROM

The boot ROM is accessed as a normal read-only memory. However, it has the slightly peculiar characteristic that, when accessed for program, it appears at location 0, but when accessed for (read-only) data, it appears at location 10000. This apparent craziness accommodates the 68010's treatment of interrupt vectors, since the "reset" vector is defined to reside at supervisor program locations 0-3 and all other vectors reside in supervisor data space beginning at location 4.

3.2 Interval Counter

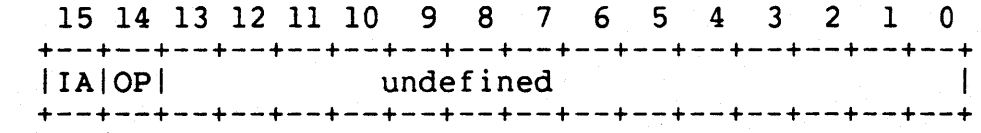
The interval counter is a read-only, 16-bit, non-byte-swapped counter that increments once every 80 chip cycles. Thus, with a 10 MHz clock, the counter ticks once every 8 microseconds. It wraps around in 8*64K microseconds, or approximately 0.5 seconds.

3.3 Serial I/O

Serial I/O is controlled by a dedicated processor (MC68701) that supplies a full-duplex communication channel to the keyboard, mouse, and a UART (RS232C). The mouse is strictly an input device. The keyboard generally provides input, but accepts output for control functions, such as altering the way in which key transitions are reported or illuminating the LEDs on the keyboard. The UART provides bidirectional data transmission and also responds to some control functions, such as altering the transmission rate.

All of the registers used to communicate with the serial I/O processor are non-byte-swapped.

The serial I/O processor has a read-only, 16-bit, non-byte-swapped status register at location 18800:



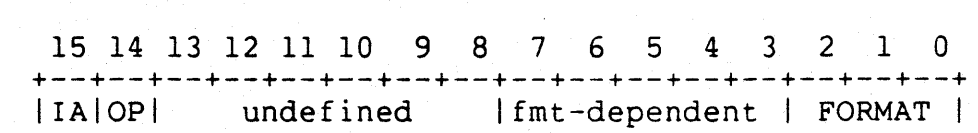
where

IA is 0 if the I/O processor has input available, or 1 otherwise. If input interrupts are enabled (see section 3.4), an interrupt will be requested when IA = 0. However, the IA flag does not necessarily become 0 as soon as input becomes available; the I/O processor normally sets IA to zero at most once every 10 milliseconds. It will do so more often if explicitly requested to do so (see section 3.3.2.3), or if its internal buffers become uncomfortably full (because software has been too leisurely in processing input messages), or if it otherwise feels inclined to interrupt.

OP is 0 if the I/O processor is ready to receive output, or 1 otherwise. If output interrupts are enabled (see section 3.4), an interrupt will be requested when OP = 0. Software is permitted to send a byte to the I/O processor only when OP = 0, as discussed in section 3.3.2.

3.3.1 Serial Input

The input side of the serial I/O processor uses two 16-bit, non-byte-swapped registers named P and A, at locations 18802 and 18804, respectively. Software can read input from the I/O processor only when both IA and OP in the status register are 0. (That is, the I/O processor must tell software that it is finished processing the current output byte before software can request an input message. See section 3.3.2 for a description of the serial I/O processor's output side.) Software then reads the P register, obtaining a "message" of the following form:



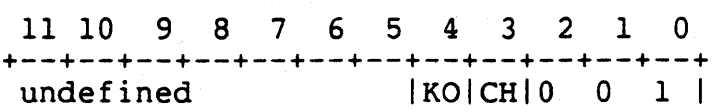
IA and OP are the same as the corresponding bits in the status register. FORMAT describes the nature of the input. Bits 7-3 are format-dependent and will be explained in subsequent subsections.

The IA and OP bits also appear as bits 15 and 14, respectively, in

the A register.

3.3.1.1 Keyboard Status

When the FORMAT field of the message in the P register is 001, the I/O processor is reporting a state change on the keyboard. In this case, the P register has the following form:



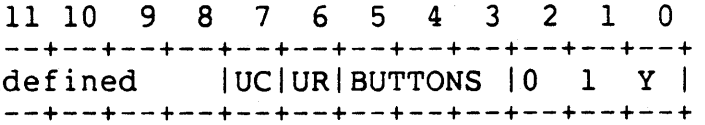
where
- KO is 1 if the keyboard is ready to accept output (see section 3.3.2.1) and 0 otherwise.
- CH is 1 if the keyboard has a character to supply and 0 otherwise. If CH = 1, the A register holds the character in its least significant 8 bits.

Software must always read the A register, even if CH = 0. It is the read of the A register that notifies the hardware that software has finished reading the keyboard status message.

Normally, CH will be 1, since the I/O processor does not normally notify the software when KO changes from 0 to 1. However, software can force the I/O processor to supply the keyboard status; see section 3.3.2.3.

3.3.1.2 Mouse Status

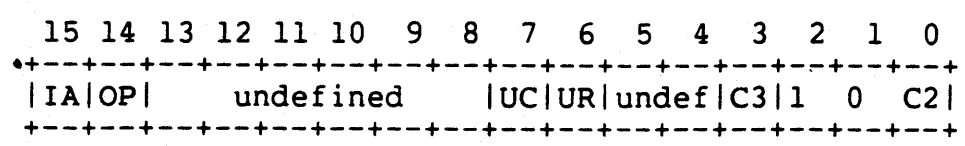
When the FORMAT field of a message in the P register is 010 or 011, the I/O processor is reporting a state change on the mouse. In this case, the P register has the following form:



where
- UC is the UART's "carrier" signal (see section 3.3.1.3).
- UR is the UART's "ring" signal (see section 3.3.1.3).
- BUTTONS encode the state of the mouse buttons (***the correspondance between the bits and the buttons, including up/down sense, is TBD).
- Y if Y = 0, the A register contains the change in the mouse's X coordinate since the last time it was reported by the I/O processor. If Y = 1, the A register contains the change in the mouse's Y coordinate since the last time it was reported by the I/O processor. In either case, the A register contains the value as a signed 8-bit quantity in its least significant byte.

3.3.1.3 UART Input

When the FORMAT field of a message in the P register is 100 or 101, the I/O processor is reporting that input is available from the UART. In this case, the P register has the following form:



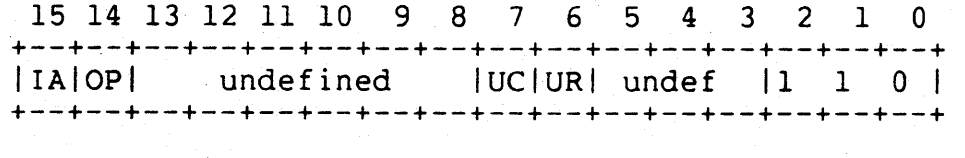
where
UC is the UART's "carrier" signal (***details TBD).
UR is the UART's "ring" signal (***details TBD).
C2,C3 indicate whether 1, 2, or 3 characters of UART input are available. In any case, software reads the first character of input from the least significant byte of the A register. If C2 = 1, either one (C3 = 0) or two (C3 = 1) additional characters are available. Software reads the P register again, obtaining the third character (from the least significant byte) if C3 = 1 or discarding the result if C3 = 0. Finally, software reads the A register again, obtaining the second character from the least significant byte. This slightly peculiar order of character presentation simplifies the hardware, at the cost of a slightly unintuitive software interface.

The baud rate for input and output is identical and can be set by an output message (see section 3.3.2.2).

3.3.1.4 UART Output Flow Control

The serial I/O processor provides for limited buffering of output to the UART. To prevent its buffer from being overrun by the CPU, the I/O processor sends the CPU explicit "credits" for output to the UART. Software is permitted to send a byte to the (output side of the) UART only when it has a non-zero credit balance, and every byte sent decreases the credit balance by one. Software must keep track of its balance; the I/O processor may not respond gracefully if presented with a byte for the UART when the credit balance is zero.

At initialization time, software sets the UART output credit balance to zero. It increments the balance by a variable amount whenever the FORMAT field of a message in the P register is 110, i.e.,



where
UC is the UART's "carrier" signal (see section 3.3.1.3).
UR is the UART's "ring" signal (see section 3.3.1.3).

After software reads this message from the P register, it reads the A register, which contains in its least significant byte an increment to the credit balance (in units of bytes). Software should add this value to its credit balance.

3.3.2 Serial Output

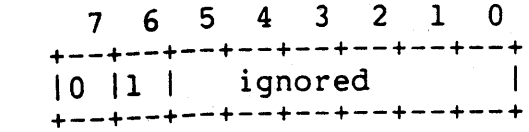
In general, output to the serial I/O processor consists of multi-byte "messages". However, the hardware path to the I/O processor is only one byte wide. For each byte of an output message, software places the byte in the least significant 8 bits of a 16-bit halfword, then writes the resulting value to the B register. The B register is a

16-bit, non-byte-swapped register that responds to any even address in the range 18800-18806. (By convention, the B register is taken to be location 18806.) As described earlier, software is permitted to load the B register only when the OP bit of the status register is 0. Furthermore, as described in section 3.3.1.4, output may be sent to the UART only when the credit balance is greater than zero.

In the following subsections, we will describe the output messages that the I/O processor accepts without repeating the preceding rules by which the message is actually transmitted.

3.3.2.1 Keyboard Output

Software sends a byte to the keyboard by sending a two byte message whose first byte is

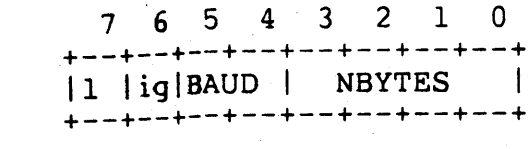


and whose second byte holds the desired value. (***The semantics of the byte are TBD.) This is the only way to output values to the keyboard; each byte must be contained in a separate message.

The keyboard processes output more slowly than the I/O processor and cannot buffer output data. Therefore, software must not send an output message to the keyboard unless it knows the keyboard is ready to accept it. The keyboard reports its willingness to receive data via the KO bit of a keyboard status message (see section 3.3.1.1), which the I/O processor generates (with KO = 1) after the keyboard finishes processing an output byte. In addition, software can poll the KO bit by forcing the I/O processor to send a keyboard status message, using one of the commands described in section 3.3.2.3 (OPCODE = 6).

3.3.2.2 UART Output

To send a sequence of bytes to the output side of the UART, software sends a message whose first byte is



where

BAUD is the baud rate to which the UART is to be set (both input and output sides of the UART operate at this rate), encoded as follows:

0	unchanged from previous value
1	9600 baud
2	1200 baud
3	300 baud

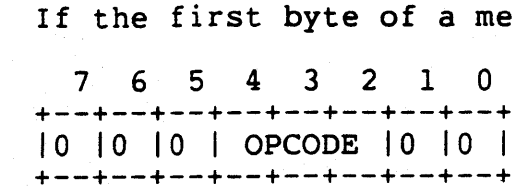
NBYTES is the number of bytes that follow.

Following the first byte are NBYTES of data to be send to the UART. Note that NBYTES can be zero, permitting software to set the baud rate without performing output. At power-up time, the baud rate is set to 9600.

Recall that UART output is subject to flow control, as described in section 3.3.1.4.

3.3.2.3 Miscellaneous Commands

If the first byte of a message has the form

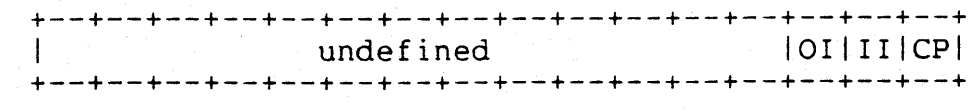


the serial I/O processor decodes OPCODE as indicated below. In all cases except OPCODE = 7, the output message is a single byte.

OPCODE	Interpretation
0	No operation.
1	Set the UART DTR line to 0. (***semantics TBD)
2	Force the I/O processor to generate a mouse status input message (with FORMAT = 010). In addition, any status changes on keyboard and/or UART will also be reported.
3	Force the I/O processor to send any pending input messages without waiting for the usual time interval (see the description of the IA bit in the status register, section 3.3). If the status of the mouse and UART have not changed since they last supplied input messages, this command has no effect.
4	Reserved.
5	Set the UART DTR line to 1. (***semantics TBD)
6	Force the I/O processor to generate a keyboard status input message (FORMAT = 001). This command is used in conjunction with keyboard output (section 3.3.2.1).
7	Treat subsequent output bytes as 68701 code. The first byte following the command is a count of the number of bytes of code, which immediately follow the count byte. The code is loaded into an internal buffer and executed; the size of this buffer is determined by available space in the I/O processor's internal RAM and is likely to be modest. A description of the operating environment for this code and the rules to which it must adhere are beyond the scope of this document.

3.4 Control Register

The control register is a write-only, 16-bit, non-byte-swapped register laid out as follows:



where

CP is written with a 0 to cause interrupts from the clock chip (see section 3.6) to occur at priority 2, or is written with a 1 to cause interrupts at priority 5. In either case,

the interrupt uses the 68000 autovector for the appropriate priority. Note that CP does not actually enable the clock interrupt; this is done by direct interaction with the clock chip (see section 3.6).

II is written with a 1 to enable interrupts from the input side of the serial I/O processor (see section 3.3), or is written with a 0 to disable interrupts.

OI is written with a 1 to enable interrupts from the output side of the serial I/O processor (see section 3.3), or is written with a 0 to disable interrupts.

Serial I/O interrupts, both input and output, occur at priority 3, using the 68000's autovectoring mechanism.

At power up time, the control register has undefined contents. It is software's responsibility to initialize the control register properly.

3.5 Encryption

The encryption facilities are provided by an AMD 9518 chip, which offers a bewildering variety of ciphering facilities. The hardware interface provided in the Firefly makes a subset of these facilities available; only these are discussed here. It is the intent of this section to make it unnecessary for a programmer to read the chip documentation; however, the intrepid and/or curious may find some additional useful information there. Both the chip description and this document assume a knowledge of the Data Encryption Standard (DES).

To the 68010, the encryption chip appears to have four control registers and six data registers. The control registers are 8 bits wide and are named ADDR, MODE, COMMAND, and STATUS. The data registers are 64 bits wide and are called INPUT, OUTPUT, E, D, IVE, and IVD. At any given instant, only ADDR and one of MODE, COMMAND, STATUS, INPUT, or OUTPUT are accessible to the 68010.

All reads and writes of 8-bit quantities to and from the encryption chip should be coded on the 68010 as halfword (i.e., 16-bit) transfers. The least significant byte of the 16-bit halfword contains the desired 8-bit value. Thus, the chip's 8-bit registers appear as 16-bit, non-byte-swapped registers in which only (at most) the low-order 8 bits are meaningful.

3.5.1 The ADDR Register

The ADDR register is an 8-bit, write-only, non-byte-swapped register whose value determines what other control or data register is accessible to the 68010. Most of the addressible registers are either read-only or write-only, and a slightly unintuitive scheme is used to access them. The hardware interface defines two locations in supervisor data space, a "read register" address (location 19800) and a "write register" address (location 19802). The 68010 can read one of the chip's registers only at the "read register" address and can write one of the chip's registers only at the "write register" address. The following table shows the correspondance between values in ADDR and accessible registers.

Value In ADDR	Readable at Read Register	Writable at Write Register

0	OUTPUT	INPUT
2	STATUS	COMMAND
6	MODE	MODE

A value is stored in ADDR by writing an 8-bit value to supervisor data space location 19800. The register(s) thus rendered addressible are described in subsequent sections.

3.5.2 The MODE Register

The MODE register is an 8-bit, read-write, non-byte-swapped register that determines the kind of ciphering operation to be performed. Its format is

```
  7  6  5  4  3  2  1  0
+--+--+--+--+--+--+--+--+
| 0| 0| 0|DE| 1| 0|CIPH |
+--+--+--+--+--+--+--+--+
```

where
- DE is 0 for decryption or 1 for encryption,
- CIPH selects the ciphering mode: 0 = "electronic code book" (EBC), 1 = "cipher feedback", and 2 = "cipher block chain" (CBC).

(Readers of the AMD documentation: note that the port configuration bits (bits 3-2) must be set as indicated to select single port mode. The Firefly interface does not use the auxilliary or slave ports.)

3.5.3 The COMMAND Register

The command register is an 8-bit, write-only register that specifies a particular operation that the chip is to perform. The operations group into two classes, data movement and ciphering control.

3.5.3.1 Data Movement Commands

Each data movement command moves a 64-bit quantity between the 68010 and one of the chip's four internal data registers (E, D, IVE, IVD). The INPUT and OUTPUT registers are used as intermediaries in these transfers. In the table below, the notation {x}k means "take the contents of register 'x' and decrypt it using electronic code book (EBC) and the key in register 'k'". The notation k{x} means "take the contents of register 'x' and encrypt it using electronic code book (EBC) and the key in register 'k'". The numeric codes are expressed in hexadecimal and are the values that software writes into the COMMAND register to cause the specified data movement operation.

	Direction			
	Store		Fetch	
Operation	Code	Function	Code	Function
Clear E	11	E := INPUT	--	--
Clear D	12	D := INPUT	--	--
Clear IVE	85	IVE := INPUT	8D	OUTPUT := IVE
Clear IVD	84	IVD := INPUT	8C	OUTPUT := IVD
Encrypt IVE	A5	IVE := {INPUT}E	A9	OUTPUT := E{IVE}
Encrypt IVD	A4	IVD := {INPUT}D	A8	OUTPUT := D{IVD}

In detail, then, software initiates a data movement operation by writing the COMMAND register with the desired value from the table above. Software then writes 0 to ADDR, which makes INPUT accessible at the "write register" location and OUTPUT accessible at the "read register" location. If the operation is a Store, software then writes eight bytes, one at a time, to the "write register" location. If the operation is a Fetch, software reads eight bytes, one at a time, from the "read register" location. In either case, the most significant byte is transferred first.

3.5.3.2 Ciphering Control Commands

There are five control commands, three that initiate ciphering, one that stops it, and one that resets the chip.

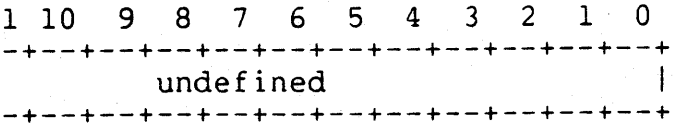
Start Encryption (hex 41), Start Decryption (hex 40), and Start (hex C0) all cause ciphering to commence, using the mode specified in the MODE register. Start Encryption first forces the DE bit of the MODE register to 1 (encrypt), Start Decryption first forces the DE bit of the MODE register to 0 (decrypt), and Start leaves the DE bit unaffected. The ciphering section of the encryption chip will then expect data, which is supplied as described in section 3.5.4. Stop (hex E0) causes the chip to cease expecting data, but ciphering operations that are underway complete normally. Reset (hex 00) forcibly resets the chip to its power-on state, which, from the 68010's point of view, is undefined.

3.5.4 Ciphering

The encryption chip is capable of "pipelined" operation, and when programmed appropriately can achieve close to its maximum throughput. We first describe the algorithm for ciphering a single 64-bit block of data, then show how to cipher multiple blocks using the pipelining feature.

To set up the chip for a ciphering operation, software first loads the MODE register (section 3.5.2) to select the desired ciphering mode. Software then loads E, D, IVE, and/or IVD as required by the selected mode. (Since these registers are unchanged by ciphering operations, they need only be loaded if they are not known to contain the proper values.) Software then writes the COMMAND register with one of the three ciphering control commands that initiate ciphering. Finally, software writes the ADDR register with the value 0 to make the INPUT and OUTPUT registers accessible. The chip is now ready to cipher data.

Two locations in supervisor data space are used to communicate and synchronize with the chip during ciphering. These are called the flags register and the data register. The flags register (location 19806) is a read-only, 16-bit, non-byte-swapped register that contains two status flags:



where
OP is 0 when the chip is ready to accept a 64-bit block of data

and 1 otherwise,

IA is 1 when the chip is ready to supply a 64-bit block of data and 0 otherwise.

The data register (location 19804) is a read-write, 16-bit, non-byte-swapped register that is used to move data to and from the chip. To cipher a 64-bit block, software first performs the set-up sequence described above, then waits until the OP flag is 0. It then writes four 16-bit words, one right after the other, to the data register. Software then waits for IA to become 1, after which it reads four 16-bit words, one right after the other, from the data register. In both cases, the first word transferred is the most significant one, and in each word the left byte is the more significant one. (More simply, the bytes are passed in the natural order for the 68010.) Finally, software writes the value 2 to the ADDR register (making COMMAND accessible), then writes the Stop operation code to COMMAND.

The preceding algorithm can be iterated with correct results, but better performance results if the chip's pipelining feature is exploited. Suppose that N blocks are to be ciphered. The software performs the following algorithm:

```
<execute set-up sequence for ciphering>
until OP = 0 do end;
write block 1 (4 words) to the data register
for i := 2 to N do
  until OP = 0 do end;
  write  block i (4 words) to the data register
  until IA = 1 do end;
  read block i-1 (4 words) from the data register
end;
until IA = 1 do end;
read block N (4 words) from the data register
<issue Stop command>
```

This algorithm reduces to the previous one for N = 1.

Note that OP and IA need not (should not) be tested during the four-word reads and writes; the timings of the 68010 and the chip interface are such that no explicit handshake is necessary. On the other hand, the 68010 is not required to read or write the data at any particular rate; the interface is properly buffered and synchronized internally in both directions. Thus, an interrupt in the middle of a four-word transfer is perfectly acceptable.

3.5.5 The STATUS Register

The STATUS register is an 8-bit, read-only, non-byte-swapped register that holds the status of the encryption chip. For the purposes of the Firefly interface, these status bits are largely subsumed by the flags register described in section 3.5.4. Therefore, we omit the description of all but two bits of the status register.

Software reads the STATUS register by writing ADDR with the value 2, then reading from the "read register" location. The format of the resulting value is

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
|       undefined       |MP|SP|AP|PA|LP|B |CP|SS|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where

PA is meaningful only when a Store Clear E or Store Clear D data movement operation is in progress and indicates the computed parity of each byte of a key as it is loaded (0 = odd, 1 = even).

LP is meaningful upon completion of a Store Clear E or Store Clear D data movement operation and contains the logical OR of the eight byte parity bits.

All other bits of the STATUS register are uninteresting to the 68010; they are explained in the AMD documentation.

3.6 Clock Chip

The clock chip includes time-of-day, alarm clock, and stable-storage facilities. All of its functions are software-controlled by accessing even locations 01A000-01A07E as halfword operands. Details of the semantics of these locations appear in the specification for the chip, MC146818, and are not repeated here. The programmer should be aware that the Firefly hardware uses a 32.768kHz external time-base, so the divider control bits DV2..0 in register A (location 01A014) should be set to 010. The Firefly has a battery (which automatically recharges) that maintains the contents of the chip's storage in the event of a power failure.

One of the clock registers (register B, location 01A016) has the interrupt enable bits for the various clock functions. The priority of the interrupt to the 68010, and therefore the autovector it uses, is determined by the control register (see section 3.4).

Data read from or written to the clock registers should appear in the least-significant byte of a 16-bit, non-byte-swapped halfword. Halfword transfers should be used to access the clock registers, even though the registers themselves are only a byte in width.

3.7 Power Failure

On a power failure, an interrupt occurs through the 68000's autovector for priority 6.

3.8 Disk Controller

The disk controller is the standard RQDX1 controller, accessed on the Q-bus. Its programming is adequately described in the following documents:

RQDX1 Controller Module User's Guide (EK-RQDX1-UG-001)
UDA50 Programmer's Documentation Kit (QP905-GZ), especially:
Storage System Unibus Port Description (AA-L621A-RK)
MSCP Basic Disk Functions Manual Version 1.2 (AA-L619A-RK)

The Firefly's Q-bus interface affects the addresses and layout of device registers, the layout of control blocks in main memory, and interrupt handling. See section 3.11.

3.9 Ethernet Controller

The Ethernet controller is the standard DEQNA module, accessed on the Q-bus. Its programming is adequately described in:

DEC Ethernet Q-bus Network Adaptor (M7504) Engineering Specification

(This is a company-confidential document. As of 8/11/84, there doesn't appear to be a publicly-available version.)

The Firefly's Q-bus interface affects the addresses and layout of device registers, the layout of control blocks in main memory, and interrupt handling. See section 3.11.

3.10 Display Controller

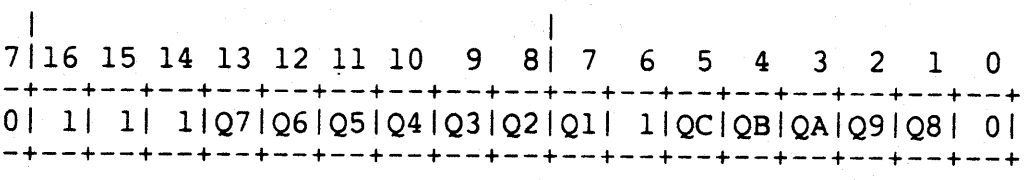
TBD

3.11 Q-bus Interface

This section describes the I/O interface between the processor-cache subsystem and the Q-bus.

3.11.1 Addressing

Q-bus locations (usually device registers) are accessed through supervisor data locations 01C000-01FFFF. The detailed mapping of an address is



where QC-Q1 are bits 12-1 of a PDP-11 I/O page address. (From a PDP-11's point of view, bits 17-13 of the address are 1's and bit 0 is 0.) For example, a PDP-11 expects the IP register of an RQDX1 controller to be at location 772150 (octal). In binary, this is

111 111 010 001 101 000

On the Firefly, this register would be addressed as

000 000 011 101 101 001 101 000 = 0000 0001 1101 1010 0110 1000

or location 01DA68 in supervisor data space.

Note that this mapping applies only to addresses of locations that are implemented on the Q-bus. Main memory addresses (i.e., for DMA) are handled differently; see section 1.2.2.

3.11.2 Data Transfers

All data transfers on the Q-bus consist of 16-bit halfwords. As each halfword passes between the Q-bus and the processor-cache subsystem, its bytes are swapped. That is, the bits on the Q-bus lines BDAL7..0

will appear in the most significant byte of a 68000 halfword, and the bits on BDAL15..8 will appear in the least significant byte of a 68000 halfword. Byte-swapping preserves PDP-11 byte addressing order, so that the order of bytes in main memory matches the order of bytes seen by a Q-bus device.

Note that byte-swapping applies to all data transfers (DMA or CPU) between the Q-bus and the processor-cache subsystem. This implies that the values that appear in device registers will have their bytes swapped on both reads and writes. Furthermore, some controllers, including the DEQNA and the RQDX1, fetch their control blocks using DMA transfers. Therefore, when constructing or interpreting the values that appear in device registers or device control blocks, software must swap the bytes in each 16-bit halfword.

One additional note: although it is possible for the processor to write a single byte to a Q-bus address, it is recommended that all processor references to Q-bus location be made using halfword (16-bit) operands.

3.11.3 Interrupts

All interrupts from Q-bus devices occur through the 68000's autovector for priority 4. The interrupt routine must read supervisor data location 01B000 to determine which device is actually requesting the interrupt. (More precisely, software can read any even halfword in the range 1B000-1BFFE to determine the device.) The read is required even if the interrupt routine doesn't care which device is involved. The 16-bit value returned is the interrupt vector offset (with its bytes swapped) that would have been used to interrupt a PDP-11; depending on the peripheral, this may or may not be under software control. The DEQNA and RQDX1 controllers both leave the choice of interrupt vector to the software.