C o m p u t i n g
S u r f a c e

Vector Processing Element (MK403) Users Guide

S1002–10M139.01

meiko

Meiko's address in the US is:

Meiko
130 Baker Avenue
Concord MA01742

508 371 0088
Fax: 508 371 7516

Meiko's address in the UK is:

Meiko Limited
650 Aztec West
Bristol
BS12 4SD

Tel: 01454 616171
Fax: 01454 618188

Issue Status:	Draft	
	Preliminary	
	Release	x
	Obsolete	

Circulation Control: *External*

Contents

Overview 1

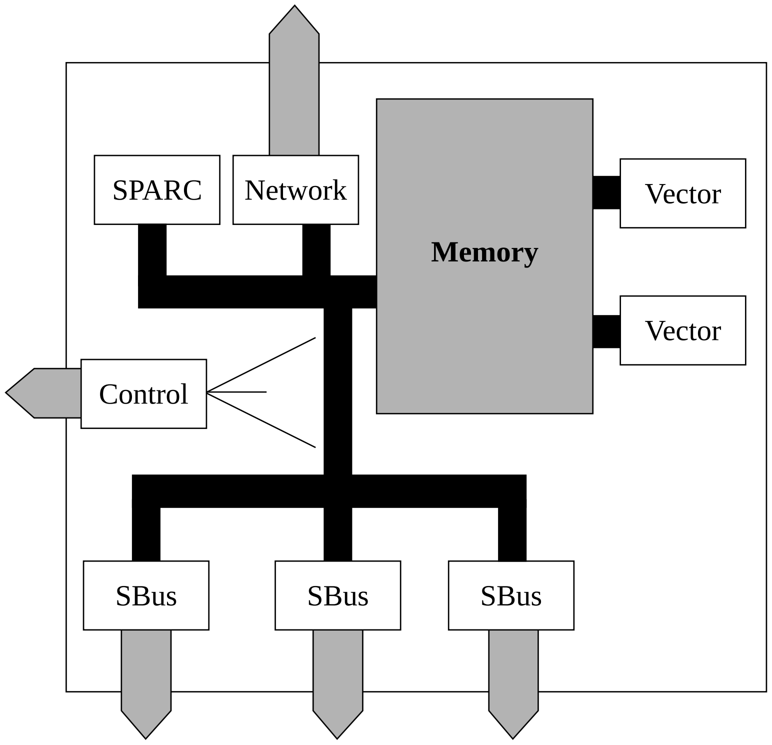
The MK403 Vector Processing Element offers high performance vector computing power and flexible I/O options.

The board design encompasses at the lowest level the principle design objectives for the CS-2, offering a scalable modular construction with easy upgrade options, a reliance on state of the art commodity components, leading edge proprietary network components, and support for system wide fault tolerance.

In outline the Vector Processing Element offers:

- Superscalar SPARC MBus module for scalar computing power and operating system services.
- Meiko Elan Communications Processor offering a high bandwidth, low latency interface to the CS-2 data network.
- Two Fujitsu Vector Processing Units (VPUs) on a plug-in module.
- Up to 128Mbytes of memory with 3 independent ports allowing simultaneous access by the SPARC MBus and the 2 Vector units.
- Three full size SBus slots for SCSI, Ethernet, or other third party options.
- Keyboard, mouse, and dual serial ports.
- Interface to the machine-wide control area network (CAN) offering remote diagnostic control and error logging facilities.

Figure 1-1 MK403 Board Overview



MK403 Board Description 2

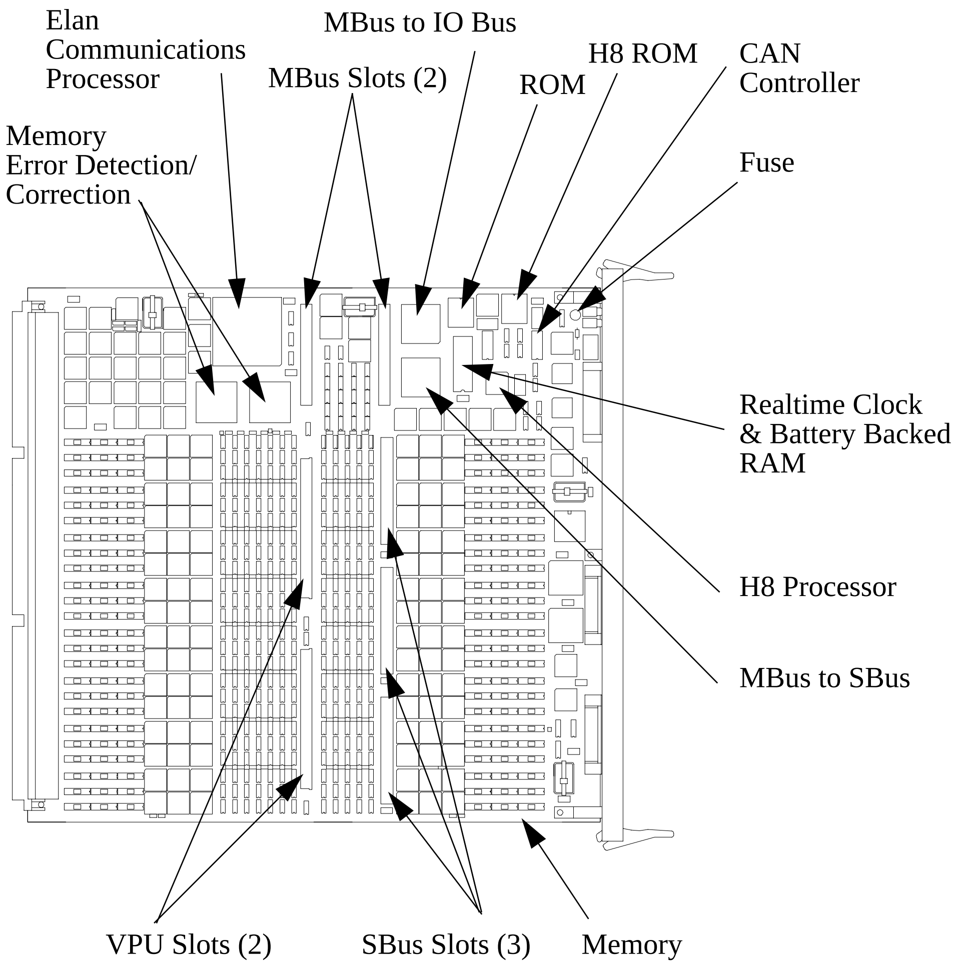
Access by the 3 processors to the memory system is via three independent memory ports. One port of the memory system is connected to an industry standard MBus interface giving access to the SPARC processor, the Elan Communications Processors, the MBus to SBus interface and its 3 SBus slots, and various other minor I/O devices on an I/O bus; in essence a SPARC workstation. Each of the remaining memory ports are used by the Fujitsu micro vector processors.

The 2 vector processing units on the MK403 are mounted on a single plug-in board which offers memory management and cache coherency with the SPARC caches. Cache coherency between the SPARC and vector processors has been achieved by defining a cache directory close to each vector processor; this cache directory attempts to replicate the contents of the SPARC cache tags/state, and is therefore potentially updated on every MBus cycle. Memory accesses by the vector processors that conflict with the SPARC cache cause the vector processor to stall until the appropriate MBus cycles have been generated.

Running throughout the whole CS-2 system is a control network (CAN) used to distribute status and configuration information, to provide remote control and diagnostics of all processors, and to create remote console connections to the processors. The MK403 has two interfaces to this network; one connected to the I/O bus (and thus providing a direct interface to the SPARC processors) and one via a dedicated micro-controller which provides board control.

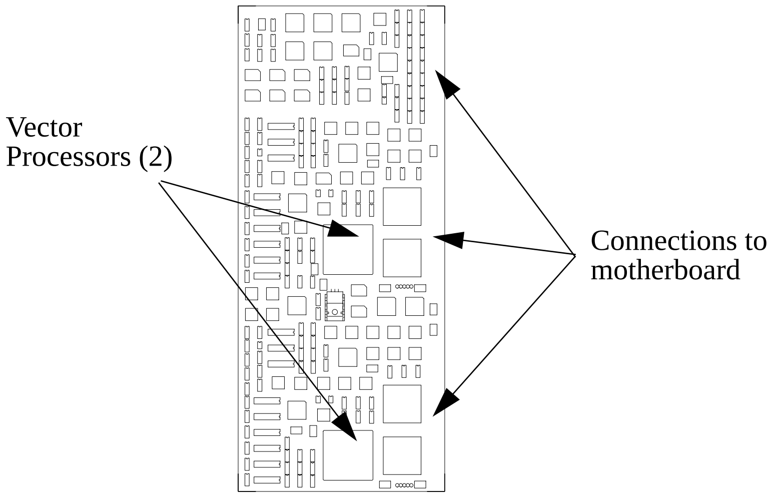
The major components and their placement on the MK403 motherboard are shown in Figure 2-1.

Figure 2-1 Board Schematic Showing Major Components (MK403)



MBus, SBus, and Vector Processor plug-in modules not shown.
Components are also fitted on the reverse of this board.

Figure 2-2 Board Schematic Showing Major Components (MK534)



MBus

Two full size MBus sites are provided. One of these is used by the plug-in vector processors, the other by a uni-processor SPARC MBus module.

The MBus is fully level 2 compliant and runs at 40MHz. The SPARC processor shares the MBus with the Elan Communications Processor, the MBus-to-SBus interface, the I/O bus controller, and vector processors (but note that the vector board uses the MBus interface solely for memory management and cache coherency with the SPARC, and that direct memory accesses are made via memory ports that are independent of the MBus). The allocation of MBus id's is:

- MBus id 0 is the I/O bus controller.
- MBus id 4 is the MBus to SBus controller.
- MBus id 6 is the Elan Communications Processor.
- MBus id's 8 and 9 are MBus slot 0.
- MBus id's 10 and 11 are MBus slot 1.

MBus slot 0 is always used by the plug-in vector board. MBus slot 1 is always used by a uni-processor SPARC module, currently either a TI Viking (with or without second level cache) or a ROSS Pinnacle.

ROSS Pinnacle Module

The MK403 may be fitted with a uni-processor Pinnacle module which includes second level cache.

The Pinnacle MBus module is built upon a tightly coupled set of three ROSS devices: the RT620 HyperSPARC CPU, the RT625 cache controller, memory management, and tagging unit (CMTU), and the RT627 cache data units (CDUs).

Features of the RT620 CPU are:

- SPARC version 8 conformance.
- 90MHz clock rate.
- 4 execution units offering parallel execution of major instruction types: Load/Store, Branch/Call, integer and floating point units.
- Dual instruction fetch per clock cycle.
- 8Kbyte 2-way set-associative on-chip instruction cache.
- Instruction pipelining including a cache stage to accommodate the latency for second level cache accesses on data. Simultaneous accesses to on-chip and second level cache for each instruction fetch.
- High bandwidth 64bit Intra Module Bus (IMB) provides the interface between the CPU and the second level cache. Use of second level cache decouples the processor clock rate from the lower MBus clock rate.

Key features of the RT625 (CMTU) and RT627 (CDU) devices are:

- Full level 2 cache-coherent MBus compatibility.
- Each CDU has integral 16Kbytes x 32bit SRAM. MBus modules use either 2 or 4 CDUs for 128Kbyte or 256Kbyte second level direct-mapped cache.

- Physical cache tagging with virtual indexing allow the cache coherency logic to determine snoop hits and misses without stalling the CPU's access to the cache.
- Both copy-back and write-through cache modes supported.
- 32 byte read buffer and 64 byte write buffer for buffering the 32 byte cache lines in and out of the second level cache.
- SPARC reference MMU offering 64 entry, fully set-associative TLB with 4096 contexts.

Texas Instruments Viking Module

Two variants of the TI Viking MBus module are available. One contains a Viking SPARC processor with direct connection to the MBus. The second includes a Viking processor with additional Cache Controller and 1Mbyte of second level external cache (E-cache).

Key features of the TMS390Z50 SuperSPARC are:

- SPARC Version 8 conformance.
- 3 instructions per cycle, instruction pipelining, 150MIPs peak performance.
- SPARC Integer Unit.
- SPARC Reference MMU. Cached translation lookaside buffers (TLBs). 32bit virtual addresses, tagged with a 16bit context (65,536 contexts), map to 36bit physical addresses.
- Single and Double precision FPU. Tightly coupled to the integer execution pipeline and allowing one floating-point operation and one memory reference to be issued in each clock cycle. The FPU maintains a 4 entry FIFO queue for FP operations.
- 20Kbyte instruction cache, 16Kbyte data cache. The instruction cache is 5-way set associative, physically addressed, and non-writable. The data cache is 4-way associative and physically addressed. Both cache's are coherent with each other and with optional E-cache or MBus. Without E-cache the instruction and data caches operate in write-through mode; otherwise they are copy-back mode.

- Store buffer. A FIFO queue of 8 entries, each 64bits, decouples the instruction execution pipelines from the E-cache or MBus.
- Multiprocessor cache coherent support; highly pipelined and non-multiplexed VBus interface to optional external cache and the TMS390Z55 Cache Controller, or direct connection to the MBus.
- Prefetch buffer.
- Support for system and software debugging including hardware breakpoint.

The external cache is managed by the TMS390Z55 Cache Controller. Key features of this device are:

- Built in support for cache coherent multiprocessing; multiple Viking modules can share a single MBus and remain fully cache coherent.
- High performance VBus interface with the SPARC processor; decouples the SPARC processor from the MBus clock speed allowing higher processor performance. Reduced MBus traffic reduces contention when multiple processors share a single MBus.
- E-cache is direct mapped, copy-back, and unified: there is a single cache location where a particular byte of the physical address space can reside in the cache (direct mapping); writes by the main processor into the E-cache do not propagate to main memory until the cache is flushed or replaced (copy-back); both instructions and data are supplied to the processor from the same cache (unified).
- 1Mbyte SRAM cache.

Vector Processing Elements

Two vector processors are mounted on a plug-in board that is fitted to MBus slot 0 and the two VPU slots. Included on the vector board is the memory management and cache coherency logic and associated memory. The MBus interface is used by the memory management and cache coherency to remain synchronised with the SPARC processor, whereas direct memory accesses are via the dedicated VPU slots.

Memory Management

The vector processors operate with their address translation units disabled (Real Mode) and instead use an external memory management unit which allows better handling of address translation faults.

Each vector processor references a single-entry TLB and a 64K×16bit MMU RAM (128Kbyte). The MMU RAM contains 16K entries for each of 4 contexts, each entry consists of a 12 bit physical page number, 2 bit read/write flags, and 2 bit reference/modify flags. With a page size of 256Kbytes the 16K entries for each context gives a virtual address space of 4Gbytes. A separate register stores the current context and is used to provide the 2 high order bits into the MMU RAM. The MMU RAM is accessible from the MBus and is kept in-step with the SPARC's page tables by modified page-in code running in the operating system kernel. The single-entry TLB contains a single translation from logical page number to a physical page number, and a two bit reference/modify field.

Memory references by the vector processor are first checked against the TLB. If the logical page number matches, and the page has already been referenced (for a vector process read) or modified (for a vector process write), then the physical page provided by the TLB is used. If the TLB cannot provide the translation the logical page number and the context register are used as an index into the MMU RAM. If the corresponding entry in the MMU RAM has the sufficient access privileges, and the reference/modify bits are set appropriately, then the physical page is read from the MMU RAM and the TLB is updated with this entry. Translations that cannot be resolved by the MMU RAM (access permissions or missing pages) are notified to the SPARC by a level 5 interrupt, which may fault-in a new page or kill the vector process.

Cache Coherency

The vector processing board includes cache coherency logic which allows the vector processors (which have no cache capability themselves) to maintain consistency with the SPARC caches.

Each of the vector processors has an associated cache directory which is used to replicate the state of the SPARC's caches. Entries within the cache directory identify modified cache lines that are held by the SPARC; they are maintained by snooping the activity on the SPARC's MBus. Every memory address generated by the vector processor is checked against the cache directory, and if there is a conflict with the SPARC cache the appropriate MBus operations are generated (one of Invalidate or Coherent-Read-with-Invalidate followed by a Writeback).

The vector processor may be used alongside either the ROSS Pinnacle or Texas Instruments SPARC modules, and must therefore accommodate the different caches used by these modules. The TI caches are indexed by physical address, whereas the Pinnacle are indexed by virtual address. The TI E-cache and Pinnacle caches are direct mapped, whereas the TI integral cache is 4-way set associative. The configuration of the vector processors' cache directories is selected by hardware jumpers and is not field configurable.

When used with a TI Viking processor each cache directory is configured as 8Kbyte×10bits of cache tag information, 8Kbytes×8bits of shared and dirty information, and 8Kbytes×1bit of flags used to show that the access was initiated by a set-associative cache. Each cache block consists of 4 lines, each line consists of 32 bytes. Each line has its own shared and dirty bits, but the tag field is shared by the whole block. There is (8K blocks) × (4lines/block) × (32bytes/line) = 1Mbyte of cache represented by this configuration.

When used with the Pinnacle each cache directory is configured as 4Kbytes×10-bits of tag information, 4Kbytes×4bits of shared and dirty information, and 4Kbytes×1bit of flags used to show that the access was initiated by a set-associative cache. Each cache block consists of 2 lines, each line consists of 32bytes. Each line has its own shared and dirty bits, but the tag field is shared by the whole block. There is (4K blocks) × (2 lines/block) × (32bytes/line) = 256Kbytes of cache represented by this configuration.

While the vector processors are idle all MBus operations are snooped into the cache directories. While the vector processors are busy and accessing store all MBus operations are queued in an inbound buffer; the queue can only be serviced when the vector processors are idle, during compute time, or during recovery time between a LDA/STA and the start of the next one. In order to guarantee coherency the vector processors will not start a new series of operations until the queue has been emptied.

Fujitsu MB92831 Micro Vector Processor

The MK403 uses 2 Fujitsu micro vector processors (µVPs) in a co-processor shared memory configuration (the vector processors share the same memory as the SPARC but access it via independent memory ports).

Key features of the Fujitsu µVP are:

- Peak performance 100MFLOPS double precision, 200MFLOPS single precision per processor. External memory bandwidth 400Mbytes/s.
- Rich command set offering 252 vector commands, 57 scalar commands, 9 general control commands. Includes: command load and branch commands; vector load or store commands using stride values or indirect addressing (“scatter-gather”); floating-point, integer, and logical vector and scalar commands; maximum and minimum value search commands; a full range of vector and scalar comparison commands.
- Pipelined memory access allows the vector processor to output memory references before it needs them.
- Compliance with IEEE 754-1985 standard for binary floating-point arithmetic. 32bit and 64bit floating point data types.
- Pipeline execution allows the vector processor to perform the same operation repeatedly for all data items in a vector (or just once for a scalar). Parallel pipelines execute more than one command at a time. Chaining allows the operation in one pipeline to use the result of another.
- Six pipelines: Add (addition, subtraction, comparison, and data conversion), Multiply, Divide, Graphic, Mask, Load-Store.
- Host processor register interface. Commands are written to the 256 command buffer, and parameters to the scalar registers. Program execution is initiated by the start register, and polled for completion via the vector busy register. An abort register allows the host to halt the vector processor.
- 8Kbyte vector registers, 128byte scalar registers, and 64byte mask registers. Vector registers are partitioned into 4 banks, each bank can be read or written on each clock cycle. The banks may be accessed concurrently by 4 pipelines.

SBus Interfaces

The MBus to SBus interface supports up to 5 SBus devices, but on the MK403 only three are used and these map directly to the 3 SBus slots. SBus slot 0 is nearest the MBus processor slots.

The SBus runs at a clock speed of 20MHz.

Memory Configuration

The Superscalar SPARC processors and Elan communication processor are connected to a standard 40 MHz MBus. The vector processors and MBus are connected to a 16 bank memory system, each bank providing 64 bits of user data (78 bits including error checking and correction, implemented using 20 by 4 bit DRAMs with two bits unused). Error detection and correction is implemented on each half word (32 bits), allowing write access to 32 bit (ANSI-IEEE 754–1985 `single`) values to be performed at full speed, without requiring a read modify write cycle.

Each bank of memory maintains a currently open DRAM page within which accesses may be performed at full speed. This corresponds to a size within the bank of 8 Kbytes, giving 128 Kbytes total for the 16 banks. When an access is required outside the currently open page a penalty of 6 cycles is incurred to close the previous page, and open the new one.

Refresh cycles are performed on all banks within a few clock cycles of each other, thus allowing the cost of re-opening the banks to be pipelined (since the VP can issue four addresses before stalling for the data from the first), and reducing the overhead of refresh to a few percent of memory bandwidth.

The memory system is clocked at the same speed as the μVP processors (50 MHz), and accesses from the 40 MHz MBus are transferred into the higher speed clock domain. When accessing within an open page each memory bank can accept a new address every two cycles (40 ηs), and replies with the data four cycles (80 ηs) later, giving a bandwidth of 8 bytes every two cycles (40 ηs), that is 200 Mbytes/s. Since there are 16 banks, the total memory system bandwidth is thus 3.2 Gbytes/s.

Each μVP can issue a memory request every cycle (20ηs), and can issue 4 addresses before it requires data to be returned. In the absence of bank contention (which will be discussed below), after a start up latency of four cycles, these requests can be satisfied as fast as they are issued, giving each μVP a steady state bandwidth of 8 bytes every 20ηs, that is 400 Mbytes/s.

Since each bank can accept a new address every two cycles (40ηs), but the μVP can generate an address every cycle (20ηs) there is the possibility of bank contention if the μVP generated repeated accesses to the same bank. With a simple linear mapping of addresses to banks, this would occur for all strides which are multiples of 16 (for 64 bit double precision accesses). Such an access pattern would then see only one half of the normal bandwidth, that is 200 Mbytes/s. All other strides achieve full bandwidth.

To ameliorate this problem as well as allowing the straightforward linear mapping of addresses to banks, Meiko also provide the option (through the choice of the physical addresses which are used to map the memory into user space) of scrambling the allocation of addresses to memory banks. The mapping function has been chosen to guarantee that accesses on “important” strides (1, 2, 4, 8, 16, 32) achieve full performance. Access on other strides may see reduced performance, but there are no strides within the open pages which see the pathological reduction to one half of the available bandwidth.

IO Bus

The IO bus is a slave-only bus used for the connection of minor peripherals to the MBus. The following devices are connected to this bus:

- A 512Kbyte EPROM holding bootstrap and diagnostic programs. The bootstrap code initialises the hardware devices, initial page table construction, and the booting of the Unix kernel.
- A realtime clock module with battery backed SRAM. This holds configuration information and node fault logs, including the log of uncorrectable memory errors. The realtime clock provides year, month, day, hour, minute, and second times.
- Dual DUART devices; one for connection of keyboard and mouse, the other for two general purpose serial ports. In the absence of a keyboard the bootstrap code in the EPROM will usually direct console I/O via serial port A. The serial

ports are clocked at 4.9152MHz, with a working capability of 38.4KBaud. Both serial ports share the same 25-way front panel connection; port A has full synchronous/asynchronous operation and a full complement of modem control lines; port B has a limited set of control lines and is asynchronous only.

- Interrupt controller; this uses registers to mask out certain types of interrupt to relieve the SPARC processor from unnecessary interrupt loading, and to share the handling between the SPARC and the Elan. The programming of the interrupt controllers is handled by the kernel device drivers.
- Periodic interrupt timer; used for maintaining the kernel clock and for kernel profiling.
- A CAN device provides the SPARC with an interface to the machine-wide control area network (CAN). The SPARC processor writes diagnostic information to this bus, and can also act as an X-CAN or G-CAN router. CAN routers transfer data between two levels of the CAN network; an X-CAN router handles transfers between the modules in a Cluster, and a G-CAN router handles transfers between Clusters. The configuration of a SPARC as a router will cause it's CAN device to generate numerous level 2 interrupts which will impact on processor performance.

Board Control Processor

The MK403 board uses an Hitachi H8/534 micro-controller (commonly referred to by Meiko as the H8) to perform basic node control functions. This controller is a single-chip 16bit RISC microcomputer with integral 2KBytes RAM, 32Kbytes EPROM, 16bit RISC CPU, and a number of I/O ports and timers.

The H8 processor runs independently of the other processors on the board and is used solely for control and diagnostic purposes. It has its own interface to the CAN bus via the second of the board's CAN interface devices. This processor receives diagnostic messages, via the CAN bus, from the local SPARC processor, and interprets incoming control messages, such as board reset, console connections, and network configuration.

Using the MK403 3

This chapter describes the usage of the MK403 in terms of its installation, hardware interfaces, and field serviceable components.

Installation

The MK403 is designed for use solely in a CS-2 Processor Module. The Processor Module supplies the board's power, cooling, and connection to the CS-2 data and control networks. The MK403 is fitted into one of the four vertical board slots behind the Processor Module's removable front panel.

Warning – You must disconnect the power fr om the Pr ocessor Module before removing or installing pr ocessor boards.

Warning – The board may be fitted with fragile or static sensitive devices. You must handle with car e and observe anti-static pr ecautions.

Removing the Module's Front Panel

The module's front panel is held in position by four clips, one in each corner. To remove the panel pull firmly away from the module.

The module's LED display is fitted to the module by two 50-way connectors. To remove the LED display pull firmly away from the module.

Use the reverse procedure to install the LEDs and front panel.

Installing the Processor Board

Insert the board so that it fits into the guide rails at the top and bottom of the module's board rack, ensuring that the component side is to the left (viewed facing the module). Gently push the board squarely on its front panel. Before pushing the board fully into position fold back the levers at each end of the front panel so that they are at 90^{o} to the board; now push the board (while holding the levers) until the base of the two levers is touching the card cage. To lever the board into its final position push both levers until they lie flat on the board's front panel. Secure the board by tightening the two captive screws.

Use the reverse procedure to remove the board.

Warning – You should take care not to damage the connectors at the rear of the board and on the module's backplane. Ensure that the board mates squarely with the module's backplane.

Warning – When removing or installing a board you should take care not to damage the RFI (copper) seals along the edge of the board's front panel.

Warning – To maintain proper circulation of cooling air and to conform to RFI regulations all board slots must be fitted with a processor board or blanking plate.

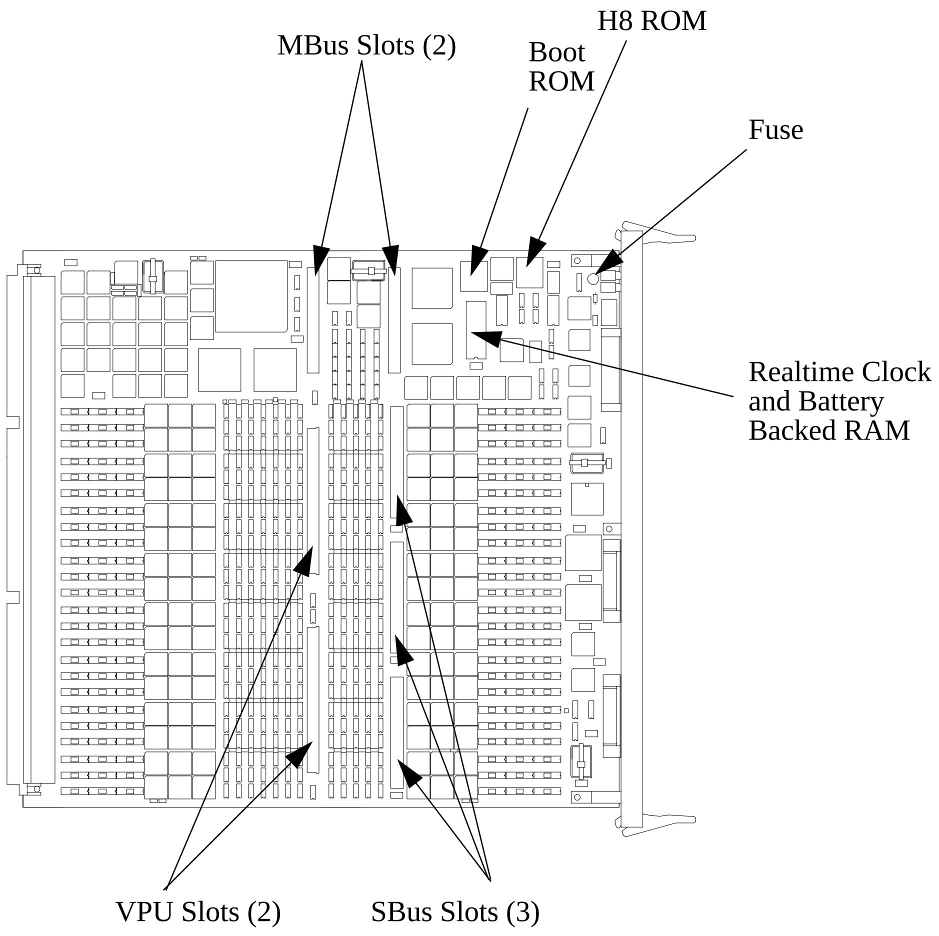
Field Serviceable Components

The MK403 has the following field upgradeable components (see Figure 3-1):

- Superscalar SPARC processor module fitted to MBus slot.
- Three SBus slots.
- Two vector processor slots.
- Boot ROM.

- H8 ROM.
- Realtime clock and non-volatile RAM module.
- Fuses.

Figure 3-1 MK403 Components

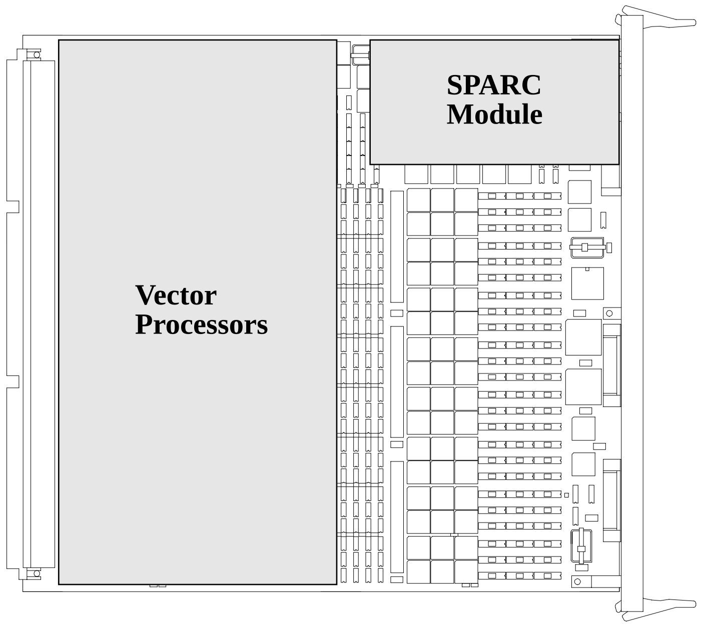


Processor Modules

The 2 vector processor are mounted on a single plug-in board (MK534) which is connected to MBus slot 0 and both of the VPU slots. The board is fixed in place by 4 M3 screws.

A single uni-processor SPARC module is connected to MBus slot 1. The MK403 motherboard includes support for either Texas Instruments Viking or ROSS Pinnacle module (although Pinnacle modules are typically used). The type of SPARC module is set when the MK403 board is manufactured and may not be reconfigured on site.

Figure 3-2 Position of the Vector and SPARC Modules

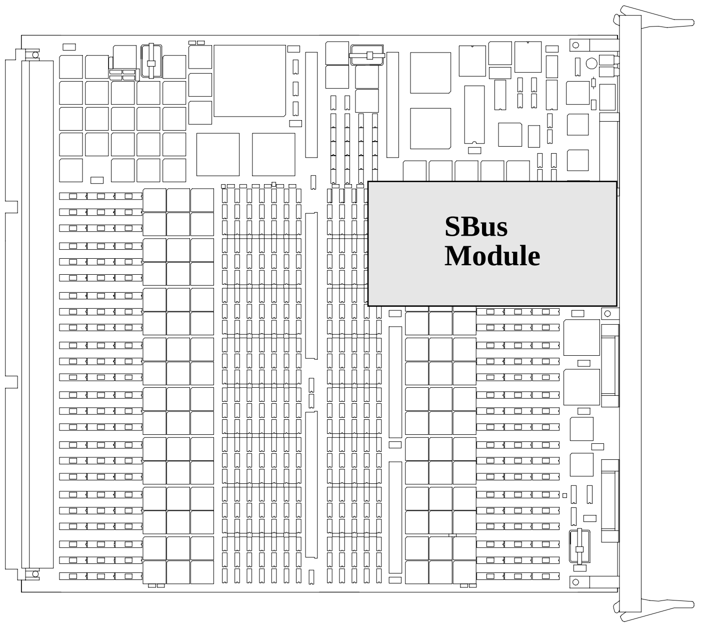


Installing SBus Modules

Three SBus slots are provided and these may be fitted with standard SBus modules; these are plugged into the SBus connectors and secured with two M3 screws. When using SBus cards that have external connections, for example a graphics card, remove the appropriate panel from the front of the MK403 — the panel is held in place by two small screws.

SBus devices are numbered from 0 to 2, device 0 being next to the processor slots (see Figure 3-3).

Figure 3-3 Position of SBus Module 0



SBus SCSI Cards

When using SBus SCSI cards to connect to disk devices within the Processor Module you must connect the front-panel output from the SBus card to the SCSI-A connector on the MK403 motherboard using Meiko cable 60-CA0217-1T.

You should note that the Processor Module's disks may be interconnected in one of three ways: each of the 4 disks connected to a separate processor board via SCSI bus A, disks connected in pairs to SCSI bus A on board's 0 and 1, or all disks connected to SCSI bus A on board 0.

Boot ROM and H8 ROM

Both of these ROMs may be upgraded from time to time. They are held in sockets and are readily replaced. Note the position on pin 1 before removing the old device (usually marked by a dot on the packaging).

Realtime Clock and Battery backed RAM

The real time clock and non-volatile RAM device is held in a DIL socket and is easily replaced. Before removing the old device note the position of pin 1 (usually marked by a dot on the packaging). Note that the information within the RAM can only be restored by Meiko's engineers.

Warning – This device contains lithium batteries; never dispose of this device in a fire or attempt to dismantle.

Fuses

There is one fuse on the MK403 motherboard to protect the keyboard/mouse circuit. This is a 250mA quick blow fuse, Meiko part number 22-FU400-02E250.

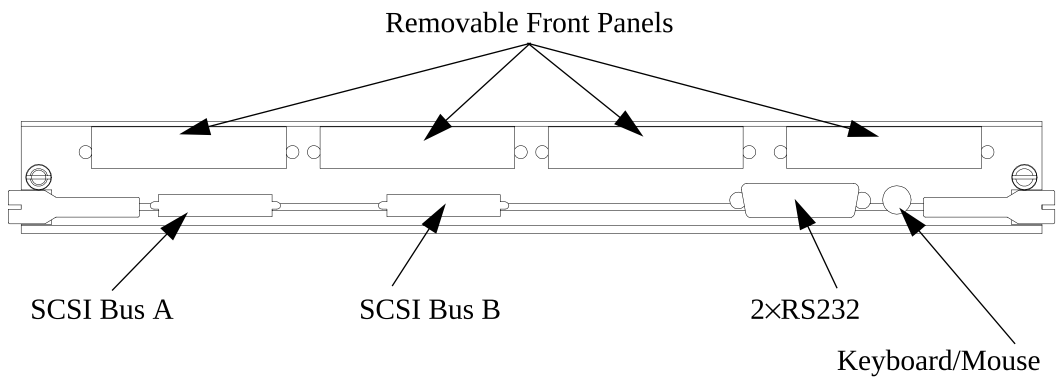
External Connections

External connections are provided for a keyboard/mouse (8 pin circular socket), serial interfaces (2 channels provided by one 25-way D-type connector), and two independent SCSI buses (each via a 50-way miniature connector).

Front Panel Connections

Removable panels provide access to connectors on the optional SBus boards.

Figure 3-4 MK403 Front Panel Connections



RS232 Connections

The two RS232 channels are output via a single 25-way connector. The connections are as shown in the following tables. Signal ground is pin 7, chassis ground in pin 1.

Table 3-1 RS232 Channel A Pinout.

Signal	Input/Output	Pin number
TXD	Out	2
RXD	In	3
RTS	Out	4
CTS	In	5
CSR	In	6
DCD	In	8
DB	In	15

Table 3-1 RS232 Channel A Pinout.

Signal	Input/Output	Pin number
DD	In	17
DA	On	24
DTR	On	20

Table 3-2 RS232 Channel B Pinout.

Signal	Input/Output	Pin number
TXD	Out	14
RXD	In	16
RTS	Out	19
CTS	In	13
DCDB	In	12

External Indicators

Two LEDs (one green, one amber) are included on the board's front panel. The green LED is the heart beat from the board's (H8) CAN controller. The amber light illuminates each time the CAN controller transmits on the CAN bus. Both should flash steadily. These indicators are also displayed on the module's LED display.

The green LED flashes at a slow steady rate (once per second) when operating normally. A quicker flash rate (2×normal) indicates that the board's SPARC processor is not responding; a very quick flash rate (3×normal) indicates that the H8 processor on the module's controller is not responding.

Each processor board within a processor module controls a 4×4 matrix of red LEDs on the module's front panel. The MK403 displays a random pattern on these when running the Boot ROM. When Solaris has been booted a circulating pattern is displayed. The pattern can be changed by user programs and various system commands and daemons.

Address Maps A

MBus Address Maps

This section gives the mapping of memory and peripherals into the MBus physical address space. All addresses are in hexadecimal, all locations are word wide unless otherwise stated in the notes. The following notes are associated with some of the items in the tables:

1. These locations are byte wide and are mapped into all 4 bytes of a word. Care should be taken to generate correct bytewide accesses to the least significant byte of the word in order to maintain future compatibility.
2. These locations are halfword wide and are mapped into both halfwords of the word. Care should be taken to generate correct halfword accesses to the least significant halfword of the word in order to maintain future compatibility.
3. These locations are bytewide memory, mapped into contiguous byte locations. Word or halfword accesses will be automatically mapped into several successive bytewide accesses.
4. These locations are byte sized registers which are only mapped into the least significant byte of the word. Accesses of larger than one byte will be mapped into several IO Bus transactions, but only one (the least significant byte access) will actually select the device. Halfword or word accesses will have no ill effects but will waste MBus bandwidth and should be avoided.

5. These locations are halfword sized registers which are only mapped into the least significant halfword of the word. Accesses of larger than one byte will be mapped into several byte transactions, but only two (the least and next-to-least significant byte) will actually select the device. Word accesses will have no ill effects but will waste MBus bandwidth and should be avoided.
6. These locations form a doubleword register.

MBus Address Map Summary

The following table summarises the MBus memory space usage by the board's principle components:

MBus Address	**Usage**
`000000000 to 007ffffff`	128MB Memory (normal mapping, no coherency between µVP and SPARC).
`020000000 to 027ffffff`	128MB Memory (scrambled mapping, no coherency between µVP and SPARC).
`200000000 to 207ffffff`	128MB Memory (normal mapping, enforce coherency between µVP and SPARC).
`220000000 to 227ffffff`	128MB Memory (scrambled mapping, enforce coherency between µVP and SPARC).
`100000000 to 10000ffff`	µVP 0 (Supervisor Access).
`100010000 to 10001ffff`	µVP 0 (User Access).
`120000000 to 12000ffff`	µVP 1 (Supervisor Access).
`120010000 to 12001ffff`	µVP 1 (User Access).
`160000000 to 16000ffff`	µVP Broadcast (Supervisor).
`160010000 to 16001ffff`	µVP Broadcast (User).
`1c0000000`	MBus EDC Error Data.

MBus Address	Usage
`1c0000008`	MBus EDC Error Diagnosis.
`1c0000010`	MBus EDC Clear.
`800000000 to 81fffffff`	µVP cache directory.
`900000000 to 9A0000018`	µVP status, MMU etc.
`e00000000 to e0fffffff`	SBus Slot 1.
`e10000000 to e1fffffff`	SBus Slot 2.
`e20000000 to e2fffffff`	SBus Slot 3.
`e60000000 to e600001ff`	MBus-to-SBus TLB.
`e70000000 to effffffff`	SBus Reserved.
`ff0000000 to ff00fffff`	Boot ROM.
`ff0100000 to ff01fffff`	Serial Port.
`ff0200000 to ff02fffff`	Keyboard and Mouse Port.
`ff0300000 to ff03fffff`	Real Time Clock and 8K SRAM.
`ff0700000 to ff07007ff`	Node Reset, IRQ.Pal’s etc.
`ff0700800 to ff0700fff`	CAN.
`ff0701000`	IRQ 0 and Timer 0.
`ff0701600`	Async. Error Pending 0.
`ff0701800`	EDC Error Mask 0.

MBus Address	Usage
`ff0701a00`	SPARC Reset Flag.
`ff0702000`	IRQ 1 and Timer 1.
`ff0702600`	Async. Error Pending 1.
`ff0702800`	EDC Error Mask 1.
`ff0703600`	LED’s.
`ff0704000 to ff07042ff`	µVP 0 Reset, Busy.
`ff0705000 to ff07052ff`	µVP 1 Reset, Busy.
`ff0800000 to ff08fffff`	STDIO Control Registers.
`ff4fffff0 to ff4ffffff`	M2S Control Registers.
`ff6f80000 to ff6ffffff`	ELAN.
`ff8000000 to ff9ffffff`	MBus Slot 0.
`ffa000000 to ffbffffff`	MBus Slot 1.

DRAM and SBus Slots

MBus Address	Usage	Rd/Wr	Note
`000000000 to 007ffffff`	128MB Memory (Normal, no coherency between µVP and SPARC).	RW	
`020000000 to 027ffffff`	128MB Memory (Scrambled, enforce coherency between µVP and SPARC).	RW	
`200000000 to 207ffffff`	128MB Memory (Normal, enforce coherency between µVP and SPARC).	RW	
`220000000 to 227ffffff`	128MB Memory (Scrambled, no coherency between µVP and SPARC)	RW	

MBus Address	Usage	Rd/Wr	Note
`e00000000` to `e0fffffff`	SBus Slot 1.	RW	
`e10000000` to `e1fffffff`	SBus Slot 2.	RW	
`e20000000` to `e2fffffff`	SBus Slot 3.	RW	

μVP Cache Directory Mappings

The cache directories for both of the vector processor are mapped into the MBus address space.

MBus Address	Usage	Rd/Wr	Note
`8uxxy0000 to 8uxxzffc0`[a]	μVP cache directory (write both, read μVP0). `x` = don't care, `u` = (binary) `00xx`, `y` = (binary) `xx00`, `z`= (binary) `xx11`. Bits 29,26:18 of address are used as data inputs to the RAMs, not as RAM index.	RW	
`8vxxy0000 to 8vxxzffc0`	μVP1 cache directory (read only). `x` = don't care, `u` = (binary) `00x1`, `y` = (binary) `xx00`, `z`= (binary) `xx11`.	R	

a. The cache directory MMU mappings should be such that Logical Address equals Physical Address, or the cache directories should be accessed by bypassing the SPARC MMU. This allows the addresses to behave the same, regardless of whether we are configured as a Pinnacle or a Viking.

μVP Status and MMU Mappings

MBus Address	Usage	Rd/Wr	Note
`900000000 to 90001fff8`[a]	μVP MMU (write both, read μVP 0)	RW	
`910000000 to 91001fff8`	μVP MMU (read-only of μVP 1)	R	
`920000000`	MMU_ADDRESS0	R	
`920000000`	LD_PF_INBOUND (note same address as above!)	W	
`930000000`	MMU_ADRESS1	R	
`930000000`	LD_PF_OUTBOUND (note same address as above!)	W	
`980000000`	MMU_CONTROL	W	
`980000008`	MMU_MASK_FAULT	W	
`980000010`	MMU_CLEAR	W	
`980000018`	KILL_uVP	W	
`980000028`	CLEAR_INVALIDATE	W	
`9A0000000`	uVP Status	R	
`9A0000010`	SET_GATE_AND_SYNCH	W	
`9A0000018`	SET_GATE_AND_SEPARATE	W	

a. The MMU RAM mappings should be such that Logical Address equals Physical Address, or the MMU RAMS should be accessed by bypassing the SPARC MMU. This allows the addresses to behave the same, regardless of whether we are configured as a Pinnacle or a Viking.

BootRom, Serial Ports, Miscellaneous

MBus Address	Usage	Rd/Wr	Note
`ff0000000` to `ff007ffff`	BootRom (512KByte).	R(W)	3
`ff0040000` to `ff007ffff`	Unused for 256KByte Flash ROM (Boot ROM Echo)		
`ff0100000` to `ff010000f`	Serial Port Controller		
`ff0100000`	Control Registers port B	RW	3
`ff0100002`	Data Buffer port B	RW	3
`ff0100004`	Control Registers port A	RW	3
`ff0100006`	Data Buffer port A	RW	3
`ff0100010` to `ff01fffff`	Unused (Serial Port Echos)		
`ff0200000` to `ff020000f`	Keyboard and Mouse Port Controller		
`ff0200000`	Control Registers mouse port	RW	3
`ff0200002`	Data Buffer mouse port	RW	3
`ff0200004`	Control Registers keyboard port	RW	3
`ff0200006`	Data Buffer keyboard port)	RW	3
`ff0200010` to `ff02fffff`	Unused (Keyboard and Mouse Port Echos)		
`ff0300000` to `ff0301fff`	Real Time Clock module and 8KByte SRAM	RW	3
`ff0302000` to `ff03fffff`	Unused (RTC Echos)		
`ff0400000` to `ff06fffff`	Unused (MBus Error)		
`ff0700000`	Node Reset Request	RW	4
`ff0700004` to `ff07001ff`	Unused (Echos)		

MBus Address	Usage	Rd/Wr	Note
`ff0700200`	MBus Grant	R	4
`ff0700204` to `ff07003ff`	Unused (Echos)		
`ff0700400`	Physical Slot Identifier	R	5
`ff0700404` to `ff07005ff`	Unused (Echos)		
`ff0703600`	LED Bargraph	RW	5
`ff0703604` to `ff07007ff`	Unused (Echos)		

Control Area Network Interface

MBus Address	Usage	Rd/Wr	Note
`ff0700800`	CAN - Control Register	RW	4
`ff0700804`	CAN - Command Register	W	4
`ff0700808`	CAN - Status Register	R	4
`ff070080c`	CAN - Interrupt Register	R	4
`ff0700810`	CAN - Acceptance Code Register	RW	4
`ff0700814`	CAN - Acceptance Mask Register	RW	4
`ff0700818`	CAN - Bus Timing Register 0	RW	4
`ff070081c`	CAN - Bus Timing Register 1	RW	4
`ff0700820`	CAN - Output Control Register	RW	4
`ff0700824`	CAN - Test Register		
`ff0700828`	CAN - TXBuf Identifier	RW	4
`ff070082c`	CAN - TXBuf RTR Data Length code	RW	4
`ff0700830`	CAN - TXBuf Data Byte 1	RW	4
`ff0700834`	CAN - TXBuf Data Byte 2	RW	4
`ff0700838`	CAN - TXBuf Data Byte 3	RW	4
`ff070083c`	CAN - TXBuf Data Byte 4	RW	4
`ff0700840`	CAN - TXBuf Data Byte 5	RW	4

MBus Address	Usage	Rd/Wr	Note
`ff0700844`	CAN - TXBuf Data Byte 6	RW	4
`ff0700848`	CAN - TXBuf Data Byte 7	RW	4
`ff070084c`	CAN - TXBuf Data Byte 8	RW	4
`ff0700850`	CAN - RXBuf Identifier	RW	4
`ff0700854`	CAN - RXBuf RTR Data Length code	RW	4
`ff0700858`	CAN - RXBuf Data Byte 1	RW	4
`ff070085c`	CAN - RXBuf Data Byte 2	RW	4
`ff0700860`	CAN - RXBuf Data Byte 3	RW	4
`ff0700864`	CAN - RXBuf Data Byte 4	RW	4
`ff0700868`	CAN - RXBuf Data Byte 5	RW	4
`ff070086c`	CAN - RXBuf Data Byte 6	RW	4
`ff0700870`	CAN - RXBuf Data Byte 7	RW	4
`ff0700874`	CAN - RXBuf Data Byte 8	RW	4
`ff0700878`	CAN - Unimplemented		
`ff070087c`	CAN - Clock Divider Register	RW	4
`ff0700880 to ff0700fff`	Unused (Echos of above)		

Interrupt Request Control and Status Registers

MBus Address	Usage	Rd/Wr	Note
`ff0701000`	IRQ pal 0 - Mask Register Read / Clear	RW	5
`ff0701004`	IRQ pal 0 - Mask Register Set	RW	5
`ff0701008`	IRQ pal 0 - Software Interrupt Reg Read / Clear	RW	5
`ff070100c`	IRQ pal 0 - Software Interrupt Reg Set	W	5
`ff0701010 to ff07011ff`	Unused (Echos)		
`ff0701200`	Timer 0 Level10	RW	4

MBus Address	Usage	Rd/Wr	Note
`ff0701204`	Timer 0 Level14	RW	4
`ff0701208`	Timer 0 Spare	RW	4
`ff070120c`	Timer 0 Control register	RW	4
`ff0701210` to `ff07013ff`	Unused (Echos)		
`ff0701400`	CPU 0 Status and Watchdog Interrupt	RW	4
`ff0701404` to `ff07015ff`	Unused (Echos)		
`ff0701600`	CPU 0 Async Error Pending	RW	4
`ff0701800`	CPU 0 EDC Mask	RW	4
`ff0701604` to `ff0701fff`	Unused (Echos of above)		
`ff0702000`	IRQ pal 1 - Mask Register Read / Clear	RW	5
`ff0702004`	IRQ pal 1 - Mask Set	RW	5
`ff0702008`	IRQ pal 1 - Software Interrupt Reg Read / Clear	RW	5
`ff070200c`	IRQ pal 1 - Software Interrupt Reg Set	W	5
`ff0702010` to `ff07021ff`	Unused (Echos)		
`ff0702200`	Timer 1 Level10	RW	4
`ff0702204`	Timer 1 Level14	RW	4
`ff0702208`	Timer 1 Spare	RW	4
`ff070220c`	Timer 1 Control register	RW	4
`ff0702210` to `ff07023ff`	Unused (Echos)		
`ff0702400`	CPU 1 Status and Watchdog Interrupt	RW	4
`ff0702404` to `ff07025ff`	Unused (Echos)		
`ff0702600`	CPU 1 Async Error Pending	RW	4

MBus Address	Usage	Rd/Wr	Note
`ff0702800`	CPU 1 EDC Mask	RW	4
`ff0702604` to `ff0702fff`	Unused (Echos of above)		
`ff0703000` to `ff0703fff`	Unused (Read Undefined)		
`ff0704000` to `ff07fffff`	Unused (Echos of above)		

STDIO IO Bus Control Registers

MBus Address	Usage	Rd/Wr	Note
`ff0800000`	Software Interrupt/Enable	RW	
`ff0800004`	Active Interrupt Level	RW	
`ff0800008`	Limit Register 0	RW	
`ff080000c`	Limit Register 1	RW	
`ff0800010`	IOBus Devices Available	RW	
`ff0800018`	Latency Delay Register	RW	
`ff080001c`	MBusID Register	R	
`ff0800020`	Timer 0	R	
`ff0800024`	Timer 1	R	

MBus to SBus, Elan, and MBus Slot Slaves

MBus Address	Usage	Rd/Wr	Note
`ff4fffff0`	M2S Virtual Address Table Base Address	RW	
`ff4fffff4`	M2S IO/MMU Control register	RW	
`ff4fffff8`	M2S Error/Status register	R	
`ff4fffffc`	M2S - MBus ID Register	R	
`ff5000000` to `ff6f7ffff`	Unused (MBus Timeout)		

MBus Address	Usage	Rd/Wr	Note
`ff6f80000` to `ff6ffdfff`	ELAN Command port area	RW	
`ff6ffe000` to `ff6ffffbf`	ELAN Hush register area	RW	
`ff6ffffc0`	ELAN Clock Hi	RW	
`ff6ffffc4`	ELAN Clock Hi	R	
`ff6ffffc8`	ELAN Clock Lo	RW	
`ff6ffffcc`	ELAN Clock Lo	R	
`ff6ffffd0`	ELAN Alarm	RW	
`ff6ffffd4`	ELAN Alarm	R	
`ff6ffffd8`	ELAN Interrupt	R	
`ff6ffffdc`	ELAN Interrupt	R	
`ff6ffffe0`	ELAN Clock Hi	R	6
`ff6ffffe4`	ELAN Clock Lo (For 64-bit accesses)	R	6
`ff6ffffe8`	ELAN Main Proc. Interrupt Mask	RW	
`ff6ffffec`	ELAN Main Proc. Interrupt Mask	R	
`ff6fffff0`	ELAN Control register	RW	
`ff6fffff4`	ELAN Control register	R	
`ff6fffff8`	MBus Port ID register for ELAN Chip	R	
`ff6fffffc`	MBus Port ID register for ELAN Chip	R	
`ff7000000` to `ff7ffffff`	Unused (MBus timeout)		
`ff8000000` to `ff9ffffff`	Used by MBus slave device in MBus Slot 0		
`ffa000000` to `ffbffffff`	Used by MBus slave device in MBus Slot 1		
`ffc000000` to `fffffffff`	Unused (MBus timeout)		

Vector Processors

Each of the two µVP's is mapped into the MBus address space twice, once for supervisor access and once for user access. There are also registers on the IO Bus to control the hardware reset line of each µVP, and to read the hardware busy signal. Only the least significant bit of these registers is used.

MBus Address	Usage	Rd/Wr	Note
`100000000` to `10000ffff`	µVP 0 (Supervisor)	RW	
`100010000` to `10001ffff`	µVP 0 (User)	RW	
`120000000` to `12000ffff`	µVP 1 (Supervisor)	RW	
`120010000` to `12001ffff`	µVP 1 (User)	RW	
`160000000` to `16000ffff`	µVP Broadcast (Super)	W	
`160010000` to `16001ffff`	µVP Broadcast (User)	W	
`9A0000010`	SET_GATE_AND_SYNCH	R	
`9A0000018`	SET_GATE_AND_SEPARATE	R	
`9A0000020`	CLEAR_GATES		
`ff0704000`	µVP 0 and µVP 1 Reset	W	4
`ff0704200`	µVP 0 Busy	R	4
`ff0705000`	Unused.	W	4
`ff0705200`	µVP 1 Busy	R	4

Within each of these images, the registers can be accessed by aligned 32-bit or 64-bit accesses.

Address Offset	Usage	Rd/Wr	Note
`0000 to 03ff`	VCB Command Buffer	RW	
`0800 to 087f`	VSR Scalar Registers	RW	
`0a00 to 0bff`	VTR Translation Registers	RW(s)	
`0e00`	VACNT Address Control	RW(s)	
`0e08`	VMD Mode	RW	
`0e10`	VLEN Vector Length	RW	
`0e14`	VCLEN Command Length	RW	
`0e18`	VSTA Start	RW	
`0e20`	VCINF Comparison Info	R	
`0e28`	VSTS Status	RW	
`0e30`	VEXB Exception Buffer	R	
`0e38`	VEXA Exception Address	R	
`0e44`	VABT Abort	W	
`0e4c`	VBSY Busy	R	

uVP Address Map

µVP Address	Usage
`00000000 to 07ffffff`	128MB Memory (Normal)
`20000000 to 27ffffff`	128MB Memory (Scrambled)
`40000000 to 47ffffff`	128MB Memory (Uncorrected)
`60000000 to 67ffffff`	128MB Memory (Scrambled Checkbit)

μVP Address	**Usage**
`80000000`	μVP EDC Error Data
`80000008`	μVP EDC Error Diagnosis
`80000010`	μVP EDC Clear

A

NVRAM Variables B

The battery-backed RAM in the realtime clock module is used to used to store basic machine start-up and communication options.

These parameters may be queried using the Forth Monitor (i.e. at the `ok` prompt):

`printenv`	Display current variable settings.
`setenv` *variable value*	Assign (or reassign) a value to a variable.
`set-default` *variable*	Restore the variables default value.
`set-defaults`	Restore the default values to all variables.

For example:

```
ok setenv output-device can
```

Alternatively the System Administrator can use the `eeprom`(1m) command to view and change the variables direct from a Unix command shell. For example:

```
root@cs2# eeprom output-device=can
```

Some of the parameters (those marked in the following list) may also be modified using the Set function in Pandora’s Network and Configuration Views.

Variable	Default	Description
`sbus-probe-list`	`43012`	Identifies the SBus slots to probe and the probe order.
`keyboard-click?`	`false`	If true, enable keyboard click.
`keymap`	*no default*	Name of custom keymap file.
`output-device` †	`screen`	Power-on output device. One of `screen`, `can`, `ttya`, or `ttyb`. Use `can` to enable console connections to be grabbed by `cancon`(1m) and Pandora.
`input-device` †	`keyboard`	Power-on input device. One of `keyboard`, `can`, `ttya`, or `ttyb`. Use `can` to enable console connections to be grabbed by `cancon`(1m) and Pandora.
`cancon-host`	`4294967295`	Used to record the host of the `cancon`(1m) remote console connection through a reboot of this processor. Do not change.
`elanip-broadcast-high` †	`4096`	Highest Elan Id in network.
`elanip-broadcast-low` †	`0`	Lowest Elan Id in network.
`ep-btxpktlifetime` †	`1000`	Elan packet characteristics.
`ep-btxtimeout` †	`1000`	Elan packet characteristics.
`ep-txpktlifetime` †	`10000`	Elan packet characteristics.
`ep-txtimeout` †	`10000`	Elan packet characteristics.
`ep-bigmsgbcastboxes` †	`4`	Elan packet characteristics.
`ep-bigmsgboxes` †	`32`	Elan packet characteristics.
`ep-bigmsgsize` †	`20416`	Elan packet characteristics.
`ep-smallmsgbcastboxes` †	`4`	Elan packet characteristics.
`ep-smallmsgboxes` †	`32`	Elan packet characteristics.
`ep-smallmsgsize`†	`4032`	Elan packet characteristics.
`elan-boot-id` †	`0`	Elan Id of node that this processor boots from.

Variable	Default	Description
`elan-node-id` †	`0`	Elan Id of this processor.
`elan-node-level` †	`1`	The processor's level in the CS-2 network.
`elan-num-levels` †	`1`	Number of levels in the CS-2 network.
`elan-top-switch` †	`0`	Specifies the level in the network that the processor sees it's topswitch. Usually this is level 0, the real top of the network.
`elan-switch-plane` †	`0`	Switch plane that this processor receives it boot code from when booting via the Elan network.
`ttyb-rts-dtr-off`	`false`	If true, Solaris does not assert RTS/DTR on ttyb.
`ttyb-ignore-cd`	`true`	If true, Solaris ignores carrier-detect on ttyb.
`ttya-rts-dtr-off`	`false`	If true, Solaris does not assert RTS/DTR on ttya.
`ttya-ignore-cd`	`true`	If true, Solaris ignores carrier-detect on ttya.
`ttyb-mode`	`9600,8,n,1,-`	ttyb (baud rate, #bits, parity, #stop, handshake). Baud rate is 110, 300, 1200, 2400, 4800, 9600, 19200, or 38400. #bits is 5, 6, 7, or 8. Parity is `n` (none), `e` (even), `o` (odd), `m` (mark), `s` (space). Handshake is `-` (none), `h` (hardware rts/cts), `s` (software).
`ttya-mode`	`9600,8,n,1,-`	ttyb (baud rate, #bits, parity, #stop, handshake). Baud rate is 110, 300, 1200, 2400, 4800, 9600, 19200, or 38400. #bits is 5, 6, 7, or 8. Parity is `n` (none), `e` (even), `o` (odd), `m` (mark), `s` (space). Handshake is `-` (none), `h` (hardware rts/cts), `s` (software).
`fcode-debug?`	`false`	If true, include name fields for plug-in device Fcodes.
`diag-file` †	`kadb`	The file and arguments to load from the root filesystem when the `diag-switch?` is true; otherwise use the `boot-file` parameter.
`diag-device` †	`elan`	Device to boot from when the `diag-switch?` is true; one of `disk`, `net`, or `elan`. Specify `elan` to boot over the CS-2 data network.

Variable	**Default**	**Description**
`boot-file`		The file and arguments to load from the root filesystem (e.g. `kadb -v`, or `/kernel/unix -vr`). No file implies `/kernel/unix`.
`boot-device` †	`elan`	Device to boot from; one of `disk`, `net`, or `elan`. Specify `elan` to boot over the CS-2 data network.
`auto-boot?` †	`false`	Boot automatically after power-on. Default value is true.
`watchdog-reboot?`	`false`	If true, reboot after watchdog reset.
`local-mac-address?`	`false`	If true, use the ethernet address taken from the `local-mac-address` parameter; otherwise use the IdPROM.
`screen-#columns`	`80`	Number of on-screen columns.
`screen-#rows`	`34`	Number of on-screen rows.
`selftest-#megs`	`1`	Megabytes of RAM to test on power-up or memory test.
`scsi-initiator-id`	`7`	SCSI bus address of host adapter, range 0–7.
`cpu-#mhz`	`40`	CPU clock rate.
`use-nvramrc?`	`false`	If true, execute the code stored in the nvramrc parameter when the boot ROM starts up.
`nvramrc`		Forth code to execute when the boot ROM starts-up (but only if `use-nvramrc?` is true).
`sunmon-compat?`	`false`	If true, come-up with old style prompt ‘>’.
`security-mode`	`none`	System security level for monitor commands; one of `none`, `command`, or `full`. None allows all commands to be executed. Command allows the continue and boot (without parameters) commands to be executed; others require a password. Full requires a password before any commands may be executed.
`security-password`	*no default*	The password used with security-mode described above.

Variable	Default	Description
`security-#badlogins`	*no default*	System set variable showing the number of times a bad password was specified.
`oem-logo`	*no default*	Byte array OEM logo (enabled by `oem-logo?`). Create a Forth array containing the logo and then copy into the `oem-logo` field.
`oem-logo?`	`false`	Enables OEM logo defined by `oem-logo`.
`oem-banner`	*no default*	Text displayed in the custom OEM banner alongside the OEM logo. Enabled by `oem-banner?`.
`oem-banner?`	`false`	Enables OEM banner text specified in `oem-banner`.
`hardware-revision`	*no default*	Hardware revision of this board (e.g. Rev D).
`last-hardware-update`	*no default*	Date of board’s manufacture or last upgrade (e.g. 25May94).
`testarea`‡		Set bit 1 to split the memory system (i.e memory test only sees half of what is available). Other bits indicate VPU silicon revision, board revision etc. and are not documented.
`mfg-switch?`	`false`	If true, perform repeated self tests.
`diag-switch?`	`false`	Run in diagnostic mode.
`meiko-sbus-slot`‡	*no default*	Identifies the SBus slot that will be used as the boot device (fitted with either a SCSI or Ethernet board). “net” is aliased to slot 2; net0, net1, net2 exist for booting off an explicit slot.
`meiko-sbus-use`‡	*no default*	Used with `meiko-sbus-slot` (above) to identify the device type; one of 0 (none), 1 (Ethernet), 2 (SCSI), or 3(Ethernet and SCSI).

† These parameters may be changed using the Set function in Pandora’s Network and Configuration Views.

‡ The definition of these variables is unique to the MK403 Vector Processing Element (all others are the same for all CS-2 boards).

Forth Monitor Commands C

The following commands have been added to the Forth Monitor and are in addition to the commands that are normally present on a Solaris system. The additional commands relate to the Control Network (CAN) and Elan network.

CAN Commands

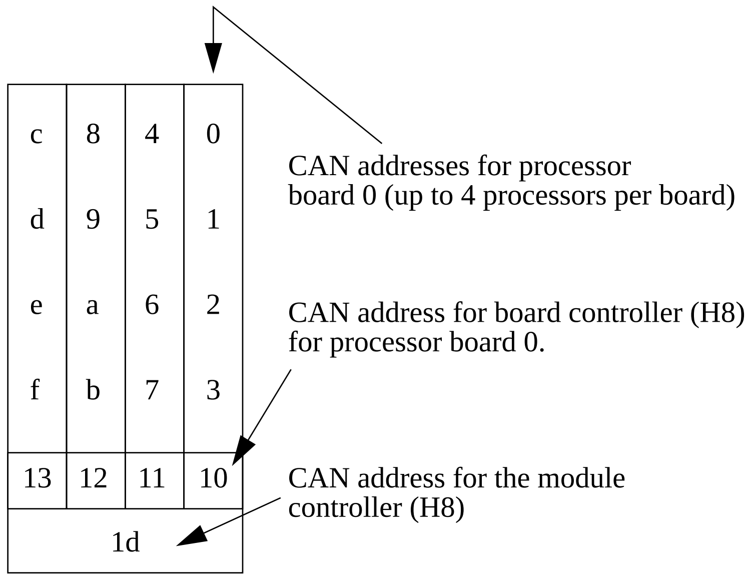
To test and use the CAN bus you need to understand CAN addresses.

Nodes are addressed by their physical position in terms of Cluster, Module, and Node id's (CMN). In CAN packets each of these id's is represented by a 6bit field; the hexadecimal representation of these three 6bit fields is a Node Id.

The module id is derived from the switch at the rear of the module. The numbering of the nodes within a module is shown in Figure C-1.

For example, the Node Id of processor with CMN 0:2:3 (processor 3, module 2, cluster 0) is 00083. The node's controlling H8 has the Node Id 00090. The node's module controller has the Node Id 0009d.

Figure C-1 CAN Addresses within a Module


c
8
4
0
d
9
5
1
e
a
6
2
f
b
7
3
13
12
11
10
1d
CAN addresses for processor board 0 (up to 4 processors per board)
CAN address for board controller (H8) for processor board 0.
CAN address for the module controller (H8)

Testing the CAN Device

Commands are provided to test the SPARC’s CAN interface device, to test the board and module controllers, and to monitor activity on the CAN bus.

Testing the CAN Interface Device

The `test` command tests the SPARC's CAN device by writing various values into its test register. The test is repeated using the test registers on the board controller's H8 and the module controller's H8.

```
ok test /can
 Register test 0x00: OK
 Register test 0xff: OK
 Register test 0xaa: OK
 Register test 0x55: OK
 Checking on-board H8: OK.
 Checking module controller: OK.
```

Testing the CAN Bus

You can test the CAN bus connection between nodes by using the `rtest` command. In the following example data is transferred from the current node to node 4:

```
ok 4 rtest
Performing remote write/read test on node 4
Remote node type is MK405
.....................................................
Time taken was 13630mSecs
```

Checking the Board and Module Controllers

You can check that both the board controller and module controller H8 processors are running by using the `ping-h8` and `ping-module` commands. Note that you need to change directory to `/can` before you use these commands.

```
ok cd /can
ok ping-h8
On-Board H8 is MK401.
ok cd ..
```

```
ok cd /can
ok ping-module
Module controller is MK515.
ok cd ..
```

Querying CAN Bus Usage

You can query the utilisation of the CAN bus by using the `perf` command. This command shows the number of CAN packets received since the machine was powered-up, and the number since the last query. You need to change to the `/can` directory before using this command:

```
ok cd /can
ok perf
Total number of messages received since power-up: 259380
No. of messages received per second since the last check: 4
ok cd ..
```

Monitoring CAN Bus Packets

You can *snoop* the CAN bus (monitor that packets on the bus) using the `snoop` command. Note that you cannot use this facility if you are connected to the Forth Monitor via a cancon connection. You need to change to the `can` directory before using this command.

```
ok cd can
ok snoop
Can’t can-snoop if you are a cancon slave
ok cd ..
```

CAN Addresses

To determine the CAN address of this node use `.can-id`. This displays the node’s address in terms of its CMN, Node Id, and Slot Id. The Slot Id is for Meiko engineering use[1].

```
ok .can-id
SlotId: 0090, CAN Node-id: 00088 [00:02:08]
```

Similarly the CAN address of the board’s controlling H8 processor can be obtained with the `.h8-id` command:

```
ok .h8-id
The on-board H8 is node 00092.
```

1. The slot id is the node’s physical position in the machine represented by a 5 bit cluster number, a 5 bit module number, a 2 bit slot number, 2 unused (always 0 bits), and a 2 bit processor number; the 2 bits that represent the slot number are transposed.

The CAN address of the H8 that controls the board's module can be determined by the `.module-id` command:

```
ok .module-id
The module controller is node 0009d.
```

You can convert from CAN node id's to Cluster, Module, Node addresses (and vice versa) by using the `canid>cmn` and `cmn>canid` commands respectively. Note that you need to change directory to `can` before you use these commands.

```
ok cd can
ok 4 canid>cmn
0 0 4
ok cd ..
```

```
ok cd can
ok 0 2 8 cmn>canid
Node Id is 00088
ok cd ..
```

Querying CAN Objects

Can packets include a 10 bit address space which, although not sufficient to map into the MBus/H8 physical address space, is adequate to map-in various status and control devices. These are referred to as CAN objects. Reading or writing to these objects allows you to query the status of a processor, board, or module, and to issue control instructions. See the header file `/opt/MEIKOcs2/include/canio/canobj.h` for a list of object addresses and their meanings.

Local CAN objects are those that relate directly to this node. Remote CAN objects maybe those of a board, module controller, or remote SPARC.

You use the `rlo` command to read a local object. You need to pass an object id on the Forth stack; in the following example we request the board type and are returned 191 (an MK401):

```
ok 0 rlo
Read: 191
```

To read a remote object you need to push onto the Forth stack a CAN node id and the object id. In the following example we request the board type of node `c1` (module 3, board 0, node 2), which is an MK405:

```
ok c1 0 rro
Read: 195
```

The following additional example fetches the board type of node `dd`, which is the controller for module 3, cluster 0:

```
ok dd 0 rro
Read: 203
```

Similar commands exist to write to CAN objects, but their direct use is not recommended (they can reconfigure and reset the machine).

Remote Console Connections

You can create a console connection to a remote node by using `cancon`. You need to pass on the Forth stack a CAN node id.

You cannot create a `cancon` connection from within an existing cancon connection. If you are remotely interacting with a node's Forth monitor via `cancon` (or Pandora) an attempt to create another `cancon` connection will fail.

```
ok 4 cancon
Connected to node 00004

cs2-4 console login:
```

If your node is currently serving a remote console connection to someone else you can force it to disconnect that connection by using `cancon-dis`. In the following example the current connection to node 8 is dropped:

```
ok 8 cancon-dis
Disconnecting node 00008 [00:00:08] ...
```

Elan Commands

The Elan device includes self test code that can be executed by the `test-all` command (which tests memory, SBus, CAN, Elan and all other devices with self test code) or explicitly by the `test /elan` command.

```
ok test /elan
Initialising Elan/Selftest software ... OK
Checking threads processor ... OK

Testing from level 1 to level 1.
Generating a route to level 1 ... OK
Ping ... OK
Check-Ping ... OK
Spraying data to top switch ... OK
Testing spray buffer ... OK

Closing down Elan/Selftest software ... OK
```