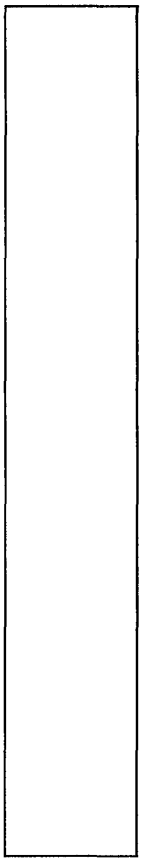
MK401 - Dino
General SPARC Node

Specification

December 16, 1992



OVERVIEW

The MK401 (Dino) is the first board in the new machine architecture. As such it incorporates several new devices and concepts. A complete mechanical redesign has resulted in a card size of 14 inches by 15 inches. The Elan/Elite communications network is supported, and a new supervisory structure based around the CAN (Control Area Network) serial bus protocol is implemented. All input/output connectors are brought to the front panel of the cards, removing the need for IO 'dongles'. Built-in SCSI drives are also supported with backplane SCSI wiring.

The board is MBus based with a single MBus (single node). On this MBus are placed two full size MBus processor card sites. This allows two SPARC CPUs to be supported, either as one dual processor card or two single processor cards. Also on the MBus is the Elan communication processor which has two Elan links to the backplane. An MBus to SBus converter chip drives an SBus which has three single sized SBus master card sites in addition to two SBus DMA controllers supporting an Ethernet port and two SCSI ports. The SBus cards and one of the MBus cards have front panel access space. A memory controller supports up to 128MBytes of error checked DRAM using 4Mbit DRAM technology. Support for 16Mbit DRAMs is included and allows a total of 512MBytes of DRAM on the node. An additional controller supports an 8bit peripheral bus, the IOBus, with a variety of devices attached. These include a Boot ROM, real time clock, two serial ports, keyboard and mouse ports, an interface to the CAN bus and miscellaneous node control functions.

IMPLEMENTATION

This section gives some details about each of the major structural blocks of the Dino. Subsequent sections will give full details, register descriptions, address maps etc.

2.1 *MBus and Processor Slots*

The MBus is fully level two compliant and runs at 40MHz. Support in the form of bus request and grant lines and interrupt signals is provided for two processors. These can either be on the same processor card or one on each card.

Each device attached to the MBus has a unique MBusID. This is used to decode addresses at which the devices may be accessed. MBus processor slot 0 is given MBusIDs 8 and 9, and slot 1 is given IDs 10 and 11. Note that most SPARC processor cards do not actually use these MBusID signals but instead rely on software programmed registers. To allow each processor to establish a unique identity there is a register which each can read to determine which MBus request and grant line it is attached to. This register is described fully later.

A table of MBusID assignments follows.

MBusID	Device
0	IOBus Controller
1	Unused
2	Unused
3	Unused
4	MBus to SBus chip
5	Unused
6	Elan Communications Processor
7	Unused
8	MBus Processor Slot J1
9	MBus Processor Slot J1
10	MBus Processor Slot J2
11	MBus Processor Slot J2
12	Unused
13	Unused
14	Unused
15	Unused

2.2 *SBus*

The MBus is connected to the SBus through an LSI Logic M2S chip (L64852C). This chip supports MBus master accesses to the SBus and also SBus accesses to the MBus. In the case of MBus master initiated transactions virtual address translation is assumed to have been performed by the master's own MMU. SBus masters however have virtual to physical address translation carried out by the M2S chip itself. To achieve this the M2S chip maintains a set of translation tables in MBus memory and 16 TLB entries on chip.

The M2S chip supports five external SBus DVMA devices (potential SBus masters). Two of these are LSI SBusDMA chips supporting the Ethernet and SCSI ports (devices 1 and 5). The other three are standard single size SBus slots into which masters or slaves can be placed (devices 2, 3 and 4). SBus arbitration is carried out by the M2S chip and the SBus runs at a clock speed of 20MHz. Complete details of the M2S chip's operation can be found in the LSI Logic L64852C, MBus to SBus controller technical manual. Details of the address mapping of the M2S chip are given in a later section. The M2S chip has an MBusID of 4.

The SBusDMA devices on the SBus are LSI Logic SBDMA2 chips (L64853A). One of these (device 1) services an AMD AM7990 Ethernet controller and an NCR N53C90A SCSI-2 controller. The Ethernet device has a standard interface to an external transceiver connected with a standard Ethernet drop cable. This allows

connection to either thick or thin wire Ethernet. The SCSI-2 controller supports an 8bit, single ended SCSI bus with connections at the backplane and a standard high-density SCSI socket on the front panel. This latter allows connection to external SCSI devices such as CD-Rom drives and scanners. The second SBus DMA device (SBus device 5) has the similar SCSI-2 circuitry but without an Ethernet connection.

Termination for the SCSI bus must be provided for correct operation. On board termination can be enabled by a front panel recessed switch. If the SCSI bus is not used this termination must be used. If there are devices plugged into the bus then, for correct SCSI operation, the bus must be terminated at both ends. If devices are plugged into only either the backplane or the front panel then the on board termination must be used. If there are devices plugged into both front and backpanels then the on board termination must be disabled.

2.3 *DRAM*

The memory controller used on the Dino is the same as that in the Sun SPARCstation10. This is the LSI chip, L64860, which uses a 128 bit wide data bus for fast memory accesses. The device supports single bit error correction, two bit error detection and multiple bits within a nybble detection. The device drives an array of 16, 36bit wide, SIMMs identical to the ones used in the TwinEngine on the existing MK083 product. The array is constructed as 4 groups of 4 SIMMs. Within each group the SIMMs must be identical, but there is no requirement for the groups to be the same. SIMMs can be either 4Mbit DRAM or 16Mbit DRAM technology, either single or double sided. This means that the minimum memory is 16MBytes and this can be upgraded in 16MByte chunks to 128MBytes and then 64MByte chunks to 512MBytes. Section **X** gives more details about the various memory options available.

Writes to memory of less than 64bits width are automatically turned into read-modify-write cycles to the array. All MBus transaction sizes and Level-1 and Level-2 transaction types are supported.

Memory accesses are error-checked with all single-bit errors being corrected. Errors of two or more bits are uncorrectable and will cause a synchronous MBus error acknowledgement to the MBus cycle. Single bit errors can be corrected and will return a normal MBus acknowledge. Their occurrence can be signalled with a level 15 interrupt to the processors. This allows error logging and scrubbing operations to be carried out. Error fault information is available for all single and multi-bit errors in registers within the memory controller. Kernel software can advise the board controller of the occurrence of uncorrectable memory errors by sending a message over the CAN bus. This can be used to log error information and cause a visible status change on the front panel LEDs.

2.4 *IO Bus*

The IOBus is a general purpose 8-bit slave only bus designed to allow easy connection of minor peripherals to the MBus. It is provided in this implementation by the LSI chip, L64851. There are several devices connected to this bus :-

- An EPROM or Flash-EPROM of up to 512KBytes. This holds the bootstrap programs, ie. OpenBoot and Meiko Diagnostics.
- A real-time clock module with 8KBytes of battery backed SRAM. This holds configuration information and node fault logs. It is software compatible with the module used on SPARCstations.
- Two dual UART devices for connection to Keyboard, Mouse and two general purpose serial ports. These are the same as used on SPARCstations.
- Board and node control registers as follows :-
  - Interrupt request control for each processor.
  - Free running timers for interrupt levels 10 and 14 to the processors.
  - CAN interface chip to the control serial bus.
  - MBus grant readback register.
  - Node "Reset Me" request register.
  - LED bargraph register.
  - Asynchronous error interrupt status latches.
  - Physical board ID register.

A brief description of each device follows. For details of address map allocation section 3.

2.4.1 *BootRom*

The code in this ROM is read by both processors on startup. This code must perform all the hardware initialisation, initial page table construction and booting of vmunix from the desired device. Note that since both processors start up together and access the same code a check will need to be made of the MBus grant line register. Once each processor is identified, processor 0 can continue with the hardware setup while processor 1 idles. When processor 0 finishes it can wake processor 1 by issuing it a software interrupt.

Board jumpers JP1 and JP2 allow the use of Flash-EPROM in the BootRom socket. This feature is recommended only for Meiko internal software development. For this reason the jumpers are Zerohm links soldered to pads on the underside of the board to dissuade user alteration. Details of the Flash-EPROM programming mechanism can be found in the AMD data sheet for part Am28F020. This is included in AMD's data book "CMOS Memory Products" of 1991. Location of the BootRom in the MBus physical memory map is given in section 3.

2.4.2 *Real Time Clock*

The Real Time Clock used on the Dino is the Mostek/SGS-Thompson part MK48T08. This is an 8KByte version of the earlier MK48T02 used on Sun SPARCstations. The extra RAM is useful for logging of system failure data such as Uncorrectable multi-bit errors. Detailed information about each fatal crash can be stored along with a date-stamp. In addition a log of time powered up could be kept along with an audit trail of board manufacture tests and modifications.

The Clock module updates several of the RAM registers once every second. The module contains features for calibration to offset crystal frequency drift with time. year, month, date, day, hour, minute and second information is available.

Details of the locations of the various registers are included in section 3. For details on programming the clock see the SGS-Thompson data sheet on the MK48T08.

2.4.3 *Serial Ports*

There are two Dual UARTs of type Z85C30 by AMD on the IOBus. One chip receives data from the keyboard and mouse connector on the front panel, the other handles to two RS232 serial ports also on the front panel. In the absence of a keyboard and frame buffer the bootstrap code has an option to direct console IO to serial port A.

These serial ports are clocked at 4.9152MHz and are thus theoretically capable of up to 76.8Kbaud. In practice 38.4Kbaud is achievable. Note that both serial port A and serial port B use the same 25-way 'D' connector. Thus port A supports full

synchronous and asynchronous operation and has a full set of modem control lines, whereas port B has a more limited set and only supports asynchronous operation.

Full details of the locations of the registers on the serial port controllers are included in section 3. For details on programming the Z85C30 see the AMD publication "Z85C30 Serial Communications Controller. Technical Manual."

2.4.4 *Interrupt Controller*

The MK401 has many sources of interrupts to the SPARCs. These are assigned to various priority levels and given to the Interrupt Controller. There is a separate controller for each processor, to enable interrupt handling to be effectively shared between the processors. Each controller is implemented in a large PAL, full details of which are given in section 5. The controller has the following major features :-

- Masks for all hardware generated interrupts.
- Software interrupts on six levels using a multi-processor compatible set/clear register structure. This allows one processor to interrupt the other by writing once to a single memory location, without having to do a read-modify-write cycle which is subject to being broken by the other processor. Supported levels are 1, 4, 6, 12, 13 and 15.
- Latches triggered by the outputs of the free running timers. These capture the timer passing zero event and cause an interrupt to the processor. When the interrupt handler accepts the interrupt and clears the request it resets this latch.
- Priority encoder to generate the 4bit encoded interrupt level that is passed to the SPARC processor module.

2.4.5 *Periodic Interrupt Timers*

Periodic interrupts are used for maintaining the Unix clock and kernel profiling. Each processor has a separate timer chip (an AM82C54) which has three free running timers integrated. Timer 0 is used for the lower priority clock ticks. This timer counts from a set value down to zero, decrementing once every 3.2us and generating a level 10 interrupt on reaching zero. Timer 1 is used for higher priority interrupts for kernel profiling. This timer decrements once every 0.8us and generates a level 14 interrupt on reaching zero.

Note that the Timer 0 for processor 0 generates the level 10 interrupts for processor 1 as well. This removes the need to initialise Timer 0 in the other timer chip.

Full details of the registers in the timer chips are given in the AMD data sheet for the Am82C54. Memory locations on the Dino are given in section 3.

2.4.6 *CAN Interface*

The SPARC processors have direct access to the control serial bus with their own CAN bus interface chip. This is intended to be used only by the node that is acting as machine controller. All Dino nodes will have the capability of becoming machine controller. The CAN chip has the capacity of generating a large number of level 2 interrupts and thus should be disabled on all nodes other than the designated machine controller to avoid impacting CPU performance.

The CAN controller is a Philips PCA82C200 device and full details of the chip are given in the Philips data sheet accompanying it. Placement of its registers within the MBus physical memory map is given in section 3.

2.4.7 *Miscellaneous Registers*

This section describes the various registers used to control the node's operation. Address map positions are summarised in section 3.

2.4.7.1 MBus Grant Readback

This is a single eight bit read only register, visible by both SPARC processors. There are two bit fields as follows :-

[3:0] Bits [3:0] are used to carry the active low MBus grant signals for this MBus transaction. An MBus master device can thus read this register to determine its unique ID. The bit that is clear when read is given by the following list :-

0	MBus Processor 0
1	MBus Processor 1
2	Elan Communications Processor
3	SBus Master accessing MBus.

[7:4] These four bits are unused and will return undefined values if read.

2.4.7.2 Node reset request

This is a single eight bit write-only register accessible by both processors. Bits [7:1] are reserved and should be written as ones. Bit zero, when written as zero, causes a Node reset to be generated. During the reset process this register is restored to its inactive state.

2.4.7.3 LED Bargraph

This is a single sixteen bit read/write register also accessible as two eight bit registers, visible to both processors. Bits [15:0] drive the sixteen module front panel LEDs to generate a performance/load Bargraph display. A zero should be written at a bit location to illuminate the corresponding LED. This register is set to all ones during a node reset operation.

2.4.7.4 Asynchronous error status latch

This is a four bit read only latch, which is cleared when written to. The bits are set when an asynchronous MBus or SBus fault or Memory error occurs and will cause a maskable level 15 interrupt to the SPARCs. The bits are defined as follows :-

0 Uncorrectable multi-bit memory error or, when enabled, correctable singel bit errors also.

1 MBus to SBus asynchronous transaction fault

2 Asynchronous Fault flagged by a slave MBus device in an MBus slot.

3 Backplane power failure signal asserted.

[7:4] These bits are unused and will have undefined values when read.

2.4.7.5 Physical board location

This is a 16 bit read only register visible by both processors used to find the physical location of the Dino card in the entire system. A twelve bit Slot ID is uniquely defined for every card slot in the machine and is read at this location, at the upper 12 bits of the halfword. The least significant nybble contains the node number on the card, which in the case of the single node Dino is always zero.

MBUS PHYSICAL ADDRESS MAP

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 Memory banks are spaced 64MBytes apart. If less than 64MBytes of memory is present in each bank, it will appear echoed throughout the 64MByte.

2 These locations are byte wide and are mapped into all 4 bytes of a word. Care should be taken to generate correct byte-wide accesses to the least significant byte of the word in order to maintain future compatibility.

3 These locations are byte-wide memory, mapped into contiguous byte locations. Word or halfword accesses will be automatically mapped into several successive byte-wide accesses.

4 These locations are byte sized registers which are only mapped into the most significant byte of a halfword. To ensure compatability with other boards only byte accesses at the correct address should be used.

5 These locations are byte sized registers which are only mapped into the least significant byte of the word. To conserve MBus bandwidth and ensure compatability with other boards only byte wide accesses at the correct address should be used.

6 These locations are halfword sized registers which are only mapped into the least significant halfword of the word. To conserve MBus bandwidth and ensure compatability with other boards only halfword wide accesses at the correct address should be used.

7 These locations form a doubleword register.

DRAM and SBus Slots

MBus Address	Usage	Rd/Wr	Note
000000000	64MB Memory in bank 0	RW	1
004000000	64MB Memory in bank 1	RW	1
008000000	64MB Memory in bank 2	RW	1
00c000000	64MB Memory in bank 3	RW	1
010000000	64MB Memory in bank 4	RW	1
014000000	64MB Memory in bank 5	RW	1
018000000	64MB Memory in bank 6	RW	1
01c000000	64MB Memory in bank 7	RW	1
020000000 to dffffffff	Unused (MBus Timeout)		
e00000000 to e0fffffff	SBus Slot 1 (J11)	RW	
e10000000 to e1fffffff	SBus Slot 2 (J12)	RW	
e20000000 to e2fffffff	SBus Slot 3 (J13)	RW	

SBus DMA chip B and SCSI

MBus Address	Usage	Rd/Wr	Note
e30000000	SBus DMA_B ID register (= 0xfe810102)	R	
e30000004 to e303fffff	Unused (Echoes of above)		
e30400000	SBus DMA_B Control/Status Register	RW	
e30400004	SBus DMA_B (Next)AddressCounter	RW	
e30400008	SBus DMA_B (Next)ByteCount	RW	
e3040000c	Reserved for testing M2S		
e30400010 to e307fffff	Unused (Echoes of above)		
e30800000	SCSI_B Transfer Count Low	RW	1
e30800004	SCSI_B Transfer Count High	RW	2
e30800008	SCSI_B Fifo Data	RW	2
e3080000c	SCSI_B Command	RW	2
e30800010	SCSI_B Status	R	2
e30800010	SCSI_B Destination Bus ID	W	2
e30800014	SCSI_B Interrupt	R	2
e30800014	SCSI_B Select/Reselect Timeout	W	2
e30800018	SCSI_B Sequence Step	R	2
e30800018	SCSI_B Synchronous Period	W	2
e3080001c	SCSI_B FIFO Flags	R	2
e3080001c	SCSI_B Synchronous Offset	W	2
e30800020	SCSI_B Configuration 1	RW	2
e30800024	SCSI_B Clock Conversion Factor	W	2
e30800028	SCSI_B Test mode	W	2
e3080002c	SCSI_B Configuration 2	RW	2
e30800030 to e3080003c	SCSI_B Reserved		
e30800040 to e30bfffff	Unused (Echoes of above)		
e30c00000 to e3fffffff	Reserved (Read Undefined)		

SBus DMA chip A, Ethernet and SCSI

MBus Address	Usage	Rd/Wr	Note
e40000000	SBus DMA_A ID register (= 0xfe810102)	R	
e40000004 to e403fffff	Unused (Echoes of above)		
e40400000	SBus DMA_A Control/Status Register	RW	
e40400004	SBus DMA_A (Next)AddressCounter	RW	
e40400008	SBus DMA_A (Next)ByteCount	RW	
e4040000c	Reserved for testing		
e40400010 to e407fffff	Unused (Echoes of above)		
e40800000	SCSI_A Transfer Count Low	RW	2
e40800004	SCSI_A Transfer Count High	RW	2
e40800008	SCSI_A Fifo Data	RW	2
e4080000c	SCSI_A Command	RW	2
e40800010	SCSI_A Status	R	2
e40800010	SCSI_A Destination Bus ID	W	2
e40800014	SCSI_A Interrupt	R	2
e40800014	SCSI_A Select/Reselect Timeout	W	2
e40800018	SCSI_A Sequence Step	R	2
e40800018	SCSI_A Synchronous Period	W	2
e4080001c	SCSI_A FIFO Flags	R	2
e4080001c	SCSI_A Synchronous Offset	W	2
e40800020	SCSI_A Configuration 1	RW	2
e40800024	SCSI_A Clock Conversion Factor	W	2
e40800028	SCSI_A Test mode	W	2
e4080002c	SCSI_A Configuration 2	RW	2
e40800030 to e4080003c	SCSI_A Reserved		
e40800040 to e40bfffff	Unused (Echoes of above)		
e40c00000	LANCE Register Data Port	RW	2
e40c00002	LANCE Register Address Port	RW	2
e40c00004 to e5ffffffff	Unused (Echoes of above)		
e60000000 to e600000ff	Unused (Invalid TLB Entry) (MBus Error?)		
e60000100 to e600001f0	M2S TLB Slices 0 through 15. (on 16-byte boundaries)	RW	
e60000200 to e6ffffffff	Unused (Invalid) (MBus Error?)		
e70000000 to effffffff	Unused (SBus Reserved) (MBus Error?)		

Memory Controller

MBus Address	Usage	Rd/Wr	Note
f00000000	Memory Enable	RW	
f00000004	Memory Delay	RW	
f00000008	Fault Status	R(W)	
f0000000c	Video Config	RW	
f00000010	Fault Address 0	R	
f00000014	Fault Address 1	R	
f00000018	ECC Diagnostics	RW	
f0000001c to f0ffffffff	Unused (Read undefined)		
f00000000 to feffffffff	Unused (No response) (MBus Timeout)		

BootRom, Serial Ports, Real Time Clock, Miscellaneous

MBus Address	Usage	Rd/Wr	Note
ff0000000 to ff007ffff	BootRom (512KByte). (Writeable only when Flash ROM installed and jumpers JP1 and JP2 are in the correct positions. See Flash ROM data sheet for details of command port structure.)	R(W)	3
ff0080000 to ff00fffff	Unused (BootRom Echo)		
ff0100000 to ff0100007	Serial Port Controller		
ff0100000	Control Registers port B	RW	4
ff0100002	Data Buffer port B	RW	4
ff0100004	Control Registers port A	RW	4
ff0100006	Data Buffer port A	RW	4
ff0100008 to ff01fffff	Unused (Serial Port Echoes)		
ff0200000 to ff0200007	Keyboard and Mouse Port Controller		
ff0200000	Control Registers mouse port	RW	4
ff0200002	Data Buffer mouse port	RW	4
ff0200004	Control Registers keyboard port	RW	4
ff0200006	Data Buffer keyboard port)	RW	4
ff0200008 to ff02fffff	Unused (Keyboard and Mouse Port Echoes)		
ff0300000 to ff0301fff	Real Time Clock module and 8KByte SRAM	RW	3
ff0302000 to ff03fffff	Unused (RTC Echoes)		
ff0400000 to ff06fffff	Unused (MBus Error)		
ff0700000	Node Reset Request	W	5
ff0700004 to ff07001ff	Unused (Echoes)		
ff0700200	MBus Grant readback	R	5
ff0700204 to ff07003ff	Unused (Echoes)		
ff0700400	Physical Slot Identifier	R	6
ff0700404 to ff07005ff	Unused (Echoes)		
ff0700600	LED Bargraph	RW	6
ff0700604 to ff07007ff	Unused (Echoes)		

Control Area Network Interface

MBus Address	Usage	Rd/Wr	Note
ff0700800	CAN - Control Register	RW	5
ff0700804	CAN - Command Register	W	5
ff0700808	CAN - Status Register	R	5
ff070080c	CAN - Interrupt Register	R	5
ff0700810	CAN - Acceptance Code Register	RW	5
ff0700814	CAN - Acceptance Mask Register	RW	5
ff0700818	CAN - Bus Timing Register 0	RW	5
ff070081c	CAN - Bus Timing Register 1	RW	5
ff0700820	CAN - Output Control Register	RW	5
ff0700824	CAN - Test Register		
ff0700828	CAN - TXBuf Identifier	RW	5
ff070082c	CAN - TXBuf RTR Data Length code	RW	5
ff0700830	CAN - TXBuf Data Byte 1	RW	5
ff0700834	CAN - TXBuf Data Byte 2	RW	5
ff0700838	CAN - TXBuf Data Byte 3	RW	5
ff070083c	CAN - TXBuf Data Byte 4	RW	5
ff0700840	CAN - TXBuf Data Byte 5	RW	5
ff0700844	CAN - TXBuf Data Byte 6	RW	5
ff0700848	CAN - TXBuf Data Byte 7	RW	5
ff070084c	CAN - TXBuf Data Byte 8	RW	5
ff0700850	CAN - RXBuf Identifier	RW	5
ff0700854	CAN - RXBuf RTR Data Length code	RW	5
ff0700858	CAN - RXBuf Data Byte 1	RW	5
ff070085c	CAN - RXBuf Data Byte 2	RW	5
ff0700860	CAN - RXBuf Data Byte 3	RW	5
ff0700864	CAN - RXBuf Data Byte 4	RW	5
ff0700868	CAN - RXBuf Data Byte 5	RW	5
ff070086c	CAN - RXBuf Data Byte 6	RW	5
ff0700870	CAN - RXBuf Data Byte 7	RW	5
ff0700874	CAN - RXBuf Data Byte 8	RW	5
ff0700878	CAN - Unimplemented		
ff070087c	CAN - Clock Divider Register	RW	5
ff0700880 to ff0700fff	Unused (Echoes of above)		

Interrupt Request Control and Status Registers

MBus Address	Usage	Rd/Wr	Note
ff0701000	IRQ pal 0 - Timer Latches	RW	5
ff0701002	IRQ pal 0 - Mask Register Read / Clear	RW	5
ff0701006	IRQ pal 0 - Mask Register Set	RW	5
ff070100a	IRQ pal 0 - Software Interrupt Reg Read / Clear	RW	5
ff070100e	IRQ pal 0 - Software Interrupt Reg Set	W	5
ff0701010 to ff07011ff	Unused (Echoes)		
ff0701200	Timer 0 Level10 Processor 0 and 1	RW	5
ff0701204	Timer 0 Level14 Processor 0	RW	5
ff0701208	Timer 0 Spare timer	RW	5
ff070120c	Timer 0 Control register	RW	5
ff0701210 to ff07015ff	Unused (Echoes)		
ff0701600	AErr and Powerfail Latch	RW	5
ff0701604 to ff0701fff	Unused (Echoes)		
ff0702000	IRQ pal 1 - Timer Latches	RW	5
ff0702002	IRQ pal 1 - Mask Register Read / Clear	RW	5
ff0702006	IRQ pal 1 - Mask Register Set	RW	5
ff070200a	IRQ pal 1 - Software Interrupt Reg Read / Clear	RW	5
ff070200e	IRQ pal 1 - Software Interrupt Reg Set	W	5
ff0702010 to ff07021ff	Unused (Echoes)		
ff0702200	Spare timer	RW	5
ff0702204	Timer 1 Level14 Processor 1	RW	5
ff0702208	Timer 1 Spare timer	RW	5
ff070220c	Timer 1 Control register	RW	5
ff0702210 to ff0702fff	Unused (Echoes)		
ff0703000 to ff0703fff	Unused (Read Undefined)		
ff0704000 to ff03fffff	Unused (Echoes of above)		
ff0400000 to ff4ffffec	Unused (MBus Timeout)		

MBus to SBus Chip, Elan Communications Processor 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)		
ff6f80000 to ff6ffdfff	ELAN Command port area	RW	
ff6ffe000 to ff6fffbf	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	7
ff6ffffe4	ELAN Clock Lo (For 64bit accesses)	R	7
ff6ffffe8	ELAN Main Proc. Interrupt Mask	RW	
ff6fffffec	ELAN Main Proc. Interrupt Mask	R	
ff6ffffff0	ELAN Control register	RW	
ff6ffffff4	ELAN Control register	R	
ff6ffffff8	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)		

NODE CONTROL STRUCTURE

This section gives more detail on the control circuitry used to monitor and supervise the operation of the node in a system. It details the function of the control microprocessor and how it interfaces with the SPARC.

4.1 Control Microprocessor Overview

The MK401 uses a Hitachi microcontroller to implement basic node control functions. The controller is a member of the H8/500 series of single chip micros, containing 2KBytes of RAM and 32KBytes of EPROM as well as a number of peripherals, IO ports, timers etc. The device is a H8/534 which has a 16-bit RISC based CPU core. On-chip peripherals include 9 IO ports of which 3 are used by an external address and data bus for off-chip peripherals and memory. Other on-chip devices include two serial ports, 3 16-bit timers and one 8 bit one, pulse-width modulated outputs, analogue-to-digital converters etc. Much of this remains unused on the MK401, however the chip and family has the potential to perform more complex tasks on future boards.

The H8 controller is responsible for performing the following actions :-

- Monitoring of the CAN bus to detect messages addressed to this node and act on them.
- Monitoring of the CAN bus to detect status and watchdog messages from the local node and interpret them.
- Monitoring of the SPARC processor performance/load LED Bargraph signals to make this information available remotely over the CAN bus.
- Placing the node in reset when directed to by CAN bus messages.
- Reporting physical slot ID over the CAN bus to allow the physical location of a board to be located. Especially useful in very large systems.

- Recoding of the node status information to drive the two front-panel status LEDs.

Most of the node to board control information flow goes over the CAN bus. Other signals are the SlotID, Bargraph lines and a node reset output from the H8. The H8 also controls the front panel LEDs.

4.2 *H8 IO Port Assignments*

Each subsection in this section is devoted to a particular IO port of the controller and gives a list of the signals connected to the port with some description of how they are used.

4.2.1 *Port 1*

0 This output carries the 10MHz H8 clock 'Phi'.
4:1 SlotID[11:8] inputs.
5 notIRQ0 input from CAN interface chip. This can be used to trigger a preprogrammed transfer of data to or from the CAN chip. See the H8 manual section 5 "Interrupt Controller" and section 6 "Data Transfer Controller" for more information.
7:6 Unused IO.

Three registers are associated with Port 1 and should be programmed as follows.

P1DDR Port 1 data direction register should have 8'b00000001 programmed. This sets unused IOs to inputs.

P1DR Carries the current input data.

SYSCR1 System Control register 1. Should be set to the value 8'b00100000 to enable the notIRQ line.

4.2.2 *Port 2*

This port is used to carry strobes and select lines to the off-chip peripherals. Its function is set automatically and no programming of the control registers P2DDR and P2DR is required.

0 This output carries the notAddressStrobe signal.
1 This output carries the ReadnotWrite signal.
2 This output carries the notDataStrobe signal.
3 This output carries the notRead strobe.
4 This output carries the notWrite strobe.

4.2.3 *Port 3*

This port carries the eight bits of the data bus to off-chip peripherals. Its function is set automatically and no programming of the control registers P3DDR and P3DR is required.

4.2.4 *Port 4*

This port carries the eight least significant bits of the address bus to off-chip peripherals. Its function is set automatically and no programming of the control registers P4DDR and P4DR is required.

4.2.5 *Port 5*

This port carries the eight most significant bits of the data bus to off-chip peripherals. Its function is set automatically if off-chip ROM is enabled (See description for JP3 and JP4 in section X). If on-chip ROM is used then P5DDR must be set to 8'b11111111. No programming of the data register P5DR is required.

4.2.6 *Port 6*

0 Unused IO.

1 This output, when low, causes the Dino node to be reset.

2 This signal drives the yellow front panel LED. The LED is illuminated when the signal is low.

3 This signal drives the green front panel LED. The LED is illuminated when the signal is low.

Two registers are associated with Port 6 and should be programmed as follows.

P6DDR Port 6 data direction register should have 8'b00001110 programmed. This sets unused IO to input.

P6DR The output pins are set to the value in this register.

4.2.7 *Port 7*

This eight bit port is used to carry the low eight bits of the SlotID value, SlotID[7:0].

Two registers are directly associated with Port 7 and should be programmed as follows.

P7DDR Port 7 data direction register should have 8'b00000000 programmed.

P7DR Carries the data input on the pins.

Other register bits have a bearing on Port 7s operation and should be programmed as follows. See the H8 manual section 10 "16-Bit Free Running Timers" and section 11 "8-Bit Timer" for more details on these registers.

FRT1.TCR.OEA Output Enable A of the Timer Control Register of Free Running Timer channel 1. This bit must be clear to enable normal Port 7.7 operation.

FRT3.TCR.OEB Output Enable B of the Timer Control Register of Free Running Timer channel 3. This bit must be clear to enable normal Port 7.6 operation.

FRT2.TCR.OEB Output Enable B of the Timer Control Register of Free Running Timer channel 2. This bit must be clear to enable normal Port 7.5 operation.

FRT1.TCR.OEB Output Enable B of the Timer Control Register of Free Running Timer channel 1. This bit must be clear to enable normal Port 7.4 operation.

TMR.TCR.CCLR1:0 These two bits of the Timer Control Register of the 8-bit Timer can be both set to one to enable resetting of the counter from an input on Port 7.3. This must be disabled (by clearing one or both of these bits) to allow normal operation of Port 7.3.

FRT2.ICR The count in Free Running Timer 2 is captured to its Input Capture Register on edges of the signal on Port7.2. Since this signal has no special time relationship to the H8, the value in the ICR will be meaningless.

FRT1.ICR The count in Free Running Timer 1 is captured to its Input Capture Register on edges of the signal on Port7.1. Since this signal has no special time relationship to the H8, the value in the ICR will be meaningless.

TMR.TCR.CKS2:0 These three bits of the Timer Control Register of the 8-bit Timer can be set to select Port7.0 as the clock signal for the 8-bit Timer. Since the signal on Port7.0 has no timing significance to the H8, this should not be done.

4.2.8 *Port 8*

This eight-bit port is used to monitor the low eight bits of the BarGraph value, notBAR[7:0].

Port 8 has no Data direction register (being input only) so the only register of interest is P8DR, which reflects the state of the input pins.

4.2.9 *Port 9*

This eight-bit port is used to monitor the high eight bits of the BarGraph value, notBAR[15:8].

Several registers have a bearing on the operation of Port 9.

P9DDR Port 9 data direction register should have 8'b00000000 programmed.

P9DR Carries the data input on the pins.

SCI1.SCR.CKE1:0 Serial Channel 1, Control Register, Clock Enable bits. These bits should be programmed to zero to allow Port 9 operation and disable serial clock output.

SCI1.SCR.RE Serial Channel 1, Control Register, Receiver Enable bit. This should be cleared to enable normal port 9 operation.

SCI1.SCR.TE Serial Channel 1, Control Register, Transmitter Enable bit. This should be cleared to enable normal port 9 operation.

SYSCR2 The P9PWME and P9SCI2E bits of the System Control Register 2 (bits 1 and 0 respectively), must both be set to zero to disable Pulse Width Modulator and Serial Channel 2 functions of Port 9 pins 2 to 4, and enable normal port operation.

FRT3.TCR.OEA Output Enable A of the Timer Control Register of Free Running Timer channel 3. This bit must be clear to enable normal Port 9.1 operation.

FRT2.TCR.OEA Output Enable A of the Timer Control Register of Free Running Timer channel 2. This bit must be clear to enable normal Port 9.0 operation.

4.2.10 *CAN Port*

The CAN chip has a total of 32 byte wide registers which are mapped into the H8 address space from 0x8000 to 0x801f. These registers are read only, write only or read/write as defined in the CAN chip data sheet. Full details of these registers are given in the data sheet. A summary of register names and addresses is given below. To enable correct operation of this port the on-chip programmable wait-state generator should be set up to give two wait states on external accesses. This is achieved by setting <register> to <value>.

0x8000	Control Register (RW)
0x8001	Command Register (W)
0x8002	Status Register (R)
0x8003	Interrupt Register (R)
0x8004	Acceptance Code Register (RW)
0x8005	Acceptance Mask Register (RW)
0x8006	Bus Timing Register 0 (RW)
0x8007	Bus Timing Register 1 (RW)
0x8008	Output Control Register (RW)
0x8009	Test Register (none)
0x800a to 0x8013	Transmit Buffer (RW)
0x8014 to 0x801d	Receive Buffer (R)
0x801e	Unused (none)
0x801f	Clock Divider Register (RW)

4.3 *Can Bus Interface*

This section describes more fully the interface between the control microprocessor and the CAN bus. The CAN chip is a Philips PCA82C200, which contains all the buffering, parallel to serial conversion and back, packet framing and CRC error checking required to implement a multi-master two-wire serial bus.

A packet is transmitted on the CAN bus by writing it to the transmit buffer and writing a transmission request to the command register. On an interrupt or by polling the receive buffer status bit in the status register, the microcontroller can detect when a new packet has been placed in the receive buffer. This can then be read into the microcontrollers memory and the release receive buffer command issued. Note that the CAN chip has two receive buffers so that a packed can be received while the previous one is being processed by the microcontroller. The definition of the message protocol and content is beyond the scope of this document. See the associated document "CAN Bus Protocol".

The following sections give more detail on how the CAN chip should be initialised. Initialisation of the CAN chip should be performed by the H8 microcontroller every time it comes out of reset. Since the CAN chip and the H8 share a reset signal the CAN chip will be out of reset by the time the H8 comes to initialise it. The H8 and CAN chip are reset by a local power-on reset and when the backplane power-good signal is de-asserted. After a reset the CAN chip will have the 'Reset Request' bit in the control register set. This allows the other configuration bits to be safely adjusted. The register contents can be set as follows.

4.3.1 *Control Register*

Reset Request	This should be held high while configuration of the chip is in progress.
Receive Interrupt Enable	Set high normally
Transmit Interrupt Enable	Set high normally
Error Interrupt Enable	Set high normally
Overrun Interrupt Enable	Set high normally
Sync Edges	Set low normally to force receiver re-synchronisation on recessive-to-dominant edges only. Thee edges are more stable in time than dominant-to-recessive edges which may vary depending on line load and reflections. A result of this is that the CAN chip can only compensate for a clock frequency variation of 209 ppm between transmitter and receiver.
Test Mode	Set low.

This register should be written with 8'b00000001 at the start of initialisation to force a reset request and disable interrupts. At the end of initialisation it should be rewritten with 8'b00011110 to enable interrupts and release the reset request.

4.3.2 *Command Register*

The command bits in this register are all reset to their inactive states by the reset request bit being set in the Control Register. No initialisation of this register is thus required.

4.3.3 *Acceptance Code Register*

This register can only be set while the reset request bit is set in the Control Register. The contents of the register are used to select which of the data packets on the CAN bus are accepted and read into the receive buffer. The value that this register should be set to is related to the message Identifier (PacketID) that is allocated to Host-to-MK401 packets in the "CAN Bus Protocol" document. The Acceptance Mask Register can be used to select which bits of the Code register are compared, according to the pseudo-code fragment below. Note that a one bit in the Mask register means that the corresponding bit in the Code register is not checked:-

```
if (((PacketID[10:3] XNOR AcceptCodeReg[7:0]) OR AcceptMaskReg[7:0]) ==
8'b11111111)
 ReceivePacket;
else
 IgnorePacket;
```

4.3.4 *Acceptance Mask Register*

See the above section for more details. This register can only be set while the reset request bit is set in the Control Register.

4.3.5 *Bus Timing Register 0*

This register can only be accessed while the reset request bit is set in the Control Register. This register defines the baud rate on the CAN bus and the maximum number of clock cycles that a bit width can be adjusted by during a single synchronisation. See the CAN chip data sheet for the derivation of the required contents of this register. The magic value to program this register to is 8'b00000000.

4.3.6 *Bus Timing Register 1*

This register can only be accessed while the reset request bit is set in the Control Register. This register defines the bit period, the position within the bit period that the line is sampled and the number of samples taken. See the CAN chip data sheet for the derivation of the required contents of this register. The magic value to program this register to is 8'b00010100 (0x14).

4.3.7 *Output Control Register*

This register can only be accessed while the reset request bit is set in the Control Register. The register controls the output driver configuration. See the CAN chip data sheet for the derivation of the required contents of this register. The magic value to program this register to is 8'b10101010 (0xaa).

4.3.8 *Clock Divider Register*

This register controls the frequency at the CAN chip ClkOut pin which is not used on the Dino. No initialisation of this register need be performed.

4.4 *Status Encoding and Watchdog Timers*

Node status information is acquired by the H8 by the CAN Bus. The CAN chip connected on the IOBus, under control of the processor node, is used to transmit periodic status messages to the H8 CAN chip. The adoption of this communication method required the architectural decision that a CAN chip would be present on every compute node.

If the H8 receives no status information within a reasonable time of releasing node reset then it will assume the node is non-functional. Node status information is coded into a form suitable for presentation to the CAN Bus (if required) and also grossly coded into a set of status indications on the front panel LEDs. The exact protocol of the status message on the CAN bus is beyond the scope of this document, and reference should be made to the "CAN Bus Protocol" document.

4.4.1 *Watchdog Timers*

The 16-bit free running timers can be used to implement watchdog timers for each processor. These counters should be reset to some value which gives an acceptable delay every time a valid status message is received by the H8 from the compute node. If a timer should reach zero due to a processor crashing, the H8 should report this over the CAN bus. The H8 may then reset the board. Note that this will also stomp on the processes that were running on the other processor, so in some cases it may be better not to reset the node until the processor itself does it. This may not be possible if the other processor died while accessing some hardware.

This section describes a watchdog timer per processor. It may be sufficient to implement a watchdog timer per node, in which case the node should be reset as soon as it's watchdog times out.

Note that the H8 based watchdog timer operate in conjunction with an MBus watchdog timer. This latter will acknowledge an MBus transaction with a synchronous timeout acknowledge if the MBus busy line has been active for 200 microseconds. The time delay on the H8 watchdog should be considerably longer than this to allow for cases where multiple accesses to non-responding MBus addresses may be performed, for example during memory sizing at boot time, or during diagnostic self tests. It may also be worthwhile disabling the H8 watchdogs if this situation is likely to arise.

The H8 has a builtin watchdog timer to monitor its own operation. It is unlikely that the program on the H8 will ever crash, but the timer can be used to catch this situation. In the event of an H8 watchdog reset it is unlikely that the program will need to reset the nodes. A message on the CAN bus to inform the host that the node has had an H8 watchdog reset may be appropriate. See the "CAN Bus Protocol" document for details of these messages.

4.4.2 *Front Panel LEDs*

Two front panel LEDs are driven by the H8, one yellow and one green. They are intended to show broadly the status of the node which is derived from status messages transmitted by the node. The LED encodings required are as shown below.

Front Panel LED Encodings

Green	Yellow	Meaning
Off	Off	Unpowered
Off	Off	Reset Asserted
Off	Flash	Reset De-asserted but not Booting
Flash	Off	Executing Bootstrap or testing
On	Off	Normal Operation
On	On	Limp Along
Off	On	Dead

4.5 *Performance Monitoring*

Node performance statistics are collated by processes running on the node SPARC processors. A summary Bargraph is presented to external LEDs using the LED Bargraph register. Exactly how these LEDs are used is defined by the collating software being used and command-line options passed to it. The 16 bit register is visible to the H8 which should respond to requests for Bargraph information by passing the 16-bit value unaltered in a return data packet to the CAN Bus host.

4.6 *Reset Action*

Command packets on the CAN Bus can request slot-id specific or global reset. This is accomplished by holding the notNodeReset line active for a suitable period (eg. 100ms). SlotID value can be read and compared to the received id to be reset.

4.7 *SlotID Report*

A 12-bit SlotID is uniquely generated for each card in the system and is visible to the H8. The H8 should respond to requests for Node SlotID information by passing the 12-bit value unaltered in a return data packet to the CAN Bus host.

SPARC INTERRUPT CONTROL

This section describes the various sources of interrupts to the SPARC processors and how they can be masked. It also describes the software generation of interrupts which can be used for inter-processor communications or low priority interrupt routine scheduling.

5.1 *Interrupt Sources and Levels*

This section describes the sources of interrupts on the Dino and what level they occur on. Level 1 is the lowest priority, level 15 the highest.

Level 1 SBus Interrupt Signal 1.

Level 2 SBus Interrupt Signal 2.
CAN Bus Interface chip on SPARC IOBus.

Level 3 SBus Interrupt Signal 3.
SCSI-2 Interrupt Controller A and B (via SBusDMA chips).

Level 4 Software generated (see below).

Level 5 SBus Interrupt Signal 4.
Ethernet Controller Interrupt.

Level 6 Software generated (see below).

Level 7 SBus Interrupt Signal 5.
Elan Interrupt Request

Level 8 SBus Interrupt Signal 6.
MBus Slave device in MBus Slot 0 or 1 (if fitted).

Level 9 SBus Interrupt Signal 7.

Level 10 Low priority interval timer.

Level 11 Unused

Level 12 SIA for Serial Ports A and B.
SIA for Keyboard and Mouse Ports.
Software generated (see below).

Level 13 Software generated (see below).

Level 14 High priority interval timer.

Level 15 Asynchronous Error Latched.
Software generated (see below).

Of these signals, Levels 2, 3, 5, 7, 8, 9 and 12 are presented to the Elan chip as signals on its notExtDecInt pins 1 to 7 respectively. This allows the Elan chip to be used to handle some of the elementary device IO using threads processes if required. This scheme would relieve some of the IO processing load on the main processors, which would then either have the interrupt levels masked out or would execute no-operation interrupt routines.

More details on the circumstances which would cause a device to generate an interrupt and on the actions which should be taken to service and clear the request can be found in the device data sheets and manuals.

Response time from interrupt request to it being cleared is critical in a multiprocessor situation. For example if Processor 0 has been assigned to serial port interrupts (ie. the device driver for the serial ports has been installed on that processor) and Processor 1 has been assigned keyboard and mouse interrupts, then while Processor 0 is servicing the level 12 request Processor 1 will be repeatedly taking the interrupt trap, polling its installed devices (which doesn't include the requesting device) and returning. Thus if an interrupt occurs on a level which both processors have enabled then one processor will be effectively idle until the other has cleared the interrupt.

5.2 *Interrupt Masks and Software Interrupts*

The various interrupt sources described above pass through a complex logic circuit which masks out levels which have not been enabled, incorporates software generated interrupt requests, and prioritises the result for presentation to the SPARC Processor. To achieve this it contains several registers which are detailed below. There are two separate instances of this logic circuit, one for each processor, making it possible for each processor to separately enable the interrupts it wishes to receive and to request interrupts on itself and the other processor.

5.2.1 *Interrupt Mask Register*

This is a halfword wide read and specially writeable register, readable at halfword addresses given below. Halfword accesses are transparently broken into two IOBus transactions. Note that the Halfword is mapped into the least significant half of the word. Only Processor 0 should write to IRQ Pal 0 Mask Register, and similarly processor 1 to its pal.

The bits in this register correspond directly to the hardware interrupt levels, eg. bit 9 corresponds to the mask for level 9. Setting a bit enables the associated interrupt. Note that bits 0, 4, 6, 11 and 13 have no mask bit since there are no associated hardware interrupt lines at these levels. Writing to these bits will have no effect and they will always read as zero. The register defaults at reset to containing all zeroes, ie. all interrupts disabled.

There are two locations at which the register can be written, a set location and a clear location. A write to the set location will set each bit in the register which has the corresponding data bit set to one. Register bits with zero data bits will be unaffected. Similarly writing to the clear location will reset all the register bits for which the associated data bit is one. Again zero bits will not affect the register. This scheme removes the need for locked indivisible IOBus cycles to read and modify the register in a multiprocessor environment.

The addresses of these registers are as follows :-

0xff0701002 IRQ Pal 0 - Mask Register Read / Clear

0xff0701006 IRQ Pal 0 - Mask Register Set

0xff0702002 IRQ Pal 1 - Mask Register Read / Clear

0xff0702006 IRQ Pal 1 - Mask Register Set

5.2.2 *Software Interrupt Register*

This is a halfword wide register of which only a few bits are used. Software interrupts can be generated at levels 1, 4, 6, 12, 13 and 15. When the corresponding bit of the software interrupt register is set an interrupt will be generated at that level. Attempts to set or clear bits other than these will have no effect and they will always read as zero.

This register is readable and specially writable; it can be read at MBus halfword location 0xff070100a for processor 0 and at halfword location 0xff070200a for processor 1. There are two locations at which the register can be written, a set location

and a clear location. A write to the set location will set each bit in the register which has the corresponding data bit set to one. Register bits with zero data bits will be unaffected. Similarly writing to the clear location will reset all the register bits for which the associated data bit is one. Again zero bits will not affect the register. This scheme removes the need for locked indivisible IOBus cycles to read and modify the register in a multiprocessor environment.

The addresses of these registers are as follows :-

0xff070100a IRQ Pal 0 - Software Interrupt Register Read / Clear

0xff070100e IRQ Pal 0 - Software Interrupt Register Set

0xff070200a IRQ Pal 1 - Software Interrupt Register Read / Clear

0xff070200e IRQ Pal 1 - Software Interrupt Register Set

Note that the halfword registers are mapped into the least significant half of the word.

5.2.3 *IRQ Pal Status Latches*

This is a halfword wide read register which operates a clearing scheme similar to the software interrupt register, ie. writing to this register will clear those bits which have a one in the data. Only the following bits are used :-

Bit 8 Latched counter timeout for low priority (Level 10) timer.

Bit 15 Latched counter timeout for high priority (Level 14) timer.

Note that all other bits are unused and will always read as zero. This register at MBus halfword location 0xff0701000 for processor 0 and halfword location 0xff0702000 for processor 1. The same location is used for reads and clear-writes. Since only one byte is used of the register this can be accessed as a single byte transaction to the above addresses, using bits 7 and 0 of the byte.

5.3 *Interval Timers*

For full details of the timer chips see the AMD data sheet, "AM82C54 Programmable Interval Timer". A summary of the required programming steps is given here.

The timers should be configured to operate in mode 2 (Programmable Rate Generator). In this mode of operation they will count down from the initial value, causing the interrupt when the count reaches 1 and reloading the counter when the count reaches zero. Thus if the counter latch value is N, an interrupt will occur every N ticks of that counter's clock signal. Note also that the timer interrupt latch in the IRQ pal cannot be cleared until the count has reached zero (ie. one timer's clock tick after the interrupt was generated).

The two timers operate at different clock rates. The level 14 timer has a clock input of 1.25 MHz giving a time per tick of 0.8 microseconds. Thus for an interrupt every 10 milliseconds the value 12500 should be loaded into the counter latches. The level 10 timer has a clock input of 312.5KHz yielding a time per tick of 3.2 microseconds. Thus for an interrupt every 100 milliseconds the value 31250 should be loaded into the counter latches. The timers and timer latches are 16-bits wide so the maximum count value is 65535, or times of 52.428 milliseconds and 209.712 milliseconds respectively for the level 14 and level 10 timers.

The third timer in the package is not connected to any hardware signals, but can be used for software interval timing. It is clocked at the same rate as the level 10 interrupt timer, ie. 3.2 microseconds per tick.

Channel 0 of timer chip 0 is used to cause the level 10 interrupts on both processors, so Channel 0 of timer chip 1 is spare.