Algorithmics P−5064
User´s Manual

Revision: 1.10
Dated: 1999/06/17

P−5064 is a single board computer which is mainly aimed at embedded systems developers wanting to build system prototypes and early-development platforms for applications using the 64-bit MIPS R5000 family of CPUs. P−5064 features a synchronous DRAM local memory system running at the CPU clock rate, and a PCI I/O system for easy expansion. 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 §1.2, "Key facts for the impatient" and §2, "Getting Started".

Algorithmics Ltd
3 Drayton Park
London N5 1NU
ENGLAND.

Phone: +44 171 700 3301
Fax: +44 171 700 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−5064 and manual

We made P−5064 because embedded systems developers need a high-performance target for big evaluation tasks and to support software development in parallel with embedded hardware design. We believe that many embedded systems in design in 1998/99 will move to system interfaces running at 75-100MHz, synchronous DRAM local memory and a PCI-based I/O system; and no other product allows you to explore the characteristics of such a system.

This manual describes boards built to revision C and higher, featuring 10/100Mbit ethernet. If you have previously worked with a revision B board, refer to “Software-visible changes between revision B and revision C boards” (Appendix D) on page 83.

1.1. The R5xx0 CPU family

How MIPS emerged from an academic project to lead the RISC charge is too long a story to tell here. However, the 64-bit R4000 CPU, introduced in 1990 for high-end workstations, was reworked by a design group called QED to make IDT’s simpler, lower-power and eminently embeddable R4600[1]. The R4600 turned out to be an ideal motor for Silicon Graphic’s low-cost “Indy” workstation, generating a need for a successor product. That became the R5000.

The R5000’s “embedded” ancestry and desktop leading application gives it a somewhat confused identity, but an R5000-200 is a potent high-end embedded CPU. In early 1997 QED (now a full-fledged microprocessor company) introduced the RM5260, first of a range of R5000-derivative CPUs delivering the power but at lower cost and power consumption. The QED family is growing, and is likely to be joined by R5000-derivative products by other MIPS vendors in 1998.

Read on to find why P−5064 is the right prototyping platform for all of them.

1.2. 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 3.1 and Figure 3.2 on page 18.
- *Memory map*: refer to Table 4.1 to find out what registers are where.
- *Physical arrangement, location of connectors and jumpers*: described in §6 on page 47 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*: §5.4.1 on page 30.
  - The *DRAM configuration register*: §5.4.2 on page 31.
  - The interrupt controller:§5.5 on page §33.
- *MIPS CPUs and their addresses*: can be very confusing for the uninitiated. If you’re not familiar with MIPS, do read Appendix A.

[1] That wasn’t *why* they made it of course; this useful part was specified and funded under the influence of a Microsoft-induced dream of MIPS-powered personal computers running Windows/NT.

1.3. Manual Sections

- *Getting started*: what we've supplied, how to switch the board on, and set-up stuff.
- *Overview and Block diagram* : a look at how the board works, at the kind of level of detail a programmer might need.
- *Memory map*: where to find memory regions and registers.
- *Programming P−5064*: gory details of registers and how devices are wired. For detailed programming information you're normally referred to manufacturer's data sheets.
- *Board layout: locating connectors and jumpers*: the physical picture.
- *Jumpers: where and what for*: just that.
- *Connectors: where, what and wiring*: we include pin-outs for all but the most familiar connectors.
- *Cables supplied*: what's in the box, and what can be bought as extras.
- *Debug and trace facilities*: about the optional extra debug board.
- *Software from Algorithmics and third parties*: a running list - our web site might be more up to date.
- *Board and logic revisions*: P−5064 is an evolving design, as we keep up with new CPU introductions and features, fix bugs and make customer-related improvements. Moreover, the soft-loadable FPGA logic can evolve separately from (and usually faster than) the logic built onto the board.

  The board revision is a letter; revision "A" was an unshipped prototype, so production boards begin with "B". It's to be found inscribed into the copper of the board, but is also encoded in a 4-bit software-readable field through the "board revision" field of the software options register. The logic revision is a number, returned by the *logic revision level register*. Both are described in section 5.4.3, which starts on page 32.

And then there's some slightly more obscure information relegated to the appendices:

- *Appendix A - MIPS CPUs, program addresses, and physical addresses*: how MIPS CPUs access external memory and I/O. You really should read this unless you are already familiar with MIPS CPUs.
- *Appendix B - Finding more information*: references to books and web sites.
- *Appendix C - Reading configuration information from DIMM modules*: the DIMM memory modules we use are equipped with a serial-access ROM full of information about the DIMM and its components. This information is supposed to be a de-facto industry standard, and this appendix tells you how to read it.
- *Appendix D - Software-visible changes between revision B and revision C boards*: we've introduced significant new functionality while slightly reducing the parts count. Here are the consequences, laid out for the programmer who's already got something working on one of the earlier boards.

1.4. What and why

Here's the basic features of the product, and why it's like that.

- *Clock rates*: embedded systems designers want CPU power on a par with desktop PCs. In 1998 that can't be done without high clock rates in the CPU interface and memory controller. In 1998, a system which can't run at 75MHz is a toy and anything less than 100MHz means aiming for second place. So P−5064 is designed to grow to 100MHz.
- *CPU choice*: P−5064 is intended to support a range of 64-bit MIPS CPUs.

CPU type/ Manufacturer	*Clock ext/int*	*Onchip cache (I+D+secondary)*	*External cache*
MIPS R5000 (NEC, IDT, NKK)	100/200	32K+32K	to 2M
QED RM5260	67/133	16K+16K	no
QED RM5270	83/166	16K+16K	to 8M
QED RM7000	100/250	16K+16K+256K	to 8M
NEC Vr5464	100/200	32K+32K	no
MIPS R10000/ R12000 (NEC)	83/250	32K+32K	to 8M

  The most popular CPU options will be available as motherboard options; less popular CPUs, later CPUs, or those with secondary cache fitted, will be provided on small daughterboards plugged into the connector which is fitted to all boards. An onboard CPU (if fitted) can be disabled when a module is attached.

  The R10000/R12000 module will be made only if we discover some demand for this beast in embedded applications; we're really hoping someone will take us up on this!

  If demand exists we will also produce modules for "last-generation" CPUs such as IDT's R4650. Check our web pages or enquire for status.

- *Local memory system*: two industry-standard 168-pin DIMM slots, for synchronous DRAM modules (3.3V, unbuffered). The DRAM always runs at the CPU's interface clock. The board's designed around 72-bit "ECC" types, but you can use 64-bit so long as you turn off CPU interface checking. But note that you've got to have parity DRAMs to use an R5000 secondary cache.

  It's a peculiar feature of the SDRAM market that 100MHz SDRAMs are unusable above 83MHz. You'll need modules one grade faster than the clock rate suggests.

- *ROM*: there's a DIL ROM socket (32-pin) which will support uV-erasable or flash types up to 512K×8; and a soldered-down 1M×8 flash ROM. The CPU can boot from either ROM. The ROMs are 8 bits wide, but hardware feeds the CPU with as many bits as it wants; the CPU can run cached from ROM too.

  When your P−5064 is delivered the ROM socket is empty; it's there for start-of-life bootstrapping, emergency rebooting, and support of useful ROM emulator products.

- *PCI*: 33MHz, 32-bit, compliant with V2.1 of the PCI standard; three industry-standard edge-connector slots.
- *ISA slot*: a single double-length card position for legacy PC cards.
- *PC-card (PCMCIA) slots*: connector for one or two cards, as used in portable PCs.
- *Interrupt controller*: custom design for maximum flexibility in interrupt routing and prioritising.
- *Choice of I/O*: a bit of everything: two serial ports, Centronics, IDE, SCSI, 10/100Mbit ethernet[2], PC keyboard/mouse, USB, real time clock, LED display, optional "front-panel" type LCD display, user-

programmable parallel I/O.

- *Re-programmable logic*: P−5064 is built with high-speed CMOS MSI chips for the data path, controlled and sequenced by logic in re-programmable FPGAs. The devices used (Xilinx 9500 family) are “flash” types, and can be reloaded through an external interface - a PC Centronics port and special cable is all the hardware you need.

  One advantage of this is that we can deliver hardware upgrades and bug-fixes over internet.

- *PC “ATX” form factor*: the board is physically compatible with an “ATX” motherboard. That should make it easy for you to buy a compatible power supply, cables, and (if you need it) a case.

Why not?

When we build a board like this we have to stop somewhere. Here’s what we turned down:

- *Wide ROM*: the 8-bit onboard ROM minimises space requirements and cost, but means that when the CPU runs from ROM it runs slow. We did this because we think that few applications using this class of CPUs will run serious code from ROM (even wide ROM is much slower than SDRAM).
- *EDO DRAM option*: the SDRAM price premium is too small for conventional DRAM to be a sensible alternative, and SDRAM is likely to be mainstream PC memory in 1998 anyway. It doesn’t make sense to get such a fast CPU and then slow it down with 33MHz burst rate memory.

  We’ll regret this decision if there’s an SDRAM famine in 1998.

- *Faster or wider PCI*: 33MHz 32-bit is the universal standard; faster or wider may never happen on a large scale.
- *Compact PCI*: we think this would take up too much board space and be incompatible with the PC-orientated mechanics of our board. Let us know if you’d find some hybrid useful.

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, and the government regulations intended to prevent trouble with it. P−5064 is not currently certified under European regulations, because it is not itself a system but only a component[3]. By design, P−5064 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−5064 open on a bench set up. Use of a 100MHz+ system without any overall metal shielding is likely to produce radiated emissions which drastically exceed the levels permissible for office (let alone domestic) equipment. European (and other national) 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−5064 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 under EC rules 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] Revision B boards (with serial numbers 1-21 or so) have only 10Mbit/s ethernet.

[3] 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−5064 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−5064*: configured with the CPU, and the amount and type of memory, you ordered.
- *Transition Cables*: there will be some standard transition cables, which you can either use to bring out connections for a board in a PC box, or to convert onboard connectors to industry standard ones. Refer to §9 on page 63 if you need a list.

You may also find (if you ordered them):

- *Extra cables*: some are optional, see §9.
- *Debug board*: makes it much easier to watch addresses/data in your program. Invaluable for driver and ROM-code debug; see §10.1 below.
- *LCD display*: an optional 16×2 alphanumeric display supplied loose with a short transition cable. It is valuable for prototyping applications which need a "front panel" of some kind (you may find the user-defined programmable I/O port usable for the front panel inputs).

  It's up to you to organise physical support, or just leave it lying around. More information in §8.14.

2.2. Initial wiring up

P−5064 is quite happy operating on a bench top. There are no dangerous voltages, and CPUs which need it will be fitted with fanned heatsinks.

You'll need to connect at least power, and possibly some other stuff.

- *Power*: "ATX" PC power supplies are cheap, electrically safe, and plug right in.
- *Serial port(s)*: if the connection from your computer terminates in a female 9-pin D-type (as it would to connect to a 9-pin PC Port), there's a good chance that you can just plug them in. If not, refer to Figure 8.2 below and settle in for the usual RS232 interface lead struggle.

  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*: if you have 100BASE-T or 10BASE-T "twisted pair" ethernet, it should plug straight in to P1.

  Alternatively, an AUI cable (the 10Mbit/s "transceiver" interface) connects to P12 via the supplied transition cable.
- *Centronics for download*: the most common PC-to-printer Centronics cables terminate in a "Centronics" style connector and are no good to you for connecting P−5064 to your PC. You should have found (in the box) a cable with a PC-compatible 25-way male D-type connector at one end wired pin-for-pin to a 25-way female D-type connector at the other end, ready to plug in to P−5064's "Centronics Peripheral" connector[4]. If it isn't long enough, buy what your PC store will call a "Centronics extender" cable.

[4] If your board is revision B or C then there's a horrible bug on the Centronics interface, and you'll probably also need a patch unit, available from Algorithmics.

2.3. Switching on

The recommended ATX power supply has a soft switch; when the mains power switch is first thrown only a low-current “standby” +5V supply is sent to the board. To switch on the P−5064’s power supply toggle the reset/debug switch to the “debug” position.

Some programs may provide a “switch off” command; otherwise you can turn the board off at the power supply mains switch.

In Revision C and higher boards a fancy power-control system is provided by the combi I/O chip and retains register-state across power-down. If you power-down such a board by switching off the power supply or pulling the plug, then it will come straight up when AC power is restored.

2.4. Boxing a P−5064

P−5064 is designed to fit into PC “ATX” metalwork and is PC-compatible in its size, fixing hole positions and standard connectors (PCI, keyboard, power supply). PC metalwork varies, so you may need some patience.

The metalwork should have fixed or optional openings for the I/O connectors along the back of the board (mouse, keyboard, serial ports, Centronics connectors and 10/100BASE-T ethernet). The SCSI cable we supply 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 J22, allowing the board to be reset from the front panel. ATX boxes will often have a power-on switch, which can be plugged into J21. The power switch will now double as a “debug” interrupt button.

2.5. Normal sign-on sequence and what it means

From power up your P−5064 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−5064 says	*What it means*
`Notice: Integrated Tests`	
`Info: Version: P5064L 711: \` `  Mon Jul  6 14:15:09 BST 1998`	PROM sign-on. "P5064B" for big-endian, "P5064L" for little-endian. The number is the AlgPOST version number. Note that the PROM contains both the power-on self-test 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`	Started cache tests
`Info: Dcache size 16 Kbytes (32/line)`	
`Info: Icache size 16 Kbytes (32/line)`	
`Info: Activity: dcache refill test`	
`Info: Activity: dcache writeback test`	
`Info: Activity: flash memory operation`	
`Info: Flash: Am29F080`	
`Info: Activity: RTC operation`	
`Info: Date: Mon 6/7/1998 16:46:00`	
`Info: Activity: quick memory address test`	
`Info: Activity: memory byte address test`	More often you only run the "quick" memory test; see PMON manual for how to choose which ones run.
`Info: Activity: memory halfword address \` `  test`	
`Info: Activity: memory word random test`	
`Info: Activity: ns16550 operation`	
`Info: Activity: keyboard operation`	
`Info: Activity: PCI operation`	
`Info: V962 silicon revision 4`	The PCI bridge chip. Revision 4 indicates the part which got renamed V360EPC - but V3 were originally just going to call it rev C...
`Info: Activity: SCSI operation`	
`Info: SCSI: silicon revision 2`	
`Notice: Integrated Tests Completed`	
`Notice: Executing PROM package 6`	Control is now being handed over from the power-on tests to PMON.

P−5064 says	*What it means*

`PCI slot 0/0: Digital Equipment DEC 21143 \`
`  (network, ethernet)`
`PCI slot 1/0: NCR 53c810 (mass storage, \`
`  SCSI)`
`PCI slot 2/0: Intel 82371SB PCI-ISA \`
`  bridge(bridge, ISA)`
`PCI slot 2/1: Intel 82371SB IDE interface \`
`  (mass storage, IDE)`
`PCI slot 2/2: Intel 82371SB USB interface \`
`  (serialbus, USB)`
	PMON is probing for active PCI devices

`de: 21143 [10-100Mb/s] pass 3.0 \`
`  address 00:40:bc:04:00:64`
`  en0: media: 1="10baseT" 2="Full Duplex \`
`  10baseT" 3="AUI" 4="100baseTX" \`
`  5="Full Duplex 100baseTX"`
	Ethernet driver initialisation, prints ethernet address and list of connected interfaces

`PMON version 0.0.214 [P5064,EL,FP,NET]`
`Algorithmics Ltd. Jul  6 1998 14:02:43`
`This software is not subject to copyright \`
`  and may be freely copied.`
	PROM monitor version and date

`Board Rev: C; FPGA Rev: 04; User Options: 7`	Revision information about your board - vital when using Algorithmics' support lines.
`CPU type R5260.  Rev 1.0.  166.63 \` `  MHz/83.31 MHz.`	From CPU ID register and measurement of the clock rate (which may sometimes be slightly off).

`Memory size  32 MB.`
`Icache size  16 KB, 32/line (2 way)`
`Dcache size  16 KB, 32/line (2 way)`
	PMON should agree with AlgPOST
	These figures are right for the RM5260 and most R5x00s, but others differ.

`PMON>`	You've got a prompt

Table 2.1: P−5064 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.

2.6. Flash memory and socketed PROM

P−5064 normally boots from an onboard 1M×8 flash memory, pre-loaded by Algorithmics. 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 (U33) which accepts an alternative bootstrap source - a 512K×8 150ns ROM, in a 32-pin dual in-line package. The board will use the ROM socket for its bootstrap if you insert jumper J17.

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.

2.7. 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 the top "page" of the onboard flash memory device. It 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−5064; we'll explain what they mean.

```
PMON> set
   netaddr = 192.168.1.7
   ethaddr = 00:40:bc:04:00:09
   itquick = y
  hostname = comm7.comm.algor.co.uk
nameserver = 192.168.1.65
   gateway = 192.168.1.65
  bootaddr = gate
         v = gate:/tftpboot/p5064/vx5260
         m = cmemram
         t = gate:/tftpboot/p5064
ittstlevel = 7
         i = gate:/tftpboot/p5064/itram
     itrom = gate:/tftpboot/p5064/fload.itrom
    dlecho = off         [off on lfeed]
   dlproto = EtxAck      [none XonXoff EtxAck]
  hostport = tty1
   heaptop = 80020000
    moresz = 10
    prompt = "PMON> "
    brkcmd = "l -r @epc 1"
    datasz = -b          [-b -h -w -d]
   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"
   showsym = yes         [no yes]
     fpfmt = both        [both double single none]
     fpdis = yes         [no yes]
PMON>
```

Table 2.2: P−5064 - typical PMON environment variables

What do all these mean?

- *ethaddr*: Without this, no network. The first part of the address ("`00:40:bc:04`") is the same for all P−5064s; the last four digits of the hex ethernet number are the board's serial number (but in hex); this board is serial number 9, which is `0x0009` in hex; but the conversion gets a bit harder for bigger numbers!
- *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 over your local network.

  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 (figured out by comparing our IP address of that with the host, subject to the “netmask”) will be sent here. Useful if you keep your prototype boards on a separate subnet.
- *bootaddr*: default network host to use when using *tftp*. You can always give an explicit host name.
- *v, m, t, i, itrom*: typical programmer-set shortcuts, allowing you to just say (for example):

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

  to load and run the program.
- *dlecho, dlproto*: control download over serial or parallel link. P−5064 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. Overview and Block diagram

The easiest way to understand the board is to appreciate that it's built in two sections, joined at the PCI controller. One includes the CPU, local memory, ROM and the "host" side of the PCI interface; the other includes almost all the I/O system, and connects via the onboard PCI bus.

Actually a few signals (interrupts, device resets) do creep across the boundary. But it's near enough right to make it worth drawing the block diagram in two sections.

3.1. CPU and memory

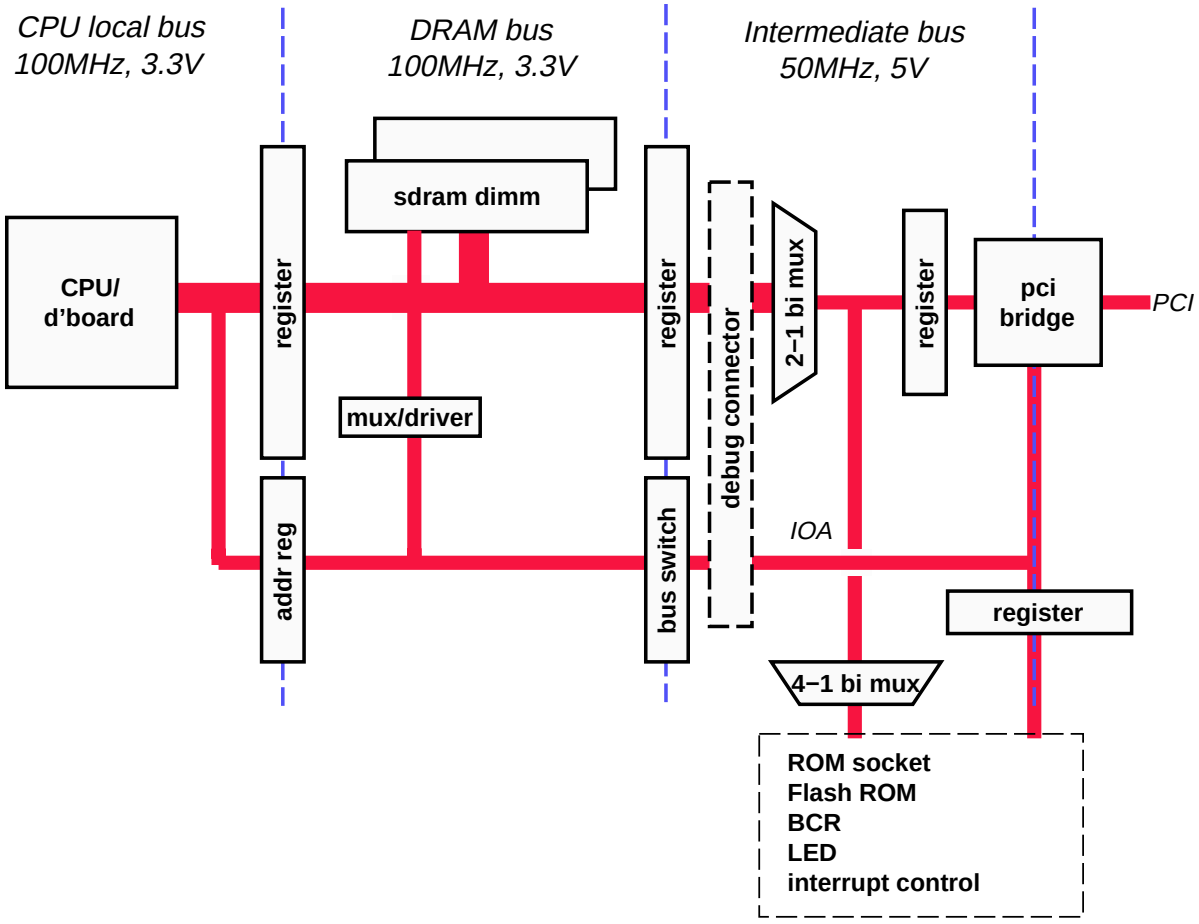


Figure 3.1 CPU, memory and local I/O bus

- *CPU/daughterboard*: P−5064 has both a position for a CPU on the motherboard and a socket for a small daughterboard for variant CPUs and/or CPUs with secondary cache RAMs.
- *Data paths*: the CPU local bus is connected to the DRAM data bus, and from there to the 32-bit intermediate bus and I/O system. The rationale for this is that CPU I/O operations are much less frequent than DRAM cycles, so it creates little overhead to route them through the DRAM bus; and in that way the critical high-speed CPU local bus has only one load point.
- *Address paths*: the CPU local bus - the MIPS designation is *SysAD*- is multiplexed, and the address registers pick up and store the address of each cycle.

- *SDRAM*: P−5064 is designed entirely around synchronous DRAM (SDRAM) memory components, fitted in DIMM modules. SDRAMs offer vastly better burst data rates than fast-page or EDO DRAMs - to 100MHz instead of to about 33MHz. P−5064 exploits this performance by having the SDRAM system very tightly coupled to the CPU for the highest performance achievable in this “discrete logic” design.
- *Connection to intermediate bus*: the SDRAM data bus is registered (for speed conversion) and multipexed down to 32 bits[5] wide to produce an intermediate bus. The bus has several purposes:
  - It provides a hospitable environment for the V360EPC PCI controller, which was designed for an i960 bus;
  - It makes a good home for the system ROM; it’s off the critical high-speed DRAM bus, but accessible without the complex configuration necessary to get PCI running;
  - It provides a point of compatibility with Algorithmics’ P−4032 design, which uses a similar bus.
- *ROM and Local I/O*: live on an 8-bit extension of the intermediate bus. During ROM cycles a byte at a time gets read from the ROM and captured by each byte-lane of the intermediate bus register in turn. We also include devices here which are required so early in system initialisation that we’d rather not have them beyond the PCI controller (because it requires fairly complex initialisation).
- *PCI bridge*: this is a V360EPC device from V3 corporation. It’s familiar (it’s also used in Algorithmics’ P−4032 product) and works OK.

3.2. PCI and I/O

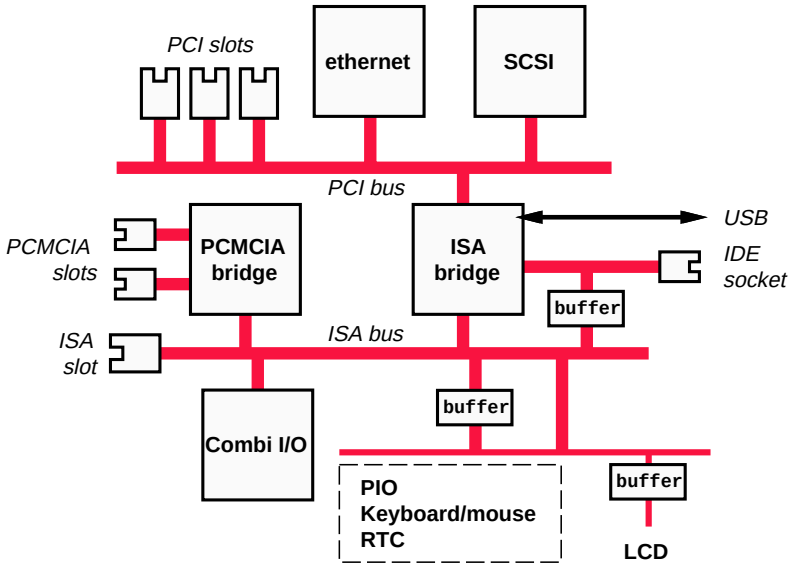


Figure 3.2 PCI, ISA, PCMCIA and PC-type I/O

- *PCI bus*: 32-bit, 33MHz, PCI 2.1 compliant onboard bus and three PC-type expansion slots.

  P−5064 does the PCI host role, supplying clocks and arbiter. You can use an external arbiter instead.

[5] Here and elsewhere in the design bus width matching is done using “QuickSwitch” components to provide a completely bidirectional multiplexer/selector.

- *Ethernet*: onboard 10/100Mbit/s ethernet controller, using the DECchip 21143[6]. Two interfaces are provided - one for twisted-pair (either speed) and one for 10Mbit/s transceiver. The interface type is software-selected, and most drivers will automatically configure themselves to talk to an active connection.
- *SCSI*: 8-bit SCSI-2, using an NCR53C810 controller. There are active terminators for the SCSI bus onboard.
- *ISA bridge*: an Intel 82371 produces an old-fashioned ISA bus for old plug-in boards and onboard dumb devices.
- *USB*: two connections, implemented by the i82371. P−5064 just has a header pin connector, and you'll need a transition cable to use standard peripherals. Note that this is a USB *host* function - USB is a highly asymmetrical bus.
- *IDE interface*: dual high-speed IDE channels for adding low-cost disks and other peripherals. Implemented by the i82371 as a high performance PCI bus master.
- *ISA bus*: single slot which can accommodate a long ISA bus card.
- *PC-card (PCMCIA) bridge*: uses a Vadem VG469 controller to support two slots for miniature add-in cards.
- *Combi I/O controller*: includes dual (16550-compatible) serial ports, diskette interface and PC-type "Centronics" parallel port with bidirectional operation extensions, and capable of playing both the host or peripheral role.
- *"PC motherboard" I/O*: includes PC-type keyboard/mouse controller, real-time clock chip and a general-purpose programmable parallel I/O controller for whatever users want.

[6] Revision B boards (serial number up to 21) have only 10Mbit/s ethernet, with a DECchip 21041.

4. Memory map

4.1. CPU's memory map

The CPU generates a 36-bit address. But for compatibility with our 32-bit P−4032 board and general unwillingness to widen the buses, we only decode bits 0−31 of that address, to give a 4Gbyte address range. And then there are some MIPS facts of life which influence the map:

- Following a reset the CPU starts execution at `0x1FC0 0000` (physical), so this area must map to onboard ROM.

  The other "hard-wired" addresses in the MIPS architecture are the exception/interrupt entry points, which are in low physical memory. That means it's important to have high-speed program memory at the bottom of the map.

- Much system software finds it easier to operate in the *kseg0* and *kseg1* "unmapped" spaces described in Appendix A, and such programs will only generate physical addresses up to and including `0x1FFF FFFF` (the low 512Mbytes of address space).

  We've therefore arranged to map all onboard resources into that first 512Mbytes - the remaining 3.5Gbytes of address range are available for producing strange PCI bus addresses.

Note that revisions of this manual before 1.7 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 4.1 a dagger (†) denotes that the address is software-configured at boot time - the value given is recommended and fits in with the hardware decodings. You can change it, but the consequences are your responsibility!

Base Address	*Size*	*Class*	*Description*
`0000 0000`	256Mb	Memory	onboard DRAM memory
`1000 0000`	8Mb†	ISA via PCI	ISA memory access window. Actually, this is programmed as the start of a window on PCI space, but not used for onboard devices, and should not be used by PCI-add-in cards. Unclaimed PCI accesses are directed at the ISA bus (called "subtractive decoding" in PCI jargon). This only reaches half the ISA's limited addressing range - but that should be enough for most purposes.
`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
`1d00 0000`	16Mb†		PCI I/O space window: used (and probably *only* used) to access the I/O space of the "ISA" bus and its PC-like peripherals.

Base Address	*Size*	*Class*	*Description*
1d00 0000	64K		ISA bridge I/O space via PCI
1d00 0000			i82371 DMA channel 1
1d00 0020			i82371 ICU 1
1d00 0040			i82371 counter/timer
1d00 0060			keyboard data register (combi I/O chip)
1d00 0064			keyboard control register (combi I/O chip)
1d00 0070			real-time clock (address, combi I/O chip)
1d00 0071			real-time clock (data, combi I/O chip)
1d00 0060			keyboard controller (combi I/O chip)
1d00 0080			i82371 DMA address base registers
1d00 0092			SYSC_PORT
1d00 00a0		ISA I/O	i82371 ICU 2
1d00 00c0			i82371 DMA channel 2
1d00 01f0			i82371 IDE registers
1d00 02f8			Combi I/O, serial port 1
1d00 03f8			Combi I/O, serial port 0
1d00 0378			Combi I/O, bidirectional Centronics
1d00 037c			Centronics peripheral data input register, for use in old-fashioned Centronics mode only.
1d00 03e2	2 reg		Configuration registers for Vadem VG469 PC card controller. All other PC-card locations are software-configurable and not yet fixed.
1d00 03f0			Combi I/O, diskette controller
1d00 ff00			Combi I/O, GPIO lines
1ee0 0000	1Mb	PCI	PCI configuration space: access to PCI devices' configuration registers. In P−5064, PCI device *IOSEL* signals are derived from high PCI bus address bits - see §5.7. ("PCI bus") on page 41 for details. You need to configure V360EPC before this will work.
1ef0 0000	64Kb†	V360EPC	PCI controller's internal registers
1ef1 0000			reserved
1f80 0000	512Kb	ROM	Byte-writable programming window on ROM: configured bootstrap, socket and onboard flash respectively
1f90 0000	512Kb		
1fa0 0000	1M		
1fc0 0000	512Kb	ROM	Boot ROM location (accesses either ROM socket or flash locations, depending on the setting of the boot jumper J17.)
1fd0 0000	512Kb		EPROM in socket
1fe0 0000	1M		Onboard flash ROM - top 64K usually reserved environment
1ff0 0000	1 reg	Local I/O	
1ff2 0010	4 reg		LED display cells (leftmost has lowest address)
1ff3 0000			LCD display
1ff4 0000 1ff5 0000			Rev B boards only - Z80 GPIO controller registers (later boards have GPIO pins on the combi I/O chip). The 0x1ff5.0000 decode is used to emulate a Z80 interrupt acknowledge cycle, to keep the chip happy.
1ff6 0000			UART on debug board (see §10.1)
1ff9 0000	9 reg		Interrupt controller registers
1ff9 003c	1 reg		Logic revision level register
1ffa 0000			board configuration bits 0
1ffb 0000			board configuration bits 1
1ffc 0000			DRAM configuration register bits
1ffd 0000			board revision and option links register
2000 0000	3.5Gb†	PCI	Available if you need access to a larger region of PCI space than is available in the lower-memory window. You'll need to program the TLB or use 64-bit pointers to get addresses bigger than 0x2000 0000 out of the CPU.

Table 4.1: P−5064 physical address map

4.1.1. PCI and ISA “DMA” memory map

PCI devices can be bus transfer initiators, and operate by moving data directly into and out of local memory - the onboard Ethernet and SCSI controllers work like that, and so may plug-in PCI devices.

In principle ISA bus devices can take advantage of the DMA service provided by the PCI/ISA bridge chip, too. However, a combination of unexpected (but documented) behaviour by the Intel ISA bridge chip and unexpected and undocumented behaviour by the V3 PCI controller means that use of ISA bus DMA may cause a deadlock, and should be avoided. That may change in future versions of P−5064.

By default, the Algorithmics monitor defines two PCI-accessible windows onto local memory (called “apertures” in the V3 documentation), as shown in Table 4.2.

Base Address		*Size*	*Description*
PCI	*CPU*		
0080 0000	0080 0000	8Mb	Maps to 8Mbytes of local memory at the same physical address. Use for ISA bus DMA (if it ever works - see note above). You may also find this window useful for a PCI device, when an existing driver believes that PCI and physical addresses are necessarily identical. But you have to figure out whether it will be possible to restrict it to operating in this range of addresses.
8000 0000	0000 0000	256Mb	Window onto all possible local memory addresses. The effect of a PCI access to addresses higher than those supported by your board’s DRAM configuration is undefined.

Table 4.2: PCI master’s windows onto local memory

4.1.2. PCI memory map (CPU-initiated cycles)

To access a PCI location: you have to program an *aperture* in the PCI bridge chip. An aperture has several components, as shown in Figure 4.1.

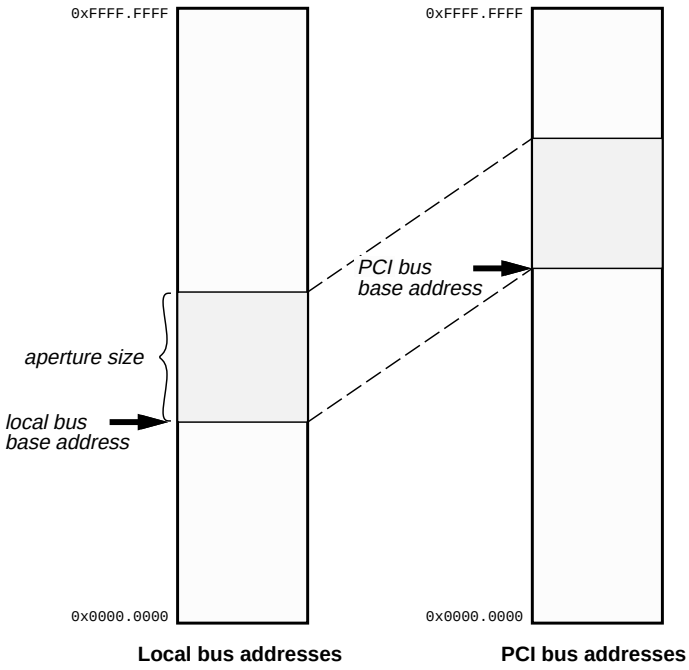


Figure 4.1 Defining an aperture onto PCI space

Here's what you need to define an aperture:

1) A local bus base address. Accesses to the region starting here will get decoded by the PCI bridge chip and forwarded to PCI.
2) A size, implemented by a mask which determines what bits to check against the base address; that lets you define apertures as small as 64Kbyte and as large as 256Mbyte.
3) A PCI base address which is used to relocate the bottom of the mapped region into PCI space.
4) Some information on how the PCI access is to be carried out. This allows you to set up apertures to talk to PCI I/O or configuration space, for example.

Cache refill or writeback cycles are not supported by the PCI subsystem. Since the MIPS CPU does accesses cached or uncached according to the program address (see Appendix A), it's the programmers responsibility to make sure you do only UNCACHED accesses to PCI space.

You also need to map the PCI address into your aperture, adding the appropriate base address.

Table 4.1 shows Algorithmics' default address setup for the PCI bus, and where you can locate onboard devices. If you want to change it you can, but it is likely to confuse everyone.

5. Programming P−5064

This chapter focusses on P−5064 from a programmer's point of view.

5.1. CPU

P−5064 can support any MIPS CPU using a 64-bit derivative of the system interface introduced with the R4000, and extended to support the R5000 on-bus secondary cache. Algorithmics intend to provide CPU daughterboards for all popular CPU/cache combinations.

CPUs announced to date in this class (and not plainly obsolete) include:

Manufacturer/CPU	*Date of* intro	*max i/f* clock	*Ext* cache?	*Notes*
IDT R4700	1994	67MHz		MIPS III instruction set
IDT R4650	1995	67MHz		Low-cost but cut-down CPU with no double-precision floating point and no memory management hardware (TLB). Integer multiply-accumulate instructions.
MIPS R5000	1995	100MHz	✓	MIPS IV instruction set
MIPS R10000	1995	83MHz?	✓	MIPS IV instruction set
QED RM5260	1997	75MHz		MIPS IV + integer multiply-accumulate
QED RM5270	1997	83MHz	✓	MIPS IV + integer multiply-accumulate
QED RM5261	1998	100MHz		MIPS IV + integer multiply-accumulate
QED RM5271	1998	100MHz	✓	MIPS IV + integer multiply-accumulate
QED RM7000	1998	100MHz	✓	MIPS IV + integer multiply-accumulate, prefetch, non-blocking reads.
NEC Vr5464	1998	100MHz		MIPS IV + integer multiply-accumulate, 8-bit "MDMX", prefetch, non-blocking reads, other enhancements.

Of these the QED RM5270 and RM7000 are pin-compatible; RM5260, RM5270 and R5000 versions of P−5064 are available now (Summer 1998). Versions for other CPUs are likely to be scheduled according to customer demand.

Differences between CPUs

In software terms, there is little to choose between most of these processors; and what differences there are, are mostly hidden by the compiler. However:

- *R4650*: is a cut-down CPU whose lack of a TLB and double-precision floating point hardware make it unsuitable for some applications; of course, the resulting small die and low price make it attractive for others.
- *MIPS III vs MIPS IV*: the R5000 and later CPUs implement the MIPS IV version of the instruction set, and both this and some implementation improvements make them likely to offer significantly better performance on floating-point intensive programs.
- *Integer multiply/accumulate*: available on all except the real "MIPS" CPUs, boosts performance on some "DSP"-like algorithms such as MPEG decoding.

  MIPS IV CPUs have a floating point multiply/add instruction, which may provide a perfectly good alternative.

- *Dual-issue*: the R5000 and its successors can all issue a floating point and an integer operation in the same clock cycle. RM7000 and Vr5464 (quite different designs) can both dual-issue a much wider range of instructions.

  This has few software-visible effects. But R5000's dual-issue shows up at the bus interface when running uncached; the R5000 fetches 8 bytes of instruction data and (so long as they are both instructions in its execution sequence) runs both of them. This habit has been criticised (the bootstrap memory must be 8 bytes wide and the system interface has to know the CPU endianness before you can run any code at all); later RM5xxx CPUs fetch instructions one at a time. As a result, R5000 runs about twice as fast uncached; rarely important, but may catch you out.

Daughterboards and on-motherboard CPUs

A P−5064 motherboard can accommodate one type of CPU directly; others are added on daughterboards. CPUs with external caches must be on daughterboards.

Optional external cache

Some daughterboards will include a CPU with an external secondary or tertiary cache. Caches could be as small as 256Kbytes or (for some CPUs) as large as 8Mbytes. The price list currently includes RM5270 and R5000 CPUs with 2Mbyte caches.

CPU configuration options

The main input clock is derived from a synthesiser IC and is set by a group of jumpers summarised in Table 7.2; likely choices are 75, 83 and 100MHz.

Options determined by the CPU itself are set up mostly on the jumper block J24 (described in §7.2), though endianness is configured by one of the jumpers on J23 (described in §7.3).

- *Basic clock rate*: configure the clock synthesiser to any of its options in the range 50-83MHz.

  Higher clock speeds are available by fitting a dedicated oscillator into position OSC1 and changing the jumper J5 to 2-3.

- *CPU clock multiplier*: CPUs run internally at some multiple of the basic clock rate; see Table 7.3.

- *Secondary cache present/absent*: R5000-like CPUs need to be told that a secondary cache is fitted; it's all in §7.2. ("CPU software and options jumper: J23, J24") on page 51.

- *Secondary cache size*: needs configuration on daughterboards; some may be pre-configured, others may have links.

- *Endianness*: regardless of your CPU choice P−5064 can be set up either big-endian or little-endian. Software binaries for big- and little-endian are different, and you will have to make sure that the boot ROM program and any other software you want to run has been built to match the CPU's configured endianness.

5.2. Local SDRAM memory

P−5064 uses synchronous DRAM 168-pin DIMM modules. These are 3.3V, "unbuffered", 72-bit (also called "ECC") types. DIMMs are fitted with encoding slots which match coded separators in the socket; so if your DIMMs won't fit in the sockets, the chances are they are the wrong type.

Speed grades for DIMM memories were rather confusing. You needed components at least one speed grade higher than the CPU interface clock rate of your P−5064. However, there's now a straightforward solution - you should buy DIMMs described as "PC-100 compatible" and they should function in P−5064 up to 100MHz.

Apart from these (manifest) features, there are also some options with no functional significance in a running board, but where the memory controller on P−5064 must be configured correctly to match your particular DIMMs.

By a convention initiated by IBM and sanctified by PC-100, each DIMM carries “self-portrait” data encoded in a in a tiny on-DIMM EEROM device, accessed through a compact 2-wire interface. P−5064 is equipped to access that data.

Most of the time, this will be done by Algorithmics’ bootstrap monitor sequence at power-on and you won’t have to touch it again; if you need to replace the bootstrap we suggest you approach Algorithmics and get a copy of our code. But the sections below explain the options; you can find how to change the memory controller setup in the “DRAM configuration register” described in §5.4.2; we’ve even included details of how to read and interpret DIMM EEROMs in Appendix C.

Modules and sizes

In each of P−5064’s two sockets:

- The DIMM module contains one or two *sides* of memory components, 72 bits wide. (We say “side” because the second set is usually assembled on the reverse side of the DIMM PCB, though it doesn’t have to be).
- Each DRAM component is divided into 2 or 4 *banks*. It’s possible to overlap actions in different banks in high-performance memory systems[7].
- Each bank consists of an almost-square two-dimensional array of memory locations. Applying the *row* address to a DRAM causes a whole slice (a *column*) of the array to be fetched to an internal holding store; the data can then be accessed by specifying a *column* address which picks the datum you want from the internal store.

  At the end of any access cycle, the entire column of data is written back to the main DRAM store[8].

DIMMs of the same capacity may differ:

- There are various speed grades (P−5064 is likely to use 83, 100 and 125MHz components in DIMMs marked as 67, 75, 83 or 100MHz). The DIMM speed grade tells you what speed of PC chip set it works with; at the time of writing that means that *all* DIMMs tend to be described as “67MHz” because that’s how fast PCs go. Read the small print in the DIMM data manual, or ask Algorithmics.
- Some have one side, some have two.
- Some use 2-bank components, some use 4-bank components.
- Components vary in how big the columns are, and how many columns make up the array - that is, how many *row address* bits and how many *column address* bits are valid.

In theory, all you do is to read and decode the DIMM configuration out of the EEPROM and write appropriate values into the DRAM configuration register (see §5.4.2).

Outcomes of out-of-range memory accesses

Once the DRAM configuration register is set to match your SDRAM components, an out-of-range memory access in the DRAM region of the memory map will fail to select any DRAMs. The data bus will therefore “float”, and a read will usually return whatever doubleword value was last seen on the bus. Quite commonly, this will come back with correct parity; but after a while the data will “decay” and you’ll get a parity error. Software which probes for memory configurations should either disable parity error checking or catch the parity exception.

---

[7] P−5064 doesn’t do this. These facilities give little extra performance for a lot of extra complexity.

[8] In fact, the main DRAM store is “leaky” and every column must be *refreshed* by being accessed often enough to prevent the data from dribbling away to unreliability; typically every column must be touched about 250 times per second. That’s why it’s called “dynamic RAM”.

If you do write and read-back memory locations to sense the presence of DRAM, make sure that you write a different doubleword bit pattern to somewhere between the write and read.

5.3. Flash ROM and the boot ROM socket

P−5064 normally bootstraps itself and runs its debug monitor out of a 1Mbyte flash ROM. But for its start-of-life bootstrap, emergencies or to use a ROM emulator device it also supports a socket for a real dual-in-line PROM.

Both devices are always visible in the memory map of the board at their own unique addresses. In addition, whichever ROM is designated as the bootstrap device is mapped into memory at location `0x1fc0 0000` physical - which is where MIPS CPUs start execution when reset. Jumper J17 should be out to boot from onboard flash, and in to boot from the socket.

5.3.1. Flash ROM

Flash ROMs can be reprogrammed under software control, but retain data indefinitely with no power. However, you can't just overwrite the bytes you want; to re-write a flash part you must first erase it (using a special software command sequence) and then program it (using yet more special sequences). The erase operation works not on individual bytes, but on large chunks ("sectors") of the memory space[9].

P−5064 features a 29F080 part, sometimes from AMD and sometimes from Fujitsu: it's 1Mbyte in size, 8 bits wide[10], and is erasable in 64Kbyte sectors.

5.3.2. PROM socket

By default, this accepts a 512K×8 uV-erasable PROM, whose access time is 120ns or less. But by moving a couple of links (specifically, moving J8 and J7 from their default "2-3" position to "1-2") you can also use an AMD 29F040 or compatible flash device. If you want to use the socket for a ROM emulator or similar, you may need the pinout so it's shown in Figure 5.1.

[9] On early devices you had to erase the whole contents of the device; it's this erase-everything-at-once feature which originally led to it being called "flash" memory.

[10] The Fujitsu part has a 16-bit bus, but we use an 8-bit compatibility mode.

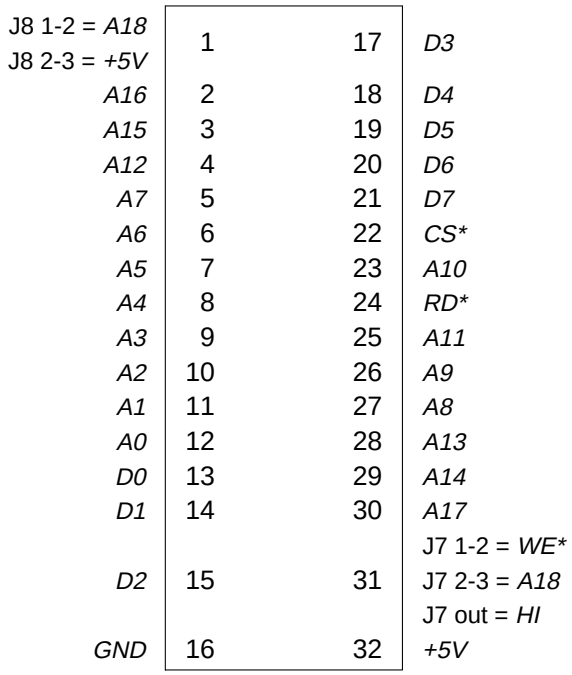


Figure 5.1 Pinout of ROM socket

5.3.3. Programming flash memory

Before you do anything, note that Algorithmics' PMON bootstrap ROM reserves the highest 64Kbyte sector of the flash ROM to keep its "environment store", which holds important information about the board and shares it between different applications. Even if you're putting completely different software in the ROM, you should keep the environment store; you can get software to read and maintain it free from Algorithmics.

An additional jumper J6 is normally fitted, but can be removed to write-protect a flash ROM fitted in the socket (it disables the write strobe signal to the ROM socket.)

The flash-programming endianness problem

Whenever the CPU reads a ROM location on P−5064 the board runs eight byte-wide ROM reads and assembles the bytes into a double-word - even if the CPU is only trying to read a byte or some other sub-word quantity[11].

The assembly process is in fact managed in two halves; four ROM bytes are built into 32-bit words on the intermediate bus and two such words are collected at the intermediate bus/memory bus interface.

For rather dubious historical reasons the assembly of ROM bytes into 32-bit words is done the same way, regardless of the processor's configured endianness; the ROM data with the lowest ROM address is presented on the intermediate bus signals 0-7, and the highest-ROM-addressed byte on signals 24-31.

However, for compelling practical reasons the assembly of two 32-bit words into one 64-bit doubleword is done according to CPU endianness; for a little-endian CPU this is consistent with the byte-within-word organisation, but for a big-endian CPU it isn't. Figure 5.2 shows what happens.

[11] This is more reasonable than it first sounds, because the CPU can only read partial words from the ROM when running uncached, which it should do only for a small part of the early bootstrap sequence.

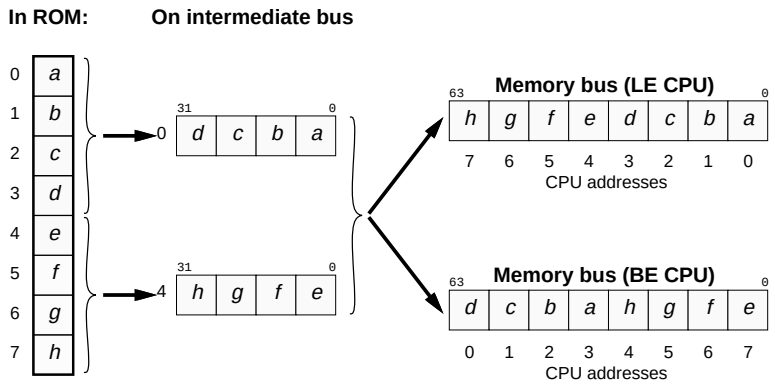


Figure 5.2 How ROM bytes become CPU data

You need to understand this organisation to be able to build a functioning ROM for a P−5064, but most ROM preparation utilities are forgiving of hardware's peculiarities and are prepared to rearrange bytes appropriately.

However, the read-8-bytes-at-once mechanism, unproblematic for normal reads, offers no way of doing writes and is quite unacceptable for the "polling" and other magic reads required as part of programming a flash device. So we've provided a separate set of memory map windows onto the ROMs, where something else happens which is more useful for programming.

In the ROM programming windows, *only* byte accesses are supported - the effect of wider accesses is undefined. But in these windows, the CPU byte address is fed directly to the ROMs. For a little-endian CPU this makes sense, but for a big-endian CPU the result is that within 8-byte aligned blocks the programming view of the bytes is horribly scrambled with respect to the regular ROM-reading, program-executing view.

You can read off the relationship between the addresses from Figure 5.2; but it's summarised directly in Table 5.1 too.

Byte address in normal region	0	1	2	3	4	5	6	7
Programming address	3	2	1	0	7	6	5	4

Table 5.1: Programming flash on big-endian CPU - ROM byte addressing

5.4. P−5064-specific hardware registers

5.4.1. Board configuration register

An array of 1-bit write-only registers used to configure the board, and to hold various subsystems in reset. You should access them with word writes, but only bit 0 of the data value is important. All the outputs of this register are set to zero (logic low) following a board reset or power-up. Eleven bits are currently used; five more are temptingly available for future use.

Offset	*Name*	*set 1 to*
0x0 0000		reserved
0x0 0004	CENTPRINTER	Use Centronics interface as a peripheral operating in “classic” old-fashioned centronics mode
0x0 0008	LED_BLNK*	Make LED display characters visible
0x0 000c	PSU_OFF	Rev B boards only - switch off power to the board. Later boards use registers internal to the combi I/O chip.
0x0 0010	CPU_PE_EN*	Disable parity checking by the CPU, when it reads from SDRAM memory. You have to do this if you use non-ECC DIMM modules; you can’t do this if you’re using a CPU with an R5000-type secondary cache.
0x0 0014	V3_PE_EN*	Mask the V360EPC (PCI interface chip) parity error signal, which is appropriate when the local memory is not storing correct parity.
0x0 0018		reserved
0x0 001c	STERM_EN*	Take P−5064’s SCSI terminators out of circuit. You should do this if P−5064 is not at either end of the SCSI cable - but it normally will be.
0x1 0000	V3RESET*	release the chip reset to the PCI bridge (V3 V360EPC)
0x1 0004	NCR_RESET*	release the chip reset to the SCSI controller (NCR 53C810)
0x1 0008	DEC_RESET*	release the chip reset to the ethernet controller (DEC 21143)
0x1 000c	ISA_PWROK	allow the ISA subsystem to power up in the normal way. With this signal inactive, the ISA bridge will hold the ISA bus in reset. Note that the ISA bridge chip waits a few mS after *ISA_PWROK* becomes active before releasing reset; if you cycle this signal wait 10mS or so before attempting any operation involving the ISA bus or ISA bridge.
0x1 0010	PCMCIA_RESET*	release reset to the PC card (PCMCIA) controller and sockets.
0x1 0014		reserved
0x1 0018		
0x1 001c		

Table 5.2: Board configuration register fields

5.4.2. DRAM configuration register

An 8-bit register which, consists of 8 individually-writable write-only bits. Six of these bits are used together as a 6-bit field.

Offset	*Description*
0x00	6-bit configuration value for a DIMM module, LS bit first. In normal operation, this configures the DIMM1 position.
0x04	
0x08	
0x0c	
0x10	
0x14	
0x18	A 0→1 transition loads configuration values into the internal register which configures the DIMM2 position.
0x1c	Reserved

Table 5.3: DRAM configuration register bits

So DIMM2 values must be loaded first and pushed into place by writing a 1 to offset 0x18. The data for each DIMM module is now interpreted as shown in Figure 5.3.

register offset	0x14	0x10	0x0c	0x08	0x04	0x00
	Sides	Banks/DRAM	Column address width		Row address width	
	`0` = 1	`0` = 2	`00` = 8 bits		`00` = “default”	
	`1` = 2	`1` = 4	`01` = 9 bits		`01` = 11 bits	
			`10` = 10 bits		`10` = 12 bits	
			`11` = 11 bits		`11` = 13 bits	

Figure 5.3 Memory configuration registers and DRAM types

5.4.3. Board (motherboard) revision

You can read this on four bits of the “software options” register described in §7.3. Value “0” indicates revision “A” and so on.

Note that the motherboard’s identity is also defined by its serial number (Algorithmics keep a list of the revision level of all boards); the serial number is in turn directly related to the ethernet address, which is available as part of the PMON boot monitor’s “environment” store.

5.4.4. Logic version

An 8-bit register, which tracks the version of the system logic loaded into P−5064’s flash-programmable Xilinx logic devices. It is incremented with new releases of the logic (a separate sequence for each board artwork and corresponding board revision register value). We don’t always publicise what gets changed; but sometimes board errata will be reported for a specific revision, and fixed by a higher one.

It will always help us resolve support queries if you tell us the board and logic revision levels of your board. Their value is reported by the bootstrap ROM at power up.

5.5. P−5064 interrupt controller

This flexible controller is implemented in one of P−5064's programmable logic devices, and has the potential to be customised to user's requirements. This section describes the standard version.

P−5064 has a lot of different possible interrupt inputs, which need to be collected, controlled and fed into one of the MIPS CPU's five usable interrupt inputs. Although the CPU inputs are identically treated in the MIPS architecture, it's common practice in MIPS software to use any CPU input to group only interrupts whose software handlers run with the same priority[12].

Given the diversity of operating systems available on P−5064, a good interrupt system should be able to provide something close to arbitrary steering of incoming interrupts onto CPU inputs. So we do that, at least for four out of six CPU interrupt inputs.

In addition, the interrupt controller:

- Allows each incoming interrupt signal's state to be read in an "interrupt request register" or IRR;
- Allows each interrupt to be individually enabled (disabled interrupts can't cause an active interrupt to the CPU);
- Latches interrupts which occur transiently, and provides a way for the latched value to be cleared down by the interrupt handler.

5.5.1. CPU interrupts

There are seven CPU interrupt pins on most MIPS CPUs: *Int0-5** and *NMI*.

P−5064's interrupt controller divides them up as follows:

- *Int0-3** are treated identically, and any normal interrupt input can be directed to any one of these under software control. These interrupt signals should be used for grouping together most interrupts into the CPU.
- *Int4** is reserved for high-priority ("panic") interrupts - fatal, serious error or other beyond-normal-device events.
- *Int5** is not used at all; MIPS CPUs after R4000 have a useful internal timer which can and usually does take over this interrupt input.
- *NMI** is not driven by the standard interrupt controller, but is wired in to be available for customers specials.

5.5.2. Interrupt sources

The different events which can potentially cause interrupts in P−5064 are listed out in Table 5.4; it also lists the signal name as used in the schematics. Signals ending with a "*" are active low - which doesn't normally matter to the programmer.

Name	*Signal*	*Significance*
berr	*BUSERR**	Timeout on intermediate bus cycle.
debug	*DBGINT**	From input on debug board, intended for trigger outputs from logic analysers or other hardware debug assistants.
	*DEBUG**	Debug/power-on switch pulse
isanmi	*ISA_NMI*	Interrupt from the ISA bridge, intended to connect to an Intel-compatible CPU's *NMI* "non-maskable" interrupt input.

[12] In a fixed-priority interrupt scheme, two interrupt handlers have the same priority if neither is allowed to pre-empt the other.

Name	*Signal*	*Significance*
pfail	*PWRGD*	5V and 3.3V power OK signal, can create an interrupt when it goes inactive to give advance warning of power loss.
IOperr	*V3_PERR**	Parity error detected by PCI bridge
ide0	*DDIRQ14*	Primary and secondary IDE ports, respectively
ide1	*DDIRQ15*	
eth e_mdint	*ETH_INT**	Ethernet interrupts. *ETH_INT** is the interrupt from DECchip 21143 ethernet controller; *MDINT** is from the 100baseT codec (QS6612 chip). They are separately enabled, and independently visible in the interrupt request register, but are steered to a CPU interrupt input together, by the “eth” field of the steering register “XBAR4”. In practice, we don’t expect anyone to enable *MDINT**; it was only included to allow users to implement a suggested work-round to an obscure QS6612 errata.
e_wake	*ETH_WAKEUP*	Connected to the 21143’s extra-interrupt pin; one function of this is to recognise a particular kind of packet arrival, even when the ethernet chip is dozing. Intended to implement deep power-down modes for server functions in intermittent use. Not available on 21143 parts before the “TD” revision. This signal is a positive-going edge, and the interrupt controller contains latch-and-clear logic.
e_coderr	*QS6612_ERR*	Error indication from the 100baseT codec chip. A positive-going edge; the interrupt controller contains latch-and-clear logic. You probably won’t need this.
isabr	*ISA_INTR*	Interrupt from i82371 ISA bridge
kbd	*LIRQ1*	keyboard/mouse controller
mouse	*MSEIRQ*	separate mouse interrupt from keyboard/mouse controller, if required. The standard controller combines mouse and keyboard interrupts, into the event we’ll call `mkbd`.
com1	*LIRQ4*	serial port 1
com2	*LIRQ3*	serial port 2
flp	*LIRQ6*	floppy disk controller
cent	*LIRQ7*	Centronics/IEEE1284 controller. Note that in some of the simpler modes the signal is a pulse, and the interrupt controller contains logic to latch the signal and clear it down under software control.
rtc	*LIRQ8**	Real-time clock
pci0-3	*PCIIRQ0-3**	PCI slot signals
scsi	*SCSI_INT**	Symbios 53C810 SCSI controller
usb	*USB_INT**	USB controller (part of i82371)
pcibr	*V3IRQ**	V3 V360EPC PCI bridge chip

Table 5.4: Interrupt sources

5.5.3. Register map

Interrupt request/enable registers

Offs (hex)	*register name*	*Dir/ Function*	*Data bits* 7	6	5	4	3	2	1	0
0x0	*LOCINT*	Rd: IRR Wr: IntEn	rtc	0	cent	com2	com1	mkbd	flp	pcibr
0x4	*PANIC*	Rd: IRR Wr: ClearInt	e_coderr	e_wake	0 cent	IOperr	isanmi ×	berr	pfail ×	debug
0x8	*PCIINT*	Rd: IRR Wr: IntEn	pci3	pci2	pci1	pci0	usb	scsi	eth	e_mdint
0xC	*ISAINT*	Rd: IRR Wr: IntEn	×	×	×	×	×	ide1	ide0	isabr
0x24	*KBDINT*	Rd: IRR Wr: ClearInt	×	×	×	×	×	×	mouse	kbd

Interrupt steering registers

0x10	*XBAR0*	Wr: Steer	com1		mkbd	flp	pcibr
0x14	*XBAR1*	Wr: Steer	timer		×	cent	com2
0x18	*XBAR2*	Wr: Steer	pci3		pci2	pci1	pci0
0x1C	*XBAR3*	Wr: Steer	×	KE†	ide1	ide0	isabr
0x20	*XBAR4*	Wr: Steer	usb		scsi	eth	×

† This bit doesn't steer an interrupt, but configures the keyboard interrupt as edge- rather than level-sensitive.

Table 5.5: Interrupt controller registers

Notes on the interrupt request/enable registers

A "1" in any `IRR` register means that the corresponding interrupt signal is active (or in the case of latched signals, an edge has been detected). Some of the interrupt signals are active-low, and some active-high, but the software can't see that; an active interrupt is always a "1" bit.

No interrupt will be passed to the CPU unless the corresponding bit is set "1" in the `IntEn` register. You can't read the `IntEn` registers (the `IRR`s occupy the same IO addresses); so you may need to keep a soft copy of the register value if you want to be able to update the bits independently. The `IntEn` registers are cleared to zero (disabling all interrupts through them) on power-up.

Most of the non-panic device interrupts in P−5064 are "level-triggered" - that is, once the device has raised the interrupt condition some kind of software intervention at the device is necessary to get it to de-assert its interrupt signal.

The first exception is the Centronics interface interrupt which (in the simplest mode) does not behave like this; the data-ready interrupt signal is a pulse. The interrupt controller stores the pulse event, and once you've serviced the Centronics port it needs to be cleared out by writing a "1" to the appropriate bit of the "panic" `ClearInt` register.

The keyboard/mouse controller interrupts also are best handled as edges, and that's an option from logic revision 4 onward. You need to set the `xbar3(KE)` bit to make this happen, and then write to `kbdint` to clear out interrupts once they've been detected. You can still only enable and steer the mouse/keyboard interrupt together.

None of the “panic” interrupts can be masked at the interrupt controller; the CPU interrupt *Int4** will be activated whenever any of these conditions happen. But some interrupts (berr, debug, IOperr, and centronics) occur as pulses and are stored by the interrupt controller. You can clear them out (individually or together) by writing to the `panic` register in its `ClearInt` function - write a “1” to clear the corresponding stored interrupt.

Although the latched interrupts are all forced to the inactive state following a system reset, it’s probably sensible to clear all of them again during software initialisation.

Notes on the interrupt steering registers

Device interrupts may be individually signalled on any of the CPU interrupt lines *Int0-3**; just write a binary value 0-3 to the two-bit field provided in one of the steering registers. At system reset the steering registers are all reset to zero, directing all interrupts towards *Int0**. This is more by accident than design, and is unlikely to be useful.

5.6. Local I/O

There are a whole bunch of simple “dumb” I/O devices on P−5064, to help you wire up a variety of possible peripherals. Support for “IDE” disks and the USB bus are provided by the ISA bridge chip, and are described in §5.10. Some of these peripherals are attached more or less directly to the intermediate bus, and some to the onboard relation of the ISA bus called the “X-bus” on PCs. You may need to be aware of the location of devices sometimes.

5.6.1. Dual Serial port

The dual serial port is one of the functions of a National Semiconductor PC93707 multi-function controller[13]; from a software point of view it’s just like programming two independent 16550 UARTs.

You’ll need to know:

- 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).
- When you wire up a serial port, important signals which you don’t connect are generally pulled up into the least-disruptive state. That makes it easy to communicate with P−5064 along a 3-wire cable, if that’s your choice.
- See Figure 8.2 in the connectors chapter 8 for a list of what signals are supported.

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

5.6.2. Debug board serial port

Some debug ports will carry a serial port; it’s there because the normal serial port (on the ISA bus) requires a great deal of the board’s logic to be functional before it will work. The debug board port is mainly intended for commissioning at Algorithmics, and most debug boards will be built without it. Enquire if you have a need for it.

5.6.3. Centronics

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

Being a Centronics peripheral

The controller was conceived to implement the host-side interface, and not the printer side. But on P−5064 the port also provides a peripheral interface on a second connector (to support functions such as downloading from a PC to the board). Table 5.6 shows how the controller signals are re-deployed when in peripheral mode.

[13] Rev B boards used a Winbond W83787F.

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

Table 5.6: Centronics connections in “peripheral” mode

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

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; one of the general-purpose PIO controller’s outputs, called *B0* in §5.6.6, is used to drive Centronics *nFault.*

There are some particular difficulties experienced when being a peripheral to the old-fashioned “compatible” mode:

- The compatible mode requires 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−5064 provides some external logic which asserts *Busy* after every datum is received, and de-asserts *Busy* again when you program an *nAck* response.

  The auto-busy logic is not required in the P1284 fancy bidirectional modes; but when you’re emulating an old-fashioned printer you can enable this logic by writing a bit in the board configuration register, described in §5.4. (“P−5064-specific hardware registers”) on page 30. When the “auto-busy” logic is enabled, we don’t need a controller output for *Busy* (we’re in peripheral mode so that’s the signal called *nAuto* on the schematics, or *AutoFd* in the controller documentation). Auto-busy is enabled from board reset, on the assumption that the most primitive use of the port is to download software from a PC.
- In compatible mode it’s common to hold data on the bus for only a very short while after the active (negative-going) edge of *nStrobe*, because most printers capture the data through an edge-triggered register. P−5064 has an extra register which captures the Centronics data bus and should be used when receiving in this mode; it’s called “Centronics peripheral data input register” in the memory map table Table 4.1.
- 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 §5.5. (“P−5064 interrupt controller”) on page 33.

ECP/EPP programming is complicated; ask Algorithmics whether they’ve got any driver software which will help you.

5.6.4. 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, somewhat as shown in Figure 5.4.

	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 5.4 Alphanumeric Display Extended Character Set

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

The LED display is attached to the board's intermediate bus, so that it works regardless of the programming of the (complicated) PCI and ISA bus subsystems. That helps it do one of its main jobs, which is to pass information back from the power-on test sequence.

5.6.5. 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 P23 with a short ribbon cable. The connection and adjustments are described in §8.14.

You can no doubt get manufacturer's data sheets, but we found programming information on the world-wide web (see Appendix B, page 77).

5.6.6. Software-configurable general purpose I/O

This part of the functions of the combi I/O chip goes beyond standard PC clone operation, and you'll need to read the datasheet.

Older boards used a separate Z80 family PIO chip, the Z84C2008PEC.

In both cases some of the I/O pins are used for onboard requirements for extra I/O bits, and as a bonus remaining signals are made available on a connector. This should 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.

The connector provides an 8-bit bank of I/O pins, individually controlled as to direction, with various latching options. The header pinout is in Table 8.5 on page 60.

GPIO bits used for onboard functions

Port/Bit	*Signal Name*	*In/Out*	*See section*	*Used for*
GPIO20	nFault	Out	5.6.3	Drives this Centronics peripheral interface signal.
GPIO25 GPIO26 GPIO27	SDA_OE* SCL SDA	Out Out BiDir		*GPIO25-27* are used to read and write the serial EEPROM devices on P−5064's memory DIMM modules.
			App C	This is not done directly because the combi chip has 5V signalling and the SDRAM modules are 3.3V only; the signals are staged through a programmable logic chip, which alters the semantics somewhat.

Table 5.7: Parallel I/O bits and onboard functions

5.6.7. Diskette

This is provided by the combi I/O chip, and emulates the NEC μPD765 device used in PCs since time began. For programming information get the combi I/O chip documentation; see Appendix B, page 77. The floppy port uses DMA service, provided by the i82371 ISA bridge.

5.6.8. Real Time Clock (RTC)

Part of the combi I/O chip (except on rev B boards) the PC-compatible real-time clock remembers the date and time with a resolution of 1 second. It provides a programmable tick and alarm which can cause an interrupt. On Rev B boards there is a *SQW* output, programmed at 32kHz by the firmware, which must not be changed - it is used to generate DRAM refresh timing.

The RTC also provides a small amount (242 bytes) of read/write memory which is retained over power-down; the monitor ROM does not use this space, but for historical reasons some OS ports do - VxWorks, for one.

The RTC chip uses a long-lifetime battery (BT1) to keep time when system power is off. The battery can be replaced when it eventually runs down; buy a 1" lithium "coin" type. Revision B boards used a soldered-in battery, a Varta 3/V60H Ni-MH or equivalent, with a 3-pin footprint.

5.6.9. Keyboard/mouse controller

part of the combi I/O chip, this provides a standard PC interface. Revision B boards used a separate keyboard controller - a standard pre-programmed microcontroller from "American Megatrends". Refer to the sample drivers for a software interface.

5.7. PCI bus

The PCI interface provides three standard slots for expansion cards, as well as hosting the ISA bridge and on-board ethernet and SCSI controllers. PCI runs at 33MHz (irrespective of the processor operating frequency)[14].

The PCI interface is built using a V3 V360EPC[15] device which is designed to convert between an Intel i960's local bus and PCI; for programming information you'll need V3's manual - see Appendix B, on page 76.

Custom interface logic converts all the CPU's non-DRAM bus cycles into a form which is compatible with the V360EPC'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.

5.7.1. 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 in the local DRAM.

5.7.2. 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[16]" where devices are addressed by means of per-device *IDSEL* signals provided by the motherboard hard-wired decoding. In P−5064 PCI IDSELs are obtained from individual PCI *AD* lines, as shown in Table 5.8 below.

Device	*AD line used for IDSEL*	*Interrupts*			
		INTA	*INTB*	*INTC*	*INTD*
PCI slot 1	29	PCIIRQ2	PCIIRQ3	PCIIRQ0	PCIIRQ1
PCI slot 2	28	PCIIRQ3	PCIIRQ0	PCIIRQ1	PCIIRQ2
PCI slot 3	27	PCIIRQ0	PCIIRQ1	PCIIRQ2	PCIIRQ3
ISA bridge	26		-	-	-
SCSI ctrlr	25		-	-	-
Ethernet ctrlr	24		-	-	-

Table 5.8: IDSEL for PCI devices/slots

In all cases the *IDSEL* line is connected to the corresponding *AD* line through a 47Ω resistor. The value on the *AD* bus is mostly "don't care" during configuration cycles, so to direct a configuration cycle at the

[14] It is theoretically possible to configure P−5064 with a half-CPU-rate PCI clock, but that will usually be too fast.

[15] Revision B boards used a V3 V962PBC rev B.2, which sounds different but almost identical in P−5064. The V360EPC adds features, particularly support of the "I2O" standard, which are valuable in some markets but largely irrelevant to P−5064.

[16] 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 defines an additional "Type 1 configuration cycle" which works across bridges. P−5064's PCI controller supports "Type 1" cycles, in an ugly sort of a way; see the V3 manual.

ethernet controller you'd set the PCI address to something like `0x0010.0000` - which would set *AD24* to a "1" and all of *AD25-29* to "0".

You will need to program the PCI configuration space window base address register within the PCI converter device; see the V3 manual for details[17].

PCI devices typically connect to one interrupt line; all P−5064's onboard devices have dedicated interrupt outputs which are handled by the interrupt controller, described in §5.5.

However, the PCI expansion slots provide a choice of four separate interrupt lines to accommodate multi-function boards. By convention, the assignment of motherboard interrupt signals to expansion slot positions is rotated for successive slots so that simple one-function boards (which should always interrupt through the PCI slot signal *IntA#*) will get independent interrupts.

5.7.3. PCI arbitration

Arbitration should be invisible to the programmer, and usually is. But see Table 5.9 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˜	PCI Slot 3
PCIGNT5˜	ISA bridge
V3GNT˜	V360EPC PCI controller

Table 5.9: PCI arbitration signals for devices

Arbitration is round-robin; the priority list gets rotated each time a new master gains the bus, with the last winner always having the lowest priority.

5.7.4. PCI interface registers

Extensively documented in V3's manual. One day we may summarise this here; meanwhile Algorithmics will supply customers with C code examples on request.

5.7.5. PCI startup

From system reset the PCI controller itself is held in reset by the signal *V3RESET˜* until released through the board configuration register; see §5.4.1. Once that reset is released, the PCI bus controller keeps the PCI bus reset is asserted until it is explicitly cleared by writing a V360EPC register.

V360EPC has no PCI-accessible "configuration space" (in PCI language it can't act as a target of configuration cycles) but must be configured by the CPU through 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 §5.4.1).

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

5.7.6. 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−5064?

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.

  Most high-bandwidth PCI peripherals are "bus masters" - they initiate data transfer cycles on PCI and read or write P−5064'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−5064'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−5064'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−5064 runs code out of its boot ROM, each cache line refill turns into 32 separate byte reads, and occupies the local bus for nearly 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.

5.8. Ethernet interface (DEC 21143)

See the manufacturer's hardware manual (hints on getting it in Appendix B on page 76), or software driver examples provided by Algorithmics, for the use of this part.

In P−5064:

- *Ethernet clock*: is defined by a dedicated 25MHz crystal.
- *Ethernet controller reset*: is a programmable signal dedicated to the DEC controller, from the board configuration register, described in §5.4.1.
- *Interface signals*: both the transceiver and twisted pair connections are protected by transformers.
- *PCI connections*: the DEC 21143's *IDSEL* signal is derived from PCI address line *AD24,* and it's interrupt output is dedicated.
- *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 by P−5064 yet, but may appear in a Revision D. Algorithmics' drivers will use the EEPROM if it's there.

5.9. SCSI interface (Symbios 53C810A)

Once again, this is too complicated to describe. You can read the manufacturer's documentation to find out more - leads to Symbios manuals are in Appendix B on page 77, or use the sample drivers. You'll need to know the following board-specific information:

- *Timing clock*: SCSI signal timing relies on the *SCLK* input to the 53C810 chip, which on P−5064 is 48MHz (it borrows the USB subsystem's master clock, which is fixed at this rate).
- *SCSI controller reset*: is a dedicated signal from the board configuration register, described in §5.4.1.
- *PCI connections*: the 53C810's *IDSEL* comes from *AD25*, and it has a dedicated interrupt line into the interrupt controller.
- *SCSI cable termination*: the board has active terminators for the SCSI cable, controlled by a jumper and a software-writable register. The terminators must be enabled if P−5064 is at one end of the cable, and disabled if it's in the middle.

  Most often the SCSI terminators are enabled under software control by a control bit in the board configuration register described in §5.4.1; but if you want the terminators to go on working while P−5064 is powered down or reset, you'll need to move the J15 jumper; see Table 7.1 on page 49.
- *SCSI terminator power*: normally, P−5064 feeds +5V power for remote termination into the SCSI cable; the line is protected by a 1.1A self-resetting fuse and a diode. However, where you know that some other SCSI bus host is providing power, you can disconnect the line completely by removing J16.

5.10. ISA controller, ISA bus and IDE channels

The i82371 ISA bridge component provides:

- One ISA bus socket for those "legacy" cards you can't easily plug in anywhere else.
- A PC-like environment, the "Xbus", which is used to connect several different onboard I/O devices. They include the combi I/O chip which integrates serial ports, centronics, floppy and real-time clock. But there's also a connector for a small alphanumeric LCD which may be useful for prototyping simple "front-panel" applications.

  The PC emulation includes address decoding, DMA provision and maintenance of miscellaneous and semi-mythological signals on the ISA bus. You probably don't want to know about most of these.

- Two IDE disk/peripheral channels. By default these operate as slave-only ports (like an old PC's IDE bus), but they can be reconfigured into high-speed ports which take advantage of DMA over the PCI bus.
- A host port for the emerging USB ("universal serial bus") interconnect standard for low-speed devices.

The part is designed for use in PC-clone systems which implement all I/O through a PCI bridge chip. It gains quite a lot of complexity from the necessity, in these systems, for the hardware to "hide" the existence of the PCI bus from early bootstrap software; such a machine must look like a PC with a directly-connected ISA bus before the bridge chip has been programmed.

You will not often have to reprogram the ISA bridge; access to the I/O registers of devices lying on the onboard PC bus or ISA slot just requires you to add the appropriate base address.

When you need to tackle functions provided by the i82371, remember that the part has to be software-compatible with the old PC devices it supplants (so to do DMA get "PC DMA" driver software). An excellent (but lengthy) manual is available from Intel, online as well as in paper. See Appendix B, page 76 for leads.

Choices made in P−5064's use of the chip include:

- *Clocks*: standard. The *OSC* input gets a 14.318MHz crystal, and *USBCLK* a 48MHz clock.
- *Controller reset*: is run by the signal called *PWROK* (it's inactive low level holds the chip in reset). It is under software control, coming from the board configuration register as shown in Table 5.2. The controller doesn't expect this to be under software control, and you should give it a substantial amount of time to recover its poise after asserting *PWROK*, before expecting it to do anything sensible. 10ms or so seems reasonable.
- *Interrupt output*: wired straight into the main interrupt controller.
- *PCI connection*: the controller's *IDSEL* comes from *AD26*.
- *Dual-use pins*: P−5064's use is very conventional; the dual-use signals *DD12-15* and *LA17-23* are all used just as data/address bits.

5.11. PC card slots (PCMCIA)

A dual PC card (the PC Card standard was formerly known as "PCMCIA") connector is attached via the onboard ISA bus and a Vadem VG469 PC-card controller. P−5064 implements the PC-card features for software control of power to the cards, to allow "hot swapping" - with appropriate software support PC cards can be safely inserted and removed with the system powered up. You'll need to read Vadem's documentation find out how to do this stuff; look in Appendix B on page 76.

5.12. PMON debug monitor compatibility

PMON loads:

- Executable fully-resolved ELF object files from an IP-network *tftp* server, accessible over ethernet. PMON can use a standard domain name server and communicate via a default gateway.

  The ELF files must be MIPS-ABI or compatible version. A symbol table, if present, is read and is available for PMON debugging.
- A variety of "plain text" download formats (including Motorola S-records) loaded via serial port or Centronics link. It's possible to share a single serial port for download and console operation; but it's not very much fun. Serial port download is very slow for large files unless your host's port will run at 36Kbaud or more. PMON on P−5064 will run up to 56Kbaud (perhaps even 115Kbaud - none of our hosts go that fast).

PMON provides a “debug monitor” service to host-based debuggers over ethernet (*MIPS* protocol) or serial port (*gdb* or MIPS protocol).

6. Board layout: locating connectors and jumpers

A diagram of what's where on P−5064 is Figure 6.1.

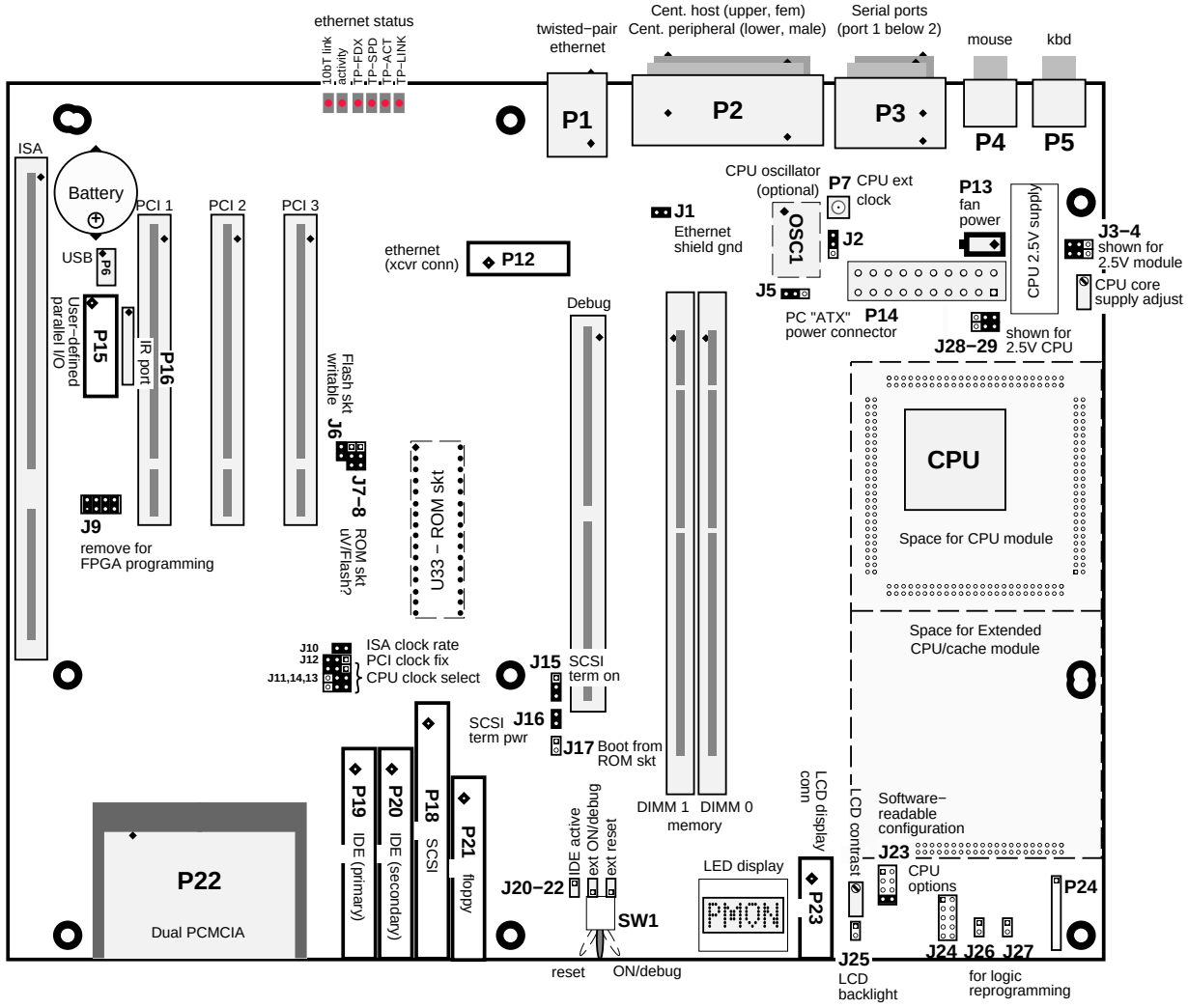


Figure 6.1 P−5064 layout, connectors and jumpers

Notes on Figure 6.1

- *Connector orientation*: pin 1 positions are usually marked with a diamond. For components where all pins are shown, the square pad marks pin 1 (the same convention is used on the PCB itself).
- *Connector pin-outs*: are described in §8 below. But we don't usually document industry-standard connectors.
- *Jumper*: functions and defaults are described in §7 below.
- *Optional components*: such as the CPU oscillator (only required for CPUs running at clock rates beyond the reach of the clock synthesiser), or the CPU daughterboard, are marked with dotted outlines.

- *Adjustable components*: R224 is a multi-turn adjustable resistor used to adjust the contrast of an LCD display, if you've fitted one.

  R85 is used to set the CPU core power-supply voltage for those CPUs which use dual power supplies. It will be factory-set to match the CPU fitted to the board, if required. CPUs can be damaged by incorrect voltages, and it will normally be wise to adjust the CPU core voltage before fitting the CPU daughterboard.

7. Jumpers: where and what for

First let's summarise all the options in Table 7.1.

Ref	*Default*	*Description*
J1	in	Connects ethernet "shield" grounds to board. Normally done, but some systems have a ground on the transceiver unit.
J5	1-2	The CPU usually runs from an onboard clock synthesiser. 100MHz CPUs (above the range of the clock synthesiser IC) need OSC1 fitted and this link set 2-3.
J2	1-2	Move this to 2-3 to inject a master CPU clock through the P7 connector, allowing you to run the CPU at peculiar speeds.
J6	in	Remove to write-protect flash memory fitted in the socket (if any).
J8	2-3	Configure ROM socket for flash. The default position 2-3 is for uV-erasable ROM; change both links to 1-2 for a 29040-compatible flash device.
J7	2-3	
J12	2-3	PCI clock set around 33MHz. 1-2 synchronises PCI to half the CPU clock rate - almost always a really bad idea.
J13	1-2	CPU clock synthesiser rate selection (control inputs called *S2-0* on the synthesiser data sheet). See Table 7.2 for settings.
J14	1-2	
J11	2-3	
J15	2-3	Enable active SCSI terminators at software command. If you set this jumper 1-2 the terminators are always enabled; essential if you want to have the terminators work with the P−5064 powered down.
J16	in	P−5064 supplies SCSI terminator power - remove if some other device on the SCSI cable does that job.
J17	out	Boot from flash. Insert this jumper to boot from socketed ROM.
J27	out	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.
J26	TBA	Insert to select the Centronics "peripheral" connector as the logic reprogramming source. To make this work you'll need to pull the jumpers J9.
J9	in	remove 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.
J20	n/a	Not a jumper; connects optional external IDE activity LED. Pin 1 is +ve.
J21	n/a	Not a jumper; connects optional external "on/debug" switch (make to turn on, or generate a "debug" interrupt.)
J22	n/a	Not a jumper; connects optional external reset switch (make-to-reset)
J24		CPU configuration - 5-way jumper block. Described in 7.2.
J25	out	If in, powers LCD backlight (supplying 5V to connector pin 15). Not all LEDs have backlights.
J10	in	Run ISA clock at the "safe" level of about 8MHz. "Out" increases ISA bus clock to 11MHz or so (PCI/3). That's generally a bad idea - ISA cards are not always reliable above 8MHz.

Ref	*Default*	*Description*
J23		Software-readable configuration - 4-way jumper block. Described in Table 7.4. In revision B boards this is an 8-position jumper block, and the four extra positions are used for the board revision register; in later boards the board revision field, although read through the same location, is hard-wired.
J28		Set 1-2 for CPUs requiring only 3.3V power; set 2-3 for CPUs needing a separate (lower) core power supply. Will be set at the factory to meet the requirements of the onboard CPU.
J29		
J4		Separately configurable core power for CPU daughterboard (if fitted) - introduced in Rev D. Set 1-2 for daughterboards requiring only 3.3V power; set 2-3 for daughterboards needing a separate (lower) core power supply.
J3		
J19		Probably wire links on most boards, these jumpers feed power to the core of the programmable logic devices - may be 5V or 3.3V, depending on the generation and speed grade of those devices.
J18		

Table 7.1: All jumpers on P−5064 (including connectors called Jxx)

7.1. CPU clock synthesiser jumpers: J13, J14, J11, J12

Jumper setting			*CPU Clock*
J13	*J14*	*J11*	*rate*
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 7.2: CPU clock rate setup

Probably fairly self-explanatory. Most P−5064 CPUs will either run at 75 or 83 MHz, or faster from a dedicated oscillator - in which case the setting of these jumpers isn't relevant.

7.2. CPU software and options jumper: J23, J24

The two main jumper blocks for CPU and board options are located at the lower RH side of the board, and their individual jumpers are shown in Figure 7.1.

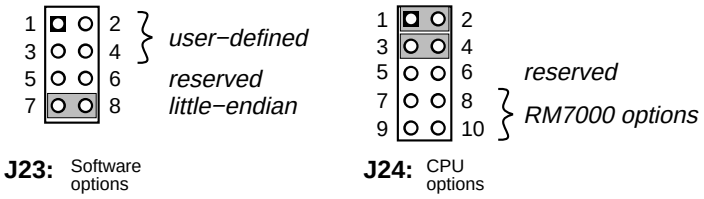


Figure 7.1 CPU options and software options jumper blocks

The CPU options bits work together to tell the onboard logic what type of CPU and secondary cache is fitted; some of these functions are essential, so that a mismatch with the CPU may prevent the board from working.

J24(7-8,9-10) should be fitted when you are using an RM7000 CPU and want the internal secondary cache to work (you probably do); they have the side-effect of enabling a set of different clock multiplier rates supported by the RM7000 - see column of Table 7.3 marked “RM7000/SC”. These jumpers are probably harmless with other CPU types, but play safe - take them out.

Add-in CPU modules for P−5064 (with or without secondary cache) use some of their module pins to send configuration information, interpreted by the logic at startup, which communicates the presence, size and type of an external cache.

The CPU options links are connected directly into the reprogrammable system logic, and their meanings may evolve with later logic revisions. Combinations of settings which don’t exactly match some combination recommended here may do strange things.

Do not change any other links from the state in which they’re delivered to you, without consulting Algorithmics first[18].

J24 jumpers		*CPU clock ratio*		*Notes*
3-4	*1-2*	*RM7000/SC*	*other CPUs*	
in	in	× 2	× 2	
in	out	× 3	× 3	
out	in	× 2.5	× 4	
out	out	× 3.5	× 5	Not available on revision B boards

Table 7.3: Selecting CPU clock divisor

[18] When I wrote these words we envisaged that the CPU module would be married up to a motherboard in production and never removed; the module was intended to be a build-time option. Since then sales pressure has persuaded us to ship modules separately, and some confusion can result. Go carefully and ask our advice if you think you need it.

Notes on CPU options

Some things aren't obvious:

- *CPU type*: in principle, some of the jumpers in the J24 block are there to select for different types of CPU. In fact, P−5064 so far supports one type of onboard CPU (the RM5260) and three plug-in CPUs (R5000, RM5270, RM7000) - and only the RM7000 needs any links; connect 7−8 and 9−10 to take advantage of the RM7000's onchip secondary cache.

  That may change with future versions of the board, so other settings of J24 links 5−6, 7−8 and 9−10 are reserved.

- *CPU Clock multiple*: P−5064 boards are built for at least a 75MHz clock rate, so the ×4 & ×5 clock multiplier options are required only for CPUs running at more than 225MHz.

- *Secondary cache size*: secondary cache size settings have no hardware effect; they are effectively a Silicon Graphics-defined convention for passing configuration information to the bootstrap software which software-configures the CPU. They are also unnecessary, since it's possible to detect the secondary cache and figure out its size with a simple software algorithm.

  However, configuration bits are now defined on the daughterboard and made available to onboard logic when required.

- *Endianness*: is (for historical reasons) defined in the "software jumper" - even though it's a hardware-only option on all 64-bit-bus MIPS CPUs[19].

7.3. Software options jumper J23

This is a 4-way jumper block and 4 hard-wired links (8 jumpers on revision B boards) which are readable by software; just one of the jumpers is also sensed by the hardware at power-up and used to set the CPU's endianness.

A link which is in reads as a "0", but a link which is out reads as "1".

The connections are in Table 7.4.

bit no	7	6	5	4	3	2	1	0
J23 link					7-8	5-6	3-4	1-2
use	board revision				LE	user-defined		

Figure 7.2 Board revision/option links register and J23 settings

Notes on software options and the options link register

This register provides links which can be read by software at the "option links register" position, but is also the place where the CPU is configured as either big- or little-endian.

Here are what the fields shown in Table 7.4 do:

- *board revision*: will return a value which reflects the artwork and hardware revision level of the P−5064. The value is hard-wired into the board when it is assembled (except on revision B boards, which had a larger jumper block with extra links you should not move). It is reported by the PMON startup sequence, which interprets it as a "board revision" letter. Production P−5064 revisions start with "B".

[19] It's a feature of the R4000/R5000 system interface that it isn't possible to run even a limited subset of instructions without the CPU and the system interface agreeing on the CPU's endianness - so it's not possible to defer the endianness option for software configuration.

- *LE*: endianness - in for little-endian, out for big-endian. For CPUs requiring reset-time endianness configuration (most of them) this determines both how the CPU is brought up, and how the CPU interface interprets CPU signals. You can read this in the register, of course, but it will often be more useful to read the endianness bit in the CPU's internal configuration register.
- *user-defined*: free for your application to use.

8. Connectors: where, what and wiring

8.1. CPU daughterboard connector

CPU daughter cards plug in to a connector made up of five banks of 25×2 2mm pitch socket strip. Four of the banks are set in a square around the CPU, and form a suitable connector for CPU-only daughterboards, with the fifth some distance away adding mechanical support, extra power and ground and extra signals to support a secondary cache. The physical layout of the connector, and where each pin number is located, is shown in Figure 8.1.

We know the pin numbering is peculiar; don't worry about it, it's just an artefact of the way our CAE system numbers connectors.

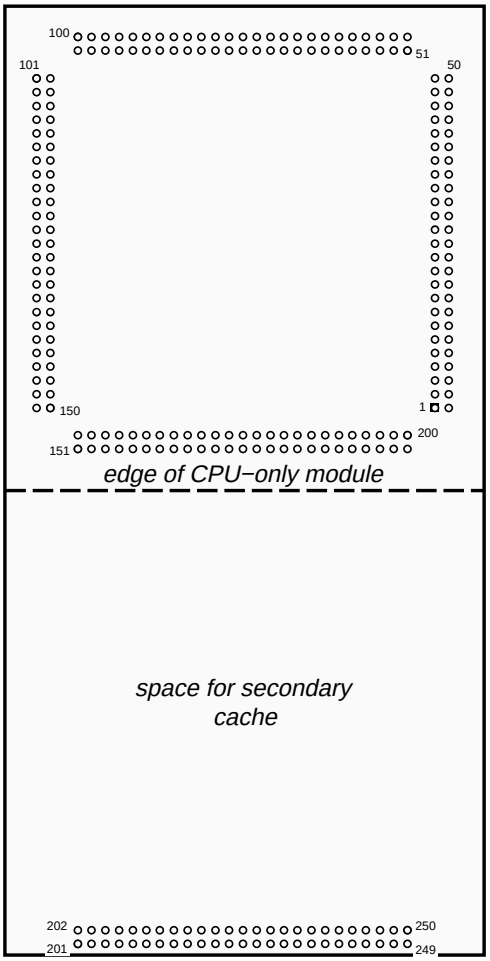


Figure 8.1 CPU daughterboard module connector layout

The signals on the CPU connector are laid out as follows:

Pin	*Signal*	*Pin*	*Signal*	*Pin*	*Signal*	*Pin*	*Signal*	*Pin*	*Signal*
1	*3.3V*	2	*VCORE*	3	*VCORE*	4	*3.3V*	5	*GND*
6	*SysAD4*	7	*SysAD36*	8	*SysAD5*	9	*SysAD37*	10	*3.3V*
11	*GND*	12	*SysAD6*	13	*SysAD38*	14	*3.3V*	15	*GND*
16	*SysAD7*	17	*SysAD39*	18	*SysAD8*	19	*SysAD40*	20	*3.3V*

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
21	*GND*	22	*SysAD9*	23	*SysAD41*	24	*3.3V*	25	*GND*
26	*SysAD10*	27	*SysAD42*	28	*SysAD11*	29	*SysAD43*	30	*3.3V*
31	*GND*	32	*SysAD12*	33	*SysAD44*	34	*3.3V*	35	*GND*
36	*SysAD13*	37	*SysAD45*	38	*SysAD14*	39	*SysAD46*	40	*3.3V*
41	*GND*	42	*SysAD15*	43	*SysAD47*	44	*3.3V*	45	*GND*
46	*ModeClk*	47	*JTDO*	48	*JTDI*	49	*JTCK*	50	*JTMS*
51	*3.3V*	52	*GND*	53	*VCORE*	54	*VCORE*	55	*VCORE*
56	*3.3V*	57	*GND*	58	*ModeIn*	59	*RdRdy**	60	*WrRdy**
61	*ValidIn**	62	*ValidOut**	63	*Release**	64		65	*ModPres**
66	*ModClk0*	67	*3.3V*	68	*GND*	69	*3.3V*	70	*GND*
71	*3.3V*	72	*GND*	73	*SysCmd0*	74	*SysCmd1*	75	*SysCmd2*
76	*SysCmd3*	77	*3.3V*	78	*GND*	79	*SysCmd4*	80	*SysCmd5*
81	*3.3V*	82	*GND*	83	*SysCmd6*	84	*SysCmd7*	85	*SysCmd8*
86	*SysCmdP*	87	*3.3V*	88	*GND*	89	*3.3V*	90	*GND*
91	*3.3V*	92	*GND*	93	*Int0**	94	*Int1**	95	*Int2**
96	*Int3**	97	*Int4**	98	*Int5**	99	*3.3V*	100	*GND*
101	*VCORE*	102	*VCORE*	103	*VCORE*	104	*VCORE*	105	*3.3V*
106	*NMI**	107	*ExtRqst**	108	*Reset**	109	*ColdReset**	110	*VDDOK*
111	*BigEndian*	112	*3.3V*	113	*GND*	114	*SysAD16*	115	*SysAD48*
116	*3.3V*	117	*GND*	118	*SysAD17*	119	*SysAD49*	120	*SysAD18*
121	*SysAD50*	122	*3.3V*	123	*GND*	124	*SysAD19*	125	*SysAD51*
126	*3.3V*	127	*GND*	128	*SysAD20*	129	*SysAD52*	130	*SysAD21*
131	*SysAD53*	132	*3.3V*	133	*GND*	134	*SysAD22*	135	*SysAD54*
136	*3.3V*	137	*GND*	138	*SysAD23*	139	*SysAD55*	140	*SysAD24*
141	*SysAD56*	142	*3.3V*	143	*GND*	144	*SysAD25*	145	*SysAD57*
146	*3.3V*	147	*GND*	148	*SysAD26*	149	*SysAD58*	150	*SysAD27*
151	*SysAD59*	152	*3.3V*	153	*GND*	154	*VCORE*	155	*VCORE*
156	*GND*	157	*VCORE*	158	*VCORE*	159	*VCORE*	160	*VCORE*
161	*3.3V*	162	*GND*	163	*SysAD28*	164	*SysAD60*	165	*SysAD29*
166	*SysAD61*	167	*3.3V*	168	*GND*	169	*SysAD30*	170	*SysAD62*
171	*3.3V*	172	*GND*	173	*SysAD31*	174	*SysAD63*	175	*SysADC2*
176	*SysADC6*	177	*3.3V*	178	*GND*	179	*SysADC3*	180	*SysADC7*
181	*3.3V*	182	*GND*	183	*SysADC0*	184	*SysADC4*	185	*3.3V*
186	*GND*	187	*SysADC1*	188	*SysADC5*	189	*SysAD0*	190	*SysAD32*
191	*3.3V*	192	*GND*	193	*SysAD1*	194	*SysAD33*	195	*3.3V*
196	*GND*	197	*SysAD2*	198	*SysAD34*	199	*SysAD3*	200	*SysAD35*
201	*PRqst**	202	*PAck**	203	*RspSwap**	204	*RdType*	205	
206		207		208		209	*TcDOE**	210	*TcTCE**
211	*TcMatch*	212	*TcWord0*	213	*TcWord1*	214	*TcWord2*	215	*3.3V*
216	*GND*	217	*3.3V*	218	*GND*	219	*3.3V*	220	*GND*
221	*SC_DCD**	222	*SC_SIZE1*	223	*SC_SIZE0*	224	*ModClk1*	225	*SC_REVB**
226		227		228		229		230	*3.3V*
231	*GND*	232		233		234		235	
236		237		238		239		240	*3.3V*
241	*GND*	242		243		244		245	
246		247		248		249	*3.3V*	250	*GND*

Table 8.1: Signal assignments on the CPU daughterboard connector

The various signals have the following meanings:

Signal Name	*Description*
3.3V	Fixed 3.3V power supply
BigEndian	Some CPUs have a dedicated signal to configure CPU endianness
*ColdReset**	Used to cycle the CPU through a from-scratch reset, including resynchronising phased-locked clocks and reading the configuration bit stream.
*ExtRqst**	System logic can use this signal to request the *SysAD* bus from the CPU; not used on P−5064.
GND	Board ground
*Int5-0**	MIPS interrupt inputs. P−5064, and some CPUs, don't implement *Int5**; even if available, it's most often unused because the corresponding interrupt is dedicated to the internal timer.
JTCK *JTDI* *JTDO* *JTMS*	JTAG signals for CPU and/or cache RAM boundary scan. Only for testing.
*ModPres**	Pulled up on motherboard, grounded on module; used by motherboard logic to sense that a daughterboard has been fitted.
ModClk1-0	CPU input clock *MasterClock*, duplicated to manage loading in the secondary cache.
ModeClk	Output clock from CPUs which need a configuration bitstream from cold reset
ModeIn	Data bitstream in response to *ModeClk* above.
*NMI**	MIPS CPU non-maskable interrupt, unused in P−5064
*PAck** *PRqst**	RM7000-specific signals for extended bus protocols. See CPU manual for definitions.
*RdRdy**	Signals MIPS CPU that the system logic can hold a read address, and the CPU can move on.
RdType	RM7000-specific signal unsupported on P−5064 up to rev C; see CPU manual for details.
*Release**	Pulsed by CPU when it expects the system logic (or secondary cache) to drive the *SysAD* bus.
*Reset**	CPU logic reset, which can be used independently of *ColdReset** to reset the CPU but retain configuration. But on P−5064, the two resets are always cycled together
*RspSwap**	RM7000-specific signal unsupported on P−5064 up to rev C; see CPU manual for details.
*SC_DCD**	Configuration bit - low if daughterboard uses "double-cycle-deselect" (R5000-special) cache RAMs. Most boards use "single-cycle-deselect" (Pentium-compatible) RAMs.
*SC_REVB**	Configuration bit - low if daughterboard provides valid *SC_SIZE1-0* signals.
SC_SIZE1-0	Encode size of external cache fitted to this board. Often unnecessary.
SysAD63-0 *SysADC7-0*	64-bit multiplexed bus, used on all 64-bit MIPS CPUs so far. *SysADC7-0* are check bits, and usually carry per-byte parity on P−5064.
SysCmd8-0 *SysCmdP*	Encoded cycle type, used on all 64-bit MIPS CPUs. *SysCmdP* carries parity on the command bus, but is a don't care on input for most CPUs.
*TcDOE** *TcMatch* *TcTCE** *TcWord2-0*	Signals used for an R5000-style secondary cache; a similar secondary cache is used on the QED RM5270 and RM7000 CPUs
VCCOK	Signal telling MIPS CPU that power is stable and it can come out of reset.
VCORE	Adjustable low-voltage "core" supply for CPUs which need it. Will probably usually be set at 2.5V or 2.9V.
*ValidIn**	MIPS R4x00 output meaning "I've put something on the bus in this clock cycle".

Signal Name	*Description*
*ValidOut**	MIPS R4x00 input asking the CPU to take note of the bus this cycle
*WrRdy**	Signals MIPS CPU that the system logic can accept a write address, and the CPU can move straight on to present the data

Table 8.2: Description of CPU daughterboard signals

8.2. DIMM memory slots (DIMM0/DIMM1)

These are pretty much industry standard, so we won't define them here. Note that the DIMM memories are unbuffered synchronous 3.3V 72-bit (ECC) types.

8.3. PCI edge connectors (P8/P9)

Industry standard connectors, not defined here.

However, you will need to know how the *IDSEL* and interrupt lines are assigned; see Table 5.8 on page 41.

8.4. Ethernet (P12, P1)

The 10Mbit/s ethernet can be connected either with the 10baseT connector P1 or by connecting a standard transceiver (directly or indirectly) to the transition cable supplied, which mates with the on-board header connector P12. The onboard connector is an 8×2 0.1" pin grid, polarised to match our cable.

If you need to replace your transition cable, the pinout of P12 is shown in Table 8.3.

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	-
-	15	16	-

Table 8.3: Pinout of ethernet connector P12

where:

- *OPTGND* signals must be grounded at only one end of the transceiver interface, normally at the host end. But if you meet a transceiver which has local grounds on these lines they can be disconnected from board ground by removing jumper J1 (called "enet conn gnd" on the board layout diagram Figure 6.1.)
- The *E12V* line, which provides 12V power to a transceiver, 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 mis-connection or extreme electrical noise.

8.5. SCSI (P18)

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 8.4.

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 8.4: SCSI connector (P18) pinout

Note that the *TermPower* is a 5V supply protected by a fuse and diode, connected to P−5064's 5V supply through the jumper J16. There is normally one "host" on a SCSI bus which supplies terminator power, but the jumper should be removed if any other SCSI bus device is driving the power line - which is likely to be the case when P−5064 is acting as a SCSI peripheral.

P−5064 has an active terminator unit - active termination is highly recommended when running SCSI reasonably fast. When P−5064 is not at one end of the SCSI cable, the terminators can be effectively taken out of circuit by moving the "terminator enable" jumper J15 from its default position.

8.6. IDE

Primary and secondary IDE channels, connected up to the i82371 controller.

8.7. RS232 (P3)

Dual serial ports implemented with a double connector. Port 1 is the lower (nearest the board) and port 2 the higher; both are standard PC-compatible 9-pin male D-type, and the pinout is shown in Figure 8.2.

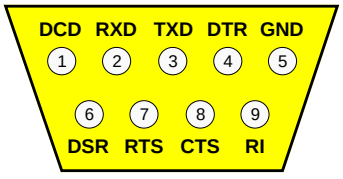


Figure 8.2 Pinout of a PC-compatible serial connector (looking into pins)

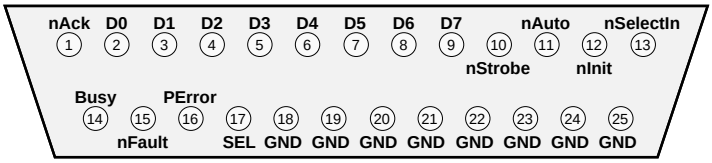
Notes on the serial port signals:

Signal	*Description*
RXD, TXD	asynchronous serial data into and out from P−5064, respectively. In many cases, you need only connect these and ground to have a working interface.
CTS	“clear to send”: input which can be used for flow control, stopping P−5064 from sending data if inactive - whether this is actually done is down to software. We pull it up, so that when you don’t make a connection to this pin it will appear active.
DSR	“data set ready”: signal into the board, sometimes used for flow control instead of *CTS*.
DTR	“data terminal ready”: programmable output - usually wired to *DSR* at the other end.
RTS	“request to send”: programmable output, usually wired to *CTS* at the other end.
DCD	“data carrier detect”: used by a modem to indicate that it has an active connection. Rarely needed when a modem not fitted.
RI	“ring indicator”: input activated by modem when the connected phone rings. Rarely used for anything.

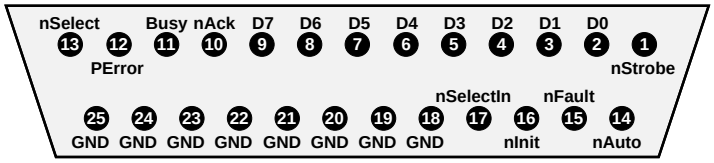
8.8. Centronics (P2)

A double-stacked connector offering either a “host” connector (same as you’ll find on your PC) or a “peripheral” connector (suitable for connecting up to a PC for download). There’s only one port, so don’t try to use both at once!

Unfortunately, on many boards (rev C as well as rev B) the Centronics interface is wrongly wired; the shorter row of signals being reversed[20]. Ask Algorithmics for a correction cable. Rev D and higher boards are correctly wired, as shown in Figure 8.3.



Centronics peripheral (male)



Centronics host (female)

Figure 8.3 Centronics/IEEE-1284 parallel port connector

[20] Surprisingly, this doesn’t stop it working as a dumb peripheral.

8.9. Diskette (P21)

Standard PC-type diskette header; not described here.

8.10. User-defined parallel I/O (P15)

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

- Polling an external logic level.
- Driving some simple external device.
- Software-controlled trigger for test equipment...

Anything you like. The pinout is in Table 8.5.

	1	2	
	3	4	
PA0	5	6	*PA4*
PA1	7	8	*PA5*
PA2	9	10	*PA6*
PA3	11	12	*PA7*
GND	13	14	*+5V*
GND	15	16	*+5V*

Table 8.5: General purpose I/O connector (P15) pinout

The signals *PA0-7* correspond to those described as *GPIO10-17* in the PC97307 combi I/O chip documentation.

8.11. PC-compatible keyboard and mouse (P5/P4)

Both use small (“PS/2” type) DIN connectors[21]. If your keyboard has a big DIN connector as used in older PCs, converters are readily available.

8.12. USB (P11)

A host port for “universal serial bus” implemented by the i82371 ISA bridge should permit the attachment of USB peripherals, just emerging as the first P−5064 boards ship. Ask Algorithmics about software support. The connector is a 2×4 pin 0.1” pin grid, with signals as shown in Figure 8.4.

+5V	1	2	+5V
USBP0˜	3	4	USBP1˜
USBP0	5	6	USBP1
GND	7	8	GND

Figure 8.4 USB (P11) connector signals

[21] Revision B boards (serial numbers up to 20, probably) have the signals on these connectors reversed; they will be supplied with some patch leads. Sorry.

8.13. IR “network” interface (P16)

We’re now really getting down into the far end of feasible. The combi I/O controller can recycle the signals from one of the serial ports into an IR connector standard used by some hand-held computers. This is for experimentation only, but P16 is a 5-pin 0.1” SIL header with pin 2 missing for orientation. It’s connections are as follows:

Pin	*Signal*
1	VCC
3	IRRX
4	GND
5	IRTX

8.14. LCD display connector (P23)

This simple I/O port on an 8×2 header is designed to attach an LCD alphanumeric display on a short ribbon cable. You may even be able to use it for something else, with ingenuity. It’s pinout is in Table 8.6.

GND	1	2	*+5V*
LCD_BRIGHT	3	4	*LCD_A2*
*LCD_W**	5	6	*LCD_E*
LCD_D0	7	8	*LCD_D1*
LCD_D2	9	10	*LCD_D3*
LCD_D4	11	12	*LCD_D5*
LCD_D6	13	14	*LCD_D7*
BackLight	15	16	

Table 8.6: LCD display header (P23) pinout

Note that in Table 8.6:

- The 8-bit IO data bus *LCD_D0-7* is buffered to protect P−5064 circuitry and help drive the LCD cable.
- The brightness of the LCD illumination is controlled by *LCD_BRIGHT,* which can be varied between 0 and 5V by adjusting R224.
- Signal *LCD_W** is a direction signal (low for write); and *LCD_E* is a kind of combined chip select and transfer strobe. *LCD_A2* is the least-significant register address (8-bit IO registers on P−5064 are always at least 4 bytes apart), supporting a princely two registers. *BackLight* can be connected to the +5V supply through the jumper J25, if your LCD display has this feature.

8.15. Power supply connector (P14)

Compatible with PC motherboards built to the “ATX” standard; we used this because it’s the most available standard which features 3.3V and 5V power. You should find it easy enough to come by an appropriate power supply, but if you can’t Figure 8.5 shows the pinout.

+5V	*+5V*	*-5V*	*GND*	*GND*	*GND*	*On**	*GND*	*-12V*	*+3.3V*
20	19	18	17	16	15	14	13	12	11
10	9	8	7	6	5	4	3	2	1
+12V	*VStdBy*	*PwrGd*	*GND*	*+5V*	*GND*	*+5V*	*GND*	*+3.3V*	*+3.3V*

Figure 8.5 ATX power supply connector pins

In Figure 8.5 the signals are as follows:

- *+5V, +3.3V, -5V, +12V, -12V, GND*: power rails. “ATX” supplies provide lots of +5V and +3.3V, a decent amount of +12V (used for PC disc drives) and just a little -12V and -5V.
- *VStdBy*: ATX supplies are software-switchable. When the PSU is off the main supply rails are all disconnected, but the *VStdBy* provides a small amount of +5V power to feed some power-up circuitry. On P−5064 that just allows the debug/reset switch to be pulled to the debug position to switch on the power.
- *On**: enables the main power rails when it’s taken low and draws some current. You switch off the PSU by taking this signal high.
- *PwrGd*: is returned high by the power supply when all rails have switched on and are stable, and goes low to provide early warning of a power failure. *PwrGd* is fed into P−5064’s reset circuitry, and its low-going transition can be used to generate an interrupt.

8.16. Logic programming connector (P24)

P−5064’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 J27 must be fitted. To download from the Centronics connector you should also fit jumper J26 and remove the jumpers J9.

The Xilinx-compatible connector is P24, and is shown in Figure 8.6. It matches Xilinx’ supplied module, so the signals connect across one to one.

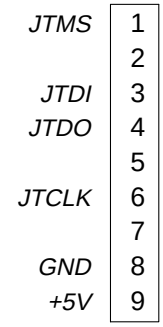


Figure 8.6 Xilinx-compatible connector (P24) for reprogramming P−5064 logic

8.17. 12V fan power (P13)

A socket providing a fused +12V supply (and ground) to connect a PC-type fanned heatsink. Pin 1 is *+12V* and pin 2 is *GND.* Some low-power CPUs fitted to P−5064 do not need a fan when run open at room temperature with no forced air movement. If you have to fit one, use any fan designed for “Pentium” class CPUs in PC clones.

9. Cables supplied

With your P−5064 you should have received a basic set of cables. Everything else you should need to get your system up and running should be readily available from your local PC superstore.

Here's the cables you'll find in the box:

- *Ethernet AUI*: converts the onboard (dual header strip) connector to the usual 15-way D-type, which will plug right into a cheap and cheerful transceiver or mate with a longer AUI cable for attachment to your building network.

  Of course, if you were using 10baseT ethernet you could plug right in and ignore this cable.
- *IDE*: two ribbon cables supplied to attach drives, though perhaps they're rather short for peripherals not in the same box.
- *SCSI* : one ribbon cable supporting two peripherals.
- *Floppy*: one cable with connectors for two drives.
- *Centronics download*: we supply a "straight-through" cable with a 25-way D-type male on one end (plugs into your PC), and a female on the other (plugs into the "centronics peripheral" connector on P−5064). These cables are sold by PC suppliers as "printer cable extenders".

10. Hardware debug and trace facilities

Since P−5064 is principally meant as a development aid, debug assistance is important. Most software development hours go into high-level functions where software tools are all-important, so P−5064 provides the communication channels those software tools need.

But you may also need to prototype and debug software on a level where software monitors can't reach; and then you'll need to be able to observe and interrupt the execution of your program using tools which don't depend on being able to run a program on the main CPU.

At that point you'll be reaching for a logic analyser; the debug board DBG−5 helps you plug it in; and most of this chapter describes it and how it's used. But we've also got notes in about the humble debug switch (which sends an interrupt to the CPU which can be used by debug monitor software) and the socket which allows you to use a ROM emulator.

10.1. The debug board

DBG−5 is a somewhat inelegant board which plugs into the edge connector behind the DIMM memory slots. The edge connector is like those used to hold plug-in cache modules on PCs, and its 160 pins are enough to connect P−5064's wide buses. DBG−5 provides you with:

- *Logic analyser connections*: present all CPU cycles and PCI cycles accessing local memory. DBG−5 buffers and re-registers the buses, so that even a relatively slow analyser will have no trouble following P−5064's buses up to 100MHz. The connectors are pinned out to allow HP logic analyser "mass terminator pods" to be plugged straight in, but can be wired pin-by-pin to any kind of analyser.
- *Address/data trigger PLD*: DBG−5 is fitted with a programmable logic device (a PAL or GAL22V10) whose inputs are bus cycle status signals, and generates trigger signals which will capture address or data. The PAL is socketed, and may be replaced with a custom version.
- *Trigger input*: takes a standard BNC trigger line from your test equipment, and feeds it as the *HPTRIG* signal into P−5064 in parallel with the usual "debug switch" signal. That allows you to break into your software debug monitor when a significant hardware even occurs.
- *Optional dual serial port*: potentially useful to a customer who wants to reprogram the PCI/ISA system in a way which temporarily or permanently obstructs access to the serial ports.

The layout of DBG−5 is shown in Figure 10.1. It's fitted with the connectors facing the CPU; but the connector is keyed and non-reversible.

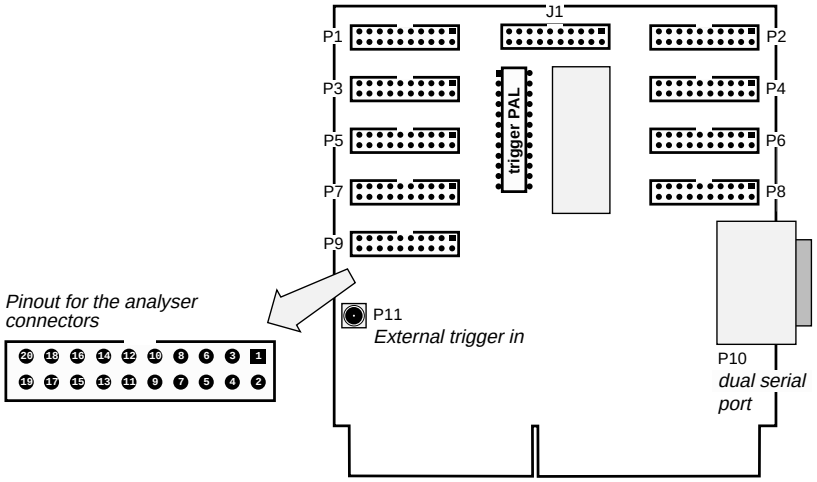


Figure 10.1 DBG−5 layout and connector positions

10.1.1. Debug board analyser connectors

The signals available on the standard analyser connectors of DBG−5 are summarised in Table 10.1. Some more signals (rarely of use to anyone who is not messing with P−5064's internals) are available on the J1 connector and listed in §10.1.3 below.

Signal	*Description*
A0-31	Addresses on the intermediate bus (where there are only 32 address bits). For CPU cycles, you'll see the low 32 bits of the CPU's address; PCI cycles' addresses get mapped through the PCI controllers "aperture" base registers. Low address bits during bigger-than-byte or block transfers may look like anything; ignore them.
ATrig	One rising edge per cycle; captures valid address and data (one data item only if the transfer is a burst. Generated by the debug board trigger PLD.
BE0-7*	Byte enables - *BE0** indicates valid data on *D0-7* etc.
Block*	Active (low) if this cycle is a burst.
D0-63 DP0-7	Data Parity for each data byte lane
GND	System ground pin
IOGnt*	Inactive for CPU-initiated cycles; active for cycles initiated by PCI master devices (or the PCI bridge's DMA channel, if you're brave enough to use it).
Rd*	Low for read, high for write.
RdRdy* Rlse* SysCmd0-8 ValIn* ValOut* WrRdy*	MIPS CPU signals, if you need to watch the system interface. These signals track the CPU at all time, so may well change on every CPU clock edge. They run one register stage behind the CPU.

Table 10.1: Signals from DBG−5

If you are using an HP or compatible analyser with mass terminator probes which plug right into the connectors P1-9, then the signals get laid out for your convenience as shown in Figure 10.2.

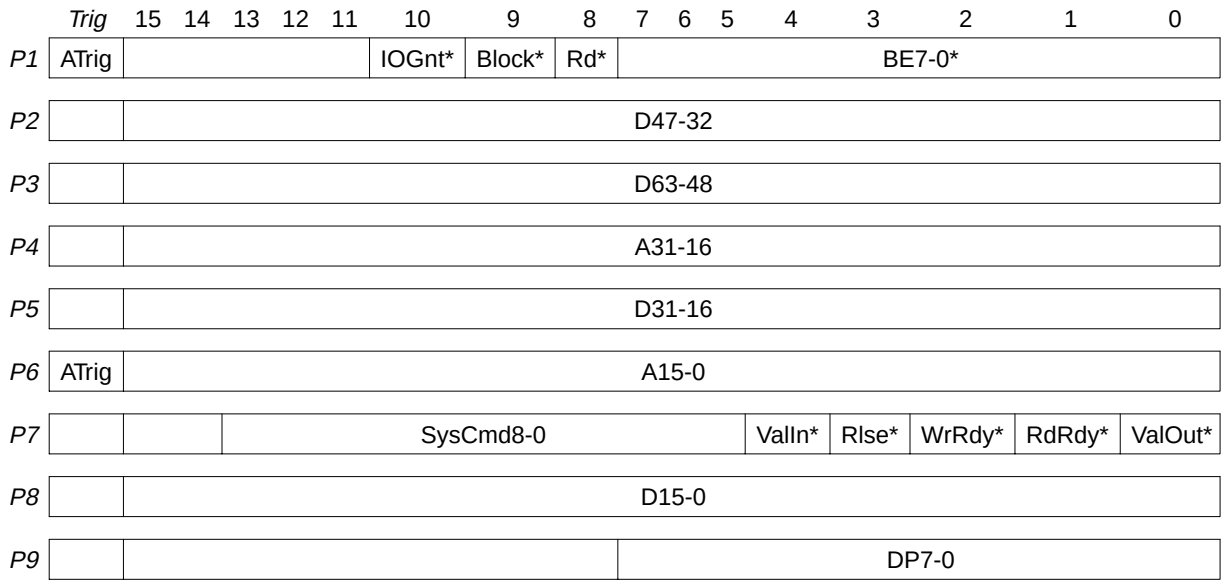


Figure 10.2 Connecting an HP or compatible analyser to DBG−5

Those of you who haven't got an HP or compatible analyser can still use the connectors, of course; but you'll probably find Figure 10.3 more helpful.

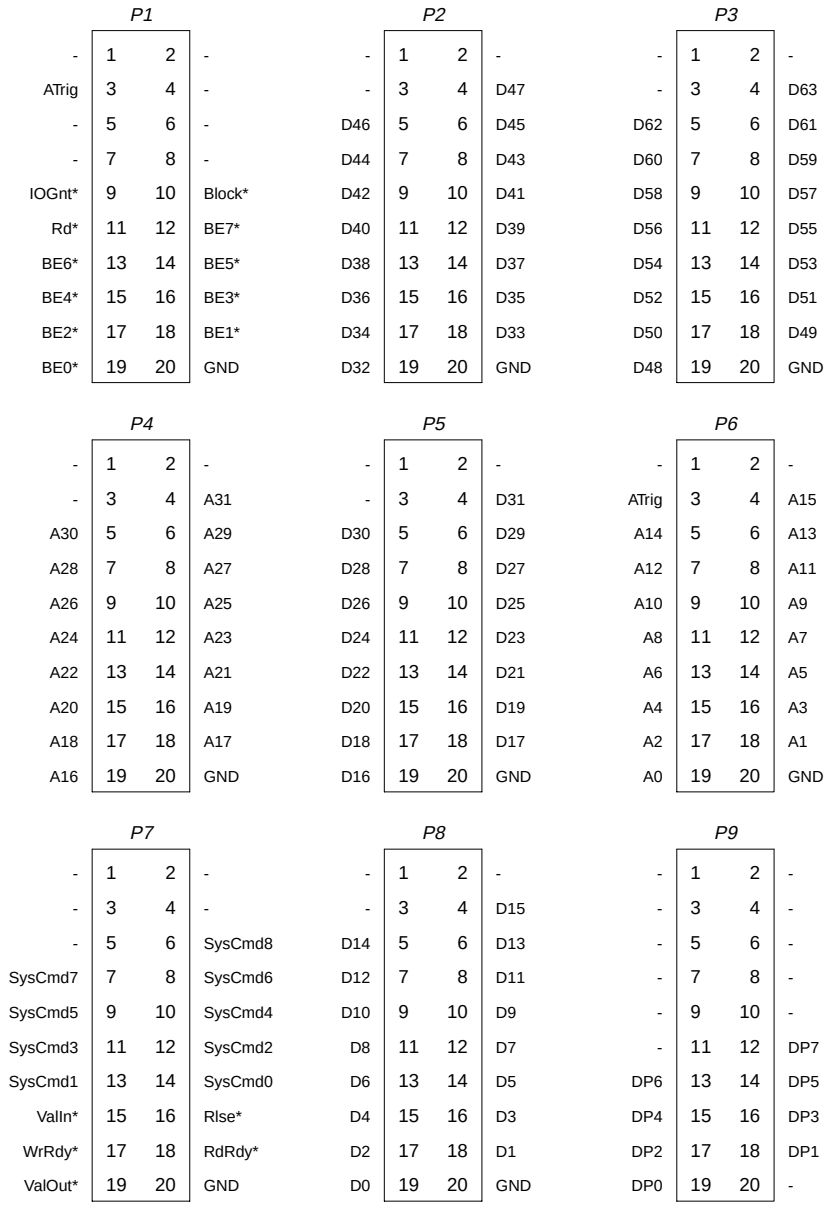


Figure 10.3 Pin-by-pin analyser connection to DBG−5

10.1.2. The trigger PAL; connections and standard version

DBG−5's trigger PAL is an electrically reprogrammable 22V10, 24-pin 0.3" DIP shape, 7.5ns speed grade. We usually use a Lattice GAL22V10, but any 22V10 will do if it's fast enough. Table 10.2 list the signals connected in and out of the PAL; and then we'll show you the logic equations. If you want to see those signals, they're also available on the J1 connector, whose pinout is Table 10.3.

Signal Name		*Description*
Schems	*Equations*	
Inputs		
	dbgclk	The CPU bus interface clock. The CPU/memory region of P−5064 all runs synchronously to this clock.
	mstrrq*	Cycle starting.
Rd*	mstrrd*	Low for a read, high for a write
Block*	mstrblock*	Low for a block transfer, high for just one word
	mstrlast*	Low for the last word of a block transfer, or the data of a single-word transfer
	mstrdval*	Data valid on write
	slvdval*	data valid on read
	slvdone*	slave has finished the transaction
IOGnt*	iomgnt*	when active, this cycle is being done by an I/O master - that is, it's a memory cycle being run by a PCI initiator. All other cycles are CPU-initiated.
	ioclk	half-rate clock. Usually the same as *v3lclk*.
	v3lclk	Clock defining events signalled to/from the PCI interface chip. Usually synchronised to half the CPU interface clock; the PAL uses it to qualify half-speed signals to/from the PCI chip.
	dbgrst	Logic reset from motherboard reset circuit.
HPTRIG		That's the signal from the external trigger connector.
Outputs		
	addrclk	Local version of *dbgclk*, buffered and somewhat delayed. That means the debug board registers sample a few ns later than the motherboard, loosening up timing requirements. Note that this only works because the debug board isn't trying to send any synchronous signals back to the motherboard.
	dbg_aclken*	Clock enable for debug board address and data registers,
	dbg_dclken*	respectively. Active (low) to sample data at the next clock edge; inactive to hold.
ATrig		Trigger for user's analyser. In the standard PAL, this is set to capture addresses of both CPU cycles and PCI-initiated memory cycles.
DBGINT*		Interrupt to CPU, can be used to feed back a hardware trigger condition so that the software stops too.

Table 10.2: Signals on the ANTRIG PAL

Probably the easiest thing is to show you the "Abel" source of the PAL we currently fit.

```
module ANTRIG
" (c) 1992,1993,1994 Algorithmics Ltd
title ’CPU boot time mode bit generator’
antrig device ’P22V10’;
"*** Pin Definitions ***
" Inputs
dbgclk pin 1;
!mstrrq pin 2;
!mstrrd pin 3;
!mstrblock pin 4;
!mstrlast pin 5;
!mstrdval pin 6;
!slvdval pin 7;
!slvdone pin 8;
!iom_gnt pin 9;
ioclk pin 10;
v3lclk pin 11;
dbgrst pin 13;
hptrig pin 23;
" Outputs
addrclk pin 20;
!dbg_aclken pin 19;
!dbg_dclken pin 18;
atrig pin 17;
!dbgint pin 14;
" Internal
as0 pin 22;
mreq_off pin 15;
"*** Macros ***
x = .X.;
astate = [dbg_aclken, dbg_dclken, atrig, as0];
IDLE = [0,0,0,0];
STRT = [1,1,0,0];
AHLD = [0,1,0,0];
DHLD = [0,0,0,1];
TRIG0 = [0,0,1,0];
TRIG1 = [0,0,1,1];
```

```
Equations
addrclk = dbgclk;
mreq_off :=
   (astate != IDLE) & (
      !mstrrq #
      mreq_off
   );
astate :=
   !dbgrst & (
      (astate == IDLE) & (
         mstrrq & (!iom_gnt # iom_gnt & !v3lclk) & STRT #
         !mstrrq & IDLE
      ) #
      (astate == STRT) & (
         !iom_gnt & (
            mstrlast & slvdone & AHLD #
            !(mstrlast & slvdone) & STRT
         ) #
         iom_gnt & (
            slvdone & AHLD #
            !slvdone & STRT
         )
      ) #
      (astate == AHLD) & (
         DHLD
      ) #
      (astate == DHLD) & (
         TRIG0
      ) #
      (astate == TRIG0) & (
         TRIG1
      ) #
      (astate == TRIG1) & (
         !mstrrq & IDLE #
         mstrrq & (
            mreq_off & IDLE #
            !mreq_off & TRIG1
         )
      )
   );
end ANTRIG
```

Can you be expected to understand all that? Let's walk you through it:

- *Signal names*: we write Abel with all-lower-case names. Active low signals, which are usually called *XX** when we're referring to the schematics, are defined with a leading "!" (exclamation mark) in Abel.
- *Macros and vectors*: the assignment ("=") operators appearing before the magic line "equations" are macro definitions which are interpreted by simple textual substitution.

  In Abel lists of signal names or levels enclosed in brackets "[]" are vectors. When you use a vector in a logic equation, you are implicitly defining a bitwise operation across all bits in the vector at once. If the operator is a logical and ("&") or logical or ("#") the result is a vector too; if the operator is something like a comparision ("==") then the result is true/false.
- *= and :=*: in Abel the ":=" assignment implies that the computed logic result is captured in a register (in this type of device, always sampled on the rising edge of the clock fed into pin 1).

- *astate :=* : this equation actually defines a finite state machine. It's a vector (the state vector) assigned through a register; and each of the main terms describes transitions from a particular state. Abel has a syntax specially defined for finite state machines, but it's very inefficient so we don't use it.

The "astate" state machine tracks cycles on the intermediate bus, and particular transitions of "astate" mark points at which we can usefully sample address and data into the registers, and other points at which we can trigger the attached logic analyser. The encoding of "astate" is chosen so that the constituent output signals control the external logic as they should.

10.1.3. The J1 connector: looking inside P−5064

-	1	2	-
-	3	4	-
-	5	6	IOGnt*
BSLVDONE*	7	8	BSLVDVAL*
BMSTRDVAL*	9	10	BMSTRLAST*
Block*	11	12	Rd*
BMSTRRQ*	13	14	V3LCLK
V3BTERM*	15	16	V3RDY*
V3W_R*	17	18	V3BLAST*
V3ADS*	19	20	GND

Table 10.3: Debug board internal signals connector J1 pinout

Perhaps there'll be more about these signals in a later edition.

10.2. ROM emulators

ROM emulators plug into the 32-pin ROM socket, described in §5.3. ("Flash ROM and the boot ROM socket") on page 28 with the pinout in Figure 5.1. As well as providing a source of boot-time code, some ROM emulator products provide a simple network connection and console; not perhaps really needed with P−5064. However, if you want to use any extended functions you'll probably need to be able to write to the ROM socket position; note that only single byte write cycles are supported.

10.3. The debug switch

P−5064's reset flip-switch is two-way; push it the other way and it powers on the unpowered board, or delivers a debug interrupt when the board is already powered up.

Note that if you attach a remote power-on switch to J21 it will double up as a remote debug interrupt switch.

It's up to the debug monitor, hanging off one of the MIPS exception handlers, to receive the interrupt and do something useful with it.

11. Software from Algorithmics and third parties

This information is necessarily a snapshot; we'll try to keep updated information on our web site.

PMON boot ROM sources

Are available free from Algorithmics' web site, at somewhere like `ftp://ftp.algor.co.uk/pub/software/pmon/pmonsrc-970821.tar.gz`. There may be a newer version by the time you look, of course.

The package is configured to build using Algorithmics' SDE−MIPS tools on a Unix host. Use of another toolchain should not be too bad, so long as it supports MIPS-standard assembly code; but building on DOS or any version of Windows with only short file names is painful.

Note that PMON, while freely redistributable source code, is not supported by Algorithmics.

SDE−MIPS for P−5064

Probably the best MIPS compiler toolkit in the world[22]. You can find out more on Algorithmics web site. SDE−MIPS has built-in support for P−5064, as it does for all Algorithmics prototyping boards and a good range of MIPS boards from third parties.

Real-time OS on P−5064

Our policy is that no reasonable RTOS should be unavailable; but license conditions can make this difficult. If you don't see what you want, please ask.

- *Windows CE*: perhaps not really real-time, but it's clear that Microsoft's baby will get significant use in embedded systems, starting with those where a user interface or ability to run third-party software are important. Algorithmics are now "Systems Integrators" for Windows CE, and at the time of writing we are developing support for P−5064, and we expect to maintain it for all future boards. Check out our web site or ask us.
- *VxWorks/Tornado*: a BSP ("board support package") for Wind River System's OS is available. With all CPUs known so far, P−5064 is compatible with the version of VxWorks/Tornado built for MIPS R4x00 CPUs. A native R5x00 version might offer better performance, but is not essential.
- *pSOS*: Algorithmics are in touch with ISI and intend to make sure this becomes available.
- *ARTX*: Algorithmics' own RTOS is a minimal microkernel supporting a POSIX threads implementation. It's available on reasonable royalty-free terms.

  One particular feature of ARTX is its "Transputer-replacement" library, which is a library of functions which emulate the de-facto standard C binding used for the scheduling functions built into the Inmos Transputer architecture. Could be useful if you're converting Transputer code.

Other OS on P−5064

- *OpenBSD*: one of the public-domain BSD-derivative OS factions, and the one we've had most success with. Running now.
- *Linux*: the MIPS version exists in a state of considerable activity. A port exists for Algorithmics' P−4032 board, and we intend to move it over to P−5064 when we can.

[22] Algorithmics' product, of course.

Appendix A: MIPS CPUs and addresses

In MIPS CPUs the addresses generated by your program[23] 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 A.1:

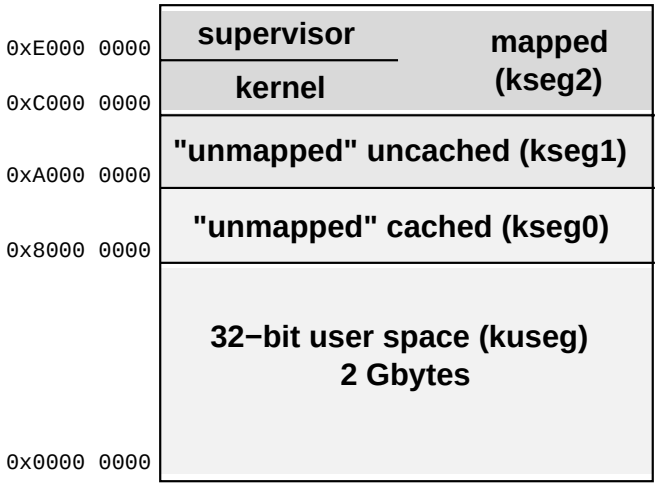


Figure A.1 MIPS program address map

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 as recommended in Appendix B below.

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 A.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 A.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 A.2 shows the big picture:

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

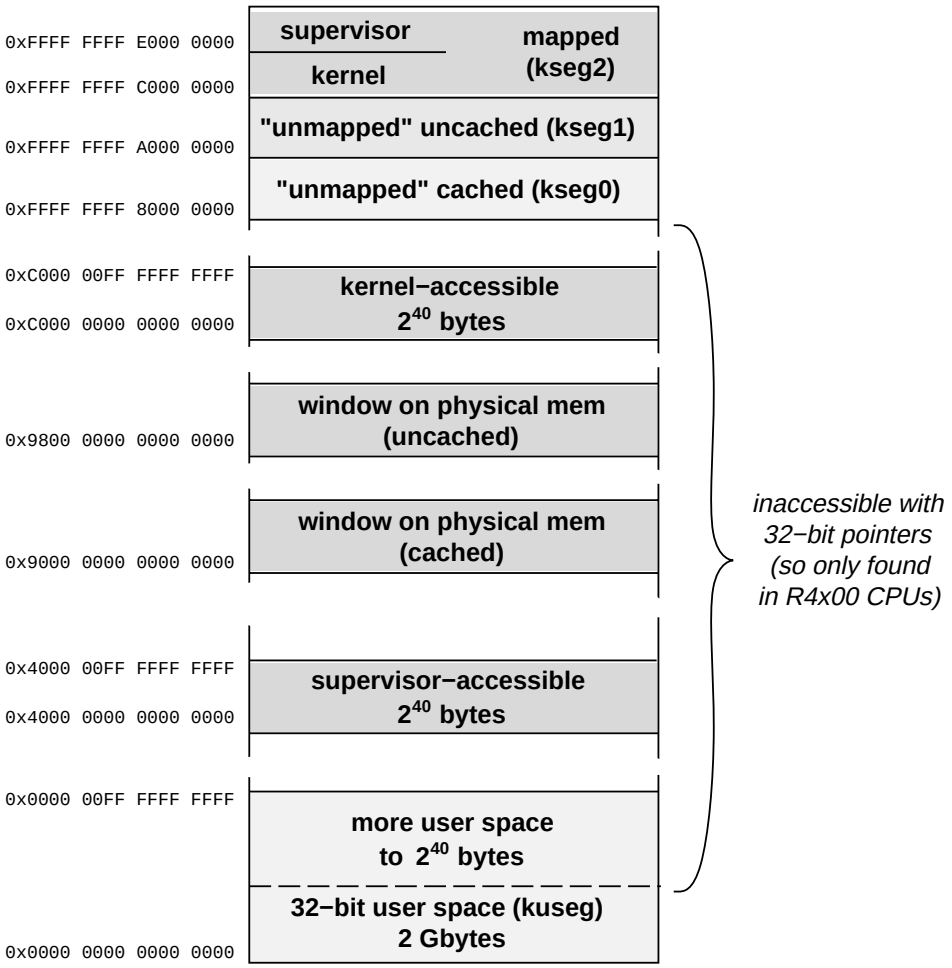


Figure A.2 MIPS program address map (entire 64-bit space)

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.

Appendix B: References - Finding more information

The world-wide-web is your best and first resource. Most MIPS semiconductor partners and most suppliers of software or hardware add-ons have web sites, and most have extensive documentation there. However, large documents and books are not much fun online, so only those of you with the right sort of printers will use those sources for the bigger documents; we'll quote all the information we can as we go along.

And of course we'd like to encourage you to visit our web site at `www.algor.co.uk`; start 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.

  A readable introduction to the practice of programming MIPS at the low level, by the author of PMON. Strengths: lots of examples; weakness: leaves out some big pieces of the architecture (such as memory management, floating point and advanced caches) because they didn't feature in the LSI "embedded" products this book was meant to partner.
- *MIPS R4000*: Joe Heinrich/Gerry Kane, *MIPS R4000 Microprocessor User's Manual*, published Prentice Hall, ISBN 0-13-1059254.

  The bible of the MIPS architecture; lots of details, but sometimes hard to find. It also takes a rather rigid view as to what is implementation specific, and can thus be left out. You can probably find a version of this to download from SGI's technical library.

CPU variants

- *R5000*: a 64-bit MIPS CPU designed by QED but made by NEC and IDT from 1995 onward. Available at up to 200MHz internal speed, on a 100MHz bus, it features limited dual-issue of instructions - essentially, it can fire off a floating-point and integer instruction simultaneously. It has dual 32Kbyte caches onchip, and onchip control for an external secondary cache attached to the system bus.

  Online documentation is only available at IDT, as far as we can determine, and it's fairly out of date. Still, look around IDT's document library. A datasheet is at `http://www.idt.com/docs/3517.pdf` at the time of writing.

  NEC has not got around to the web much. You may be able to find some data starting at `http://www.nec.com/necel/`.
- *QED RM5260, RM5270*: R5000 derivative products, designed for lower cost. The primary caches are trimmed to 16K+16K, and the RM5260 has no secondary cache controller; expect speeds up to about 166MHz internal, 83MHz interface. Launched in 1997, faster versions may be available in 1998.

  QED keep datasheets online, start at `http://www.qedinc.com/` and keep looking; they want you to sign your name to get in, but it's not a password, just a visitor's book.
- *QED RM7000*: next-generation product (but pin-compatible with RM5270) featuring a large onchip secondary cache and a more symmetric dual-issue pipeline; the first is very welcome - R5xx0 implementations have generally been cache-limited - but the second is of more dubious value. Buy a module for your P−5064 and try it for yourself!

SGI Technical Library

At `http://techpubs.sgi.com/library/` you'll find information on MIPS-designed CPUs (look under the "hardware" menu) and the only available description of MIPS assembler language - that will be under various "Irix" versions, and you may be best off using the *oldest* version, starting at "Irix 5.3".

Algorithmics' manuals

Most of Algorithmics' manuals are freely available for download from our web server in either PDF ("Acrobat") or gzipped postscript formats. Start at `http://www.algor.co.uk/algor/info/ftplist.html`.

You'll find this manual there, and also one for PMON - a reformat of the original LSI manual, which has been updated (though probably not enough). If you've bought a P−5064, you'll have a PMON manual.

Hardware information on P−5064

This is not quite freely available. We'll send the following to those who've bought a P−5064 on request (or send you a password and let you download them). Here's what you could have:

- *P−5064 schematics*: just the circuit diagrams.
- *P−5064 logic equations*: the logic equations, in Abel, for the whole of the board.
- *P−5064 design*: a theory of operation manual - which isn't actually available yet, but should be one day soon.

Note that all of these remain our copyright, which means you need to ask us before you use chunks of our design in your product; though if you do ask, we'll be very nice to you and find a way to let you do it.

Data sheets

If you're serious about programming P−5064 you're going to have to tackle the programmable devices. Here's what we know about.

Bus controllers

- *V360EPC PCI bridge*: V360EPC (V962PBC-40LP Rev B.2 in revision B boards). You can download a reasonable manual from V3's web site. A list of what's available is at `http://www.vcubed.com/databook.htm`.
- *Intel i82371 PCI→ISA adapter*: look on the `http://developer.intel.com/` web site. At the time of writing you could get links to download the data sheet on the page `http://developer.intel.com/design/intarch/datashts/290562.htm`.
- *Vadem VG469 PC card controller*: look at `http://www.vadem.com/download/` for a list of files to access.
- *Micrel MIC2563A-0 PC card power controller*: this is controlled almost entirely by the VG469 to cycle power to the PC card slots in accordance with the hot-swap rules. You probably won't need any additional data; but if you do there is a `http://www.micrel.com/pcmcia.html` and datasheets underneath it.

PCI chips

- *DEC21143 ethernet controller*: despite the name, these products are now being marketed by Intel as part of Intel's acquisition of DEC's foundries. You can find data under `http://developer.intel.com/`, at the time of writing listed on the page `http://www.intel.com/design/network/new21/techdocs/index.htm`.

  If you have a revision B board (serial number 21 or lower) with a 10Mbit/s DEC21041 ethernet controller, you'll find information in the same place.
- *Quality Semiconductor QS6612 ethernet physical-layer chip*: go to `http://www.qualitysemi.com/products/network.html` and follow the link to the QS6612 manual.

- *Symbios 53C810A SCSI controller*: Symbios have a summary page for their SCSI controller chips at `http://www.symbios.com/semi/scsi1.htm`, but they're recent converts to online data and older products like this one don't have online manuals. They have a web-based literature request form, and they probably think that the US postal service encompasses the universe.

PC-compatible devices

Most P−5064's are built with a National Semiconductor PC97307 combi I/O controller, which combines two serial ports, parallel port, floppy disk controller, real-time clock, PC keyboard and mouse, and a general-purpose parallel I/O port. However, before you go off and fetch the manual bear in mind that this part has been carefully designed to operate as part of PC clone hardware; once it has been initialised by the standard ROM at power-up it behaves pretty much like any other PC hardware.

If you need details, start at `http://www.national.com/design/`, and search on the part number "PC97307".

PC-compatible devices in revision B boards

The revision B boards were built with a lower-integration "Winbond" combi I/O chip, with separate real-time clock, keyboard and parallel I/O controllers. Here's information for those customers; but do note that for most purposes *all* these devices can be programmed as generic PC clone hardware.

- *Winbond combi I/O for ISA*: it's the W83877F. It used to be difficult to get Winbond data; but now Winbond have a web presence at `www.winbond.com.tw`, and there's a manual at `http://www.winbond.com.tw/sheet/W83877F.pdf`. You may not be able to go straight to the manual without filling in a visitors-book form.
- *Benchmarq BQ3285E real-time clock*: it's basic operation is just like every PC battery-backed-up clock calendar there's ever been, but there's data online at `http://www.benchmarq.com/prod/bq3285E_L.html`
- *AMI Key-2WP keyboard/mouse controller*: supplied by American Megatrends, who program chips which (underneath the AMI logo) are Intel 8742 microcontrollers. These are compatible with all PC keyboard controllers - Intel used to call the standard "UPI-41". AMI make more money out of BIOS software and more sexy stuff these days, and it's hard to get documentation.

  However, Intel's developer web site claims it will post you a manual to an earlier, largely-compatible product; go and visit `http://developer.intel.com/design/periphrl/manuals/`

Flash memory

- *AMD flash*: 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`.
- *Fujitsu flash*: MBM29F080, datasheet listed on the products page: `http://www.fujitsumicro.com/products/memory/flash.html`

Display

- *LCD display*: ignore the manufacturer's data, refer instead to Peer Ouwehand's online application note: `http://www.iaehv.nl/users/pouweha/lcd.htm`
- *Siemens DLR2416 LED display*: don't seem to be able to find much about that. Rely on our drivers, or find it yourself.

Odds and ends

- *Centronics ECP*: there ought to be some standards online, but probably aren't. There's a good summary at `http://www.lvr.com/parport.htm`
- *Z80 PIO controller (Z84C2006PSC)*: fitted only on revision B boards; later boards exploit the parallel I/O features of the PC97307 combi I/O chip.

  For programming information, ask Zilog at `www.zilog.com` for a data sheet.

Memory modules

Particularly for 100MHz, unbuffered SDRAM DIMMs can be problematic. Intel have spoken as to the required standards, and you can take advantage of their efforts by looking at `http://developer.intel.com/design/pcisets/memory/index.htm`.

Standards

- *PCI Local Bus Specification, Revision*: irritatingly, this isn't available online because the PCI consortium fund themselves by charging $100 or so for the hard copy. The delay and complication this causes for those of us outside the USA are considerable.

Appendix C: Reading configuration information from DIMM modules

IBM defined a specification for 168-pin DIMM modules which seems to have progressed to being a de-facto industry standard. This involves use of a small EEPROM device to store a bunch of information about the SDRAM size and organisation. P−5064 depends on this, in that the supplied bootstrap program reads out this information to decide how to program the memory controller.

This appendix tells you both how to read the ROM values, and what the ones which are significant to P−5064 mean.

How to read the DIMM's EEPROM

The DIMMs use a 2-wire interface to read (and if necessary, write) the EEPROM devices. The interface is sometimes called "I2C" and was pioneered by Xicor.

Each DIMM socket's EEPROM has an address to which only it responds, configured by static voltage levels on the three DIMM inputs called *SA0-2*. In P−5064, the DIMM1 slot gets the address 1, and DIMM2 gets the address 0.

The two signals used for reads and writes are data and clock: *SDA* and *SCL*. On the P−5064 schematics these are called *D_SDA* and *SCK* respectively.

The EEPROM signals are wired using signals controlled by the general-purpose I/O (GPIO) port, described in §5.6.6. It would be fairly straightforward to route those signals directly to the DIMMs; but unfortunately the GPIO chip is a 5V device and the EEPROM needs 3.3V levels. So instead the interface goes via one of the FPGA devices.

The signals concerned are:

GPIO signal	*I2C function*	*Notes*
B5	*SDA*	driven open-collector. Set *B5* to low to disable (which will allow *SDA*'s to be pulled up to a high), and high to actively pull *SDA* to a low
B6	*SCL*	I2C clock
B7	*SDA*	Read data. When sending to the EEPROM, a logic fossil requires that you make this signal into an output from the GPIO chip and program it low.

I2C access protocol

The basic I2C transfer uses *SDA* to convey a bit-stream of commands or data, using successive *SCL* high periods to sample *SDA*. *SDA* is defined as "open collector" and pulled up with a high-value resistor.

During data transfers *SDA* is always stable before, during and after each *SCL* high pulse. So a change on *SDA* while *SCL* is high can be used as a recognisable condition, used to delimit transfers. So the basic bit-level coding is:

- *Start condition*: a low-to-high transition of *SDA* with *SCL* high. Once the start condition timing requirements are met, it is then usual to lower *SCL*, ready for the first data bit.
- *Stop condition*: a high-to-low transition of *SDA* while *SCL* is high.
- *Writing data*: output value to be transferred on *SDA* , wait a while, raise *SCL*, wait a while, and lower *SCL*.

- *Reading data*: lower *SCL,* wait a while, raise *SCL,* pick up data on *SDA*, wait a while.

Once you can send bits, you can communicate. First of all, there is a rule for avoiding hurling bits into a black hole:

- *Byte acknowledgement*: most commands are organised as a number of 8-bit transfers. After each 8-bit transfer the EEPROM will try to send a “0” back to you to prove it is there. You should release the data line (changing it into input mode) immediately the 8th bit is safely sent.

  If you don’t see an acknowledgement, the EEPROM isn’t talking to you and nothing is happening. Perhaps the EEPROM is still busy stashing away some data you just wrote...

  Whenever the EEPROM sends you a byte of data, you have to acknowledge it, send a stop, or send a start.
- *Interpreting data*: 8-bit groups are interpreted with the first-transmitted bit regarded as the most significant (bit 7) (this is the opposite convention to most serial communications).

Now you can send 8-bit groups to the device, we can define commands and responses.

Each transfer starts with a command like this:

Device type		
1 0 1 0	001=DIMM1 000=DIMM2	0=write 1=read

Figure C.1 Command for an I2C slave device

Where:

Device type — this is a fixed code which is decoded by EEPROM devices of the particular type used on the DIMMs.

Select — another fixed code, this time to match the configuration of the *SA2-0* pins of the DIMM; as mentioned above this should be binary 1 for DIMM0, and binary 0 for DIMM2.

read/write — determines whether the next command is a read or a write.

If the EEPROM processes the command it will send an acknowledgement.

After a read/write command you need to specify an address - the 256×8 store requires an 8-bit address, and you’ve given one bit already. Don’t forget that the address is sent most significant bit first.

EEPROM write

In fact, you should never write the EEPROM on one of P−5064’s DIMMs, since the information stored therein is important configuration information. But you can’t run the protocol without doing a zero-length write...

The sequence goes like this:

- issue start
- send write command
- EEPROM ack
- send byte address
- EEPROM ack
- send data
- EEPROM ack
- issue stop

The EEPROM has an internal counter, and it is possible to write from one to 8 bytes of data by sending more data before issuing the stop. Only the low three bits of address count up, so a burst which goes over an 8-byte boundary will “wrap round” - perhaps not what you wanted.

Acknowledge Polling

After you complete a write of 1-8 bytes, the EEPROM goes off-line while it does its internal write cycle (typically 5ms). If you have more transactions to perform with it you can poll for it's completion by repeatedly sending a write command, and testing for the ACK.

Read

You can't directly supply an address for a read command. Read data is obtained from an internal "current address" register set by writes and incremented by reads.

Unless the last access was to the byte before the one you want, you have to setup the internal register with a "write" without any data, then issue another "start". So to perform a read:

- issue start
- send write command
- EEPROM ack
- send byte address
- EEPROM ack
- issue start
- send read command
- EEPROM ack
- EEPROM data

At this point you can either issue a stop (to read just one byte) or an acknowledge (in which case the EEPROM will continue with the next byte). It is possible to read through the whole of a 256-byte "page" of the EEPROM like this.

Timing requirements

You need some software mechanism for policing the frequent 5μs minimum timings. Take particular care when doing I/O writes to change the *SCL* and *SDA* signals, since the CPU's write buffer (and other "write posting" buffers in the PCI bridge and ISA bridge) can cause successive writes to come out closer together than you expected.

- *SCL frequency*: a maximum of 100kHz - so at least 10μs must elapse between successive rising and falling edges.

  In practice, you should keep all low and high periods of SCL over 5μs. It is easy to get programmed I/O from a MIPS CPU to go faster than this!

- *Data setup and hold*: change the data "as soon as possible" after the falling edge of SCL and data timing will take care of itself.

  The actual rules are that data must be stable for at least 250ns before the rising edge of SCL; and must remain stable until after SCL falls.

- *Start/stop condition rules*: SCL must be high 5μs before and after the SDA transition

- *Writes take a long time*: after you write to the EEPROM it goes away and stores the data in its non-volatile locations. This takes about 5ms, and during this period it takes no notice of you. Once you have accomplished a write you should expect to see no acknowledgement of a subsequent command for a while (see the "write polling" command above).

DIMM EEPROM data

The data provided by a DIMM module which is important to P−5064 are as follows:

Byte Addr	*Description*	*Typical Values*
0	Number of bytes reserved for this table	128
1	Number of bytes in the EEPROM device (divided by 32, so the usual 256-byte size becomes 8)	8
2	Memory type = SDRAM	4
3	Number of row address bits	11-13
4	Number of column address bits	8-11
5	Number of sides	1-2
17	Banks/DRAM	2,4
31	Module size/side (4 = 16Mbytes)	4

Table C.1

A full description of all the 32 bytes actually used on Micron DIMMs is available in the manufacturer's data sheet.

Appendix D: Software-visible changes with different versions

Revision B to C

10/100Mbit Ethernet introduction

The 21143 is a new controller, and there are many detailed differences. It also has significant extra functionality. Having said that, it is clearly a member of the same family as the Revision B's 21041 chip, and many drivers will already be configurable to the new part.

Algorithmics can supply a number of example drivers showing how the changes are made in various contexts.

Change of Combi I/O chip

The National Semiconductor PC97307 used in Revision C and later boards replaces several components from the Revision B: the Winbond combi I/O, the keyboard controller, and the real-time clock (RTC).

- *RTC*: although it still has 256 bytes of non-volatile memory, access to the top 128 bytes needs different handling. If you use Algorithmics' SDE−MIPS routines the change is transparent.
- *Power-off*: is now controlled by a register in the PC97307 "APC" module. The APC is more sophisticated than the simple control circuit on Revision B, and stores options in battery-backed-up registers.

  For example, if you power-down the P−5064 by removing the mains supply (without executing a software power-down command) the APC remembers that the board was switched on and will restart the board as soon as power returns. In the same circumstances the Revision B board would have required to be switched on with the on/debug switch.
- *Start-up configuration*: a lot of the ISA devices in the PC97307 need to be configured and enabled by software. This is handled in the startup code in the supplied boot ROM, but something similar will need to be reproduced by any customer fitting a different ROM.

  In particular, note that the address of the Centronics-compatible parallel port is programmed by startup to be compatible with the Revision B board - it defaults to something different.

Disappearance of Z80 GPIO controller

The Revision B's GPIO controller functions are taken over by the GPIO pins of the PC97307 combi chip. The PIO port address is set to 0xff00 in our startup code.

The interface to the SDRAM DIMM's ID ROMs is now built with the new GPIO controller.

Interrupt system

There is no longer an interrupt which can be generated from the user GPIO port - that was a unique feature of the Z80 PIO chip.

The new ethernet chip adds three new interrupts: *MDINT~*, *ETH_WAKEUP*, and *QS6612_ERR*. See the interrupts section §5.5 for how they're programmed.

Revision C to D

There are no systematic changes wrought between these two boards; the most pressing reason for the change was incorrect wiring to the Centronics connectors - rev C boards require the use of a special “fixer” cable when you’re using all the Centronics signals.

However, logic revision 4 has brought a change to the interrupt system, described in this version of the manual; the keyboard and mouse interrupts can now be more reliably handled by making the interrupt controller detect edges on these signals. You need to set a register bit to obtain this desirable behaviour; by default, the board continues to act as it always did. It’s laid out in section 5.5.