Algorithmics P–6032
User´s Manual

Revision: DraftFor1.0
Dated: 100/9/22

P–6032 is a single board computer for embedded systems developers wanting to build system prototypes and early-development platforms for applications using 32-bit versions of MIPS R4x00, R5x00 and MIPS–32 CPUs - and particularly for those using Algorithmics' BONITO system controller.

P–6032 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
The Fruit Farm
Ely Road
Chittering
Cambridge CB5 9PH
ENGLAND

Phone: +44 1223 706200
Fax: +44 1223 706250
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−6032 and manual

We made P−6032 initially as an evaluation and development vehicle for our BONITO system controller (both as an ASIC and as customisable IP loaded into a Xilinx FPGA). But we're familiar with supplying embedded systems developers with high-performance MIPS targets with rich I/O and excellent software and debugging support - and P−6032 should replace our P−4032 design in these markets.

This manual describes revision A boards.

1.1. The R4xx0, R5xx0 CPU families

How MIPS emerged from an academic project to lead the RISC charge is too long a story to tell here, so see [Sweet99][1]. Ever since NEC produced the Vr4300 to power the Nintendo−64 games console CPUs with the full 64-bit MIPS architecture but a low-cost 32-bit bus interface have been price/performance leaders in their sector. The other major supplier of these CPUs is QED, whose RM523x has long been the performance leader. They're widely used in printers and set-top boxes.

Read on to find why P−6032 is the right prototyping platform for all of them.

Programmers should make sure they also have the BONITO manual in front of them; if not, download it now from `ftp://ftp.algor-uk.com/pub/hardware/bonito/mips-controller.pdf`.

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 on page 16.
- *Memory map* : refer to the "Memory Map" table in the BONITO manual as well as Table 4.1 to find out what registers are where.
- *Physical arrangement, location of connectors and jumpers* : described in §6 on page 29 below.
- *Board-specific programming* : no matter how familiar you are with BONITO and the other devices on P−6032, to program the board from scratch you'll need to know about how programmable I/O pins are wired for onboard use, as shown in §5.6.
- *MIPS CPUs and their addresses* : can be very confusing for the uninitiated. If you're not familiar with MIPS, do read Appendix A (and of course [Sweet99]).

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. Some of this duplicates information available in the BONITO manual, but there's more board-specific information here.
- *Programming P−6032* : gory details of registers and how devices are wired. For detailed programming information you're normally referred to manufacturer's data sheets; we'll provide a jump page with world-wide web URLs at `http://www.algor.co.uk/algor/info/p6032-devices.html`
- *Board layout: locating connectors and jumpers* : the physical picture.
- *Jumpers: where and what for* : just that.

[1] Here and elsewhere the thing in square brackets is a reference, and you can find them in the bibliography at the end of this manual.

- *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 in-built debug unit, which allows you to see cycles in the system with a logic analyser.
- *Software from Algorithmics and third parties* : a running list - our web site might be more up to date.
- *Board revisions* : P−6032 is an evolving design, as we keep up with new CPU introductions and features, fix bugs and make customer-related improvements. Moreover, FPGA logic can evolve separately from (and usually faster than) the logic built onto the board.

  The board revision is a letter, starting a "A". There will never be more than 20 revision A boards. The board revision is printed on the board, but is also encoded in a 3-bit "bus" readable through software-readable input ports on the south bridge device; see 5.9.

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 about MIPS programming.
- *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.

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, which means fast memory systems. P−6032 will support CPUs and SDRAM at 100MHz (maybe 125MHz, not tried yet), but you might want to try different clock rates: see 5.1.
- *CPU choice* : P−6032 is intended to support a range of 32-bit bus MIPS CPUs with full 64-bit internals, which include:

CPU type/ Manufacturer	*Clock ext/int*	*Onchip cache (I+D)*	
QED RM5230	83/166	16K+16K	no
QED RM5231	100/250	32K+32K	no
NEC Vr43x0	66/133	16K+8K	
NEC Vr5432	83/166	16K+16K	
IDT R4640	66/133	8K+8K	
IDT RC64474	100/200	16K+16K	
IDT RC64574	100/200	16K+16K	

  CPUs are provided on small daughterboards. They are compatible with daughterboards for Algorithmics' earlier P−4032 product.

- *BONITO system controller* : largely invisible to software, BONITO glues the system together. While P−6032 is partly intended to advertise the wonders of this chip, it does a job which has to be done somehow in any MIPS system.

You'll program BONITO, and refer to its manual (all customers should have got one) when you set up the memory map, use its GPIO signals and handle interrupts.

- *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. Use PC-100 64-bit types - parity is not available.
- *ROM* : as supplied the board boots from a soldered-down 1M×8[2] flash ROM. There's also a DIL ROM socket (32-pin) for devices with access time 150ns or better, which will support any of:
  - A uV-erasable or flash ROM usable as a bootstrap (that's how we first fire up boards in manufacture);
  - A "NetROM"™ ROM emulator - a useful piece of equipment if you're developing ROM code.
  - A high-capacity "disk-on-chip"™ flash memory unit.

  The CPU can run cached from ROM.
- *PCI* : 33MHz, 32-bit, compliant with V2.1 of the PCI standard; four industry-standard edge-connector slots.
- *Interrupt controller* : built into BONITO.
- *Choice of I/O* : a bit of everything: two serial ports, Centronics, IDE, 10/100Mbit ethernet, PC keyboard/mouse, USB host, real time clock, 8-character LED display and user-programmable parallel I/O.

  I/O is provided by an Intel "south bridge" chip (the Intel FW82371AB), a PC-style multi-I/O chip (National PC97307–ICE/VUL), with the AMD AM79C973KC/W ethernet controller.
- *Debug unit* : you can plug in a logic analyser (directly, if it has 0.1" "mass termination" adapters compatible with an HP unit) and watch the address and some of the data of all cycles in the system. The debug unit uses BONITO's debug mode and an FPGA to decode its somewhat cryptic outputs.
- *Patchable logic* : outside of the main system controller, P–6032's logic is built around re-programmable FPGAs. The devices used (Xilinx 9500 family) are "flash" types, and can be reloaded through an external interface.
- *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:

- *SCSI* : our earlier computers have had SCSI controllers; but the controller we used has got obsolete, and relatively few customers seemed to be using it. Low-cost plug-in PCI controllers are readily available, so we swapped an extra PCI slot.
- *Faster or wider PCI* : this is a decision we already made for by BONITO: 33MHz 32-bit is the universal standard. Wider PCI buses are not much used; faster ones slow to the speed of the slowest peripheral (and we know of no south bridge or ethernet controller which runs at 66MHz).
- *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.

[2] The board can be built with 1M×16 flash memories, and probably will be once the year-2000 flash famine eases.

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−6032 is not currently certified under European regulations, because it is not itself a system but only a component[3]. By design, P−6032 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−6032 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−6032 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.

[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−6032 user's manual* : but you got that, because you're reading it.
- *BONITO - PCI/SDRAM System Controller for MIPS CPUs* : you'll certainly need this manual to program the board at a low level.
- *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−6032* : configured with the CPU, and the amount and type of memory, you ordered.

2.2. Initial wiring up

P−6032 is quite happy operating on a bench top. There are no dangerous voltages, and these CPUs don't need heat sinks or fans.

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 P2.

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−6032'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.

2.4. Boxing a P−6032

P−6032 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, USB, Centronics connectors and 10/100BASE-T ethernet).

Most PCs have a reset button lead, which will mate with the 2-pin header J4, allowing the board to be reset from the front panel. ATX boxes will often have a power-on switch, which can be plugged into J5. 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−6032 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−6032 says	*What it means*

`Notice: Integrated Tests`
`Info: Version: P6032L xxx: \`
`  Tue Sep  5 13:59:50 GMT/BST 2000`
	PROM sign-on. "P6032B" for big-endian, "P6032L" 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`
`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: Tue 5/9/2000 16:46:00`
`Info: Activity: quick memory address test`
`Info: Activity: memory byte address test`
`Info: Activity: memory halfword address \`
`  test`
`Info: Activity: memory word random test`
`Info: Activity: ns16550 operation`
`Info: Activity: keyboard operation`
`Info: Activity: PCI operation`
	Started cache tests
	
	More often you only run the "quick" memory test; see PMON manual for how to choose which ones run.

`Notice: Integrated Tests Completed` `Notice: Executing PROM package 6`	Control is now being handed over from the power-on tests to PMON.

`PCI slot 0/0: Digital Equipment DEC 21143 \`
`  (network, ethernet)`
`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

P−6032 says	*What it means*

`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 [P6032,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 R5230.  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−6032 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−6032 normally boots from an onboard 1 or 2Mbyte 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 (U26) 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 set the switch option shown in Figure 7.2.

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–6032; 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−6032 - 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−6032s; 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−6032 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 ports* : use the “load” command of PMON, and can accept a variety of download formats such as S-records.

3. Overview and Block diagram

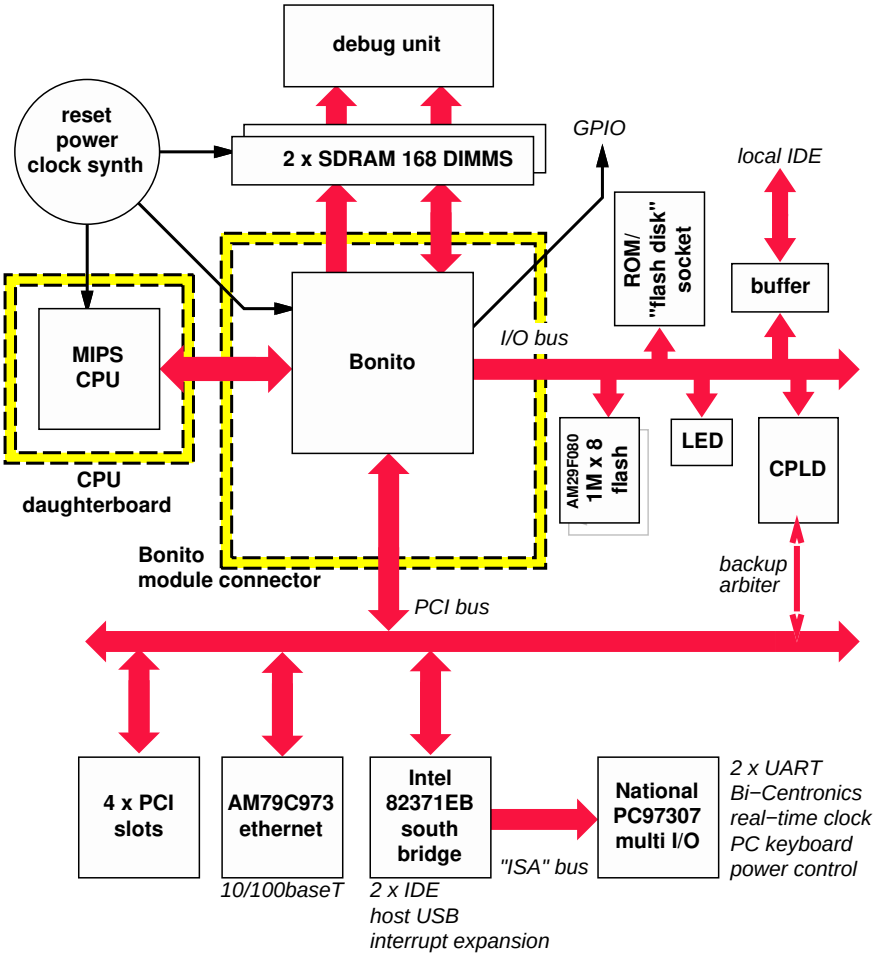


Figure 3.1 P−6032 block diagram

- *CPU/daughterboard* : P−6032 has the CPU on a small daughterboard, available for variant CPUs. Boards are available for the NEC Vr43x0, QED RM5231, IDT 79RV4640, IDT 79R64474, and IDT 79R64574. If you want some other similar CPU supported, ask.
- *BONITO* : everything is held together here - all major data and control paths go through the system controller. Note that a connector allows for variant BONITO controllers to be plugged in and tried out - the plug-in controller is implemented in an FPGA, providing a vehicle for prototyping new system controller features.
- *SDRAM* : two DIMM sockets for PC-100 parts. The SDRAM bus also connects the debug unit.
- *ROM and Local I/O* : live on a dedicated 16-bit data bus, but borrow the SDRAM data bus for addresses. The only device on this bus in P−6032 is the 8-character LED display.
- *PCI bus* : 32-bit, 33MHz, PCI 2.1 compliant onboard bus and four PC-type expansion slots.

  Of course P−6032 does the PCI host role, supplying clocks and arbiter.

- *Ethernet* : onboard 10/100Mbit/s ethernet controller, using the AMD AM79C973 controller. The same connector provides either 10 or 100Mbit operation.
- *South bridge* : an Intel FW82371AB (“PIIX 4”) south bridge provides an onboard-only bus something like an old “ISA” bus for legacy peripherals, and other functions built-in:
- *USB* : two connections, implemented by the south bridge. P−6032 has a dual connector on the back panel. 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 south bridge as a high performance PCI bus master.

  A third IDE interface is implemented directly by BONITO and should be used only to prototype BONITO systems.
- *Mouse/keyboard interface* : standard PC programming, implemented by the 82371.
- *Multi I/O controller* : a National Semiconductor PC97307−ICE/VUL controller. It includes dual (16550-compatible) serial ports, diskette interface, real-time clock and PC-type “Centronics” parallel port with bidirectional operation extensions, and capable of playing both the host or peripheral role.

4. Memory map

4.1. CPU's memory map

Although these CPUs internally generate large address ranges, they only produce a 32-bit physical 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 mapping extra PCI bus addresses.

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 (bytes)*	*Class*	*Description*
`0000 0000`	256M	Memory	local SDRAM memory
`1000 0000`	64M	PCI_Lo0	PCI low-memory bus window for most CPU accesses to PCI space. Each of the three 64Mbyte windows can be separately positioned in PCI space with its own base register.
`1400 0000`	64M	PCI_Lo1	
`1800 0000`	64M	PCI_Lo2	
`1c00 0000`	32M	ROM	soldered-down flash memory - either 1M×8 or 1M×16. You can find out which by reading **`bonponcfg.romCs0width`**, which is in turn initialised by a pullup/pulldown installed together with the flash device.
`1e00 0000` `1f80 0000`	24M 4M		unused ROM socket for emergency bootstrap, NetROM or high capacity "virtual disk" flash module.
`1fc0 0000`	1M	Boot	Bootstrap memory location - starts at the magic MIPS reset-time entry point. The position of the fifth slider on SW3 determines whether this maps to onboard flash memory or the ROM socket; see Figure 7.2.
`1fd0 0000`	1Mb	PCI I/O	PCI I/O space - window to the low megabyte of PCI address range: used (and probably *only* used) to access the I/O space of an attached device which is compatible with some old PC "ISA" device.

`1fe0 0000`
 	
`1fe0 0100`
`1fe0 0200`
	256
	
	256
		BONITO
		
		BONITO
			BONITO's own PCI configuration space registers available to other PCI bus masters
			BONITO's internal registers.
			unused

`1fe8 0000`	512K	PCI	PCI configuration space reads/writes. Low parts of the address value driven on PCI comes from this address; high order bits from BONITO's **`pcimap_cfg`** register.

`1ff0 0000`
 	
`1ff4 0000`
`1ff8 0000`
`1ffc 0000`
	256K
	
	256K
	256K
	256K
		Local I/O	Any register defined in the "CPLD_ARB" chip which provides backup logic for this board. An 8-bit register at address zero seems to write and read back. If there's any serious logic in future versions, there will be a logic revision register here.
			the HDSP-2532 8-character LED display
			Chip selects 0 and 1 (respectively) on the "BONITO IDE connector" P20.

`2000 0000`	1.5Gb	PCI_1.5	Maps 1-1 onto PCI addresses. Most likely not very useful.

Base Address	*Size (bytes)*	*Class*	*Description*
8000 0000	2Gb	PCI_2	PCI access window. Optionally mapped with either 1-for-1 addresses, or mapped down to the low 2Gb of PCI space. 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 MIPS TLB or use 64-bit pointers to get addresses bigger than 0x2000 0000 out of the CPU.

Table 4.1: P−6032 physical address map

4.1.1. PCI memory maps

To see how PCI masters view local memory, or the MIPS CPU sees PCI locations, refer to the BONITO manual chapter "Address Maps".

5. Programming P−6032

This chapter focusses on P−6032 from a programmer's point of view.

5.1. CPU

P−6032 can support any MIPS CPU using a 32-bit derivative of the system interface ("*SysAD*") which was introduced with the R4000. Algorithmics will provide daughterboards for all sufficiently popular compatible CPUs.

Differences between CPUs

In software terms, these CPUs look rather similar. They have different caches (size and set-associativity), but there are a few instruction set differences too:

- *R4640* : 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 introduced the instruction set version called "MIPS IV", and CPUs included RM5231, Vr5432 and RV64574 are MIPS IV compatible. They're likely to offer somewhat better performance, particularly on floating-point intensive programs.
- *Integer multiply/accumulate* : available on all except the Vr43x0, boosts performance on some "DSP"-like algorithms.

  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. Vr5432 CPUs can dual-issue a much wider range of instructions.

  This has few software-visible effects.

CPU configuration options

The main input clock is derived from a synthesiser IC and is set by the switches on SW1; likely choices are 75, 83 and 100MHz.

Other CPU options are configured by SW4 and SW5, as described in Figure 7.2.

- *CPU clock multiplier* : CPUs run internally at some multiple of the basic master clock rate; see Figure 7.3; you set them using SW5.
- *Endianness* : regardless of your CPU choice P−6032 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.

Most supported CPUs require a hardware strap to change their endianness, which can be achieved with a switch on SW4. But the NEC Vr43x0 is changed just by software.

5.2. Local SDRAM memory

P−6032 uses synchronous DRAM 168-pin DIMM modules compliant with the PC-100 specification. These are 3.3V, "unbuffered", 64-bit types. They must also support a CAS latency of 2 cycles - now common with PC-100 SDRAMs. DIMMs are fitted with encoding slots which match coded separators in the socket; so if your DIMMs won't fit in the sockets, they are the wrong type.

Apart from these (manifest) features, there are also some options with no functional significance in a running board, but where the memory controller on BONITO 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. BONITO 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. Otherwise, get out your BONITO manual and read the section on "SDRAM Configuration".

5.3. Flash ROM and the boot ROM socket

P–6032 normally bootstraps itself and runs its debug monitor out of the onboard flash memory. 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. Switch SW3.5 should be "on" to boot from onboard flash, and "off" 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[4].

P–6032 features either:

- a 29F080 part, sometimes from AMD and sometimes from Fujitsu: 1Mbyte in size, 8 bits wide[5], and is erasable in 64Kbyte sectors; OR
- an AM29LV160 part from AMD: 2Mbytes organised as 1M×16.

You can tell which arrangement your board has by inspecting the register bit **`bonponcfg.romCs1width`**; it will be "1" for a 16-bit ROM, and "0" for an 8-bit ROM.

Flash programming software is included as part of Algorithmics' "SDE–MIPS" toolkit, whose home page is `http://www.algor.co.uk/algor/info/sde-benefits.html`.

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 J12 and J13 from their default "2-3" position to "1-2") and ensuring J11 is installed, 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.

[4] 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.

[5] The Fujitsu "29F080" has a 16-bit bus, but we use an 8-bit compatibility mode.

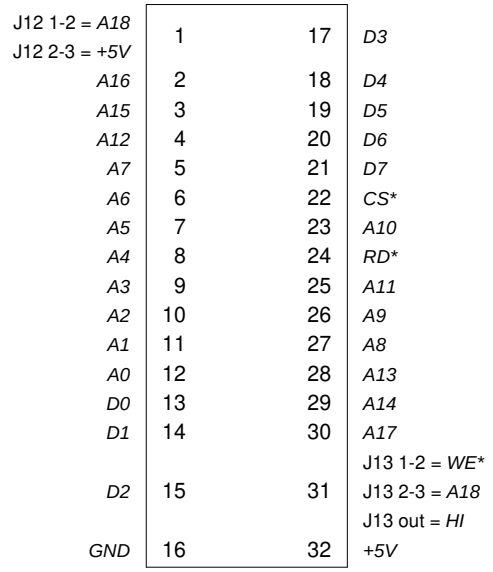


Figure 5.1 Pinout of ROM socket

5.3.3. Notes on programming flash memory

Before you do anything, note that Algorithmics' PMON bootstrap ROM reserves the highest 64Kbyte sector of the onboard 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 J11 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.)

Refer to the BONITO manual section "Endianness and ROM cycles" to understand the relationship between programming and reading flash data - it's not always obvious.

Flash programming is complex, so you might like to consider getting software which does it already, perhaps as part of Algorithmics' SDE–MIPS product, see `http://www.algor.co.uk/algor/info/sde-benefits.html`.

5.4. P–6032-specific hardware registers

There aren't any of these: everything goes through one of the complicated chips. BONITO, the south bridge and the multi I/O controller all have programmable general-purpose I/O pins to spare.

5.5. LED display

The LED display is an Agilent[6] HDSP-2532 or equivalent - an eight-character ASCII display.

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

The LED display is attached to BONITO's local I/O bus and chip select *IOCS1**, so that it works regardless of the programming of the (complicated) PCI and south bridge subsystems. That helps it do one of its main jobs, which is to pass information back from the power-on test sequence.

The LED display requires six I/O addresses (*IOA7-2* on the schematics) so BONITO must be set up to generate extra addresses with this particular chip select.

[6] Formerly the component division of Hewlett-Packard, now floated as a separate company.

5.6. Software-configurable general purpose I/O

Three devices on P−6032 provide programmable I/O pins: BONITO, the south bridge and the multi I/O controller. Some of these are used for fixed onboard functions, and some are made available for user applications on the connector P6.

GPIO bits used for onboard functions

Port/Bit	*Signal Name*	*In/Out*	*See section*	*Used for*
Bonito GPIN0	ISA_NMI	In	South bridge	Error condition (“non-maskable interrupt” on a PC) from the south bridge when it detects various kinds of bus error.
Bonito GPIN1	ISA_INTR	In	South bridge	summary interrupt output from the south bridge’s internal interrupt controller.
Bonito GPIN2	ETH_INT~	In	ethernet	interrupt notification (active low) from the ethernet controller
Bonito GPIN3	BONIDE_INT	In	BONITO IDE	interrupt input from BONITO’s own IDE channel at the P20 connector.
Bonito GPIN4-5	IRQ3−4	In	Multi I/O	*IRQ3-4* signals from the multi I/O chip. The multi I/O interrupts are also wired into the south bridge chip, so you have the option of dealing with their interrupts using south bridge facilities.
Bonito GPIO0-3	PCIIRQA−D~	In	PCI bus	PCI interrupts. These too are also wired to the south bridge chip; they’re all active low and should be level-triggered.
Bonito GPIO4	ExtPCIArbEn	Out		Software-controlled enable for the backup PCI arbiter.
South bridge GPI13-16	ModState0−3	In		Used to record PCB modifications in a software-readable version
South bridge GPI17-19	PCBRev0−2	In		major design issue number, changes with new PCB artwork. Reads all-low for rev “A” PCB.
South bridge GPI21		In	Diskette connector	Diskette change detection

Table 5.1: Parallel I/O bits and onboard functions

GPIO signals for whatever you want

Connector P6 brings a mix of BONITO and multi-I/O chip programmable I/Os for whatever you need.
See Table 8.2 on page 36 for connection details.

5.7. PCI bus

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

5.7.1. PCI accesses

The CPU can access PCI devices - usually through one of the "PCI_Lo" windows set up in BONITO. The CPU can read and write any PCI space in single cycles, but BONITO does not support bursts, so you can't access PCI through cached space.

> Some CPUs (notably NEC's Vr43x0 and Vr5432) use a 2-word burst to carry *uncached* load/store operations with 64-bit data - double-precision floating point values, or double integer data. Such transfers *do not work* to PCI space.

PCI masters can access the local memory, as described in the BONITO manual.

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" where devices are addressed by means of per-device *IDSEL* signals, generated in some system-specific manner. The BONITO manual describes how to perform configuration cycles, but you also need to know how P–6032 generates each device's *IDSEL* from the PCI *AD* lines - it's shown in Table 5.2 below.

Device	*AD line used for IDSEL*	*Interrupt controller input for*			
		INTA#	*INTB#*	*INTC#*	*INTD#*
PCI slot P9	24	GPIO0	GPIO1	GPIO2	GPIO3
PCI slot P10	25	GPIO1	GPIO2	GPIO3	GPIO0
PCI slot P11	26	GPIO2	GPIO3	GPIO0	GPIO1
PCI slot P8	29	GPIO3	GPIO0	GPIO1	GPIO2
south bridge	28	GPIO0	GPIO1	GPIO2	GPIO3
ethernet controller	27	GPIN2			

Table 5.2: IDSEL for PCI devices/slots

IDSEL generation

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 ethernet controller you'd set the PCI address to something like `0x0080.00XX` - which would set *AD27* to a "1" and all other high-order *AD* lines to "0". The low address selects the particular configuration space register.

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

PCI device interrupt assignments

PCI devices typically connect to one interrupt line; the ethernet controller has just one interrupt output wired into BONITO's interrupt controller as shown above.

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.

PCI device reset

In P−6032 the PCI reset signal *PCIRESET~* is driven by the south bridge chip[7] and acts to reset all other PCI slots and the onboard ethernet controller.

5.7.3. PCI interface registers

See BONITO manual.

5.7.4. 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−6032?

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−6032's local memory.

By the standards of onboard buses, PCI is built for fairly high bandwidth (peak 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; BONITO's "I/O buffer cache" should work well for you. But don't expect to see an average of 133Mbytes/s from a PCI master into local memory - the CPU needs some memory cycles too!

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...
- Don't expect to mix high-throughput PCI master I/O with ROM accesses; ROM accesses occupy BONITO's local bus for painfully long periods, stalling the PCI bus transfer. P−6032 is just not meant to run more than bootstraps from ROM.
- Always, always enable the I/O caching features in BONITO.

5.8. Ethernet interface (AMD AM79C973KC)

It's very complicated, and won't be described here. See the manufacturer's hardware manual or software driver examples provided by Algorithmics.

However, you may need the following P−6032-related facts:

- *Interface* : 10/100baseT socket only. All previous Algorithmics boards have had an old-fashioned "transceiver" interface - but we believe none of you use that any more.
- *Ethernet clock* : is defined by a dedicated 25MHz crystal.
- *Ethernet controller reset* : is the bussed *PCIReset** signal driven by the south bridge.
- *Interface signals* : the connection is protected by a transformer.
- *PCI connections* : the AM79C973KC's *IDSEL* signal is derived from PCI address line *AD27*, and it's interrupt output is readable at BONITO's *GPIN2* input.

[7] It would be driven by BONITO, but there's a chip bug in the 32-bit ASIC used on P−6032 which means that doesn't work properly.

- *EEROM interface* : (provided by the chip) is connected to a EEROM device, which drivers may use. However the board's ethernet address (at least) is defined by a PMON environment variable (see Table 2.2 above), which overrides any value set in the local memory.
- *LAN wakeup and magic packets* : P−6032 is built to be able to be powered up from the LAN. The ethernet controller can be kept powered from the "standby" power supply, available when the main 5V and 3.3V supplies are off; it will detect a "magic packet" and signal the event through its *RWU* pin, which eventually reaches the multi I/O controller's power-up circuit. This facility is not fully implemented in most revision A boards.
- *LEDs* : four are provided as shown on Figure 6.1. Three show receive, transmit and link activity, but the fourth (called *LED2* in the AMD manuals) is programmable.

5.9. South bridge - local I/O bus, IDE, USB etc

The Intel FW82371AB south bridge component (it's software manuals call it by the type name of "PIIX 4") provides:

- 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.
- Power-up and reset control.
- Real-time clock. On revision A boards this is not battery driven; use the RTC in the multi-I/O controller instead.
- "SMB" serial bus, which is I2C by another name. It's used to get information from the two SDRAM DIMMs and the ICS9148 clock generator which provides the selectable master clock for the CPU.
- The PCI arbiter is unused.
- The use of some of the general-purpose programmable inputs and outputs are documented in §5.6 below.
- Interrupt controller (PC legacy compatible) which is connected up to PC legacy devices, though some of them are also connected directly to the master interrupt controller provided by BONITO.
- A PC-like environment, the "Xbus", which connects the multi I/O chip and its serial ports, centronics, floppy and real-time clock.

  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.

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 south bridge chip has been programmed.

You will not often have to reprogram the ISA device base addresses; 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 for leads.

Choices made in P−6032'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 *ISA_PWROK* (it's inactive low level holds the chip in reset). It's derived from an onboard power-monitor on the 3.3V line, or forced inactive by operation of the "reset" switch SW2.

You should give the south bridge a substantial amount of time to recover its poise after asserting *ISA_PWROK*, before expecting it to do anything sensible. 10ms or so seems reasonable.

- *Interrupt output* : the output *INTR* (called *ISA_INTR* in the P−6032 schematics) is wired into BONITO's master interrupt controller through the *GPIN2* input. So is *NMI* - see §5.6.
- *PCI connection* : the south bridge controller's *IDSEL* comes from *AD28*.

5.10. Multi I/O controller

A National Semiconductor PC97307−ICE/VUL multi-function controller provides a variety of useful peripherals: a dual serial port, parallel port, real-time clock, power control and PC mouse/keyboard controller.

Dual Serial port

Programming the dual serial port is just like programming two independent 16550 UARTs.

You'll need to know:

- The serial port timing source is a 48MHz clock sourced from the standard clock generator. It can be divided by 26 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−6032 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.

Centronics

The multi 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.

But note: P−6032 is not able to behave like a traditional centronics "peripheral" as Algorithmics earlier boards are. Let us know if that causes you trouble.

Diskette

This is provided by the multi I/O chip, and emulates the NEC μPD765 device used in PCs since time began. For programming information get the multi I/O chip documentation; see Appendix B, page 27. The floppy port uses DMA service, provided by the south bridge.

The multi I/O's interface has no way of monitoring the disk change signal, so that's wired into a general-purpose programmable input port *GPI21* on the south bridge.

Real Time Clock (RTC)

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.

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 circuit 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.

Keyboard/mouse controller

part of the multi I/O chip, this provides a standard PC interface. Refer to the sample drivers for a software interface.

Power control (“APC”)

The “ATX” power supplies compatible with P−6032 provide a continuous low-current +5V supply (called *VSTBY* on the P−6032 schematics) whenever the mains is connected. The standby supply is used to power the reset circuitry, and the *VCCH* input of the multi-I/O chip. Circuits on the PC97307 are responsible for bringing up the main power when the reset/debug switch SW2 is toggled to the “debug” position.

P−6032 is built to permit the ethernet controller subsystem to be run off standby power, so that the board can be powered up by sending it a “magic packet”. This facility is not fully implemented in most revision A boards.

IR interface

A separate set of signals are available for infra-red connections. They’re brought to a connector P7, and the rest is up to you.

Hardware options

multi I/O signals *SOUT1/CFG0*, *RTS1*/BADDR1* and *SOUT2/CFG3* are pulled up with 10KΩ resistors. The results are documented in the PC97307 manual thus:

- *CFG0* set 1: “FDC, KBC and RTC wake up active”.
- *CFG1* set 0: “no X-bus data buffer”.
- *CFG3-2* set 10: “clock source is 48MHz fed via *X1* pin”. Seems wrong; *X1* is really connected to a 32.768Khz crystal.
- *BADDR1-0* set 10: “PnP motherboard, wake in Config state, Index `015Ch`”.

5.11. 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 object files must be compatible with MIPS-ABI. A symbol table, if present, will be loaded and used 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−6032 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−6032 is Figure 6.1.

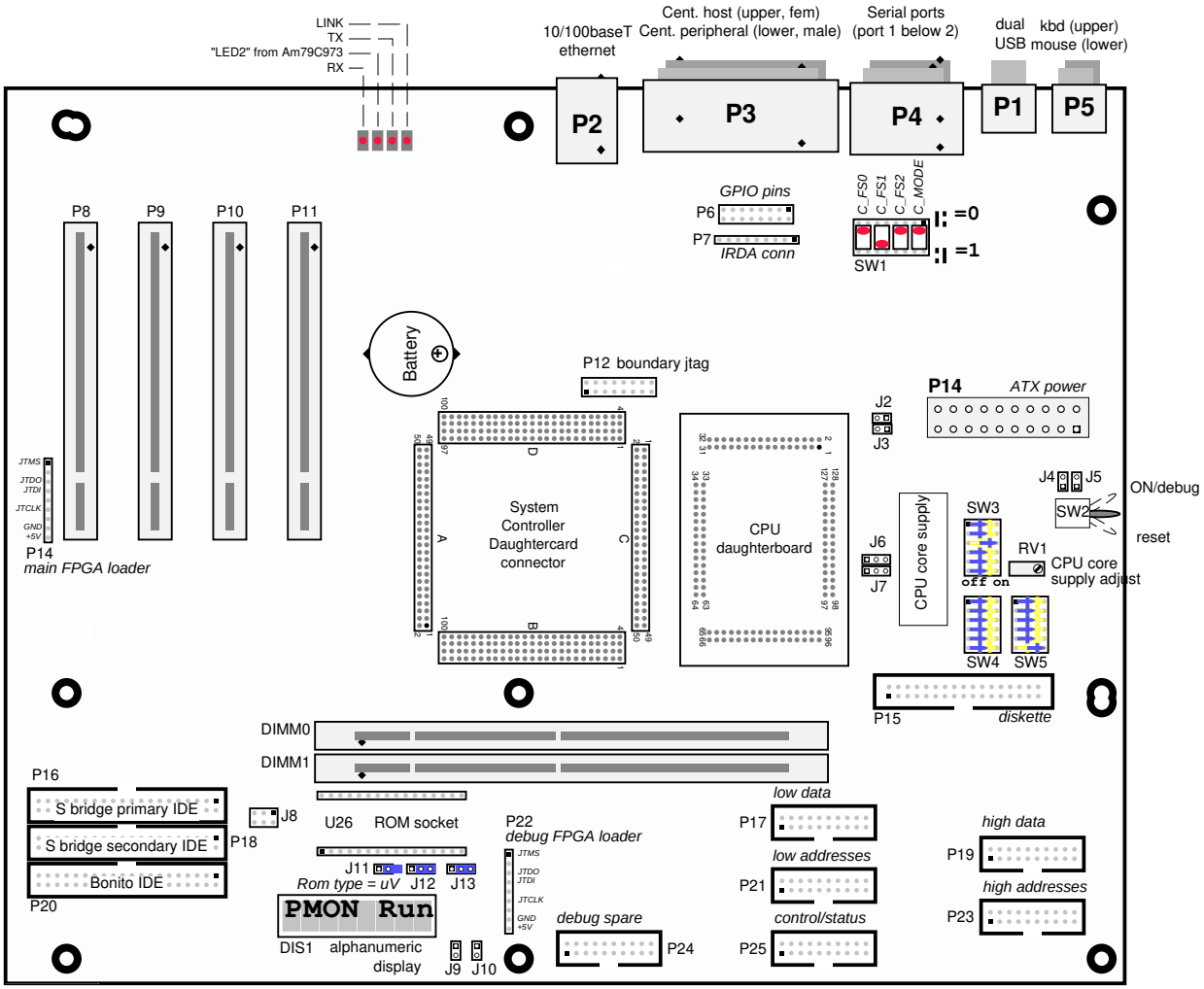


Figure 6.1 P−6032 layout, connectors and jumpers

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 where the standard is effective and you can "just plug in").
- *Switches and jumpers* : functions and defaults are described in §7 below.
- *Adjustable components* : RV1 is used to set the "adjustable" 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 - but most CPUs will be accommodated by one of the three fixed settings (1.8V, 2.1V, 2.5V) available at the flick of a switch.

  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. Switches and jumpers: where and what for

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

Ref	*Default*	*Description*
SW2		Two functions. In one direction, this is a system reset switch. Push the other way to work both as a power-up switch and (once the system is switched on) as a "debug" high-priority interrupt button. The power-up switch relies on power-on features of the multi-I/O controller. The interrupt goes directly to the CPU's *Int3** input.
SW3		6-way switch blocks, see Figure 7.2.
SW4		
SW5		
SW1		4-way switch block for CPU clock rate etc, see Figure 7.1
J11	in	Remove to write-protect flash memory fitted in the socket (if any).
J13	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.
J12	2-3	
J3/ J2	out	Usually provides an attachment point either side of a 9mΩ precision resistor in series with the 3.3V supply to the CPU, for measuring the current being used. You can short these links out if you're worried about the effect of the resistance on the 3.3V supply.
J7/ J6	out	Attachment point for measuring CPU "core" current, through a 9mΩ precision resistor.
J5		Connector to attach a remote instantaneous-on switch which will perform the same combined power-on and "debug" function as SW2.
J4		Connector to attach a remote reset switch to perform the same function as the "reset" direction of SW2.
J9	-	Reserved for debug unit
J10		

Table 7.1: All switches and jumpers on P−6032 (including connectors called Jxx)

7.1. CPU master clock rate setting - SW1

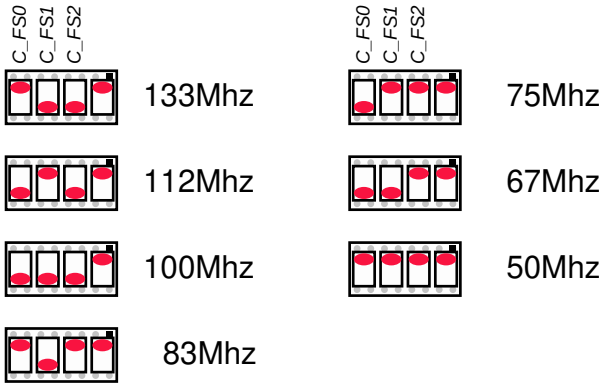


Figure 7.1 CPU master clock rate setup with SW1

Use the diagram Figure 6.1 to locate the switch, if necessary. You're unlikely to find a CPU which won't run at 67MHz, and 83 and 100MHz are common. BONITO is (in theory) at its limits at 100MHz - but 112MHz is certainly worth a try!

The last switch of SW1 should never be moved from its normal ("0") position.

7.2. CPU type and software options switches: SW3, SW4, SW5

These three switches - all shown in Figure 7.2 - tell the logic what kind of CPU is fitted, and set the CPU's hardware-determined characteristics.

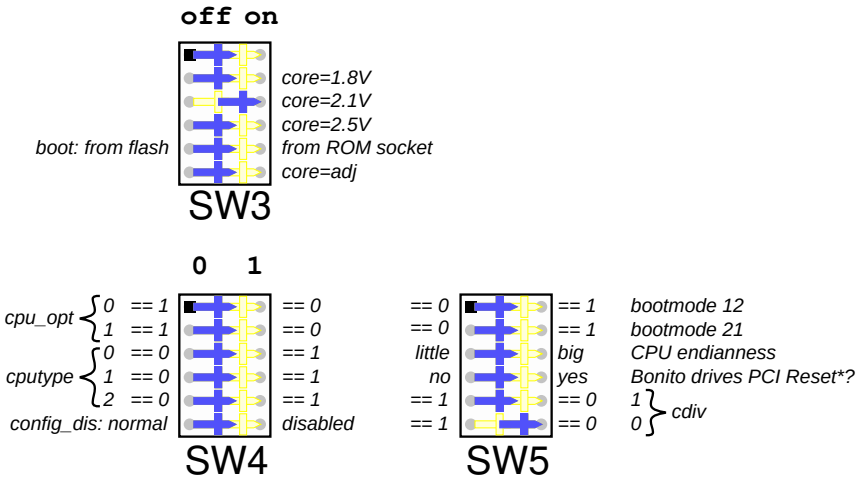


Figure 7.2 CPU type and options - SW3, SW4, SW5

The fields are as follows:

- *core=??* : many newer CPUs have dual power supply: 3.3V (*VCCIO*) for the I/O pins, but a lower voltage (*VCORE*) for the CPU core. P–6032 supports 2.5V, 2.1V and 1.8V options, and also has an adjustable setting (tweak RV1 with the CPU socket out).

  3.3V-only CPU modules are wired to take all power from the *VCCIO* pins of the module. In this case *all* these switches should be set to the "off" position - on some of those modules the *VCCIO* and *VCORE* pins are all connected together.

- *boot from...* : set to allow bootstrap from either the onboard flash (normal) or the ROM socket.

- *cpu_opt* : reserved, please leave "off".

- *cputype* : the three *CPUTYPE0-2* switches in the block SW4 are there to select for different types of CPU. This is best done by settings wired into the CPU module which is plugged into the board, but older CPU modules won't do this - so you've got switches too. Setting any switch "on" pulls up the corresponding BONITO configuration signal *IOD0-2*; the BONITO manual is authoritative for how to set these, but popular settings are:

CPU	*cputype switches*		
	2	*1*	*0*
NEC Vr4300	off	off	off
NEC Vr5432	off	off	on
QED or IDT CPU	on	off	off

Table 7.2: CPU types and SW4 switch settings

- *config_dis* : reserved, please leave “off”.
- *bootmode12, bootmode21* : used to configure two of the serial configuration bits fed to QED and IDT CPUs at reset time. The *bootmode21* bit should be set when you use a QED RM523x CPU.
- *CPU endianness* : is sometimes software-settable (with NEC’s Vr4300 CPU, for example); in that case, leave this switch “off”.

  When the CPU is settable with a static configuration pin, this switch will do it. Note that BONITO will (soon after boot-up) need to be set up to match the CPU, but that needs to be done by software.
- *Bonito drives PCI reset* : on revision A boards this is reserved and should be set to “no” (“off”).
- *cdiv* : NEC CPUs select the ratio between the master clock and the internal CPU clock with static configuration signals *CDIV1-0*, and you can set them here. The following table is correct for the NEC Vr43x0 CPU:

Field name	*Link position*	*Effect*
CDIV0,1	off off	CPU runs at input clock $\times$ 3
	on off	CPU runs at input clock $\times$ 2 (default)
	off on	CPU runs at input clock $\times$ 1.5 (Vr4310 only)
	on on	CPU runs at input clock $\times$ 1

Figure 7.3 CDIV settings and effect on NEC Vr43x0 CPU clock rate

8. Connectors: where, what and wiring

8.1. CPU daughterboard connector

The CPU daughterboard connector is made up of 2mm dual socket strip/headers, in four banks each of 2×16 pins, as shown in Figure 8.1. The signals on the connector are shown in Table 8.1. You probably won't have to know this very often.

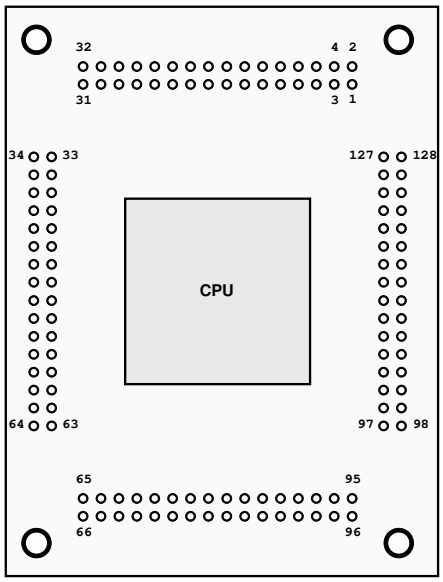


Figure 8.1 CPU daughterboard layout

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

Table 8.1: Pinout of CPU daughterboard (MIPS names)

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 64-bit (non-ECC) types.

8.3. PCI edge connectors: P9, P10, P11, P8)

Industry standard connectors, not defined here.

However, you will need to know how the *IDSEL* and interrupt lines are assigned; see Table 5.2 on page 24.

8.4. Ethernet (P2)

The 10/100Mbit/s ethernet has a 10/100baseT connector P2. 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. IDE

There are primary and secondary IDE channels (P16, P18), connected up to the south bridge controller. There is a third channel on the P20 connector, wired directly to BONITO's IDE port. The BONITO port is simpler and will probably give better performance, but you're much more likely to find an OS driver for the south bridge IDE channels.

8.6. RS232 (P4)

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.

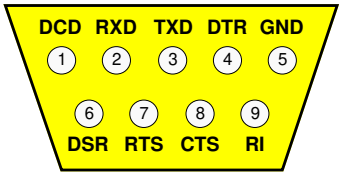


Figure 8.2 Pinout of a PC-compatible serial connector (looking into pins)

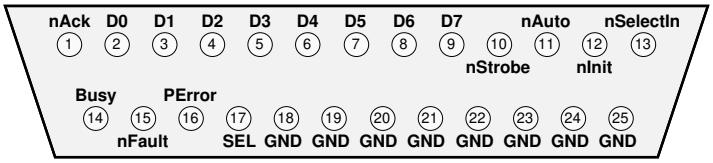
Notes on the serial port signals:

Signal	*Description*
RXD, TXD	asynchronous serial data into and out from P–6032, 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–6032 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.

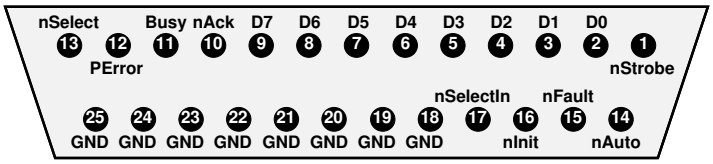
8.7. Centronics (P3)

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!

Figure 8.3 shows the pinouts.



Centronics peripheral (male)



Centronics host (female)

Figure 8.3 Centronics/IEEE-1284 parallel port connector

8.8. Diskette (P15)

Standard PC-type diskette header; not described here.

8.9. User-defined parallel I/O (P6)

This 8×2 pin header brings together individually programmable I/O signals - *GPIO5-8* from BONITO and *GPIO10-17* from the multi I/O controller. They’re available for whatever function you like, such as:

- Polling an external logic level.
- Driving some simple external device.
- Software-controlled trigger for test equipment...
- Detecting an interrupt on a low or high level, or a rising or falling edge.

Anything you like. The pinout is in Table 8.2.

		multi I/O GPIOs				BONITO GPIOs	
GND	GND	13	12	11	10	8	6
15	13	11	9	7	5	3	1
		pins					
16	14	12	10	8	6	4	2
		17	16	15	14	7	5
VCC	VCC	multi I/O GPIOS				BONITO GPIOS	

Table 8.2: GPIO connector (P6) pinout

8.10. PC-compatible keyboard/mouse connector (P5)

A dual small (“PS/2” type) DIN connector. If your keyboard has a big DIN connector as used in older PCs, converters are readily available.

8.11. USB (P1)

A host port for “universal serial bus” implemented by the south bridge should permit the attachment of USB peripherals, just emerging as the first P−6032 boards ship. Ask Algorithmics about software support. The two sockets in the dual connector are wired identically.

8.12. IR “network” interface (P7)

We’re now really getting down into the far end of feasible. The multi I/O controller provides signals to drive an infra-red transceiver supporting a standard used by some hand-held computers. This is for experimentation only, but P7 is a 9-pin 0.1” SIL header. It’s connections are as follows (signal names are as used in the PC97307 manual):

Pin	*Signal*
1	`+5V`
2	–
3	`IRRX`
4	`GND`
5	`IRTX`
6	`IRRX2/ID0`
7	`IRSL1`
8	`IRSL2`
9	–

8.13. Power supply connector (P13)

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 8.4 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 8.4 ATX power supply connector pins

In Figure 8.4 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−6032 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−6032's reset circuitry, and its low-going transition can be used to generate an interrupt.

8.14. Logic programming connectors: P22, P14

P−6032 has two Xilinx 9500 series programmable logic devices. One is dedicated to the debug unit (see §10), and the other is available as a programmable resource for fixing bugs or providing extra functions in experimental applications of the board.

These chips retain their logic programs using "flash" ROM storage, but can be reprogrammed in-circuit with a JTAG cable. The instructions given here assume you're using a Xilinx download cable (not too expensive).

The Xilinx-compatible connectors are 9-pin 0.1" SIL header strips, and the pinout is shown in Figure 8.5. The Xilinx cable terminates with eight colour-coded individual cables.

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

Figure 8.5 Xilinx-compatible connectors P22, P14 for reprogramming P−6032 logic

8.15. JTAG boundary scan test connector: P12

The scan test chain threads through the CPU module, ethernet controller and Bonito ASIC, as shown in Figure 8.6.

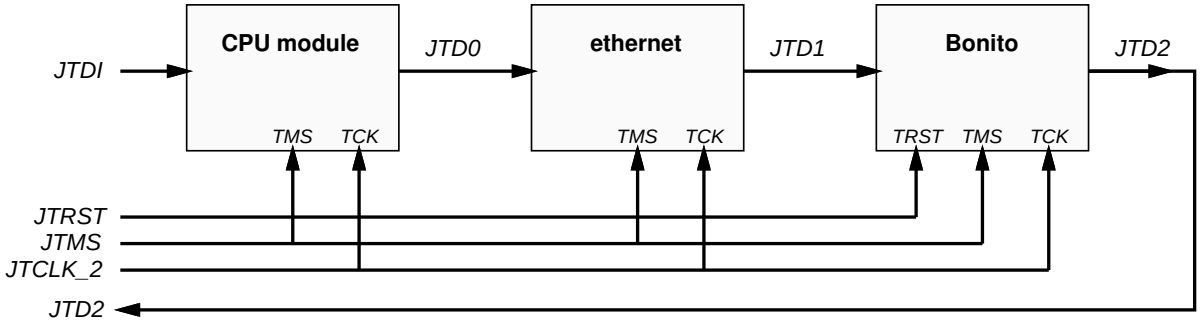


Figure 8.6 JTAG boundary scan chain

The signals of the chain are accessible through the P12 connector shown in Table 8.3:

JTMS	1	2	*GND*
JTCLK_2	3	4	*GND*
JTDI	5	6	*5V*
	7	8	*5V*

	9	10	
JTD2	11	12	
JTD1†	13	14	
JTD0†	15	16	*JTRST*

Table 8.3: Pinout of JTAG boundary scan connector P12

9. Cables supplied

With your P−6032 you should have received a couple of cables:

- *IDE* : two ribbon cables supplied to attach drives, though perhaps they're rather short for peripherals not in the same box.
- *Floppy* : one cable with connectors for two drives.

Everything else you should need to get your system up and running should be readily available from your local PC superstore.

10. Hardware debug and trace facilities

Since P−6032 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−6032 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 unit 10 gives you somewhere to 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.

The debug unit has logic analyser connectors which present all CPU cycles and PCI cycles accessing local memory. It buffers and re-registers the buses, so that even a relatively slow analyser will have no trouble following P−6032'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.

	1	2			1	2			1	2			
ATRIG	3	4	*SPARE1*	*ATRIG*	3	4	*A15*		3	4	*D15*		
SPARE0	5	6	*AMUX13*	*A14*	5	6	*A13*	*D14*	5	6	*D13*		
AMUX12	7	8	*BE3*	*A12*	7	8	*A11*	*D12*	7	8	*D11*		
BE2	9	10	*BE1*	*A10*	9	10	*A9*	*D10*	9	10	*D9*		
BE0	11	12	*DP3*	*A8*	11	12	*A7*	*D8*	11	12	*D7*		
DP2	13	14	*DP1*	*A6*	13	14	*A5*	*D6*	13	14	*D5*		
DP0	15	16	*WR*	*A4*	15	16	*A3*	*D4*	15	16	*D3*		
SIZE2	17	18	*SIZE1*	*A2*	17	18	*A1*	*D2*	17	18	*D1*		

† In the revision A P−6032, a bug means that the signals *JTD0-1* are inadvertently shared with the corresponding JTAG signals of the debug FPGA, normally driven from the programming connector P22. This bug is not critical because the *JTMS* and JTAG clock signals are *not* shared; so long as the debug FPGA is kept in a state where its JTAG output is tristate, everything still works.

SIZE0	19	20	*GND*

P25

A0	19	20	*GND*

P21

D0	19	20	*GND*

P17

	1	2	
	3	4	*A31*
A30	5	6	*A29*
A28	7	8	*A27*
A26	9	10	*A25*
A24	11	12	*A23*
A22	13	14	*A21*
A20	15	16	*A19*
A18	17	18	*A17*
A16	19	20	*GND*

P23

	1	2	
	3	4	*D31*
D30	5	6	*D29*
D28	7	8	*D27*
D26	9	10	*D25*
D24	11	12	*D23*
D22	13	14	*D21*
D20	15	16	*D19*
D18	17	18	*D17*
D16	19	20	*GND*

P19

Table 10.1: Pinout of the debug connectors

The signals on the debug connectors are as follows:

Signal	*Description*
A31-0	Address of this cycle
ATRIG	Analyser trigger - rising edge on this signal denotes something worth capturing.
BE3-0	Byte enables, where *BE0* shows a valid transfer on *D7-0* (etc)
D31-0	Data bus
DP3-0	Parity on data bus, if supported.
WR	High for a write, low for a read.
GND	Board ground to connect to analyser pod ground
AMUX13-12	SDRAM multiplexed addresses
SIZE2-0	Encoded transfer size

Table 10.2: Debug connector signals described

10.1. ROM emulators

ROM emulators plug into the 32-pin ROM socket, described in §5.3. (“Flash ROM and the boot ROM socket”) on page 21 with the pinout in Figure 5.1[8]. 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−6032. 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.

[8] P−6032 revision A boards have a problem with some ROM emulators. The emulators may have resistive pull-ups on their data lines, which are joined to BONITO’s I/O data bus *IOD0-7*. But BONITO uses pullups on those lines to configure some important chip features, so when you plug in the ROM emulator chaos ensues.

This will probably be fixed in later versions of P−6032.

10.2. The debug switch

P−6032'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 J5 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−6032

Probably the best MIPS compiler toolkit in the world[9]. You can find out more on Algorithmics web site. SDE−MIPS has built-in support for P−6032, as it does for all Algorithmics prototyping boards and a good range of MIPS boards from third parties.

Real-time OS on P−6032

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.

- *VxWorks/Tornado* : a BSP ("board support package") for Wind River System's OS is available. With all CPUs known so far, P−6032 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.
- *Linux* : the MIPS version is much better than it was a year ago (written September 2000). We expect to make a reasonable port available with early boards, and offer some support and help to customers.
- *AlgRTX* : Algorithmics' own RTOS is a minimal microkernel supporting a POSIX threads implementation. It's available on reasonable terms with source code available and no per-unit royalties.

  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−6032

These are less strongly in demand, and will be provided as and when customer demand meets resource availability.

- *Windows CE* : Microsoft's baby OS has not really taken off (yet) outside those applications where Microsoft provide all the software. But if you are interested in Windows CE and P−6032, then Algorithmics are "Systems Integrators" for Windows CE, and have delivered board support packages ("OAL"s) for our P−4032 and P−5064 boards.
- *OpenBSD* : one of the public-domain BSD-derivative OS factions, and the one we've had most success with.
- *pSOS* : a major Algorithmics customer for the P−4032 use pSOS, and it may be extended to the P−6032 - ask.

[9] Algorithmics' product, of course.

Appendix A: MIPS CPUs and addresses

In MIPS CPUs the addresses generated by your program[10] 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:

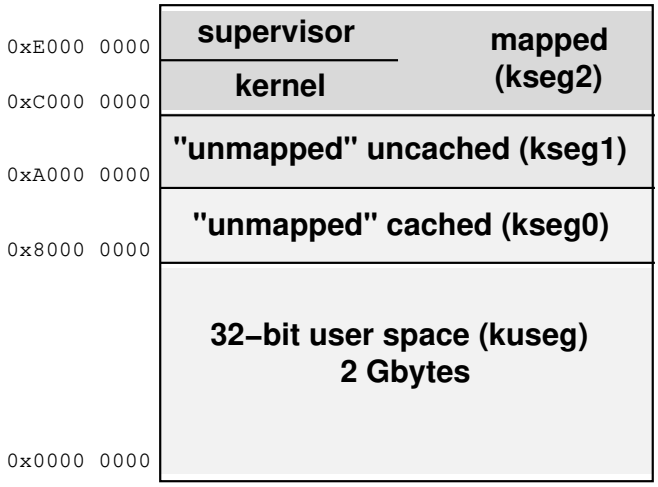


Figure A.1 MIPS program address map

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:

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

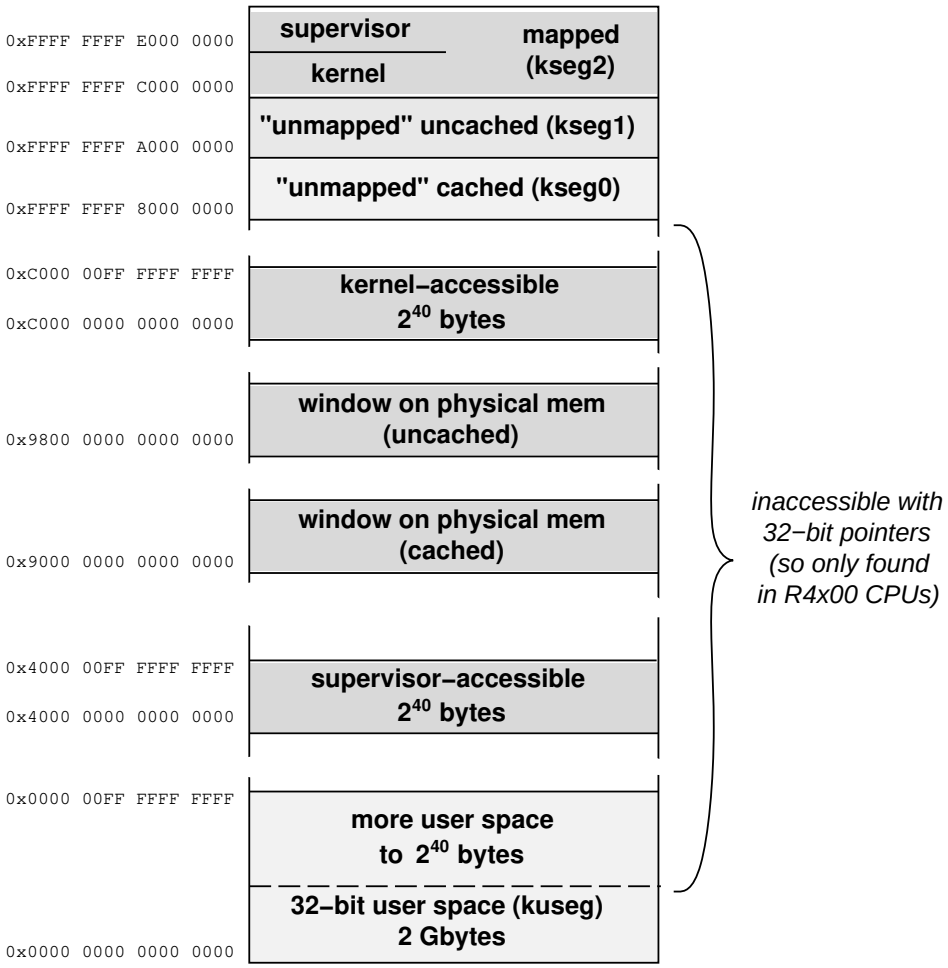


Figure A.2 MIPS program address map (entire 64-bit space)

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* : [Sweet99] Dominic Sweetman, *See MIPS Run* published by Morgan Kaufmann, ISBN 1-55860-410-3.
- *MIPS R4000* : [R4000man] 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.

Data sheets

There's a jump page at `http://www.algor.co.uk/algor/info/p6032-devices.html`, and you should go there and browse.

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–6032, you'll have a PMON manual.

Hardware information on P–6032

This is not quite freely available. We'll send the following to those who've bought a P–6032 on request (or send you a password and let you download them). Here's what you could have:

- *P–6032 schematics* : just the circuit diagrams.
- *P–6032 logic equations* : the logic equations, in Verilog, for the add-on logic.

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.

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.
- *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`

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−6032 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−6032 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−6032, 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−6032 schematics these are called *D_SDA* and *SCK* respectively.

The EEPROM signals are wired using the "SMB" signals of the south bridge device, as described in §5.9. 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−6032’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 BONITO and the south 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−6032 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.