Algorithmics P−4032
User´s Manual

© 2000 Algorithmics Ltd

Revision: 2.5
Dated: 1999/08/17

P−4032 is a single board computer for prototyping and evaluating 32-bit MIPS R4x00 CPUs like NEC's Vr4300, IDT's R4640 or QED's RM5230 CPU for your application. It's fast, efficient, and economical, with excellent software support, and the design can be licensed in whole or in part.

We all know that you only read the manual if all else fails. But can we at least recommend that you read §1.1, "Key facts for the impatient" and §2, "Getting Started".

This manual is ©1996,97,98 Algorithmics Ltd, but anyone may reprint this document in whole or in part, so long as this copyright message is preserved.

Algorithmics Ltd
3 Drayton Park
London N5 1NU
ENGLAND.

Phone: +44 71 700 3301
Fax: +44 71 700 3400
Email: ask-algor@algor.co.uk

WWW: http://www.algor.co.uk/
FTP: ftp://ftp.algor.co.uk/pub/

Contents

1. Introduction to the P−4032

1.1. Key facts for the impatient

If you know quite a lot about MIPS already, and are familiar with programming at a low level, you'll still need the following parts of this manual:

- *Block Diagram*: you'll probably find it helpful to glance at Figure 1.1 on page 9.
- *Memory map*: refer to Table 3.1 to find out what registers are where.
- *Physical arrangement, connectors and jumpers*: described in §11 on page 42 below.
- *Board-specific programming*: no matter how familiar you are with the devices we've used, to program the board from scratch you'll need to know about the board configuration register, described in §9.1 on page 32.

1.2. P−4032 evolution

Early P−4032's were fitted with either an NEC Vr4300-133 or IDT R4640 CPU as a build-time option. More recent boards (built since January 1997, serial number 1000-) provide for the QED RM5230 CPU onboard and have a daughterboard interface for other choices.

This manual specifically describes the third revision of the daughterboard-ready P−4032. The hardware of this board has been fairly extensively redesigned, because the programmable logic device we used in early boards had become obsolete (thank you, Altera). However, apart from the loss of a couple of connectors which were really customer-specific, these changes should not affect use or programming of the board.

1.3. Features

- *Processor*: any of QED's RM5230/31, NEC Vr4300/4310 or IDT's R4640 CPU running at a system interface speed of 50, 60 or 66MHz. When we want to refer to any CPU we'll call it R4x00.

  These CPUs can be configured to run at a multiple of the system interface speed - usually twice. The CPU has 64 bit registers and a complete 64-bit instruction set, but uses a 32-bit external bus interface to save cost and power.

  Early P−4032 boards have positions for either a Vr4300 or R4640 CPU; recent variants bear the name "P−4032Q" and accept either an RM5230/31 soldered down or a CPU daughterboard. Daughterboards are available for the NEC Vr4300, IDT R4640, and RM5230/31 (for customers who want to be able to change their CPU).

  The CPUs are fairly compatible, but there are software-visible differences:

  Vr4300 100% compatible with earlier R4x00 CPUs; 16K I-cache and 8K D-cache, both direct mapped. Floating point and integer pipelines combined, which slows floating point operations somewhat but saves power and space.

  R4640 Less compatible (lacks double-precision floating point, no hardware memory management); 8K I- and D-caches, 2-way set associative with locking options. It features an integer multiply-accumulate instruction, useful for some DSP-like algorithms.

  RM5230/31 upward compatible with everything including the R5000 and a full MIPSIV instruction set. 16K I- + 16K D-cache (double size on '31), 2-way set associative. Includes R4640-compatible multiply-accumulate.

  Vr4100 low-power CPU likely to be used mostly as an ASIC core. No floating point hardware, small caches, much slower. The hardware interface is rather different, and it will be

difficult for customers to reconfigure P−4032 between Vr4100 and other CPUs. In fact, there may be no Vr4100 support available in later boards; check with Algorithmics.

- *Local Memory*: supports one or two industry-standard 72-pin SIMMs. Any standard EDO or page-mode 32-bit module can be used - including 32-bit flash ROM modules if you want to prototype a ROM-based application.

  The size and type of memory is user-configurable. The maximum theoretically achievable memory size is 256Mbytes, and is 128Mbytes with modules available in early 1996.

- *Boot ROM*: supports both socketed conventional EPROM and flash PROM for economical, field-upgradeable bootstrap.

- *PCI expansion bus*: is PCI2.1 compatible[1], 33MHz, 32-bits wide. There are two PC-standard 5V edge connector sockets.

- *Ethernet*: implemented with the DEC 21041 controller, attached onboard via the PCI bus.

- *High-performance SCSI interface*: using an onboard Symbios 53C810 controller, is available for local disk or tape devices, or customer-specific SCSI peripherals.

- *PC-compatible I/O*: a combination I/O controller provides dual serial ports (16550 compatible), bidirectional Centronics and a diskette interface. The Centronics port can play the role of a “peripheral” to your PC host, or a host to your printer or other peripheral.

  So the board can talk to you when all else fails, there’s a 4-character alphanumeric LED display used by the boot ROM and other code which likes it. To bridge the software/hardware gulf there’s a general purpose parallel I/O (PIO) connector, which allows you to wiggle or watch any of 8 lines under software control.

  And we provide a PC-compatible keyboard controller, real-time clock, and an optional LCD “front panel” display.

- *EEPROM*: a simple serial device provides 4Kbits of non-volatile “environment” storage.

- *Interrupt controller*: a custom design which can be software-configured to suit a wide range of customer or OS requirements.

- *Field-reconfigurable logic*: most of the board’s logic is implemented in re-programmable FPGA devices. We can supply logic upgrades over internet.

- *Software support*: delivered with the PMON boot monitor [PMON], for which full source code (including all drivers) is available free on request. The standard bootstrap also includes a ROM-based power on self test (“AlgPOST”). Algorithmics SDE−MIPS toolkit [SDE−MIPS] provides a library for P−4032.

  Algorithmics can supply BSP (“board support packages”) for real-time OS’ such as VxWorks from Wind River Systems and pSOS from ISI. Versions of freely-redistributable operating systems (notably OpenBSD and Linux) are available too.

  Free software for P−4032 can be found on our internet ftp server at `ftp.algor.co.uk`.

- *Reset/debug switch*: that little switch (SW1). It can be pushed left (away from the center of the board) for a full reset, and right to produce a “debug” interrupt. The debug interrupt is wired into a regular interrupt, not the *NMI* (non-maskable interrupt), and does interesting things to the boot-up sequence.

[1] PCI is a fast-moving target; see the online errata for deviations. They don’t affect everyday uses.

1.4. Block Diagram

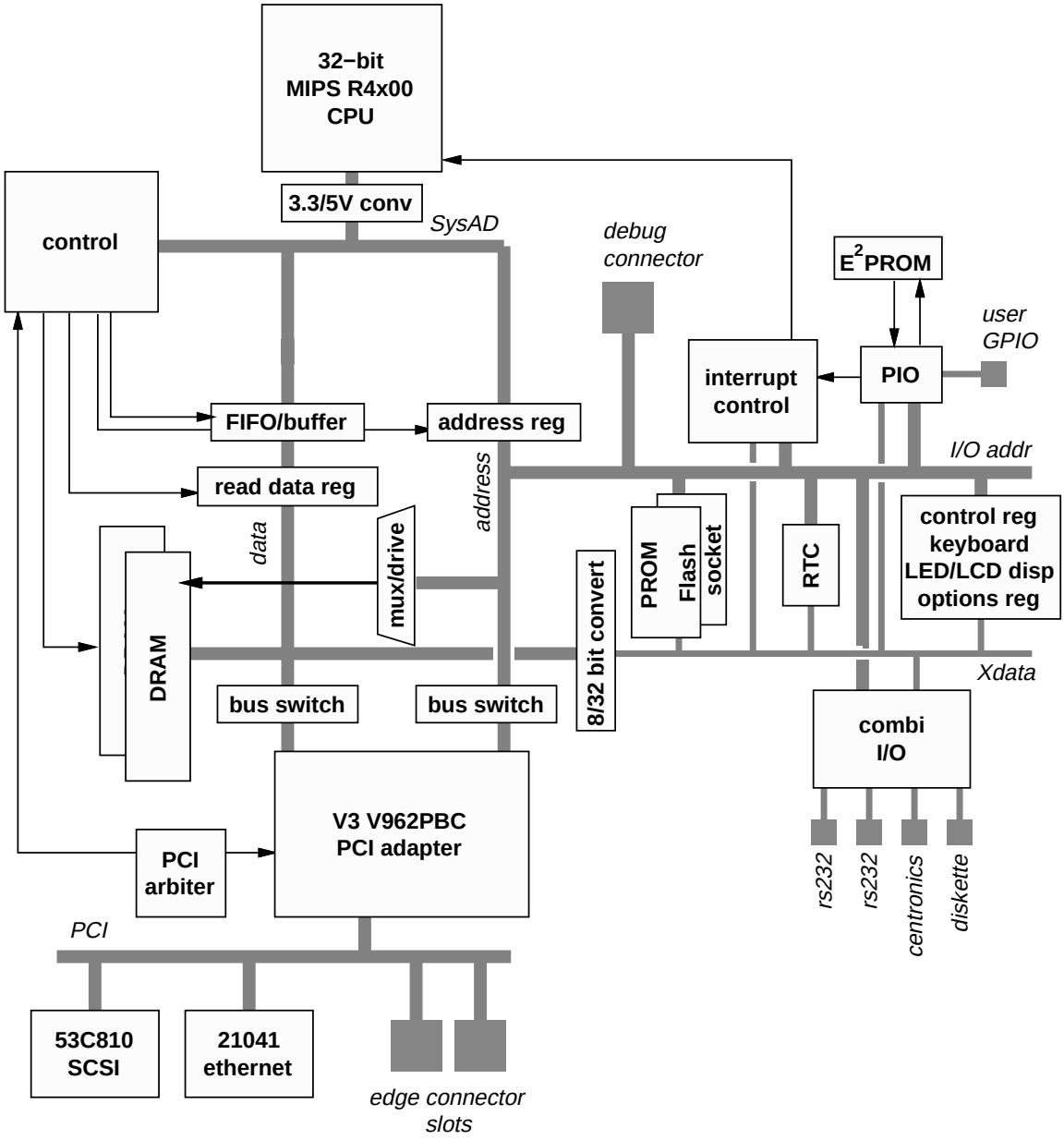

Figure 1.1 P−4032 block diagram

Notes on the block diagram

- *Buses*: the CPU local bus (“SysAD”) is a 32-bit wide, 5V-signalled, multiplexed bus running synchronously at the CPU’s interface clock rate - up to 67MHz.

  CPU data passes through a write FIFO, allowing CPU cache writeback bursts to run at full speed.

  The intermediate bus has a dual personality. During CPU/memory transactions, it runs at CPU interface speed; but during transfers to or from PCI it runs at one-half CPU speed, and with timings compatible to the local bus of an Intel i960 CPU.

- *V3 V962PBC*: is a local-bus/PCI converter, originally designed for i960 applications.
- *Bus switch*: decouples the PCI converter chip when the intermediate bus is carrying CPU/memory or CPU/local I/O traffic.
- *DRAM*: two sockets for 32-bit SIMM modules.
- *IO bus and Xdata*: provides an 8-bit bus for onboard PC-type peripherals. The “8/32-bit converter” can steer IO bus data to any byte lane of the read data register; when the CPU reads from the 8-bit ROM space the hardware performs four byte-wide reads and assembles them into a 32-bit word.

1.5. A note on EMC

The electronics industry in both Europe and the USA is now concerned with stray emissions (and sensitivity to) electromagnetic radiation. P−4032 is not currently certified under European regulations, because it is not itself a system but only a component[2]. By design, P−4032 is relatively insensitive to incoming radiation; it may be affected by power glitches, but it is the power supply’s job to filter those.

Many of you will be using P−4032 open on a bench set up. Use of a 100MHz+ system without any overall metal shielding is likely to produce radiated emissions above the levels permissible for office (let alone domestic) equipment. The European regulations specifically provide for laboratory set-ups, on the basis that it is your responsibility to ensure that no nuisance is caused to a third party. The best shielding is distance; don’t set up your board a few feet away from someone else trying to watch TV!

P−4032 is designed to be compatible with widely available “PC” boxes, power supplies and cables, and its radiation will be sharply reduced if those are of good quality. Algorithmics may at some point issue a boxed system product or specification, which would need to be certified and “CE”-marked. Write to us if you need that. Meanwhile, the board is a component for use in laboratory environments, and the user is responsible for managing radiated emissions.

[2] There is some debate in Europe about whether all assembled PCBs should be covered by the “CE” registration scheme, but it’s still an open question.

2. Getting started

Most of you should read this section.

2.1. What's in the box?

Everybody should find:

- *P−4032 user's manual*: but you got that, because you're reading it.
- *PMON user's manual*: describing the boot monitor and startup sequence. A useful reference for when things go wrong, but many of you won't really have much to do with it.
- *P−4032*: configured with the CPU, and the amount and type of memory, you ordered.
- *Transition Cables*: there should be a bunch of standard transition cables, which you can either use to bring out connections for a board in a PC box, or just to mate with standard bought-in cables.

You may also find (if you ordered them):

- *Extra cables*: some are optional, see §13.1.
- *Debug board*: makes it much easier to watch addresses/data in your program. Invaluable for driver and ROM-code debug; see §14 below.
- *LCD display*: an optional 16×2 alphanumeric display supplied loose with a short transition cable. It's up to you to find a mounting point, or just leave it lying around. More information in §13.12.

2.2. Initial wiring up

P−4032 is quite happy operating on a bench top. There are no dangerous voltages, and nothing will get too hot. You'll need to connect at least power, and possibly some other stuff.

- *Power*: PC power supplies are cheap, electrically safe, and plug right in. The connector is in two parts, but put the black (ground) wires together and you'll be safe.
- *Serial port(s)*: if the connection from your computer terminates in a female D-type (9-pin for PCs, or 25-pin for old RS232 standard), then there's a good chance that you can wire them up with the supplied cables. If not, or you've lost the cables, refer to Table 13.4 below for the connections on board.

  The PROM monitor signs on at 9600 baud, sends 8-bit characters with no parity, and (in its default configuration) accepts pretty much anything back again.
- *Ethernet*: you need an external transceiver. The supplied cable plugs in.
- *Centronics for download*: if you have a normal PC centronics cable then you'll need the centronics peripheral transition cable, which is an optional extra.

2.3. Boxing a P−4032

P−4032 is designed to fit into PC metalwork - older rather than "ATX" PCs - and matches a small PC motherboard in its size, fixing hole positions and standard connectors (PCI, keyboard, power supply). PC metalwork varies, so you may need some patience.

The serial and centronics transition cables supplied with the board terminate on PC back-panel fingers. The SCSI cable supplied is suitable for use in-box. You'll need to make your own arrangements for other I/O.

Most PCs have a reset button lead, which will mate with the 2-pin header J2, allowing the board to be reset from the front panel.

2.4. Normal sign-on sequence and what it means

From power up your P−4032 will show signs of life by writing enigmatic codes to its LED display (just in case you expected English, it starts by saying "U*U*"). At the same time it's sending rather more meaningful messages to both serial ports. Here's a typical example:

P−4032 says	*What it means*
`Info: Version: P4032 (EB) 1.17: (chris) Thu \ Feb 13 13:35:05 GMT 1997`	PROM sign-on. "(EB)" for big-endian, "(EL)" for little-endian. Note that the PROM contains both the power-on selftest code (AlgPOST) and the ROM monitor (PMON). This is AlgPOST starting up. And the "\" shows where I've folded a single line which is too long for this table.
`Info: Activity: ICU operation`	"Info:" denotes a test starting. If you get nothing but "Info" and "Notice" lines from the power-on tests, then they didn't find anything really wrong.

`Info: Activity: cache tests`
`Info: Dcache size 8 Kbytes (16/line)`
`Info: Icache size 16 Kbytes (32/line)`
`Info: Activity: dcache refill test`
`Info: Activity: dcache writeback test`
	Started cache tests

`Info: Activity: RTC operation` `Info: Date: Fri 25/4/1997 10:38:00 UTC`	
`Info: Memory Size 8Mb, Simm0 4Mb SIMM1 4Mb \ DCR 0xdd`	"DCR" is the hex value of the DRAM configuration register, described in §5.1 below.
`Info: Activity: quick memory address test`	More thorough tests are available, see PMON manual for how to make them happen.
`Info: Activity: flash memory operation` `Info: Flash: Fujitsu 29F080`	
`Info: Activity: ns16550 operation`	That's the dual serial port controller
`Info: Activity: keyboard operation` `Notice: No keyboard attached`	
`Info: Activity: PCI operation`	
`Info: V962 silicon revision 2`	The PCI bridge chip. Revision 2 indicates a "B.1" part, which is not so good. See our web site for bugs inherent in the B.1 chip, and let's hope your board announces a revision of 3 or higher.
`Notice: Integrated Tests Completed` `Notice: Executing PROM package 6`	Control is now being handed over from the power-on tests to PMON.
`PCI slot 5: Digital Equipment DECchip 21041 \ ("Tulip Pass 3") (class: network, \ subclass: ethernet)`	PMON is probing for active PCI devices
`PCI slot 8: NCR 53c810 (class: mass \ storage, subclass: SCSI)`	
`de0: P4032 DC21041 [10Mb/s] pass 1.1 Ethernet address 00:40:bc:03:00:44`	Ethernet driver initialisation.

P−4032 says	*What it means*

`PMON version 3.1.155 [P4032,EB,FP,NET]`
`Algorithmics Ltd. Jan 24 1997 11:46:17`
`This software is not subject to copyright \`
`and may be freely copied.`
	PROM monitor version and date

`CPU type R4300. Rev 2.0. 133.3 MHz.`
`Memory size 8 MB.`
`Icache size 16 KB, 32/line.`
`Dcache size 8 KB, 16/line.`
	From CPU ID register and measurement.
	PMON should agree with AlgPOST
	These figures are right for the Vr4300, others differ

`PMON>`	You've got a prompt

Table 2.1: P−4032 ROM sign-on sequence

Startup troubleshooting and switch flipping

As the board powers up, the LED shows a code for each set of tests. The display blinks out briefly as each individual test is started.

Lower-case codes are good, but upper case codes from AlgPOST are bad (at least, after it's initial "`U*U*`" stuff). Upper-case test names from AlgPOST mean a warning or worse; always stay around for long enough for you to read them; and are accompanied by a console message unless the console is not working or configured off.

Confusingly, PMON puts upper-case messages on the display and those aren't errors; but they tend to zoom past really fast until you get a gently flashing "`PMON`" - and that indicates that the system is up to the PMON prompt.

If the board seems to be expiring really early, you may want to turn up the thoroughness and verbosity of the power-on tests. Usually, this is controlled by environment variables; but if you can't reach the PMON prompt you can't change those. So you can do it by wiggling the debug/reset switch; reset the board in the usual way, but instead of releasing the switch move the switch all the way over to its other ("debug") position, and hold it there for a couple of seconds. AlgPOST will now test everything (including some rather tedious memory tests) and tell you pretty much everything about it.

2.5. Flash memory and ROM socket

P−4032 normally boots from an onboard 1M×8 flash memory, pre-loaded by Algorithmics with power-on tests and the PMON ROM monitor program. You can create and write your own bootstrap; software running out of DRAM can update the flash memory in place.

If your board won't boot and you believe that the flash memory may be corrupted, there is a socket (U18) which accepts an alternative bootstrap source. The device is usually a 512K×8 150ns EPROM, in a 32-pin dual in-line package, but you can use a suitable flash part (AMD 29F040 or equivalent), by changing the jumpers J9 and J10 from their default (2−3) position to (1−2).

The board will use the ROM socket for its bootstrap if you insert jumper J8.

A copy of PMON in S-record format, ready to run in your board, can be downloaded from Algorithmics' web site `www.algor.co.uk`. You can also download a program to run under PMON, which will write a clean bootstrap image to your flash memory.

Flash memory updates are performed through two separate address windows onto the onboard and socketed devices.

2.6. PMON

PMON is the bootstrap monitor program supplied in ROM, described much more fully in the “PMON User’s Manual” which all board customers should have received. Many users will make use of only a fraction of PMON’s facilities:

The environment store

The board environment is implemented in an EEROM device separate from the main memory map, and is intended to be shared by any software which wants to store small amounts of per-board configuration information. In PMON you use the “set” command to inspect or create environment entries. To edit existing entries, the “eset” command gives you line editing.

Table 2.2 shows a typical dump of variables from P−4032; we’ll explain what they mean.

```
PMON> set
   ethaddr = 00:40:bc:03:00:44
   itquick = t
   netaddr = 192.168.1.67
  hostname = comm67.comm.algor.co.uk
nameserver = 192.168.1.65
   gateway = 192.168.1.65
  tftphost = oval
         v = gate:/vol/tornado/target/config/p4032/vxWorks
    dlecho = off         [off on lfeed]
   dlproto = EtxAck      [none XonXoff EtxAck]
  hostport = tty1
   heaptop = 80020000
    moresz = 10
    prompt = "PMON> "
    brkcmd = "l @epc 1"
    datasz = -b          [-b -h -w -d]
     bootp = no          [no sec pri save]
   inalpha = hex         [hex symbol]
    inbase = 16          [auto 8 10 16]
  regstyle = sw          [hw sw]
   regsize = 32          [32 64]
    rptcmd = trace       [off on trace]
   trabort = ^K
      ulcr = cr          [cr lf crlf]
     uleof = %
   validpc = "_ftext etext"
```

Table 2.2: P−4032 - typical PMON environment variables

What do all these mean?

- *ethaddr*: Without this, no network. The first part of the address (“`00:40:bc:03`”) is the same for all boards; the last four digits of the hex ethernet number are the board’s serial number (but in hex); this board is serial 0068, which is “`0044`” in hex.
- *itquick*: Suppresses long-running power-on memory tests. See PMON manual for how to ask for more power-on tests.

- *netaddr, hostname, nameserver*: You need either a “netaddr” or both a (suitably registered) “hostname” and “nameserver” to be set up. Either gives the board an identity for communication.

  If you need more information about setting up the network, read the PMON manual.

- *gateway*: default gateway. Network data for any host which is not on the local network (does not respond to ARP request) will be sent here. Useful if you keep your prototype boards on a separate network.
- *tftphost*: default network host to use when using *tftp*. You can always give an explicit host name.
- *v*: a typical programmer-set shortcut, allowing you to just say:

  ```
  PMON> boot $v; g
  ```

  to load and run the program.

- *dlecho, dlproto*: control download over serial or parallel link. P−4032 can echo characters (if the link is bidirectional) or use a character-based flow control protocol.
- *hostport*: select which device is to be used for download. The device can either be shared with the PMON console, or separate. Possible download device names are:

  `tty0` is the first serial port, “com1”, also used for the PMON console.

  `tty1` is the second serial port, “com2”.

  `tty2` is the centronics port, using peripheral mode.

- *heaptop*: how much DRAM memory PMON uses, starting from zero represented as a MIPS “kseg0” address in hex. This is the lowest address at which you can load your program. You can set “`heaptop`” somewhat lower; but not to zero (PMON has to have some writable memory to operate in) and PMON may be unable to do some things for you without enough free memory.
- *moresz, prompt*: PMON user interface controls.
- *brkcmd etc*: these variables configure the operation of PMON as a debug monitor, and you’ll have to look in the PMON manual for them

Instant PMON

There’s so much more in the PMON user manual, but worth mentioning:

- *Command editing*: use emacs/unix style keys to move around and edit characters.
- *Booting from ethernet*: uses the “boot” command from PMON, and loads ELF object files.
- *Booting from serial or parallel ports*: use the “load” command of PMON, and can accept a variety of download formats such as S-records.

3. Memory map

3.1. R4x00 addressing - program and physical addresses

In MIPS CPUs the addresses generated by your program[3] are never the same as the physical addresses which come out of the CPU and affect the rest of the system.

This is different from most familiar CISC architectures, and this often causes confusion. CISC CPUs often have a mode bit which enables memory translation - and without that mode bit set the physical address is exactly the same as the program address. MIPS has no such mode bit. Instead, the CPU' program address space is split into regions, as shown in Figure 3.1:

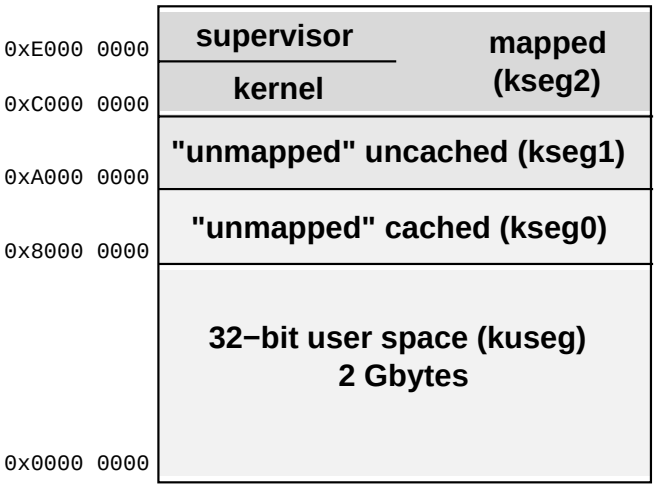

Figure 3.1 MIPS program address map

The regions *kuseg* and *kseg2* are designated for translation; addresses in these regions will be presented to the hardware's memory translation unit (the *TLB*), and what happens then is beyond the scope of this section. If you want to know more, read an architecture book such as [MIPS R4000] or [Using MIPS].

Embedded software more often runs in *kseg0* and *kseg1*, each of which offers a window onto the low 512Mbyte of physical memory (cached and uncached respectively). *kseg1* is essential to run startup code (before the caches are initialised), and is also needed for access to hardware I/O registers. Once the system is running most system code and data will be accessed through *kseg0*.

Actually, the picture shown above in Figure 3.1 is not complete. The R4x00 is, after all, a 64-bit CPU and not 32-bits, and the full program address space is 64 bits big. Figure 3.1 is useful because, so long as you only use the 32-bit-compatible part of the MIPS instruction set, registers will only contain 64-bit values whose top 32 bits are all set to the same value as bit 31 - such values look like a "sign extension" of a 32-bit value.

So the 32-bit memory map is in fact the view you get of the whole 64-bit memory map when you leave the middle out. Figure 3.2 shows the big picture:

[3] Called *program addresses* here - the term *virtual address* means exactly the same thing but is unfamiliar outside the exotic realms of big operating systems

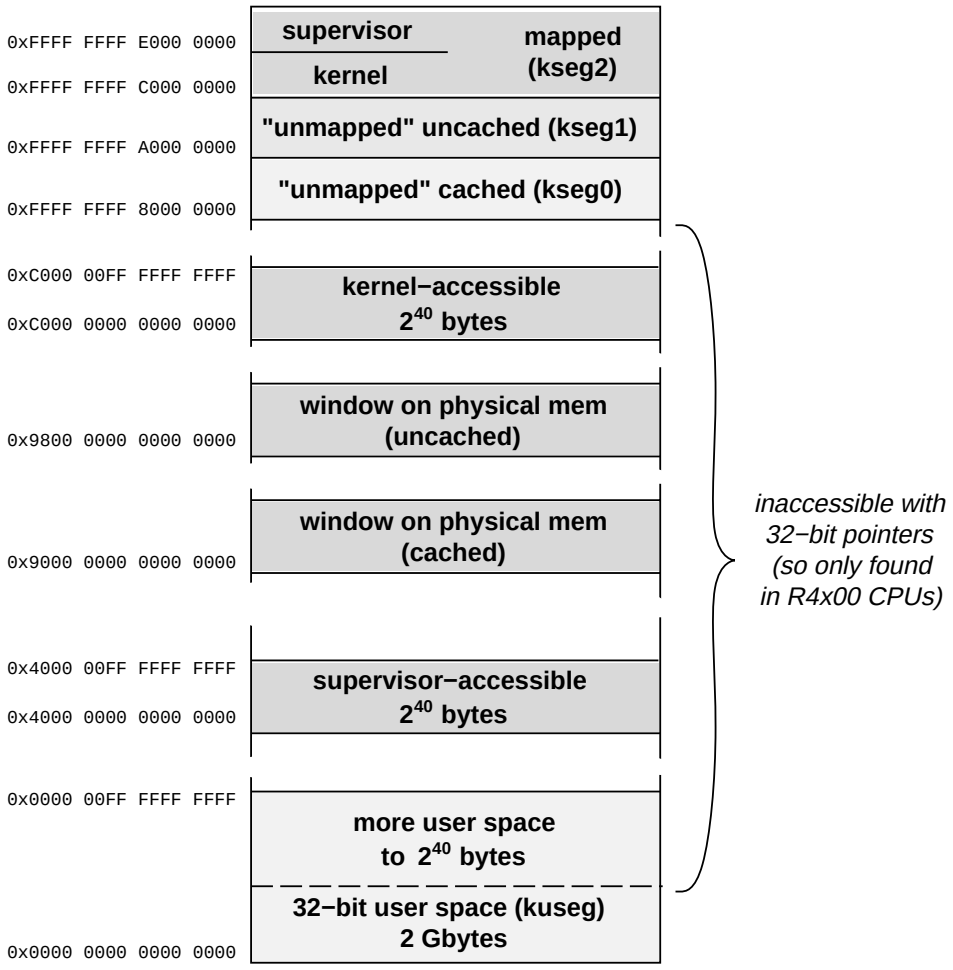

Figure 3.2 MIPS program address map (entire 64-bit space)

Handling pointers as 64-bit objects is an extravagant use of memory space for an embedded software application; and we reckon most users won't bother. If you need access to the R4x00's 32-bit physical address range outside the low 512Mbytes (so can't just use *kseg0* and *kseg1*) you can use the TLB.

3.2. Physical memory map

The CPU generates a 32-bit address. However:

- Following a reset the CPU starts execution at `0x1FC0 0000` (physical) - which must therefore map to onboard ROM.
- Much system software finds it easier to operate in the *kseg0* and *kseg1* "unmapped" spaces described above in §3.1, and such programs will only generate physical addresses up to and including `0x1FFF FFFF` (the low 512Mbytes of address space).

Note that revisions of this manual before 2.5 describe a different memory map, which turned out to make ISA bus programming difficult. If you wrote software which used the original map, and don't need ISA bus access, the PMON boot monitor will restore the original map if the environment variable "pcimap" is set to "`old`".

In the memory map Table 3.1 a dagger (†) denotes that the address is software-configured at boot time - the value given is recommended and fits in with the hardware decodings.

Base Address	*Size*	*Class*	*Description*
0000 0000	256Mb	Memory	onboard DRAM memory
0400 0000			Flash SIMM module memory if fitted
1000 0000	8Mb†	ISA via PCI	“ISA” memory access window. This is a window on the first 8Mbytes of PCI memory space, and should not be allocated to either onboard or add-in PCI controllers. It’s there because some PC legacy controllers - particularly video cards - are hard-wired to respond to some low addresses.
1080 0000	8Mb†		Reserved. This range of memory, as decoded on the PCI bus, provides an unmapped window onto local memory. If the MIPS CPU accessed these locations nothing very useful would happen - the self-decode obstructs access to the ISA bus
1100 0000	112Mb†	PCI	Window on PCI memory space (by default, generates addresses 0x1000 0000 lower on the PCI bus). PCI devices get dynamically allocated addresses starting at PCI address 0x0100 0000 which is CPU address 0x1100 0000 Although PCI devices’ base addresses are programmable, you should normally leave them where the bootstrap program left them. Find a particular device by reading PCI configuration space and getting the values already programmed into the base registers.
1800 0000	109Mb		reserved
1ed0 0000	1Mb		PCI I/O space window†: you’ll only use PCI I/O space for PC “legacy” controllers, but this space is reserved for it if needed.
1ee0 0000	1Mb		PCI configuration space: access to PCI devices’ configuration registers. In P−4032, PCI device *IOSEL* signals are derived from high PCI bus address bits - see §6.2. (“PCI wiring”) on page 28 for details. You need to program some V962PBC registers to make this work.
1ef0 0000	64Kb†	V962PBC	PCI controller’s internal registers
1ef1 0000			reserved
1fc0 0000	1Mb	ROM	Boot ROM location. Reads either ROM socket (512Kb only) or flash locations, depending on configuration)
1fd0 0000	512Kb		Programming window for socketed flash memory
1fe0 0000	1Mb		Programming window for onboard flash.
1ff0 0000	1 reg	PC-type I/O	Real time clock “pointer” register
1ff0 0004	1 reg		Real time clock data
1ff1 0000	2 reg		Keyboard controller
1ff2 0010	4 reg		LED display cells (leftmost has lowest address)
1ff3 0000	x reg		LCD display
1ff4 0000	x reg		General-purpose parallel I/O (GPIO)
1ff8 07c0	x reg	Combi I/O chip	Diskette port
1ff8 0800	-		unused (Games port)
1ff8 0be0	x reg		com2 serial port
1ff8 0fe0	x reg		com1 serial port
1ff8 0de0	x reg		Centronics port
1ff9 0000	3 reg	ICU	Interrupt requests/masks
1ff9 000c	3 reg		Interrupt crossbar registers

Base Address	*Size*	*Class*	*Description*
`1ff9 001c`	1 reg	Ver	P−4032 design revision
`1ffa 0fd4`	1 reg		Diskette “DMA acknowledge”
`1ffb 0000`	8×1-bit	PC-type	Board configuration register (1 bit/location)
`1ffc 0000`	8×1-bit	I/O	DRAM configuration register (1 bit/location)
`1ffd 0000`	1 reg		Option register
`2000 0000`	3.5Gb	PCI	Use this if you have to (you’ll need to program the TLB or use 64-bit pointers)

Table 3.1: P−4032 physical address map

4. Processor and options

4.1. Clock rate/Clock rate multiplier

The CPU's input clock is derived from a synthesiser IC and is set by a group of jumpers summarised in Table 11.1; likely choices are 50, 60 and 67MHz.

The CPU runs internally at a higher speed (2, 3, 4 or 5 times higher) than the system clock, according to the setting of the "CDIV" field of the jumper J22, described in §9.2. ("Option register") on page 33. Most often, the CPU will run at at 2× or 3× the bus speed.

4.2. CPU daughterboard

Recent boards (serial 1000-) accept a wide range of CPU types, mounted on a small daughterboard. Daughterboards are available for Vr4300, R4640, and RM5230/31. The daughterboard connector is described in §13.2 below, and may be a useful place for looking at raw CPU signals.

The idea of the daughterboard is so we can configure and ship boards with a range of CPUs. We don't want to expose the CPU connector as an "open" interface, because the documentation job would be way beyond cost-effective. Customers who want to be able to switch between two different CPUs should contact Algorithmics for instructions.

But we can tell you that changing a CPU will involve some or all of the following:

- Change the daughterboard itself.
- Change the system logic, which is stored in "flash"-programmable logic chips and can be downloaded through the Centronics port using software and files obtained from Algorithmics' web site.

  Some pairs of CPUs (eg. R4640 and RM5230) share a common set of logic, and you may just need to change a jumper.
- Change the CPU input clock rate by changing the clock control jumpers.

4.3. Endianness

All MIPS CPUs can be configured with either "endianness" - if you're not sure what this means look at §12. ("CPU endianness") on page 46. With the exception of certain hand-crafted simple sequences, program binaries for big- and little-endian MIPS are different and incompatible; so you have to match the CPU configuration with the software you're running.

On P−4032 there are three pieces of hardware to configure:

- The CPU itself. The NEC Vr4300 is setup in software (it starts up big-endian, but changes sex with an internal register bit). Other CPUs require some kind of reset-time hardware sequence.

  The CPU's endianness is set with the "BigEnd" jumper in the option register jumper block, described in §9.2. ("Option register") on page 33. It is always software-readable.
- The CPU bus interface, which interprets the CPU's read and write commands. Partial-word read/writes are signalled by the CPU in an endianness-dependent way.

  The CPU bus interface is set up by software, responding to a bit in the configuration register described in §9.1 below. It's initially little-endian, so big-endian CPUs must set this bit before attempting any partial-word accesses.
- The PCI bus interface. PCI is inherently little-endian, and it's wise to have a byte lane swapper between PCI and the rest of the system when the CPU is big-endian.

  The V962PBC bridge chip provides a software-selectable byte lane swapper.

Of course, you also have to provide software (including a boot ROM) for the appropriate endianness. The standard boot ROM will detect a mismatch between CPU and ROM endianness and give you a diagnostic

message.

Note that getting even this minimal ROM code to operate bi-endian requires that P−4032 be wired so that CPU instructions (and 32-bit loads and stores) are consistently interpreted regardless of the state of any of the above.

Note again: if the CPU and bus interface options don’t match, partial-word transfers will transfer garbage.

5. Local memory

5.1. DRAM configuration

You can fit one or two 72-pin, 36-bit wide DRAM SIMMs - as used on most PCs. If you only have one SIMM, it goes in SIMM0. If you are using a DRAM-compatible flash memory SIMM, it goes in SIMM1.

P−4032 handles a wide range of memories (size, type and speed). The most common options are software selectable; larger sizes and more unusual DRAM types require a logic firmware variant; contact `p4032@algor.co.uk`.

Modules and mixes supported

With the standard decoding logic you can use single-bank or double-bank SIMM memory modules in sizes of 1M×32, 2M×32, 4M×32 and 8M×32 - the last must be a double-bank type. If you use two double-sided DRAM modules they must be the same size.

You can use 70ns fast page mode or 60ns EDO modules; if two modules are fitted they must be of the same type (a 60ns EDO will work with page mode timings, but slower).

You can fit a flash SIMM module (1M×32 or 2M×32 double-banked) in the second slot.

- *DRAM size*: some decoding changes are required to accommodate different sizes of DRAM SIMM, and a range of common options can be setup by programming a set of write-only registers, documented below. Algorithmics' startup code figures out the sizes of the installed modules and configures the DRAM controller accordingly.
- *DRAM speed and type*: P−4032 supports several different DRAM timing sets. The timing is configured under software control by the DRAM configuration register (defined in §5.1 below); but to avoid confusion software should always initialise the DRAM controller as instructed by the "Mem Type" and "Flash SIMM" fields of the option register, described in §9.2. ("Option register") on page 33 below.

  All timings are configured in CPU system interface clock cycles. The options are:

  1. Fast page mode DRAM with 1-clock CAS (suitable for 50MHz system with 60/70ns DRAMs).
  2. 60ns EDO DRAM with 1-clock CAS (for system at any speed).

     There was to be a "burst" system, which would give the best memory performance achievable with our SIMM pinouts; but it has never been implemented.
  3. Fast page mode DRAM with 2-clock CAS (required for 67MHz system).
  4. Flash ROM - curious SIMM parts using multiplexed addressing. Initial access time is similar to 80ns DRAM, but CAS cycling is much slower.

     If flash ROM is fitted, it's base address is always 64Mbytes (`0x0400 0000`).

Jumper configuration for DRAM type/speed

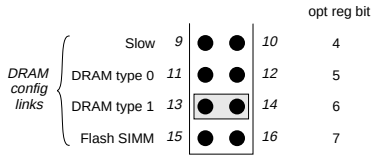

Figure 5.1 Software option jumper J22 (DRAM bits)

In Algorithmics' startup code the configuration bits relating to the type and speed of the SIMM modules are copied from the "software option" jumper block to a write-only configuration register on power up. The jumper positions are shown in Figure 5.1.

So long as this convention is maintained the links need to be set up to match the installed SIMM modules, as shown in Table 5.1.

Jumper Name	*in/ out*		*Effect*
Slow	`in`		70ns DRAM on 67MHz system
	J22 13-14	*J22 11-12*	
	`out`	`out`	reserved
DRAM type	`out`	`in`	regular EDO
	`in`	`out`	reserved
	`in`	`in`	fast page mode
Flash SIMM	`in`		Flash ROM module in SIMM1

Table 5.1: Configuring SIMM module types

DRAM Configuration register

Implemented as an array of single-bit registers. This is a bit odd - but you don't expect to change it very often.

In Table 5.2:

- *BankSz*: bank size of the memory array plugged into SIMM0. Accesses at and beyond "BankSz" will either go to SIMM1 (single-banked module in SIMM0) or the second bank on SIMM0.

  If SIMM1 is single-banked, then the hardware doesn't need to know it's size; it extends all the way up to the maximum DRAM address space limit. The software sizing routine can figure out what it really is.

  But if SIMM1 is double-sided, the hardware has to know when to switch to the second bank; and under these circumstances SIMM1 is assumed to have the same bank size as SIMM0.

- *Sngl0/Sngl1*: set 0 if the memory array in SIMM0/SIMM1 respectively has two separate 32-bit banks using four separate *RAS** signals.

Offset	*name*	*Value*	*Effect*
0x1c	Flash SIMM	1=DRAM, 0=flash	0 if SIMM1 module is a flash ROM module
0x18 0x14	DRAM type	0 0 = Page mode DRAM 0 1 = reserved 1 0 = regular EDO 1 1 = reserved.	Select type of DRAM in use
0x10	Slow	1=normal, 0=slow	Set “slow” for 70ns DRAM in a 67MHz system
0x0c	Sngl1	1=single, 0=double	SIMM1/SIMM0 module “double banked” - it has a second bank of DRAM on the module, accessed by alternate *Ras** signals
0x08	Sngl0		
0x04 0x00	Bank Size 0	0 0 = reserved 0 1 = 4 Mbyte 1 0 = 8 Mbyte 1 1 = 16 Mbyte	Bank size of SIMM0 module. For single-bank modules, this is the same as the module capacity; for double-bank modules, it’s half of the module capacity.

Table 5.2: DRAM configuration register fields

- *Slow*: set 0 to slow the DRAM access speed down, when using 70ns DRAM in a 67MHz system.

  A 67MHz CPU’s burst read timing (for cycles not delayed by external access or DRAM refresh) are as follows:

DRAM type	*Access rate*
70ns page mode	8-3-3-3
60ns EDO	7-2-2-2

How to size the DRAM

How should you establish how much memory of what type is fitted, in order to program the controller to talk to it? First of all, look at Table 5.3, which shows how CPU addresses are fed to the DRAM SIMMs in the different sizes:

Type	*DRAM address row+column*	*RAS-time bits*	*CAS-time bits*	*Bank size(s)*
1M×32 2M×32†	10+10	A12-21	A2-11	4Mbyte 2×4Mbyte
2M×32	11+10	A12-22	A2-11	8Mbyte
4M×32 8M×32†	11+11	A12-22	A2-11,23	16Mbyte 2×16Mbyte
8M×32	12+11	A12-22,24	A2-11,23	32Mbyte
16M×32	12+12	A12-22,24	A2-11,23,25	64Mbyte

Table 5.3: How CPU addresses reach the DRAM SIMMs

Some 2M×32 and 8M×32 modules (marked with †) use separate *Ras** signals to access two banks of RAM on the same module. We call these “double-banked” - they are often physically double-sided, with chips on both sides of the module.

Startup code is expected to figure out whether each of the SIMM modules is double-sided, and their bank sizes.

To sense what size banks you have, set up the board for the *maximum* DRAM size, and for single-bank operation. If the real SIMMs are in fact smaller than the maximum size some address bits will get lost and you will see the memory "wrap around".

The wraparound will happen at the first location which is too big for the actually-installed DRAM.

Once you've discovered and configured the bank size, set the double-bank configuration bit. If the SIMM module is single-banked, you'll now get a "hole" above the good bank addresses in which no DRAM will respond. Take care not to get confused by "ghost" data, returned when a floating bus holds the value you last wrote (see the note on "Outcomes of out-of-range memory accesses" below).

A Flash SIMM module is always accessible from 64Mbytes (`0x0400 0000`) and, if double-sided, is assumed to have 4Mbyte banks.

Now repeat the process for SIMM1. The only irritation is that you can't (yet) support a different bank size for SIMM0 and SIMM1 when SIMM1 is double-banked.

A good algorithm for sizing a bank is:

- Set the word at the bank's base address *Base* to some recognizable pattern such as `0x1234 5678`.
- For each *n* which is a plausible size for the SIMM memory size (starting with the *largest* value) set the word at physical address *Base + n* to the value *n*. The easiest way to generate the physical address is to use the kseg1 address *Base + n* + `0xa000 0000`.

  Plausible sizes start at 1Mbyte, and go up in multiples of 4.
- Read the word at physical address zero (ie `0xa000 0000`) to obtain the size of the SIMM in bytes.

Outcomes of out-of-range memory accesses

The memory controller responds to any access in the region dedicated to "onboard DRAM memory". An access to any part of the region for which DRAM is not configured and installed may result in any of the following:

- *Random data returned*: the data returned is unpredictable. A write to the location will have no effect.
- *Ghost data returned*: the data returned is the same as the last data written to ANY location in the DRAM system. A write to the location will have no effect. Take care that these "ghost" values don't confuse your memory-sizing code into believing that memory must be there.
- *Access wraps around*: an access to an address higher than any legitimately configured and installed may read or write another location, whose address is typically a large power of 2 lower than the address produced by the CPU.

Software which computes the size of the onboard memory at initialisation must be designed to cope with any of these behaviors. To detect wrap-around you can use an address-in-address test; to avoid being confused by ghost data you should separate the write and read back of a test location with a cycle which writes the bitwise complement of the test data to some other location.

5.2. 8-bit ROM

P−4032 is bootstrapped from one of a pair of 8-bit ROM devices.

CPU word-size reads from ROM space are accomplished by reading four consecutive bytes from ROM and presenting them on the data bus. Note that byte 0 of PROM is always presented on bits 0-7 of the data bus, regardless of CPU or board "endianness".

To update the flash ROM you need to grapple with some strange effects caused by the ROM's endianness; see below.

5.2.1. Boot PROM socket

Accepts a 32-pin uV-erasable or flash PROM of up to 512Kbytes (4Mbits). Add the jumper J8 to map this part into the MIPS startup location `0x1fbc0000`. With the jumper out, the board will bootstrap from the onboard flash PROM.

By default, your board is setup for a uV-erasable ROM. To use a flash device (AMD 29F040 or equivalent) Change the jumpers J9 and J10 from their default (2−3) position to (1−2).

5.2.2. Flash PROM

One 1M×8-bit flash memory device is provided onboard. Your board will have been delivered with bootstrap monitor code in flash. Refer to the monitor manual for information on how to upgrade the software.

For peculiar reasons a CPU running big-endian can only read the flash ROM's own mapped regions as words, or cached. Uncached reads to the flash area are handled oddly, and return the wrong byte for normal purposes.

If your board is set up to boot from flash, the flash ROM behaves properly when read through the boot window.

Notes about writing to P−4032's flash PROM

Flash memories require special programming sequences to erase and write data.

There are special commands available to erase the ROM (a "sector", usually 64Kbytes, at a time), to prepare it for programming. The programming routine loops programming a location until a status port bit indicates success. Programming is described in the data sheet [FlashData], or you can obtain example software from Algorithmics.

It's only possible to write one byte at a time to the flash PROM.

ROM data stored at ROM address 0 mod 4 is always sent to or received from the CPU on data bus lines 0-7. By making this association independent of CPU endianness, the ROM can support "bi-endian" code for its bootstrap[4].

This is exactly what a little-endian CPU would want to see, but is not consistent with the normal conventions for big-endian data.

When reading from the "bootstrap" memory region (which reads flash memory or ROM socket depending on a jumper setting), the ROM control logic always reads a whole word; the logic actually reads four bytes from the ROM and assembles them for the CPU. A MIPS CPU doing a partial-word read then selects the byte(s) it wants, and will be happy.

We can't use the same technique for writing, or for reading flash memory status. So byte accesses to the flash memory programming windows are handled specially; the byte-within-word address fed to the flash ROM is taken directly from the CPU bus.

The net effect is that a big-endian CPU sees its data swapped in the flash ROM when it's reading or writing bytes, and sees correct data when it's reading any other memory size. In general, you can't run code from the flash ROMs own memory window on a big-endian CPU - byte reads would return the wrong value. Note that you *can* run cached code (cache refills always fetch whole words) and you could write a special routine avoiding all byte operations...

When programming the flash ROM on a big-endian CPU, we recommend that you build up aligned 32-bit words of PROM data, and then copy it to flash in four 8-bit chunks, with bits 0-7 going to the low-addressed byte and bits 24-31 to the high-addressed byte.

[4] This works because MIPS instructions are aligned 32-bit objects, always read 32 bits at a time. Think of instructions as an array of words, and then pack bits 0-7 of the first word into byte 0 and so on.

5.3. Serial EEPROM

P−4032 is fitted with an 4Kbit 93x66-type serial PROM, controlled through some of the PIO controller's pins (see §9.9. (“General-purpose parallel I/O (PIO)”) on page 36). Refer to the sample driver which implements a simple “environment store”. This is used by Algorithmics' power-up test software, and boot monitor, to store essential information like the ethernet address. If your software needs to store parameters in non-volatile memory, this provides a good way to do it.

Here's the relationship between PIO controller pins and EEPROM bus signals:

GPIO pin	*EEPROM pin*	*EEPROM function*
B4	DI	Data to be written to EEPROM
B5	CS	EEPROM chip select
B6	SK	EEPROM data clock
B7	DO	data from EEPROM

Table 5.4: EEPROM signals and GPIO pins

For use of the chip refer to a data sheet. We used [Atmel93C66].

6. PCI interface

The PCI interface provides two standard slots for expansion cards, as well as hosting the on-board ethernet and SCSI controllers. PCI runs at 33MHz (irrespective of the processor operating frequency).

The PCI interface is built using a V3 V962PBC device [V962PBC] which is designed to convert between an Intel i960's local bus and PCI. Custom interface logic converts all the CPU's non-DRAM bus cycles into a form which is compatible with the V962PBC's "i960" signalling and its half-CPU-rate interface clock. The DRAM is effectively dual-ported to the local bus, and thus accessible from PCI bus masters.

There are two PC-style PCI edge connector sockets.

6.1. PCI accesses

The CPU can access PCI devices through the "aperture" programmed into the PCI controller. This provides some simple high-address-substitution memory mapping. The CPU can read and write any PCI space in single cycles, but the CPU-to-PCI logic does not support bursts, so you can't access PCI through cached space.

PCI masters can access the local memory, again through the apertures programmed into the PCI controller. PCI master burst cycles will result in burst accesses, up to 8 words long, in the local DRAM.

6.2. PCI wiring

6.2.1. PCI configuration space, IDSELs and interrupt assignments

In normal use, PCI devices respond to accesses relative to base addresses set up by initialisation software. There must be some way of programming devices before they are set up, so PCI defines a "configuration space[5]" where devices are addressed by means of per-device *IDSEL* signals provided by the motherboard hard-wired decoding. In P−4032 PCI IDSELs are obtained from individual PCI *AD* lines, as shown in Table 6.1 below.

Device	*AD line*	*Interrupts*			
		INTA	*INTB*	*INTC*	*INTD*
Ethernet ctrlr	16	PCIIRQ1	-	-	-
PCI slot 1	17	PCIIRQ2	PCIIRQ3	PCIIRQ0	PCIIRQ1
PCI slot 2	18	PCIIRQ3	PCIIRQ0	PCIIRQ1	PCIIRQ2
SCSI ctrlr	19	PCIIRQ0	-	-	-
Custom extended connector	21	PCIIRQ0	PCIIRQ1	PCIIRQ2	PCIIRQ3

Table 6.1: IDSEL for PCI devices/slots

In all cases the *IDSEL* line is connected to the corresponding *AD* line through a resistor. The value on the *AD* bus is mostly "don't care" during configuration cycles, so to direct a configuration cycle at the ethernet controller you'd set the PCI address to something like `0x0001.0000` - which would set *AD16* to a "1" and *AD17-22* to "0".

PCI devices typically connect to one interrupt line, and the signal used is by convention related to the *IDSEL* number.

[5] The original PCI configuration mechanism (based on what are now called "Type 0 configuration cycles") proved inadequate to handle large systems using multiple buses connected by bridge chips. PCI 2.1 includes an additional "Type 1 configuration cycle" which works across bridges. P−4032 does not ever provide nice support for Type 1 cycles, and boards fitted with "B.1" revision PCI controllers don't do Type 1 at all. If you need Type 1 cycles, contact Algorithmics.

P−4032 has four separate interrupt lines, as recommended by the PCI standard (interrupts are not part of the formal PCI specification, so this is just a recommendation). PCI add-in cards are offered a choice of four interrupt signals; single-function cards also use only the “INTA” position, but multi-function cards can use any interrupt signal.

To generate an *IDSEL* for PCI slot 1, you need to generate a host address which is:

a) within the “PCI configuration space” window as shown in Table 3.1 above; AND

b) translates to a PCI address where bit 17 is a “1”, and bits 16 and 18-22 are “0”.

You will need to program the PCI configuration space window base address register within the PCI converter device; see [V962PBC] for details[6].

6.2.2. PCI arbitration

Arbitration should be invisible to the programmer, and usually is. But see Table 6.2 for which signal is attached to which device/slot.

Arbitration signal	*Device*
PCIGNT0~	Ethernet
PCIGNT1~	SCSI
PCIGNT2~	PCI Slot 1
PCIGNT3~	PCI Slot 2
PCIGNT4~	Extended customer slot (P7)

Table 6.2: PCI arbitration signals for devices

6.3. PCI interface registers

Extensively documented in [V962PBC]. One day we may summarise this here; meanwhile Algorithmics will supply customers with C code examples on request.

6.4. PCI startup

From system reset the PCI bus reset is asserted, and is held until it is explicitly cleared by writing a V962PBC register. V962PBC has no PCI-accessible “configuration space”, but can be configured by the CPU in its own local-space window. You need to fully configure the chip before releasing the PCI bus reset signal.

Note that the PCI bus reset is used for the PCI slots, but is not wired to the onboard ethernet and SCSI controllers; they are held in reset by dedicated signals from the board configuration register (see §9.1).

6.5. PCI performance notes

PCI is capable of delivering very high throughput. It’s also capable of performing miserably. What do you need to get good performance on P−4032?

There are two dimensions of performance.

- *Latency*: the delay experienced when making a single access over the bus, typically characterised as the time taken to read one location.
- *Bandwidth*: the rate at which data is transferred across the bus between a data source and sink.

[6] If you have the “SDE−MIPS” toolkit from Algorithmics, you should find you have sample code to do this as part of board initialisation.

Most high-bandwidth PCI peripherals are “bus masters” - they initiate data transfer cycles on PCI and read or write P−4032’s local memory.

By the standards of onboard buses, PCI is built for fairly high bandwidth (133Mbytes/s) but latency can also be quite high (a few μs is quite normal). Getting good bandwidth in the face of transfer delays is quite an art. Unfortunately, the data buffers in P−4032’s PCI chip are inadequate in size and poorly designed; so 20-30Mbyte/s is pretty good in this environment.

But much worse than that is possible. The bridge chip and local i960-like bus protocols will impose a significant delay on returning read data to a remote initiator. If this delay exceeds 16 PCI clocks for the first word, or 8 PCI clocks between words in a burst, then PCI rules require the current transfer to be stopped (“disconnected”). When the delays are working against you, you can end up transferring data across the bus one word at a time; and you’ll be lucky to see 2Mbyte/s like that.

Here’s some simple recommendations:

- If you can, program your PCI bus master to attempt bursts of 16 or 32 bytes, and try to set up your buffers to get those “naturally” aligned to 16- or 32-byte memory boundaries.
- Pushing data (where the initiator is writing memory) is much faster than pulling (initiator reading). If you only had the choice...
- To speed transfers from memory to PCI device, enable “prefetch” in the PCI bridge chip’s local memory aperture. Unfortunately, the bridge chip has had some bugs which make this difficult; consult the online bug list.

  There’s also a PCI bridge control bit called something like “RD_POST_INH”. In PCI terminology a “read post” is a read cycle which is deliberately and promptly terminated with read retry by a target which has reason to believe that it can’t get the data back within 16 clocks. P−4032’s PCI bridge will do this for every external read from local memory unless you set the RD_POST_INH bit to “1”.
- When P−4032 runs code out of its boot ROM, each cache line refill turns into 32 separate byte reads, and occupies the local bus for over 4μs. This can cause big delay problems for PCI devices trying to access local memory. Run your code out of DRAM - the boot ROM really is just for booting.

7. Ethernet interface (DEC 21041)

See the manufacturer’s data sheet [DEC21041] and user guide, or the software driver examples provided by Algorithmics, for the use of this part.

In P−4032:

- *Ethernet clock*: is defined by a dedicated 20MHz crystal.
- *Ethernet controller reset*: is a programmable signal dedicated to the DEC controller, from the board configuration register, described in §9.1.
- *Interface signals*: the chip is protected by a transformer.
- *PCI connections*: the DEC 21041’s *IDSEL* signal is derived from PCI address line *AD16,* and it’s interrupt output is wired to *PCIIRQ1˜,* as shown in Table 6.1.
- *Transceiver power*: +12V is provided on the interface, protected by a 0.75A fuse.
- *Shield ground*: normally connected to board ground, the shield signals can be isolated by removing jumper J1.
- *EEPROM interface*: (provided by the chip) is not used in this design.

8. SCSI interface (Symbios 53C810)

Once again, this is too complicated to describe. You can read the manufacturer's documentation [Symbios53C810] to find out more, or use the sample drivers.

SCSI signal timing relies on the *SCLK* input to the 53C810 chip, which on P−4032 runs at the CPU input clock frequency (usually 67MHz, but may be anywhere down to 50MHz).

The board has active SCSI termination, which can be disabled (by removing jumper J5) if P−4032 is not attached at the end of the SCSI cable.

A +5V power source for remote termination is provided by default, but can be disconnected by removing jumper J6; but it's diode protected, so it's probably always safe to leave the jumper in. The terminator power supply is protected by a self-resetting fuse; if you should short it out disconnect the SCSI cables and wait a while before trying again.

The SCSI chip reset input is a dedicated signal from the board configuration register, described in §9.1.

9. Onboard I/O

You will not find very much information here about programming the onboard I/O devices. But the good news is:

- You can obtain C include files describing their registers, and simple polled-mode drivers, free from Algorithmics.
- Better still, the same basic drivers are available as part of Algorithmics' SDE−MIPS cross-compiler package. You can build simple example programs for P−4032 (either to download under PMON, or to run on the raw hardware) straight out of the box.

  SDE−MIPS is available at a special price to P−4032 purchasers. See "SDE−MIPS for P−4032" on page 64.
- Since most of the minor devices are compatible with PC clones, freely redistributable drivers are available as part of various operating systems which run on PC hardware; look at Linux, NetBSD and FreeBSD.
- We've included references to the device manufacturer's data sheets in Appendix A ("References and further reading") on page 62.

One significant difference from the PC standard is the way interrupts are handled; see §10 below.

9.1. Board configuration register

The board configuration register consists of an array of single-bit registers described in Table 9.1; on a write only bit zero of the data supplied is significant. All bits are cleared to zero by system reset.

Offset	*Name*	*set 1 to*
`0x00`	PCICRst	Release PCI controller from reset
`0x04`	PrAutoBsy	Enable centronics "auto-busy" logic, see §9.6
`0x08`	LedOn	Unblank LED alphanumeric display
`0x0c`	SCSIRst	Release SCSI controller from reset
`0x10`	FlpTC	Assert TC to diskette controller; used to mark the last DMA acknowledge cycle of a block
`0x14`	EthRst	Release ethernet controller from reset
`0x18`	LE	Set bus interface for big-endian

Table 9.1: Board configuration register fields

Here's what the bits do:

- *PCICRst*: when 0 (as it is from power-up) it resets V962PBC and hence the whole PCI subsystem. V962PBC (designed to operate in a specific i960 environment) appears to be unhappy about the power-on reset sequence. To be safe, the firmware deliberately cycles reset to the controller during system startup.
- *PrAutoBsy*: used when operating the centronics port in unidirectional "peripheral" mode; but you should read §9.6. ("Centronics") on page 34 for details.
- *LedOn*: set 0 to make the display dark, 1 to enable.
- *SCSIRst, EthRst*: active-low resets for the SCSI and ethernet subsystems. Write a 1 to permit these systems to operate normally.
- *FlpTC*: signal to simulate the "terminal count" output of a PC-compatible DMA controller during diskette data transfers. Should be asserted as soon as the penultimate byte of a block has been acknowledged. See §9.7. ("Diskette interface") on page 35 for details.

- *LE*: sets the CPU interface up to adapt to a little-endian CPU. The CPU's own endianness is controlled separately, either by an internal register or by hardware; the PCI byte swapper (which should be set to swap when the CPU is in big-endian mode) is implemented by the PCI controller.

9.2. Option register

This is a link-defined read-only register. Most bits have no hardware effect. Non-readable options which affect the hardware are setup in J24, described in §11.4. ("Jumper options") on page 43 below.

The software register returns the values setup on the jumper J22. Note that it reads 0 for jumper *in*, and 1 for jumper *out*.

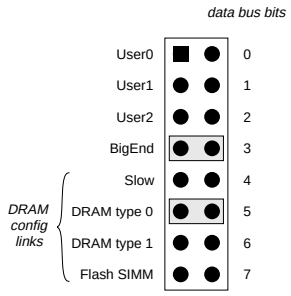

Figure 9.1 Option header/register (J22)

Where:

- *User0-2*: no fixed function - available for OS/user functions. See software manuals and help screens for existing uses.
- *BigEnd*: 1 (jumper out) for big-endian, 0 (jumper in) for little-endian. This is a software-only bit for the Vr4300 CPU, which sets its endianness in software. "Bisexual" startup code should read this bit to decide whether to be big or little.

  Note that even where the CPU is hardware-configured by this bit, the external logic is controlled separately by the board configuration register LE bit defined in Table 9.1 above, and by byte swap registers inside the PCI controller.

The highest four bits (Reserved, Flash SIMM, and DRAM type) are software-only, but by convention are copied by reset-time software to the corresponding bits of the DRAM configuration register (described in §5.1). You should read that section for their definitions; but note that the "Reserved" jumper should be left out.

The "default" setting shown in Figure 9.1 is for a little-endian CPU with standard 60ns EDO DRAM.

9.3. Design revision register

The revision register (actually implemented by the same FPGA as the interrupt controller) returns an 8-bit number which helps get you appropriate support by tracking changes in P−4032's design. Every bug-fix or feature enhancement leads to a unique value in this register. However, the revision register itself was only introduced in boards from serial number around 1100 onwards.

Our convention is:

- This location returns a zero value on all boards prior to the proper implementatino of the revision register.
- Circuit changes are reflected in the top two bits of the register, with the value zero denoting the "D" artwork revision current when the register was first implemented.

- Logic updates within a particular circuit revision are reflected in the rest of the register.

One day soon the PMON monitor will report the revision register's value on startup.

9.4. Combination RS232/centronics/diskette interface

The Winbond W83787F/W83877F provides two (16550-compatible) serial ports, a centronics parallel port, and a diskette drive controller. It's described in [Winbond]. There really are two different parts used, and some programming is slightly different.

Because this is built to implement standard DOS serial and parallel ports, basic programming is compatible with every PC in the universe. Because the chips it emulates are originally separate functions, they're described separately in the next three sections.

9.5. Dual UARTS

The serial ports are software-compatible with the 16550 (like the 16550, they have useful-sized data transmit and receive FIFOs), and operate at speeds up to 115Kbaud. The serial port timing source is a 24MHz crystal, which is divided by 13 to give a UART clock of 1.846154 MHz. This is only 0.16% higher then the usual PC UART clock of 1.8432 MHz (well within RS232 tolerances).

Programming is PC-compatible; or refer to the sample drivers.

9.6. Centronics

The Winbond chip implements a subset of the ISA Extended Capabilities Port (ECP) interface standard, defined by Microsoft and HP [Centronics ECP]; with appropriate software it can support the full set of modes described in the IEEE1284 standard.

The controller was conceived to implement the host-side interface, and not the printer side. But on P−4032 the port also provides a peripheral interface on a second connector (to support functions such as downloading from Windows/95 to the board)[7]. Table 9.2 shows how the controller signals are re-deployed when in peripheral mode.

Controller Signal		*Centronics Signal*
nStrobe	→	nAck
nAuto	→	Busy
nInit	→	PError
nSelectIn	→	nSelect
nAck	←	nStrobe
Busy	←	nAuto
PError	←	nInit
nSelect	←	nSelectIn
PIO(B0)	→	nFault

Table 9.2: Centronics connections in "peripheral" mode

In "peripheral" mode the Winbond controller lines are wired to the connector to exchange the roles of complementary signals, as shown in Table 9.2.

These connections allow the Winbond's FIFO-based, high-speed handshaking to continue to work in ECP mode.

[7] Alas, we've had a lot of trouble making this work well, though it is reliable with Windows95 if you configure your PC to use the "ECP" driver. Read the online buglist, or contact Algorithmics, if you have trouble.

Note that since there are normally 5 printer→host inputs, but only 4 host→printer outputs, we need one extra output bit in peripheral mode; the general-purpose PIO controller's *B0* output (described in §9.9 below) is used to drive Centronics *nFault.*

Moreover, in peripheral mode the Centronics protocol (even the simplest "compatible" mode), require that the *Busy* signal should be asserted immediately (you have less than 1μs to do this) after the host has sent data by asserting *Strobe.* This is too fast for software, so P−4032 provides some external logic which asserts *Busy* after every datum is received, and de-asserts *Busy* again when you program an *Ack* response.

This logic is not required in the P1284 fancy bidirectional modes; but when you're being an old-fashioned printer you can enable this logic by by writing a bit in the board configuration register, described in §9.1. ("Board configuration register") on page 32. When the "auto-busy" logic is enabled, the *Busy* output (we're in peripheral mode so that's *AutoFd* from the Winbond chip) is ignored. Auto-busy is enabled from board reset.

No DMA controller is provided on the local bus, so the DMA features of the Centronics interface can't be used. You can do everything with the CPU - and the Winbond chip's FIFO will reduce the interrupt load.

The Centronics interrupt does not remain active until serviced, but consists of a pulse. It is latched by the interrupt controller, and can be cleared by writing the appropriate bit to the "Interrupt Clear" register described in §10. ("The Interrupt system") on page 38.

9.7. Diskette interface

This emulates the NEC μPD765 device used in PCs since time began; see [Winbond] for details[8]. The floppy port uses "DMA" service; although there is no DMA controller available for this device, there's enough hardware support to simulate DMA cycles using a fast, simple, high-priority interrupt routine, which responds to DMA requests by reading or writing diskette data. The interrupt routine has to respond within about 16μs - this requires a very low-level routine, and you must avoid disabling interrupts for significant periods. But looking on the bright side, the DMA crisis time is enough for the CPU to run two thousand instructions!

Simulating DMA requires three things:

- Diskette "DMA requests", indicating that the chip is ready to accept or supply diskette data, are implemented as the interrupt called *DRQ2* in Table 10.2.

  The service routine is simplified because you can set up the interrupt controller so that this is the only interrupt cause on this CPU input.
- On early boards (serial no 190 and less): read from the diskette data port to obtain a byte of data.
- Acknowledge the DMA request with a read from the "DMA acknowledge" port shown in the memory map (Table 3.1). On later boards, this will also do the data transfer.
- During the last pseudo-DMA acknowledge cycle of a block, you need to stop the diskette controller by asserting its *TC* signal. This is under software control as part of the "board configuration register" described in §9.1.

[8] There are significant differences in programming the diskette interface on the two variant Winbond chips. Consult Algorithmics if you are changing the Winbond programming on a P−4032 board whose serial number is 0190 or lower.

9.8. Real Time Clock (RTC)

The real-time clock remembers the date and time with a resolution of 1 second, and uses a PC-compatible Benchmarq BQ3285E device. It provides a programmable tick and alarm which can cause an interrupt. Note that the *SQW* output is programmed at 32kHz by the firmware and must not be changed - it is used to generate DRAM refresh timing.

It also provide a small amount (242 bytes) of read/write memory which is retained over power-down.

The RTC chip uses a long-lifetime 3.6V battery (BT1) to keep time when system power is off. The battery can be replaced when it eventually runs down; most boards use a Varta 3/V60H Ni-MH (3-pin footprint) - a few early-production parts used a 2-pin type.

The real time clock is connected on P−4032 so that it takes two cycles to make any register access. Write the relevant register number to the “register select” address, and then read/write the data at the “data” address.

In normal use jumper J12 is set to short 1-2, to feed battery power to the RTC. But it can also be set 2-3, which has the effect of resetting the RTC clock registers and non-volatile RAM; or left out entirely, which leaves the battery open-circuit - which saves running the battery down while the board is in storage.

9.9. General-purpose parallel I/O (PIO)

This chip is provided because it is both a cost-effective way to fulfill some onboard requirements for extra I/O bits, and it can also help customers who need to build custom processor-controlled interfaces consisting of a few input and output lines; it can be a godsend when you need to work around a bug in a piece of experimental hardware, or when using hardware test equipment to trace software execution.

It provides two 8-bit bidirectional ports, with various latching options and the ability to generate an interrupt on various input conditions. Outputs and inputs are brought to a pin header.

9.9.1. GPIO bits used for onboard functions

Port/Bit	*Signal Name*	*In/Out*	*See section*	*Used for*
B0	`nFault`	Out	9.6	Drives this Centronics interface signal when the interface is being used in peripheral mode.
B4	`DI`	Out	5.3	Used to read/write the serial E^2PROM, as described in that section.
B5	`CS`	Out		
B6	`SK`	Out		
B7	`DO`	In		

Table 9.3: Parallel I/O bits and onboard functions

9.10. LED or LCD display

These are two separate devices, and in principle both could be fitted; in practice only one will be.

LED display

The LED display is a Siemens DLR2416 or equivalent - a four-character ASCII display. Each display position is accessed as a separate writable 7-bit register (the most significant bit is don’t care) - curiously, the lowest-addressed register is the *rightmost* character position.

Each position can display any of 128 characters. A familiar US ASCII character set is used for character values of 0x20-0x7e (’ ’ − ’˜’). In addition 32 special European and graphic characters are available in character positions 0x00-0x1f, as shown in Figure 9.2.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0-0xF	ì	↑	←	↓	→	?	À	Φ	φ	Ò	Ù	n	ç	ê	É	é
0x10-0x1F	è	Æ	æ	Å	å	Ä	ä	Ö	ö	Ü	ü	C	F	ß	£	¥

Figure 9.2 Alphanumeric Display Extended Character Set

The sample driver conveys longer messages by scrolling at a human-readable speed.

LCD display

The LCD display is a 16×2 back-lit LCD display suitable for panel mounting, based on the Hitachi HD44780 or compatible controller.

It should be connected to the header P13 with a short ribbon cable. The connection and adjustments are described in §13.12.

You can no doubt get manufacturer's data sheets, but we found programming information on the world-wide web [LCD-Ouwehand].

9.11. PC keyboard controller

Uses a standard pre-programmed microcontroller, usually from "American Megatrends". Refer to the sample drivers for a software interface.

10. The Interrupt system

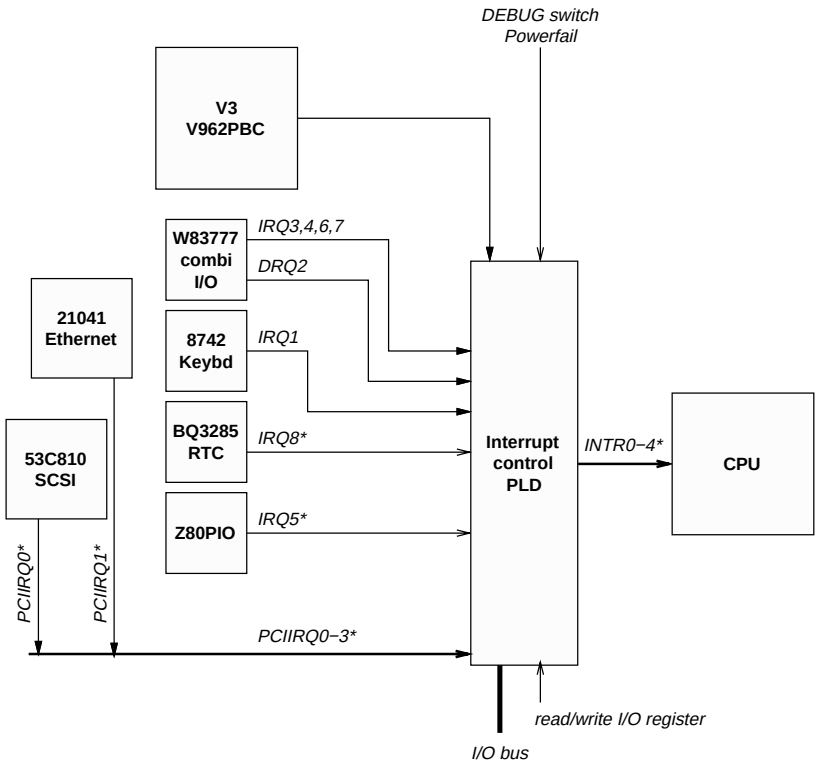

Figure 10.1 Interrupt system block diagram

The interrupt system is pictured in Figure 10.1. The interrupt control PLD takes in all the possible interrupt signals and provides:

- *Interrupt request registers (IRR)*: software readable locations which return the current level of any device interrupt.
- *Interrupt mask registers (IMR)*: provides an individual enable/disable for most of the interrupt inputs (but not the high-priority “panic” inputs). A “1” enables, and “0” disables, a particular interrupt.
- *Interrupt crossbar registers (IXR)*: for most of the interrupts, a two-bit field determines which of the CPU interrupt inputs *Intr0-3** will be asserted when it is active and unmasked. Some interrupts (panics and the diskette interface’s “pseudo-DMA”) have fixed assignments to CPU inputs.
- *Interrupt clear register (ICR)*: most interrupt inputs are “level-triggered”; they remain in the active state until programmed otherwise by the device driver. But three of the interrupt inputs (the debug switch, the bus error indication, and the interrupt from the Centronics controller) are signalled with pulses, and the interrupts are latched in the interrupt controller. You need to write to the ICR register to un-latch them; a “1” written to the appropriate bit clears the interrupt, a “0” leaves it unchanged.

All the registers are 8 bit wide. The interrupt registers are summarised in Table 10.1; the interrupt signals are summarised in Table 10.2; and the register layouts are defined in Figure 10.2.

Address	*Register*	
	Read	*Write*
1ff9 0000	8-bit device IRR	8-bit device IMR
1ff9 0004	Panic IRR	ICR
1ff9 0008	PCI IRR	PCI IMR
1ff9 000c	-	local IXR 0
1ff9 0010	-	local IXR 1
1ff9 0014	-	PCI IXR

Table 10.1: Interrupt register addresses

What	*Signal*	*From Device*	*CPU input*
R4x00 timer		inside CPU	*Int5*
Powerfail	PWRGD	PSU	
Bus timeout	BUSERR~	bus logic (latched)	*Int4*
Debug	DBGSW~	switch (latched)	
Diskette “DMA”	DRQ2	W83787F	*Int3* or masked
SCSI/PCI slot	PCIIRQ0	53C810	configure to any of *Int0-3.* *Int3* is usable only if the diskette “DMA” interrupt is disabled.
Ethernet/PCI slot	PCIIRQ1	DEC21041	
PCI slots	PCIIRQ2-3	PCI devices	
PCI bridge	V3IRQ~	V962PBC	
Keyboard	IRQ1	8742	
Serial ports com1/com2	IRQ4/3	W83787F	
Diskette	IRQ6	W83787F	
Centronics port	IRQ7	W83787F (latched)	
User GPIO port	IRQ5	Z0842004PS	
RTC tick	IRQ8	BQ3285E	

Table 10.2: Interrupt sources

Notes on Table 10.2, “Interrupt sources”

The interrupts are as follows:

- *R4x00 timer*: all R4x00 CPUs (to date) have contained a simple, flexible, programmable timer attached to the CPU *Int5* interface signal. Some CPUs gave you the option of using *Int5* for an external interrupt, but the Vr4300 does not even have the interface pin. So on P−4032 the timer is always enabled.
- *Powerfail*: generated by the power-monitoring circuit to give a few ms’ advance warning of loss of power.
- *Bus timeout*: occurs when you produce an address which doesn’t match anything local or a PCI “aperture”. The interrupt will usually happen after the offending load/store, sometimes a fairly long way after - this is mostly a diagnostic interrupt. There is no simple way of reconstructing the offending address after a timeout.

  This interrupt is latched inside the interrupt controller, and you need to write to the ICR register to make it go away.
- *Debug*: happens when you push the button. There’s no hardware “de-bounce” logic, so if you service the interrupt very fast it may appear to happen again. Latched, cleared with the ICR register.
- *Diskette DMA*: can be used to drive a low-level interrupt routine to transfer diskette data between the controller and memory, using a magic read/write data register. The interrupt is active during

reads/writes, when the controller wants to transfer data. See §9.7 for details.

- *PCI Ints*: the signals *PCIIRQ0-3* are connected to the onboard PCI devices (SCSI and ethernet) and to the expansion slots as shown in Table 10.3 (the same information is summarised in Table 6.1 above.) PCI interrupts may be shared; but if they are your interrupt handler will have to use some device-specific way of finding out which device is asserting a particular interrupt signal.
- *PCI bridge*: the PCI controller generates interrupts in various circumstances; see the manual [V962PBC] for details.
- *Keyboard*: from the PC-compatible keyboard controller.
- *Serial ports*: from the dual serial port in the Combi chip. Note that *IRQ4* comes from com1, and *IRQ3* from com2.
- *Diskette*: from the PC-compatible diskette controller in the Combi chip.
- *Centronics port*: from the centronics port. The Centronics interrupt signal is a pulse in some modes, so it is latched inside the interrupt controller and cleared with the ICR register.
- *User GPIO port*: from the general-purpose I/O controller. None of the onboard “port B” functions uses interrupts - the interrupt is available for events happening through the interface.
- *RTC tick*: delivered at a programmable rate by the real-time clock unit.

Device	*Interrupts*			
	INTA	*INTB*	*INTC*	*INTD*
Ethernet ctrlr	PCIIRQ1	-	-	-
PCI slot 1	PCIIRQ2	PCIIRQ3	PCIIRQ0	PCIIRQ1
PCI slot 2	PCIIRQ3	PCIIRQ0	PCIIRQ1	PCIIRQ2
SCSI ctrlr	PCIIRQ0	-	-	-

Table 10.3: Interrupt assignments for PCI devices/slots

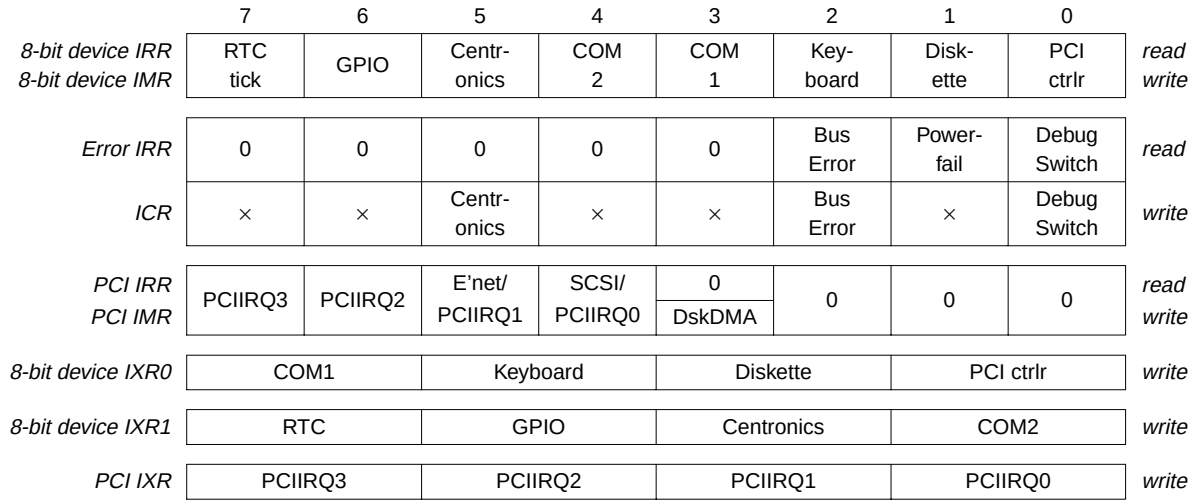

Figure 10.2 Interrupt register bit fields

Notes on the interrupt registers

All writable interrupt registers are undefined from power up or reset.

- *IRR*: a “1” indicates that the interrupt request is active.
- *IMR*: a “1” enables the request to cause an interrupt.
- *IXR*: each two-bit field encodes the number of the CPU interrupt *Intr0-3* which will be asserted when the interrupt is active and enabled.

  It’s common practice to have a number of different devices set to interrupt on any particular CPU input. However, when the diskette DMA interrupt is enabled, it takes over CPU input *Intr3* and no other interrupt will occur there.

11. P−4032 layout and user-selectable options

In P−4032, many of the functions which might usually be found on option jumpers are software selectable and programmed through two registers described respectively in §9.1. (“Board configuration register”) on page 32 and “DRAM Configuration register” on page 23.

But some options are still setup by jumpers, and they’re described here.

The board layout is shown in Figure 11.1.

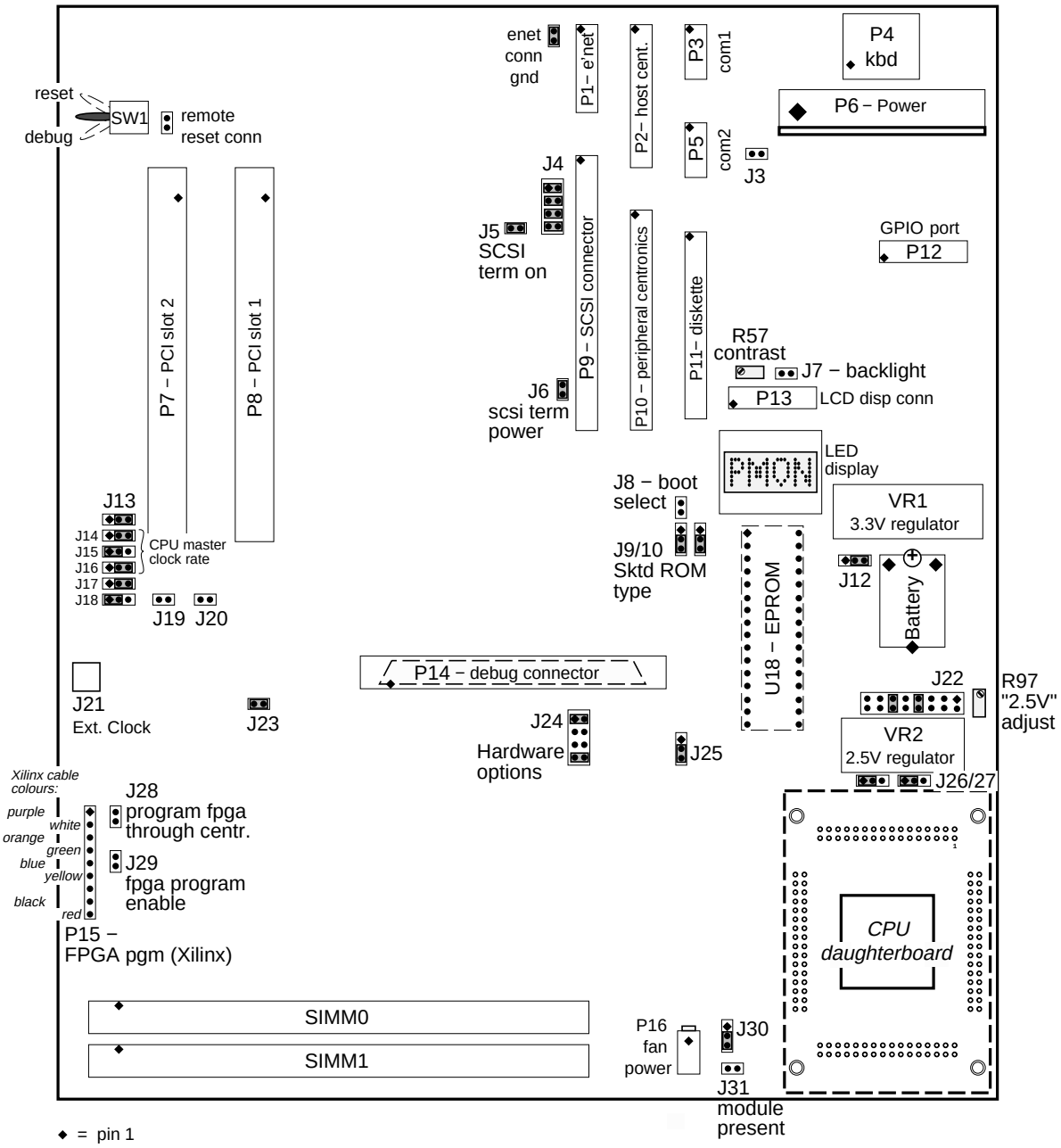

Figure 11.1 Board layout and jumper defaults

11.1. Notes on Figure 11.1

- *Pin1 on connectors*: is shown with a diamond.
- *2/3-way jumpers*: the default setup is shown.

11.2. CPU options - newer boards

If your board is marked “P−4032Q” and carries a serial number of 1000 and up, it is designed to accept any of a variety of CPU types on a personality module.

There is a position for one CPU type soldered directly to the motherboard - currently the RM5230. However, P−4032 can carry the CPU on a small daughterboard. Several daughterboards are available:

- *P4300MOD*: for the NEC Vr4300 CPU.
- *P4640MOD*: IDT’s RV4640 CPU.
- *P4100MOD*: NEC’s Vr4100 CPU (if still available).
- *P5230MOD*: QED’s RM5230 on a daughterboard.

It is possible to reconfigure a P−4032 to use a different CPU type. This will usually involve changing the system logic ROM, and a number of jumpers.

11.3. CPU options - older boards

Your board may be fitted with either a Vr4300 or R4640 CPU. CPUs are soldered down when the board is built, and the two types use slightly different system logic.

11.4. Jumper options

- *J22*: an 8-way jumper array readable by software (jumper in = “0”, out = “1”). It’s fully described in §9.2. (“Option register”) on page 33. Some functions are of significance to the hardware; all are readable by software.
- *J24*: a 3-link jumper, whose link layout is shown in Figure 11.2.

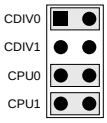

Figure 11.2 Hardware option jumper (J24) positions

The fields are used as follows:

Field name	*Link position*	*Effect*
CDIV0,1	out out	CPU runs at input clock × 3
	in out	CPU runs at input clock × 2 (default)
	out in	CPU runs at input clock × 1.5
	in in	CPU runs at input clock × 1
CPU0,1	out out	reserved
	in out	R4640 CPU
	out in	Vr4300 CPU
	in in	RM5230 CPU

The “CPU type” field will not always be meaningful; in particular the NEC Vr4300 uses a different version of system logic and the CPU type is then irrelevant.

- *J26, J27*: determine whether the CPU (whether onboard or on-module) runs entirely from 3.3V (jumpers set 1-2), or gets a lower core voltage supply (jumpers set 2-3). **Caution**: setting dual-voltage with a single-voltage CPU or module is likely to result in damage to the CPU, the board, or both; setting single-voltage with a dual-voltage CPU may damage the CPU. Don't change these lightly.
- *J8*: determines whether the system boots from the socketed ROM (jumper in) or the onboard flash ROM (jumper out).
- *J9, J10*: allow the board to use either a JEDEC 32-pin uV-erasable ROM (2-3, default) or a 29F040-compatible 32-pin DIL flash device (both 1-2).
- *J29*: Make it possible to reload the board's logic patterns into the reprogrammable logic devices. Required only when upgrading the logic, which you should not do without guidance from Algorithmics.
- *J28*: Insert to select the Centronics "peripheral" connector as the logic reprogramming source. To make this work you'll need to pull the jumpers J4.
- *J4*: remove only when you want to update the board's logic using the Centronics cable. Removing these jumpers isolates the Centronics lines used for logic reprogramming from the onboard Centronics controller.
- *J3*: jumper attached to the keyboard controller, conventionally read to establish whether a PC had a monochrome or color display. May disappear from future versions. Meanwhile, you can use it as an additional software-readable configuration bit.
- *J12*: a 3-pin link whose center pin feeds battery power to the real-time clock chip. Three options for the jumper:

  2-3 (normal operation) feed battery power to the RTC chip.

  1-2 connects the RTC battery input to ground, which has the effect of resetting it's registers and internal SRAM.

  out isolates the battery, saving battery life when the board is not being used for a long period of time.
- *J6*: remove this jumper to isolate the SCSI bus "terminator power" signal from the onboard supply. Used when another SCSI device is feeding terminator power; conventionally, the host provides the power and peripherals use it, so you will only need to do this when attaching another "host" to your SCSI bus.

CPU adaptation options

These are only available on later boards, distinguished by serial numbers from 1000 upwards. Users would not usually alter these settings, but since this board is designed for grown-ups here's what they do:

- *R97*: dual-voltage CPUs are increasingly common. Usually they require 2.5V supply for the CPU core functions (the I/O pins use 3.3V power). But some CPUs may use voltage levels higher or lower than the 2.5V nominal value, and you can adjust it here.

  The safest approach is to first move J30 to (1-2), which disconnects most of the CPU's I/Os. Then take the jumpers J26 and J27 out, isolating the core power. Monitor the voltage level on pin 1 of J26, and adjust to the desired value. With board main power off put J26 and J27 back in the (2-3) position, and replace J30.
- *J30*: allows you to power-down the "QuickSwitch" components which surround the CPU, isolating CPU inputs while the voltage is adjusted. J30(2-3) is normal, while connecting 1-2 isolates the CPU.
- *J17*: move this jumper from its default position (2-3) to (1-2) to supply the CPU master clock from an external frequency generator via the BNC input connector J21. The input is terminated with a 50Ω resistor.

- *J18, J25, J23*: adapt the CPU clocking scheme to cope with the Vr4100 CPU, which has special requirements. However, Vr4100 support is questionable in later boards; ask Algorithmics. If they work, you should set these jumpers as follows:

Jumper	*Vr4100 setting*	*All other CPUs*
J18	2-3	1-2
J25	1-2	2-3
J23	out	

  Vr4100 expects to generate the system interface clock, whereas all other CPUs are fed with an input clock which itself defines CPU interface transitions (there are multiple CPU interface clocks generated by a low-skew or PLL buffer). J18 feeds the raw oscillator output into the Vr4100 CPU, and J25 manages the Vr4100 reset sequence.

  The 2-pin jumper J23 selects between two possible clock buffer setups - “in” for the same low-skew buffer used in earlier P−4032 units, and “out” to use a smart PLL clock buffer. With Vr4100 the PLL buffer is mandatory; with other CPUs it’s optional.

  Lastly, Vr4100 CPUs run at 40MHz or less, so it makes sense to run the intermediate bus at the same speed as the CPU, not half speed as normal. You can do this by inserting jumper J19.

PCI clock source

The PCI clock is driven by the frequency synthesiser IC, and is fixed at 33MHz.

CPU system interface clock frequency

Jumper setting			*CPU Clock rate*
J16	*J15*	*J14*	
2-3	2-3	2-3	50 MHz
2-3	2-3	1-2	60 MHz
2-3	1-2	2-3	66.67 MHz
1-2	2-3	2-3	55 MHz
1-2	2-3	1-2	75 MHz
1-2	1-2	2-3	83 MHz
1-2	1-2	1-2	no clock

Table 11.1: CPU clock rate setup

Probably fairly self-explanatory. Most P−4032 CPUs will run at 67MHz.

Later boards have a miniature BNC connector which can be used to inject a different or variable clock frequency; you’ll need to alter the jumper J17 too.

12. CPU endianness

MIPS CPUs are conceived as being statically configured as either big-endian or little-endian. This has two effects:

- *Software*: it changes the way bytes are packed into multi-byte quantities (such as a 32-bit C unsigned int, or a 64-bit machine register). In the big-endian CPU low-address bytes are found in the most significant (high-numbered) bits; in a little-endian CPU the low-addressed bytes are found in the least significant (low-numbered) bits.

  This has no effect on the hardware, but big effects on software.

- *Hardware*: it changes the way data is transferred on the CPU system interface data lines. For a big-endian CPU, a byte whose address is 0 mod 4 is transferred on the *SysAD24-31* byte lane; for a little-endian CPU, a byte whose address is 0 mod 4 is transferred on *SysAD0-7.*

  Interface hardware has to know about this change, in order that it can identify the active byte lane(s) during partial-word transfers.

However, the MIPS switch-over is accomplished in such a way that *transfers of aligned 32-bit words are unaffected by endianness*. Since all *instructions* are aligned 32-bit words, that makes it possible to write chunks of code whose operation is unaffected by the CPU's endianness - you just have to avoid using any partial-word data. MIPS code which uses partial-word data is inherently endianness-specific; if you want to build a "bisexual" boot ROM, it will probably contain a hand-crafted bisexual start-up and two copies of everything else.

The Vr4300 CPU is the first ever MIPS device whose endianness is configured by a software-writable internal register bit. It starts up big-endian.

In P−4032 a configuration register bit (see §9.1 above) is used to adapt the CPU interface logic to either a big-endian or little-endian CPU. It also is reset to big-endian from power-up or system reset.

During periods when the two bits are set inconsistently (as is bound to happen transiently when putting the system into little-endian mode) partial-word transfers don't work at all - reading or writing garbage.

When you wire up a system with changeable endianness, you have two choices about how a memory or peripheral is attached:

- *Bit-order preserving*: in this case full-bus-width transfers (ie of 32-bit words) continue to work, and aligned 32-bit data is transferred in an endianness-independent way. But byte strings get mangled (the bytes in each word get swapped around); and misaligned data gets *really* messed up.
- *Byte-order preserving*: that makes strings work, but now all multi-byte quantities will need to be byte swapped between the CPU and the target.

In P−4032:

- The ROM, DRAM and local devices are wired to preserve bit-order. So if you want to write endianness-independent code which accesses local devices, always read/write words and pick the bits you need.
- All local 8-bit devices are always read/written over the low bits (D0-7) of the CPU data bus.
- The PCI bus should be configured to preserve byte-order (so ethernet and other DMA data ends up with the correct byte order). The V962PBC PCI controller has an inbuilt byte-swapper which should be enabled when the CPU is configured as big-endian.

  If you want to write endianness-independent code which accesses PCI devices, always think of it as read/writing a group of bytes.

  PCI is a little-endian bus, so if the CPU is being big-endian you'll need to byte swap multi-byte integer values (eg when programming a base address or count in the ethernet controller).

On P−4032, the 8-bit boot ROM and flash ROMs are designed to be read as 32-bit words - every time you read the ROM special ROM interface logic reads four bytes and assembles a 32-bit quantity for the CPU. The ROM byte assembly logic is not affected by the CPU endianness (otherwise you couldn't write the little bits of bisexual code which are essential to setting up a little-endian CPU). Instead, the contents of ROM address zero are always passed to the CPU on *SysAD0-7.* You'll need to remember that when setting up your PROM programmer, or downloading code into the flash ROM.

13. Connectors and cables

Most connector pin-outs are defined here. Where they're not, and you really need to know, ask Algorithmics for a set of schematics.

Although many connections are made with non-standard dual-in line headers, Algorithmics supply a set of short cables with each board which convert to recognised standard connectors.

13.1. Cables supplied

Every board comes with a pack of short transition cables, designed to interface to "standard" cables and devices. On flat cables, you can recognise pin 1 by the red line on the outer strand of the cable.

- *Ethernet*: connects from onboard header to standard 15-way D-type transceiver connection.
- *SCSI* : a 50-way ribbon cable to connect one or two external SCSI devices to P−4032.
- *Floppy*: the 34-wire flat cable is for a standard 3.5" diskette.
- *Centronics host*: converts from the onboard header to a 25-way D-type female connector.
- *2×RS232*: convert from the onboard header to a pair of D-type connectors, 9-pin male and 25-way male, as used on PCs.

  The serial and parallel cables go through back panels.

Not included in the standard pack, but an optional extra:

- *Centronics peripheral* : convert from the onboard "reverse host" connector to the Centronics connector used on most PC-world printers.

13.2. CPU daughterboard connector

The CPU daughterboard connector is made up of 2mm dual socket strip/headers, in four banks each of 2×16 pins, as shown in Figure 13.1. The signals on the connector are shown in Table 13.1. You probably won't have to know this very often.

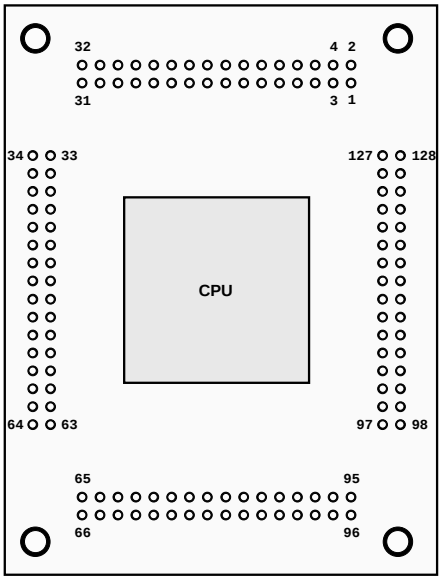

Figure 13.1 CPU daughterboard layout

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	*CDIV1*	33	*ModeIn*	65	*NMI**	97	*NC*
2	*CDIV0*	34	*RdRdy**	66	*EReq**	98	*NC*
3	*VDD*	35	*EOK*/WrRdy**	67	*Reset**	99	*NC*
4	*GND*	36	*EValid**	68	*ColdReset**	100	*NC*
5	*SysAD4*	37	*PValid**	69	*VccOK*	101	*VDD*
6	*SysAD5*	38	*PMaster*/Release**	70	*BigEndian*	102	*GND*
7	*VCORE*	39	*VCCQ*	71	*VDD*	103	*SysAD28*
8	*GND*	40	*VSSQ*	72	*GND*	104	*SysAD29*
9	*SysAD6*	41	*ClkIN*	73	*SysAD16*	105	*VCORE*
10	*SysAD7*	42	*VCORE*	74	*VCORE*	106	*GND*
11	*SysAD8*	43	*GND*	75	*GND*	107	*SysAD30*
12	*SysAD9*	44	*SysCmd0*	76	*SysAD17*	108	*SysAD31*
13	*VDD*	45	*SysCmd1*	77	*SysAD18*	109	*SysAD34*
14	*GND*	46	*SysCmd2*	78	*SysAD19*	110	*VCORE*
15	*SysAD10*	47	*SysCmd3*	79	*VCORE*	111	*GND*
16	*SysAD11*	48	*VDD*	80	*GND*	112	*SysAD35*
17	*VCORE*	49	*GND*	81	*SysAD20*	113	*VDD*
18	*GND*	50	*SysCmd4*	82	*SysAD21*	114	*GND*
19	*SysAD12*	51	*SysCmd5*	83	*VDD*	115	*SysAD32*
20	*SysAD13*	52	*GND*	84	*GND*	116	*SysAD33*
21	*SysAD14*	53	*SysCmd6*	85	*SysAD22*	117	*SysAD0*
22	*VCORE*	54	*SysCmd7*	86	*SysAD23*	118	*SysAD1*
23	*GND*	55	*SysCmd8*	87	*SysAD24*	119	*VCORE*
24	*SysAD15*	56	*SysCmdP*	88	*SysAD25*	120	*GND*
25	*VDD*	57	*VCORE*	89	*VCORE*	121	*SysAD2*
26	*GND*	58	*GND*	90	*GND*	122	*SysAD3*
27	*ModeClk*	59	*Int0**	91	*SysAD26*	123	*VDD*
28	*QJTDO*	60	*Int1**	92	*SysAD27*	124	*GND*
29	*QJTDI*	61	*Int2**	93	*VDD*	125	*PReq**
30	*QJTCK*	62	*Int3**	94	*GND*	126	*TClk*
31	*QJTMS*	63	*Int4**	95	*ModPres**	127	*NC*
32	*VDD*	64	*Int5**	96	*NC*	128	*MClkOut*

Table 13.1: Pinout of CPU daughterboard (MIPS names)

13.3. SIMM memory slots (SIMM0/SIMM1)

Take standard SIMM devices with up to 12 multiplexed addresses and 4 *Ras** signals - which would be 128Mbytes per SIMM. They don't come that big yet.

Read §5.1. ("DRAM configuration") on page 22 to find out how to get P−4032 to work with different types of module.

13.4. PCI edge connectors (P8/P7)

These are two standard PCI Rev 2 high density edge connector sockets: 5V, 32-bit, 33MHz. The pinout is not documented here - see [PCI Standard] for signal description, and section 6.2.1 of this manual for P−4032's PCI setup.

13.5. Ethernet (P1)

A 7×2 0.1" pin grid, laid out to allow a simple ribbon cable to an IDC 15-way D-type socket, implementing the ethernet-standard transceiver cable connector - a conversion cable is included in the standard set. The onboard connection pinout is in Table 13.2.

OPTGND	1	2	*COLPRES˜*
COLPRES	3	4	*TRANSMIT˜*
TRANSMIT	5	6	*OPTGND*
OPTGND	7	8	*RECEIVE˜*
RECEIVE	9	10	*E12V*
GND	11	12	*OPTGND*
OPTGND	13	14	-

Table 13.2: Ethernet connector (P1) pinout

where:

- *OPTGND* signals should be grounded at only one end of the transceiver interface (normally here); but if so required they can be disconnected from board ground by removing jumper J1 (called "enet conn gnd" on the board layout diagram Figure 11.1.
- The *E12V* signal is protected by a self-resetting fuse.
- The active ethernet interface signals are transformer-coupled to the controller to reduce the risk of damage to the board through misconnection or extreme electrical noise.

13.6. SCSI (P9)

A 25×2 0.1" pin grid, laid out to allow a ribbon cable to common peripherals; we include an unshielded cable which will connect two disk/tape units within an enclosure or on a bench.

This should plug right in to your SCSI device, but the pinout is shown in Table 13.3.

GND	1	2	*DB0˜*
GND	3	4	*DB1˜*
GND	5	6	*DB2˜*
GND	7	8	*DB3˜*
GND	9	10	*DB4˜*
GND	11	12	*DB5˜*
GND	13	14	*DB6˜*
GND	15	16	*DB7˜*
GND	17	18	*PAR˜*
GND	19	20	*GND*
GND	21	22	*GND*
GND	23	24	*GND*
-	25	26	*TermPower*
GND	27	28	*GND*
GND	29	30	*GND*
GND	31	32	*ATN˜*
GND	33	34	*GND*
GND	35	36	*BSY˜*
GND	37	38	*ACK˜*
GND	39	40	*RST˜*
GND	41	42	*MSG˜*
GND	43	44	*SEL˜*
GND	45	46	*C_D˜*
GND	47	48	*REQ˜*
GND	49	50	*I_O˜*

Table 13.3: SCSI connector (P9) pinout

Note that the *TermPower* signal (currently called *STPWRC* on the schematics) is a 5V supply protected by a fuse and via the jumper J6. There is normally one “host” on a SCSI bus which supplies terminator power, but the jumper could be removed so long as some other SCSI bus device is driving the power line - which is likely to be the case when P−4032 is acting as a SCSI peripheral.

P−4032 is fitted with an active SCSI terminator. The terminator should be disabled if P−4032 is connected to a mid-point of your SCSI cable, by removing jumper J5.

13.7. RS232 (P3/P5)

These are 5×2 0.1” pin headers, and a ribbon cable connector to a 9-pin male D-type implements the PC 9-pin serial connector (the nearest thing to a standard which RS232 has ever seen.) The pinout of each IDC connector is shown in Table 13.4.

DCD	1	2	*DSR*
RXD	3	4	*RTS*
TXD	5	6	*CTS*
DTR	7	8	*RI*
GND	9	10	-

Table 13.4: Serial connector (P3/P5) pinout

13.8. Centronics (P2/P10)

There are two connectors provided, with almost identical signal sets.

nStrobe	1	2	*nAuto*
D0	3	4	*nERROR*
D1	5	6	*nInit*
D2	7	8	*nSelectIn*
D3	9	10	*GND*
D4	11	12	*GND*
D5	13	14	*GND*
D6	15	16	*GND*
D7	17	18	*GND*
nAck	19	20	*GND*
Busy	21	22	*GND*
PAPEMT	23	24	*GND*
nSelect	25	26	-

Table 13.5: Centronics host connector (P2) pinout

nAck/`nStrobe`	1	2	*GND*
D0	3	4	*GND*
D1	5	6	*GND*
D2	7	8	*GND*
D3	9	10	*GND*
D4	11	12	*GND*
D5	13	14	*GND*
D6	15	16	*GND*
D7	17	18	*GND*
nStrobe/`nAck`	19	20	*GND*
nAuto/`Busy`	21	22	*GND*
nInit/`PError`	23	24	*GND*
nSelectIn/`nSelect`	25	26	*PAPEMT/*`nInit`
Busy/`nAuto`	27	28	*PIO(B0)/*`nFault`
-	29	30	-
-	31	32	-
-	33	34	-
-	35	36	*nSelect/*`nSelectIn`

Table 13.6: Centronics peripheral connector (P10) pinout

- *Host connector (P2)*: pinned out for a ribbon cable to the 25-way male D-type traditionally for the Centronics connector on PCs; shown in Table 13.5.
- *Peripheral connector (P10)*: a 20×2 header, pinned out for a ribbon cable to the big 38-way shrouded male connector traditionally used on parallel printers. It's shown as Table 13.6.

  In general, as related in §9.6 above, the standard control signals in peripheral mode are connected to the complementary signal on the Winbond IO device. Where this happens the "standard" name is shown in Table 13.6 in `courier` typeface.

Signal names are currently an uneasy mixture of IEEE1284 and local schematic versions.

The centronics controller does not drive any output which can reasonably be used to implement the peripheral's *nFault* signal, so this is provided by the port B, data line 0, signal from the general-purpose parallel IO port described in §9.9. (“General-purpose parallel I/O (PIO)”) on page 36.

13.9. Diskette (P11)

a 17×2 header matching the connector found on every PC. This is so standard that we won't bother to print the signals.

13.10. User-defined parallel I/O (P12)

An 8×2 header making “port A” of the parallel I/O controller available for user functions. These include:

- Allowing a user's logic to generate an interrupt;
- Polling an external logic level.
- Driving some simple external device.
- Software-controlled trigger for test equipment...

Anything you like. The pinout is in Table 13.7.

ASTB˜	1	2	*BSTB˜*
ARDY	3	4	*BRDY*
PA0	5	6	*PA4*
PA1	7	8	*PA5*
PA2	9	10	*PA6*
PA3	11	12	*PA7*
GND	13	14	*VCC*
GND	15	16	*VCC*

Table 13.7: General purpose I/O connector (P12) pinout

13.11. PC-compatible keyboard (P4)

Plug your PC/AT-compatible keyboard into this 5-pin DIN socket, and it should work. P−4032 does not support PS/2 keyboards, or any kind of keyboard-compatible mouse. The world is, fortunately, overrun with RS232-interfaced mice. We won't bother with the pinout.

13.12. LCD display connector (P13)

This simple I/O port on an 8×2 header is designed to attach an LCD alphanumeric display [LCD-Ouwehand] on a short ribbon cable. You may even be able to use it for something else, with ingenuity. It's pinout is in Table 13.8.

GND	1	2	*VCC*
BRIGHT	3	4	*XA2*
LIOR_W˜	5	6	*Enable*
XD0	7	8	*XD1*
XD2	9	10	*XD3*
XD4	11	12	*XD5*
XD6	13	14	*XD7*

Table 13.8: LCD display header (P13) pinout

Note that in Table 13.8:

- Connections to the 8-bit IO data bus *XD0-7* are made via 22Ω resistors, to provide some protection and noise suppression.
- The brightness of the LCD illumination is controlled by *BRIGHT,* which can be varied between 0 and 5V by adjusting R57.
- Signal *LIOR_W~* is a direction signal (low for write); and *Enable* is a kind of combined chip select and transfer strobe. *XA2* is the least-significant register address (8-bit IO registers on P−4032 are always at least 4 bytes apart), supporting a princely two registers.

13.13. Power supply connector (P6)

A PC-compatible power connector. PC supply mating connectors are supposed to be encoded to make it impossible to plug in the wrong way round, but impatient people often seem to cut the encoding lugs off. Place the black wires together to be safe.

The pinout is shown in Table 13.9, if you're unlucky enough to have to make your own. On second thoughts, don't; you can buy a perfectly workable PC-clone power supply for $40 or less, so it can't be worth the time you'll spend making up the connector!

Pin	*Name*
1	Power on reset
2	NC
3	+12 Volts
4	-12 Volts
5	0 Volts (GND)
6	0 Volts (GND)
7	0 Volts (GND)
8	0 Volts (GND)
9	-5 Volts (Unused)
10	+5 Volts (VCC)
11	+5 Volts (VCC)
12	+5 Volts (VCC)

Table 13.9: Power connector pinout

13.14. 12V fan power (P16)

A socket providing a fused +12V supply (and ground) to connect a PC-type cooling fan. Pin 1 is *+12V* and pin 2 is *GND.* Most CPUs fitted to P−4032 do not need a fan when run open at room temperature with no forced air movement; it will not be necessary to fit a fan to the CPU when the CPU is running at 133MHz or below, unless airflow is very restricted. Use any fan designed for "Pentium" class CPUs in PC clones.

13.15. Logic programming connector (P15)

P−4032's logic is mostly implemented in a number of Xilinx 9500 series programmable logic devices. These chips retain their logic programs using "flash" ROM storage, but can be reprogrammed in-circuit.

Reprogramming is possible either with a Xilinx download cable, or via a Centronics cable from a PC running appropriate software. The jumper J29 must be fitted. To download from the Centronics connector you should also fit jumper J28 and remove the jumpers J4.

The Xilinx-compatible connector is P15, and is shown in Figure 13.2. It matches Xilinx' supplied module, so the signals connect across one to one.

JTMS	1	purple
	2	white
JTDI	3	orange
JTDO	4	green
	5	blue
JTCLK	6	yellow
	7	
GND	8	black
+5V	9	red

Figure 13.2 Xilinx-compatible connector (P15) for reprogramming P−4032 logic

14. Logic analyser (debug) board

P−4032 is equipped with a connector P14 and an optional additional board DBG−4 which allows you to hook up a logic analyser to P−4032's main address and data bus with minimal interference with normal function of the logic. DBG−4 has several functions:

- Registers address/data so your test equipment doesn't need to be able to keep up with the 67MHz clock rate, nor to capture signals with very short setup and hold times.
- Isolates P−4032's high-speed buses from any detrimental effect which test probe cabling might have on operation.
- Provides convenient headers for attaching logic probes; it connects directly to HP logic analyser pod adaptors.
- DBG−4's PAL device (a dual in-line GAL26CV12-7) generates strobes to capture address and data, and to trigger your test equipment. But you can also reprogram the device with custom triggers; a very powerful tool for catching fast and obscure events.

With the standard PAL the trigger signal *ATRIG* fires once for every cycle, with timing which will capture both the address and one word of data - by default, the last of a block. To see a different word of data inside a block, set the option header J4 as described below.

The debug board's connectors are shown in Figure 14.1.

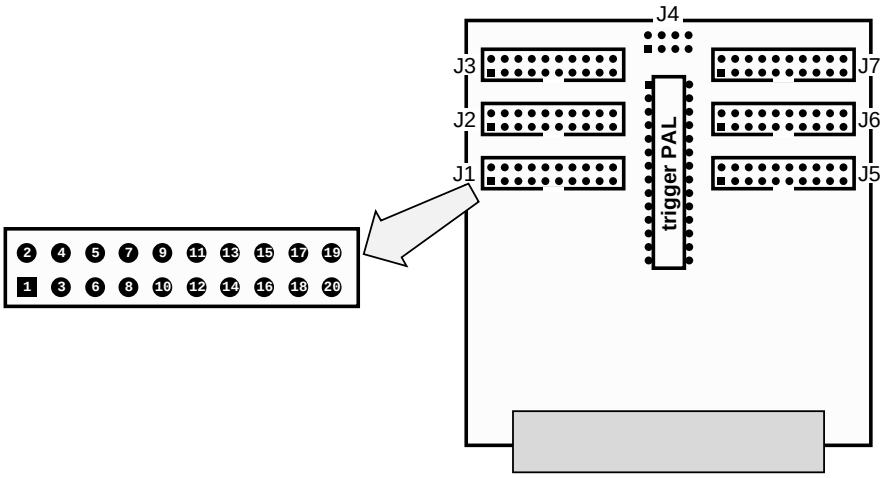

Figure 14.1 DBG−4 - P−4032's debug board

The signals available on DBG−4 are summarised in Table 14.1.

Signal	*Description*
A2-31 BE0-3˜	Address and byte enables for any transaction. Note that *BE0˜* active (low) indicates that *D0-7* will carry valid data in this transaction.
	These signals are caught on an edge-triggered register and held through the cycle by the signal *DBG_AHLD*, generated by the debug board control PAL ANTRIG.
D0-31	data, captured by an edge-triggered register and held by *DBG_DHLD*.
ATRIG	Rising-edge trigger timed to sample address, cycle qualifiers, and data into your analyser.
W_R˜	Raw direction signal (low to read) from intermediate bus.
HLDA	When 1, indicates that this is not a CPU cycle, so it must be a PCI-initiated cycle. This signal is not latched, so potentially runs a bit ahead of the address/data; but it's valid at the *ATRIG* rising edge.
BLOCK	Active during a multi-word (block or burst) transfer.
BLADS˜ BLRDY˜ BLW_R˜ BLBLAST˜ BLBTERM˜ BHLDA	i960-like control signals on the intermediate bus, delayed by one *LCLK* register stage (*LCLK* is the half-rate clock which runs the local bus side of the V962PBC PCI controller).
BPVALID˜ BEVALID˜ BEOK˜ BSYSCMD0-4 PREQ˜ BPMASTER˜	MIPS CPU signals (Vr4300 names) delayed by one CPU bus interface clock register stage

Table 14.1: Signals from DBG−4

The connectors J1-3, J5-7 are compatible with HP logic analyser multi-way pod headers; with an HP or compatible analyser you can connect it as shown in Figure 14.2.

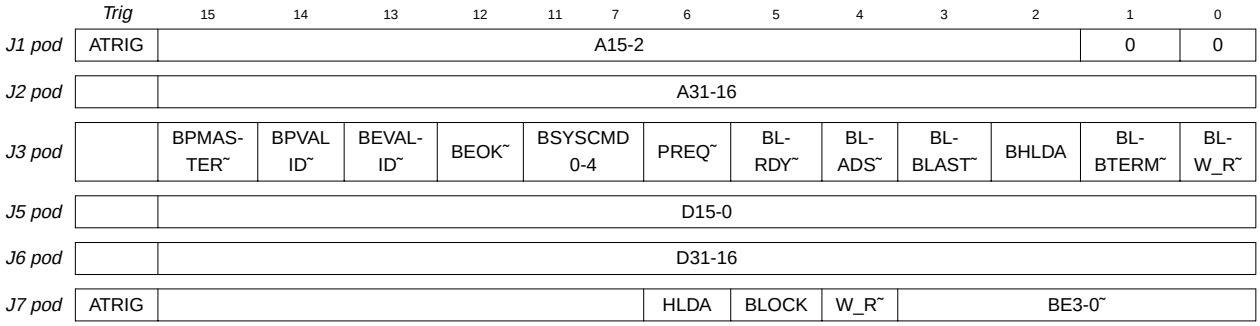

Figure 14.2 Connecting an HP or compatible analyser to DBG−4

You can of course connect up any logic analyser pin-by-pin. The signals are available as shown in Figure 14.3.

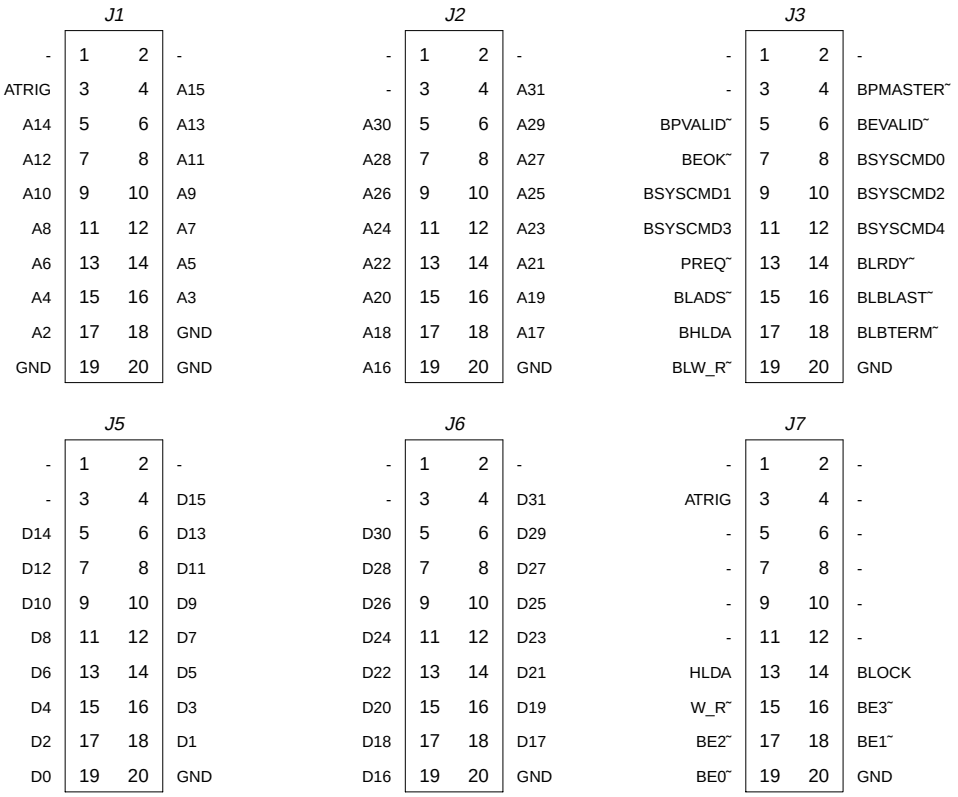

Figure 14.3 Pin-by-pin analyser connection to DBG−4

14.1. DBG−4 option header (J4)

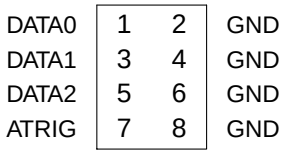

Figure 14.4 The J4 option header - pins

Where:

- *DATA0-2* encode the initial binary value of a down counter (one per data transfer) which picks the data word to be captured in a burst read. The signals *DATA0-2* are pulled up onboard and grounded by the link, so each bit is set “1” for no link, and “0” for link in. By default (no links in) the initial value is 7 and the *last* word of a block is always captured.

  If you elect to reprogram the PAL ANTRIG you can use these lines for any purpose. You could even use them to carry external signals into the PAL - the pull-ups are 4.7KΩ and won’t cause any trouble if you don’t want them.

- *ATRIG* is provided again here, in case you want to connect up more than one piece of test equipment.

J4 may be pressed into more mundane use as a supply of extra ground pins.

Reprogramming the ANTRIG PAL

Start from Algorithmics' standard PAL, available on request to any P−4032 customer. All the signals presented to the PAL are described below in Table 14.2. Your job is to generate the trigger signal *ATRIG* and the latch-closing signals *DBG_AHLD* and *DBG_DHLD*.

14.2. Pinout of debug connector (P14)

For those of you who might want to do your own debug board, Table 14.2 describes the signals available and Table 14.3 describes where the signals are available on this 96-pin "SBus"-type straight female connector. The right-angled mating connector used by Algorithmics to build DBG−4 is available as Honda part number "PCS-96LMD".

Signal	*Description*
DBGCLK CPUCLK3	Two CPU bus interface clocks: *DBGCLK* is for the use of the debug board; *CPUCLK3* is a spare, but is already used by four onboard registers.
PVALID˜ EVALID˜ EOK˜ PMASTER˜ PREQ˜ SYSCMD0-4	MIPS CPU control signals, see CPU manual for meanings. Note that while these signals come straight from the CPU (apart from voltage conversion), the address and data lines are two register stages separated from the CPU (more for write data).
LCMD2	The Vr4300 CPU signal *SYSCMD2* is low for single- or partial-word transfers, and high for bursts. *LCMD2* is latched with the address bus *LADDR2-31* with the same meaning.
LCLK	The clock for the local bus side of the V962PBC PCI bus controller; the basic clock for i960-like transactions on the local bus
LADDR2-31	Addresses
MD0-31	Data - endianness as for the CPU
LBE0-3˜	Byte enables: *LBE0˜* indicates that the byte lane *MD0-7* is being used for data in this transaction
LADS˜	synchronous "address strobe" indicating the start of a cycle
LRDY˜	activated to indicate that there's valid data on (some of) *MD0-31*.
LBTERM˜	The target's signal for terminating a burst transfer. The transfer completes on this or the next clock edge where *LRDY˜* is active.
LBLAST˜	Initiator's signal for terminating a block transfer, meaning that the next data transfer will be the last.
LW_R˜	Direction (low for read)
HLDA	high when this is a PCI-initiated cycle
LRESET˜	active-low reset
SLVGO˜ SLVDVAL˜ SLVLAST˜ SLVREAD˜	signals from the bus control state machine

Table 14.2: Debug connector signal description

1	*GND*	2	*DBGCLK*	49	*VCC*	50	*LADDR8*
3	*LCMD2*	4	*LADDR2*	51	*LADDR9*	52	*LADDR10*
5	*LADDR3*	6	*LADDR4*	53	*LADDR11*	54	*LADDR12*
7	*LADDR5*	8	*LADDR6*	55	*LADDR13*	56	*LADDR14*
9	*LADDR7*	10	*LBE0˜*	57	*LADDR15*	58	*LBE1˜*
11	*LADDR16*	12	*LADDR17*	59	*LADDR24*	60	*LADDR25*
13	*LADDR18*	14	*LADDR19*	61	*LADDR26*	62	*LADDR27*
15	*LADDR20*	16	*LADDR21*	63	*LADDR28*	64	*LADDR29*
17	*LADDR22*	18	*LADDR23*	65	*LADDR30*	66	*LADDR31*
19	*LBE2˜*	20	*LADS˜*	67	*LBE3˜*	68	*LCLK*
21	*LW_R˜*	22	*LRDY˜*	69	*HLDA*	70	*LRESET˜*
23	*LBTERM˜*	24	*GND*	71	*LBLAST˜*	72	*EOK˜*
25	*PVALID˜*	26	*PREQ˜*	73	*EVALID˜*	74	*PMASTER˜*
27	*SYSCMD0*	28	*SYSCMD1*	75	*SYSCMD2*	76	*SYSCMD3*
29	*CPUCLK3*	30	*SLVLAST˜*	77	*SYSCMD4*	78	*SLVREAD˜*
31	*SLVGO˜*	32	*MD0*	79	*SLVDVAL˜*	80	*MD8*
33	*MD1*	34	*MD2*	81	*MD9*	82	*MD10*
35	*MD3*	36	*MD4*	83	*MD11*	84	*MD12*
37	*MD5*	38	*MD6*	85	*MD13*	86	*MD14*
39	*MD7*	40	*MD16*	87	*MD15*	88	*MD24*
41	*MD17*	42	*MD18*	89	*MD25*	90	*MD26*
43	*MD19*	44	*MD20*	91	*MD27*	92	*MD28*
45	*MD21*	46	*MD22*	93	*MD29*	94	*MD30*
47	*MD23*	48	*GND*	95	*MD31*	96	*VCC*

Table 14.3: Debug connector (P14) pinout

Appendix A: References and further reading

Most of all, we encourage you to visit our web site at `www.algor.co.uk`; look at the “Documents and software to download” section.

General MIPS information

- *MIPS architecture and programming*: Dominic Sweetman, *See MIPS Run* published by Morgan Kaufmann, ISBN 1-55860-410-3.
- *Using MIPS*: Erin Farquhar and Philip Bunce, *The MIPS Programmer’s Handbook* published by Morgan Kaufmann, ISBN 1-55860-297-6.
- *MIPS R4000*: Joe Heinrich/Gerry Kane, *MIPS R4000 Microprocessor User’s Manual*, published Prentice Hall, ISBN 0-13-1059254.

CPU variants

- *NEC Vr4300*: datasheets available from NEC offices worldwide.
- *IDT R4640*: from IDT distributors. However, you can get IDT manuals over internet (`www.idt.com`) in Adobe “.pdf” format; you can read them online and print them using Adobe’s “Acrobat” software (free for PC and other common platforms). If you don’t have a common platform, then recent versions of the free “ghostscript” suite also reads PDF files, and are available from Aladdin Software.
- *QED RM5230*: available from QED (`www.qedinc.com`). We believe online manuals will be available at some point.

Algorithmics’ manuals

- *PMON*: *PMON: Users Manual*, Algorithmics Ltd/LSI Logic Corporation, 1994.
- *P−4032 schematics*: *P−4032: Schematics*, Algorithmics Ltd 1996.
- *P−4032 pals*: *P−4032: PAL (programmable logic) listings*, Algorithmics Ltd 1996.

For more information about Algorithmics, and updated information including a P−4032 buglist and the latest version of this manual in postscript form, start at Algorithmics’ WWW home page: `www.algor.co.uk`.

Other software

- *VxWorks*: there’s lots of official documentation, but you could start at `www.wrs.com`.

Data sheets

- *Atmel93C66*: describes the EEPROM. Available in Atmel’s “Non-volatile Memory data book”, 1995/96 edition.
- *LCD-Ouwehand*: If you want to program the LCD display, read Peer Ouwehand’s WWW page currently at `http://www.iaehv.nl/users/pouweha/lcd.htm`. If it’s moved, keywords “Ouwehand” and “HD44780” should get you something.
- *FlashData*: AMD parts are 29F040 (in the socket) or 29F080 (possibly fitted on the motherboard). Look at the reference page `http://www.amd.com/products/nvd/techdocs/techdocs.html`.

  The onboard flash memory is often a Fujitsu MBM29F080, datasheet listed on the products page: `http://www.fujitsumicro.com/products/memory/flash.html`.
- *V962PBC*: there’s a “User’s Manual” from V3 Corporation of Toronto, Canada. The manual we used is marked “Revision 1.0 of June 1995”. They’re on the web at `www.vcubed.com`.

- *DEC21041*: documentation on the DECchip 21041 PCI Ethernet LAN controller. This includes:

  Hardware Reference Manual, April 1995 (preliminary), DEC Order Number EC-QAWXA-TE. Oddly enough, this is where to look for programming information.

  Product Brief, DEC Order number EC-QAWVA-TE.

  Data Sheet, DEC Order number EC-QAWWA-TE.

  Web: Digital Semiconductor's Documentation Library is at:

  ```
  http://ftp.digital.com/pub/Digital/info/semiconductor/literature/dsc-library.html
  ```

- *Symbios53C810*: Symbios Logic produce the 53C810 chip, originally developed by NCR. You can find them online at `www.symbios.com`, and they have a reasonably useful range of sample drivers.
- *Centronics ECP*: *Extended Capabilities Port Protocol and ISA interface standard*, Microsoft/HP. We used revision 1.14 dated July 14th, 1993.
- *Winbond*: a PC-world manufacturer of our W83877F combi I/O chip (W83777F on early boards). Online at `www.winbond.com.tw`. We can supply an "acrobat" format data sheet if you can't get one.

Standards

- *PCI Local Bus Specification, Revision*: from the PCI Special Interest Group, PO Box 14070, Portland, Oregon 97214 USA.

Appendix B: Software support

SDE−MIPS for P−4032

Algorithmics' SDE−MIPS cross-development toolkit provides an easy way to get started programming the P−4032. SDE−MIPS is a comprehensive toolkit based on the GNU C compiler, running on most favourite hosts. But it also contains specific support code for the P−4032; there is enough in the standard SDE release to allow you to build either a PROM or downloadable "hello world" application for the P−4032.

For technical and sales information about SDE−MIPS contact Algorithmics on sales@algor.co.uk (internet email), +44 171 700 3301 (voice), or +44 171 700 3400 (fax).

SDE's P−4032 support files are as follows:

```
include/p4032
kit/...
kit/P4032/sbdreset.sx   assembler board support functions
kit/P4032/sbd.c         C board support functions
...
```

Real-time OS on P−4032

- *VxWorks/Tornado*: runs on the board on Vr4300 and RM5320 CPUs. BSPs ("board support packages") are available from Algorithmics. Unsupported sources are free to VxWorks licensees.
- *pSOS*: in development at the time of writing. Contact Algorithmics for details.

Other OS on P−4032

- *OpenBSD*: (a close relative of NetBSD, FreeBSD etc) is a freely-redistributable unix system which runs on P−4032 on Vr4300 and RM5320 CPUs. Contact Algorithmics to get sources.
- *Linux*: for P−4032 is under development.