ALPHA PC
Hardware and System Programming Guide

David G. Conroy
Digital Equipment Corporation
HLO2-3/J03, dgc@dgcvax.cdad.hlo.dec.com

This document is a hardware and system programming guide for the ALPHA PC, a simple, moderate performance, low cost, ISA bus based ALPHA machine. It does not contain programming information for the major LSI components or for any of the ISA bus peripherals.

Introduction

The ALPHA PC is a simple, moderate performance, low cost, ISA bus based ALPHA desktop computer, which combines the DC288 CPU chip with inexpensive industry standard power, packaging, and peripherals. It is being designed and built to better understand how to put high performance processors in low cost systems (in particular, what changes should be made to the external interfaces of high performace processor chips to make them better suited to low cost applications), and to gain experience with the poorly documented world of ISA peripherals.

Products decended from the ALPHA PC could be used both as stand alone computers, and, with the addition of an ISA based network interface card, as members of a distributed computer system. Products decended from the ALPHA PC would not be suited to applications which demand the highest possible CPU performance, large memories, I/O performance beyond what is possible on the ISA bus, or graphics beyond VGA or 8514. These applications would be better suited to other, more capable, members of the ALPHA product set.

The ALPHA PC consists of:

1) A system board, which contains a DC228 CPU chip and its support electronics, a 128K byte write-back cache, an 8 to 64 megabyte memory system built using industry standard memory SIMM's, an interface to the ISA bus (including the DMA and interrupt controllers), and some low speed peripherals (serial lines, printer, keyboard, mouse, TOY clock, interval timer). All of the low speed periperals have their connectors mounted directly on the system board.

2) A standard IBM PC/AT case and power supply. Any IBM PC/AT case could have been chosen; we chose a low-profile case from the Enlight corporation. The low-profile case has the feature that only a single ISA slot needs to be implemented on the system board (since the ISA cards mount horizontally, and plug into the T-card), making more space available on the system board.

3) An IBM PC/AT keyboard, which is plugged into the standard IBM PC/AT keyboard connector on the system board. Any IBM PC/AT keyboard could have been chosen; we chose the BCT-5339 101 key keyboard and the "Carry 1" 84 key keyboard (JDR Microdevices). An IBM PS/2 keyboard can also be used with the appropriate adaptor cable (an IBM PS/2 keyboard is the same as an IBM PC/AT keyboard except that it has a different connector on the end of the cable, and its microcontroller implements a few extra commands, most of which are useless).

4) An IBM PS/2 mouse, which is plugged into the system board using a special cable. We wanted 3 buttons, so we did not have many choices; we chose a Logitech serial PS/2 mouse. Of couse, a standard RS232 serial mouse can also be used by plugging it into one of the serial ports.

5) An ISA video adaptor, which is plugged into one of the ISA slots. Any of the ISA video adaptors could have been chosen; we chose the Modular Circuit Technology MCT-VGA-1024 (JDR Microdevices again, using the Western Digital/Paradise chip set) 16 bit SVGA and the ATI 8514-ULTRA 8514 display adaptors. We chose the DEC VRT13 monitor, which is a Sony monitor in disguise.

6) An ISA disk adaptor, which is plugged into one of the ISA slots, and disks that are compatible with it. Any ISA disk adaptor could have been chosen; we chose the Adaptec AHA-1522 combination SCSI and floppy controller, Quantum 100 and 200 megabyte SCSI hard disks, and TEAC FD-235-HF 3 1/2 inch floppy disks. The somewhat unusual (for the IBM PC world) choice of SCSI disks was made so that we could transport them between the ALPHA PC, the ADU's, and our 3MAX's.

7) An ISA Ethernet adaptor, which is plugged into one of the ISA slots. Any ISA Ethernet adaptor could have been chosen; we chose the Western Digital WD8013 (16 bit) Ethernet adaptor.

Hardware

CPU Chip

The ALPHA PC uses a DC228 (EV4) or DC227 (EV3) CPU chip.

The CPU chip cycles at 10nS. The CPU chip divides the 10nS CPU clock by 4 to generate a 40nS system clock, which is distributed throughout the machine. The ISA bus interface divides the system clock by 3 (normal mode) or 2 (turbo mode) to generate an 8.33MHz (normal mode) or 12.5MHz (turbo mode) clock for the ISA peripherals.

Since the DC228 CPU chip has a programmable system clock divisor and programmable backup cache timing it is fairly straightforward to run the CPU chip at higher clock rates (although faster backup cache SRAM's may still be

needed to make all of the timing work out, particularly on writes). The system clock period must be kept near 40nS, which makes CPU clock periods of 8nS (8nS * 5 = 40nS) and 6.6nS (6.6nS * 6 = 39.6nS) feasable. The system board has been designed to make experimentation with higher clock rates easy; all timing parameters come from programmable devices, the -3.3V regulator can supply enough current to run the CPU chip at full speed, and the CPU chip's heat sink can cool an 8nS part without change (and can probably cool a 6.6nS part chip with the addition of a small fan mounted in the large open space in the box above the CPU chip).

The initial program is loaded from a Xilinx XC1736 serial ROM. The initial program must fit in 1K bytes if a DC227 CPU chip is used.

Backup Cache

The backup cache is built from the 25nS 8Kx16 and 8Kx18 cache SRAM's intended for use with the Intel 80385 cache controller. The cache data store is 128K bytes in size, is 128 bits wide, and is protected by longword parity. The cache tag store contains 4K tags, and both the address part of the tag and the control part of the tag are protected by parity. The address part of the tag only stores CPU address bits cA[25..17]. This implies that the CPU tagAdr[33..26] inputs are tied low, that the parity trees that generate tag address parity assume that cA[33..26] are all 0's, and that the tag comparator ignores cA[33..26] completely. Software must ensure that any address with cA[33]=0 also has cA[32..26]=0.

Normally the backup cache is controlled directly by the CPU chip. However, during DMA the CPU is forced off the backup cache SRAMS (using the holdReq/holdAck mechanism) and the backup cache is controlled by the I/O system. The backup cache supplies data to the I/O system on DMA reads that are hits, and the backup cache accepts data from the I/O system on DMA writes that are hits (without changing the state of the dirty bit). There are no backmaps, so the D-cache is blindly invalidated on write hits.

Software loaded from the serial ROM must configure the backup cache interface for parity mode, output enable mode, read speed of 4 cycles, write speed of 4 cycles, write pulse of 2 cycles centered in the 4 cycle write interval, and enable it only in quadrant 0 (these timing parameters are for a 10nS CPU cycle time). Software loaded from the serial ROM must also validate the backup cache before enabling machine checks or expecting memory cycles to do anything sensible. Here is a sequence that will do it.

1) Enable the backup cache. Disable machine checks.

2) Read some location in each of the second 4096 cache blocks, and then read some location in each of the first 4096 cache blocks, which will result in either one (rarely) or two (usually) cache misses. The external interface will read garbage from memory, and put it in the cache as a clean block with correct tag parity.

3) Write something to all locations in the first 4096 cache blocks. All of these writes will be cache hits, so the data will be put into the cache with correct data parity.

4) Enable machine checks, if desired.

Lock Logic

The lock logic consists of only a lock flag. There is no lock address register or lock address comparator. The lock flag is cleared by reset, set by load locked instructions, tested and cleared by store conditional instructions, and cleared by all DMA writes (hits or misses).

XBUS

The I/O system is built around the the XBUS, which is, essentially, a clone of the pin interface of an Intel 80386DX microprocessor. The 25MHz system clock generated by the CPU chip is used as CLK2 on this bus, so the XBUS runs at 12.5MHz. Normally the XBUS is controlled by the CPU. However, during DMA the CPU is forced off the XBUS (using the holdReq/holdAck mechanism) and the XBUS is controlled by the Intel 82380 integrated system peripheral.

The mapping of CPU cycles into XBUS cycles is a little wierd, because it is necessary to generate interrupt acknowledge cycles, a memory versus I/O space distinction, byte masks, low order address bits, and other Intel-isms. CPU addresses with cA[33]=1 are XBUS addresses. CPU address bits cA[26..05] are copied directly to XBUS address bits xA[26..05]. CPU address bits cA[29..27] are copied onto XBUS address bits xA[04..02]. The XBUS read/write signal is determined by looking at the CPU's command. The rest of the XBUS address and command signals are generated by table lookup, based on CPU address bits cA[32..30]. Here is the table.

cA[32..30]	xIO	xCmd	xBE	xA[31..27]
000	T	F	FFFT	10000
001	T	F	FFTF	10000
010	T	F	FTFF	10000
011	T	F	TFFF	10000
100	F	F	FFFT	10000
101	F	F	FFTF	10000
110	F	F	TTTT	10000
111(R)	T	T	FFFT	00000

Cycle types 000, 001, 010, and 011 generate byte cycles in the XBUS I/O space. They are used only to access the registers inside the Intel 82380 integrated system peripheral.

Cycle types 100, 101, and 110 are used to generate low byte, high byte, and longword cycles in the XBUS memory space. They are used to access the local I/O devices and the ISA bus interface. Longword cycles are used to access word wide periperals.

A read with cycle type 111 generates an XBUS interrupt acknowledge cycle, and is used by PAL code to pull interrupt vectors out of the interrupt controller. A write with cycle type 111 generates a longword write cycle with arbitrary byte mask in XBUS memory space. Byte N (0 <= N <= 3) of the data in longword 0 is written if bit N (0 <= N <= 3) is set in the data in longword 1. A quadword store instruction is used to write the data and the mask as a unit. This cycle isn't used; it's a historical leftover from a proposed 32 bit wide local bus slot that was never implemented.

Reads from the XBUS are neither cached nor parity checked. On reads, the XBUS data appears on longword [0] of cache block, and longwords [7..1] are full of garbage. On writes the data on longword [0] is sent to the XBUS, independent of the state of the CPU chip's longword write masks.

Memory

Two banks of 64 bit wide longword parity protected memory are mounted directly on the system board. Each bank consists of 8 70nS page-mode SIMM's, 2 of which must be 9 bits wide, and 6 of which can be either 8 bits or 9 bits wide (actually 80nS SIMM's should be fine as long as you don't buy them from Hitachi; the specifications say Hitachi DRAM's can't run quite as fast in page-mode as the others). Both 1 megabyte deep SIMM's and 4 megabyte deep SIMM's are supported, so systems as small as 8 megabytes and as large as 64 megabytes can be configured. Software determines the presence and size of the memory banks by sniffing at them, and must deal with what is effectively a 24 megabyte hole in the physical address space on machines with 2 banks of memory and 1 megabyte deep SIMM's in bank 0.

CPU addresses with cA[33]=0 are memory addresses. Address bits cA[32..26] are ignored by the memory system (although they must be 0's for the cache to work properly), and address bit cA[25] selects between the two banks. Memory banks built with 1 megabyte SIMMS ignore address bits cA[24..23], and use address bits cA[22..05] to select the 32 byte block in the bank. Memory banks built with 4 megabyte SIMMS use address bits cA[24..05] to select the 32 byte block in the bank.

XBUS addresses with xA[31]=0 are DMA memory addresses. Address bits xA[30..26] are ignored, address bit xA[25] selects the bank, address bits xA[24..05] select the 32 byte block within the bank, and address bits xA[04..02] select the longword within the block. The programming of DMA transfers is simplified by the fact that a CPU memory space address can be converted into the equivelent XBUS memory space address by throwing away cA[33..32]. The byte enables are ignored by the memory on DMA writes, so all DMA buffers must be longword aligned and be full longwords in size.

Intel 82380

An Intel 80380 integrated system periperal attaches directly to the XBUS, and provides a number of system functions. On reset, the 82380 places its 256 byte-wide registers in the lowest 256 bytes of I/O space. The location of these registers can be changed, but there is no good reason to do so, since the 82380 is the only device in XBUS I/O space.

The 82380 determines which of its registers should participate in a cycle by looking at the xA[15..02] (I/O space addresses in the Intel world are 16 bits wide) and ~xBE[3..0]. When the CPU reads from the 82380 the single byte of data is replicated in all 4 bytes of the XBUS data longword. When the CPU writes to the 82380 it must place write data that would normally go in byte 3 in byte 1, and write data that would normally go in byte 2 in byte 0, because the 82380 knows that the 80386DX replicates the high word of its data bus on the low word of its data bus when it does a byte or word write.

DMA Channels

The Intel 82380 contains 7 independent DMA channels, which service DMA requests from the ISA bus. In true ISA style, they are numbered 0, 1, 2, 3, 5, 6, and 7.

The DMA channels can be run in many modes. The intention was that flow-through (2 cycle) mode would be used. In this mode the DMA channel would move data between the DMA channel and the ISA bus DMA periperal as bytes or words, and would move data between the DMA channel and memory as longwords. This would mesh well with the longword parity bits in memory and the fact that the memory would ignore the byte enables. The word "would" is used in all the descriptions since DMA transfers in which the peripheral is the source (that is, DMA transfers from the peripheral to memory) do not work the way they are described; contrary to all of the implications in the data sheet, a DMA channel only assembles bytes and words into longwords for the duration of a demand mode burst. Software can kludge around this bug by setting the source width to 32 bits, the destination width to 32 bits, the count to either 2 or 4 times the desired count, moving the data through a temporary buffer which contains a longword for each transfer unit, and manually packing and unpacking the data.

The requester for DMA cycles is always something on the ISA bus. The DACK signals and the AEN signal on the ISA bus are generated by a hardware decode of status signals generated by the 82380. The DMA controller's requester address must be programmed with a constant address that will generate an ISA I/O space cycle. The actual address bits transmitted on the ISA bus don't matter, since the peripheral is selecting on the DACK (and, in fact, the address decoder on the periperal has been disabled by AEN).

Interrupt Controllers

The Intel 82380 contains 3 cascaded 8259-like interrupt controllers which are used to prioritize interrupt requests from the DMA channels and the ISA bus. The ISA bus interrupt requests are wired to the 2 high-priority interrupt controllers in the cascade in the same relative arrangement as a IBM PC/AT, which may make emulation software a little easier.

Software can generate interrupt acknowledge cycles on the XBUS to pull interrupt vectors out of the interrupt controller. Software must mimic the interrupt acknowledge behaviour of an Intel 80386DX microprocessor (that is, it must perform two interrupt acknowledge cycles, and throw the result from the first one away).

Timers

The Intel 82380 contains 4 general purpose timers, each of which has a number of software selectable operating modes.
Timer 0 is intended to be used as a periodic interrupt source, and is internally wired to an input on the interrupt controller. It may be of some use to software, but cannot be used as the interval timer for ALPHA architectural reasons.

Timer 1 is used as the refresh timer. Low level software programs it to generate a refresh request every 15 microseconds, and leaves it alone.

Timer 2 is used as the tone generator. Its output is buffered, and fed to the speaker.

Timer 3 is intended to be used as a periodic interrupt source, and is internally wired to an input on the interrupt controller. It may be of some use to software, but cannot be used as the interval timer for ALPHA architectural reasons.

The input clock to the 82380 timers is fixed at 6.25MHz. The 6.25MHz is generated synchronously to the system clock in such a way that the documented race condition in the timer unit of the 82380 is completely avoided.

Refresh Generator

The 82380 refresh generator is triggered by the output of timer 1, and generates specially designated read cycles which refresh main memory, and can optionally refresh ISA bus memory.

Main memory is always refreshed. The refresh address output by the 82380 is ignored; main memory is refreshed using a cas-before-ras refresh cycle.

ISA bus memory is refreshed if ISA bus refresh is enabled in the ISA control register. If ISA bus refresh is enabled main memory refreshes normally, but the main memory controller does not generate a ready on the XBUS to terminate the cycle. Instead, the ISA bus interface generates the ready on the XBUS, after running a read cycle (with RFSH true) to perform the actual refresh. If ISA refresh is enabled the bus width of the refresh controller must be set to 16 bits. This will cause two read cycles to be run at each address (one with a byte mask of FFTT, and one with a byte mask of TTFF), which will generate an incrementing pattern on the ISA address bus.

The DRAM's used in the memory system require 8 RAS cycles before proper device operation is achieved (they also require a 100uS delay after power up, but this is guaranteed by the reset network). This can be achieved either by enabling refresh and waiting, or by simply reading each bank 8 times.

Ready Generator

The Intel 82380 generates ready for all cycles aimed at it. It can generate ready for other cycles as well, but this logic is not used; the wait state select inputs are strapped in the "disabled" position, and there is no need to initialize any of the registers in the ready generator.

Local I/O

The system board contains a number of integrated periperals. XBUS addresses in memory space with xA[31]=1 and xA[26..25]=00 address the local peripherals. Address bits xA[24..23] select the actual peripheral. Address bits xA[20..5] become address bits [15..0] at the VTI 82C106 combo chip and at the ROM. The combo chip and the ROM are wired onto byte 0 of the XBUS data longword.

VTI 82C106 (xA[26..23]=0000)

A VLSI Technology 82C106 ISA combo chip provides a number of low speed I/O devices.

Although the combo chip contains fully programmable address decoders, it reverts to hard-wired addresses on reset, and there is no good reason not to use the default addresses; serial line A at 0x3F8, serial line B at 0x2F8, printer at 0x3BC, keyboard/mouse at 0x060, and TOY/RAM at 0x170 (RTCMAP = GND).

Serial Lines

The combo chip contains 2 serial line interfaces. The serial lines are compatible to the WD16C450 used in newer ISA computers, and upward compatible with the NSC8250 used in the IBM PC/AT.

The serial lines have full modem control, and program selectable line format and data rate. Speeds range from 50 to 38.4K bits per second (the serial ports are only double buffered, so it may not be possible to run them at the highest rates). The external line drivers and receivers comply with EIA standard RS-232-C. The serial lines are ESD protected,

EMI filtered, and terminate in ISA standard serial port connectors (male DB9's).

The inerrupt request lines from the serial ports are or'ed together, and brought into a dedicated interrupt pin on the CPU. PAL code can either make the serial lines interrupt in a normal way (once per character), or it can manage ring buffers in main memory, and make the serial lines interrupt on interesting events. The second approach has the advantage that characters will not be lost if the CPU is running at high IPL for a long time.

Parallel Printer

The combo chip contains an interface for a standard IBM PC ("Centronics") parallel printer. The printer port can be used as either a printer port, or a a general purpose bidirectional I/O port.

The printer port is EMI filtered, ESD protected, and terminates in an ISA standard printer connector (a female DB25). The data wires and the STB wire have 2200pF capacitors on them, in the traditional style.

The interrupt request line from the printer port is brought into a dedicated interrupt pin on the CPU. If run in PS/2 mode (which is the only reasonable mode) then the printer interface generates its interrupt by clocking a flipflop with the printer's ACK signal, avoiding the bug in the traditional IBM PC printer interface which makes the interrupt more or less useless.

Keyboard and Mouse

The combo chip contains an interface for a standard IBM PC/AT or IBM PS/2 keyboard (the hardware seems to be able to support an IBM PC/XT keyboard, but if you read the specification closely, the keyboard select bit is not available in PS/2 mode, and you want to run in PS/2 mode to allow access to the mouse port, and besides, if you read the latest errata sheet for the chip, the IBM PC/XT keyboard interface has a bug in it, and it doesn't work), and for the standard PS/2 serial mouse.

The keyboard wires are EMI filtered, ESD protected, and are brought out to what appears to be a standard IBM PC/AT keyboard connector (a female 5 pin DIN). The mouse clock and data wires are ESD protected, and brought out to 2 extra pins on the keyboard connector (so the keyboard connector is, in reality, a female 7 pin DIN). The 5 volt power supply brought to the keyboard connector is short circuit protected by a PTC device, and is EMI filtered.

The interrupt request lines from the keyboard and mouse are or'ed together, and brought into a dedicated interrupt pin on the CPU. PAL code can either make the keyboard and mouse interrupt in a normal way (once per character), or it can manage ring buffers in main memory, and make the keyboard and mouse interrupt on interesting events. The second approach has the advantage that characters will not be lost if the CPU is running at high IPL for a long time.

TOY Clock

The combo chip contains a TOY clock which is program compatible to the Motorola MC14818A TOY clock. A battery pack keeps the TOY running when the ALPHA PC is turned off, and a charging circuit both powers the TOY and charges the battery any time the power is on.

Configuration RAM

The combo chip contains 50 bytes of battery backed up RAM which can be used to store configuration information.

Periodic Interrupt

The combo chip contains a source of periodic interrupts with programmable rate, one of which (976.562uS) meets the ALPHA architectural requirement. The periodic interrupt is brought into a dedicated interrupt pin on the CPU so that PAL can always take the interrupt, as required by the ALPHA archtecture..

Parallel I/O

The combo chip contains a number of parallel I/O ports, read and written indirectly through the keyboard and mouse interface. These parallel I/O ports are used to read switch closures and control LED's. The description of the parallel I/O ports in the combo chip's specification is more or less incomprehensible until you realize that it is making references to the port numbers on the Intel microcontroller normally used as a keyboard interface in IBM PC/AT computers.

The bits called P10 through P17 are inputs. The keyboard lock switch is brought in through P17 (also called KKSW). This input is low when the key is pointing at the locked padlock. The turbo switch is brought in through P16 (also called KCM). This input is low when the switch is depressed. The other inputs are unused, and are tied low.

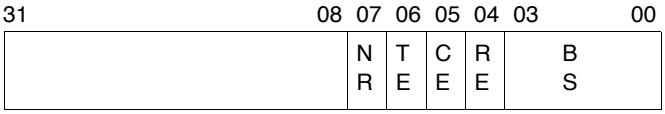
The bits called P20 through P27 are outputs. The HARD DISK LED is controlled by P21 (also called KA20). The other outputs are either unused or have dedicated functions.

ROM (xA[26..23]=0001)

A single 27C512 EPROM holds 64K bytes of bootstrap firmware. The initial profram loaded from the serial ROM copies the contents of the EPROM into memory before execution.

ISACR (xA[26..23]=0010)

The 8 bit ISACR is a read-write register which contains a number of control bits for the ISA interface. The ISACR is cleared by system reset.



The NR bit, if clear, holds the ISA bus in reset (that is, RESET is asserted), and holds the combo chip in reset (that is, the RES pin is asserted). The reset signal to the combo chip is synchronized to avoid the documented (errata 1.5) combo chip bug. SInce the ISACR is cleared by system reset, the ISA bus and the combo chip are also cleared by system reset.

The TE bit, if set, enables turbo mode. In turbo mode the ISA bus CLK signal runs at 12.5MHz (the system clock clock divided by 2) and has a 50/50 duty factor. In non-turbo mode the ISA bus CLK signal runs at 8.3MHz (the system clock divided by 3) and has a 66/33 duty factor. The turbo led on the front of the machine indicates the state of the TE bit.

The CE bit, if set, enables CPU interrupts on IOCHCHK conditions on the ISA bus (which includes data parity errors on the backup cache and the memory system detected during DMA read cycles). Software sets this bit to enable IOCHCHK interrupts, and toggles this bit false-then-true to reset the IOCHCHK interrupt. The IOCHCHK interrupt is brought into a dedicated interrupt pin on the CPU.

The RE bit, if set, enables the refresh of ISA bus memory.

The BS field supplies address bits [23..20] to the ISA bus (address bits [19..00] are directly encoded in the CPU address.

ISA I/O

The system board contains an interface to the ISA bus. Since the ALPHA PC is packaged in the low profile IBM PC case, the system board only needs to implement a single ISA slot. A T-card plugs into this slot, and the 3 16-bit and 2 8-bit ISA cards plug into the T-card. The T-card is supplied with the case.

XBUS addresses in memory space with xA[31]=1 and xA[26]=1 address the ISA bus. In this range most address bits are simply copied onto the ISA bus, but some specify additional details about the cycle.

Address bit xA[25] selects between memory space and I/O space. If xA[25]=0 then the ISA cycle is in memory space, and (S)MEMR and (S)MEMW are the data strobes. If xA[25]=1 then the ISA cycle is in I/O space, and IOR and IOW are the data strobes.

Address bit xA[24] selects between XT-style or AT-style timing. If xA[24]=0 then the ISA cycle has XT-style timing. The read or write strobe asserts on the falling edge of CLK at the start of what would be T2, and the SRDY signal on the ISA bus is ignored. If xA[24]=1 then the ISA cycle has AT-style timing. The read or write strobe asserts on the rising edge of the CLK at the start of what would be the first TS cycle, and the SRDY signal can terminate the ISA cycle at any time.

Address bits xA[23..21] encode the speed of the cycle. They specify the delay, in ISA bus T-states, between the start of the cycle and the T-state which contains the first sample of IOCHRDY. Since IOCHRDY is true by default, and since most ISA devices do not even connect to IOCHRDY, this lets the ALPHA program control the length of the cycle. Note that if the cycle is an AT-like cycle, and the device asserts the SRDY signal on the ISA bus, this field has no effect. Also note that if the cycle is an XT-style cycle, the shortest read or write strobe is 1 T-state long, unlike on a real 8088, where it is 2 T-states long.

Selecting the best cycle speed and cycle style is a tricky business, because the style and quality of design of ISA option cards is so variable. A good way to begin is to mimic the timing of an 8 MHz IBM PC/XT. In memory space an IBM PC/XT runs with what Intel calls 0 wait states, and what the ALPHA PC calls 1 wait state. In I/O space an IBM PC/XT runs with what Intel calls 1 wait state and what the ALPHA PC calls 2 wait states.

Address bits xA[20..2] are buffered, and driven onto the ISA bus as address bits [19..01]. Address bits [23..20] come from the ISACR. In addition, the SBHE signal on the ISA bus is driven by the XBUS signal ~xBE[1] (don't let the name fool you; the SBHE signal is active low), and address bit 0 is driven by the XBUS signal ~xBE[0].

The data path between the ISA bus and the XBUS is always 16 bits wide. On reads, all 16 bits of data are latched from the ISA bus, and are driven onto the low 16 bits of XBUS data. On writes, the low 16 bits of XBUS data are driven onto the 16 data bits of the ISA bus. The data repositioning required for 8 bit reads and writes to odd addressed locations on 16 bit devices is performed by software. The interface does not automatically break 16 bit wide transfers aimed at 8 bit devices into 2 transfers, nor is there any way to determine the width of a device (that is, there is no connection to the MEMCS16 and IOCS16 signals on the ISA bus).

The ISA bus on the ALPHA PC does differ from the ISA bus on a machine built with Intel microprocessors and custom ISA controller chips in a number of ways.

1) The address space is divided into 16 pages, each 1 megabyte in size. The 4 page select bits come from the ISACR,

and the 20 offset bits come from the address bus. The SMEMR and SMEMW signals are generated on the bus only if the BS field of the ISACR is 0000. This simplifies the address map, and does not seem to be much of a restriction, since generating addresses above 1M on the bus of a real ISA machine is clumsy at best.

2) On a real IBM PC the high order address bits (LA[23..17]) work differently on CPU cycles and on DMA cycles. On CPU cycles they are unlatched, and the ALE signal pulses asserted when they are stable. On DMA cycles they are latched, and the ALE signal is asserted throughout the cycle. On the ALPHA PC the high order address bits hold steady throughout the cycle, but the ALE signal is pulsed asserted on all cycles.

3) The MASTER signal is not implemented. The MASTER signal lets an ISA peripheral take complete control of the ISA bus, and, subject to a long list of restrictions, directly run byte and word width ISA cycles (which are nearly always aimed at memory). Because the CPU wants to have longword parity in the memory and in the cache, these byte and word width cycles would have to be implemented as (slow) read-merge-write cycles. In the ALPHA PC, all DMA must be done using the DREQ/DACK mechanisim, with the DMA controller in the Intel 82380 performing byte/word packing and unpacking. Of course, an ISA peripheral which requires the MASTER mechanism cannot be used.

4) Although the ISA bus interface can generate cycles with both XT-like and AT-like timing, the 8.33MHz CLK signal always has a 66/33 duty factor like the clock of an XT.

5) The OSC signal on the ISA bus comes from a 14.3181 MHz oscillator, and has no trimming capacitor. This causes no problems for modules which use it as a clock, but may cause the colors to be wierd on the CGA, which uses it to manufacture NTSC composite color video. Given that the resolution of a CGA isn't good enough for anything, it's hard to get very excited about this.

Firmware

The 64K byte EPROM on the system board provides contains initialization, console, and bootstrap firmware. The console and bootstrap firmware is unique to the Alpha PC. No attempt has been made to be compatible with the ALPHA SRM or any other ALPHA products.

Serial ROM Firmware

The ALPHA PC's serial ROM program is very simple (about 150 instructions).

1) The CPU chip itself is initialized.

2) Timer 1 in the 82380 is set to tick every 15 microseconds, the memory refresh controller's bus width is set to 16 bits, and memory refresh is enabled. This must be done before any other references on the XBUS, since until memory refresh is enabled the signal on the XBUS that designates refresh cycles (and stops XBUS devices from accidently selecting) is undefined. Each bank of memory is then read 8 times to ensure that the startup specification of the DRAM's is met.

3) The backup cache timing is initialized. The first 256K of memory is read, which initializes the backup cache's tag RAM's. Then all locations in memory are read and written, which initializes the backup cache's data RAM's and initializes main memory's parity bits. No attempt is made to size memory; 64 megabytes are always copied.

4) The contents of the EPROM are copied to main memory at location 0. The EPROM holds, among other things, PAL code (actually a derivative of an early version of the OSF PAL code) which, courtesy of the spaced-out PAL entry points, contains large blocks of 0's. These large blocks of 0's are squashed out by a simple run-coding scheme, which is undone on the way to main memory.

5) The instruction cache is flushed, and the EPROM firmware is started by jumping to location 0.

EPROM Firmware

The EPROM contains the bulk of the initialization, console, and bootstrap firmware. The expanded contents of the EPROM, along with data areas used by the firmware, occupy the lowest 128K bytes of main memory. Other software is loaded at location 20000 (that is, 128K from the bottom of main memory).

The console initialization code looks at the setting of the keyboard lock switch (on the front panel) to determine if it should communicate via serial port A (keyboard locked) or via the standard keyboard and video display (keyboard unlocked). The serial console runs at 9600 baud. The console keyboard driver looks at the keyboard and determines if it is an 84 key keyboard or a 101 key keyboard (and uses the appropriate translation tables). The console display driver uses the ATI 8514-ULTRA if it seems to be present in the system, otherwise it assumes that the MCT-VGA-1024 is is present.

With no arguments, the "help" command in the console prints the names of all of the commands. With an argument, it provides a brief synopsis of the commad and its parameters. There is no additional documentation.

Most console commands are examine and deposit commands of various types. Addresses supplied to console examine and deposit commands are always in natural form; the wild address manipulations needed are performed by the console. For example, the addresses used in ISA bus I/O space examine and deposit commands are ISA bus port numbers.

The SCSI bootstrap load command assumes an Adaptec AHA-1522 SCSI adaptor is plugged into the ISA bus. The Ultrix file structure on the specified SCSI device is searched for the named file, and the contents of the file are loaded

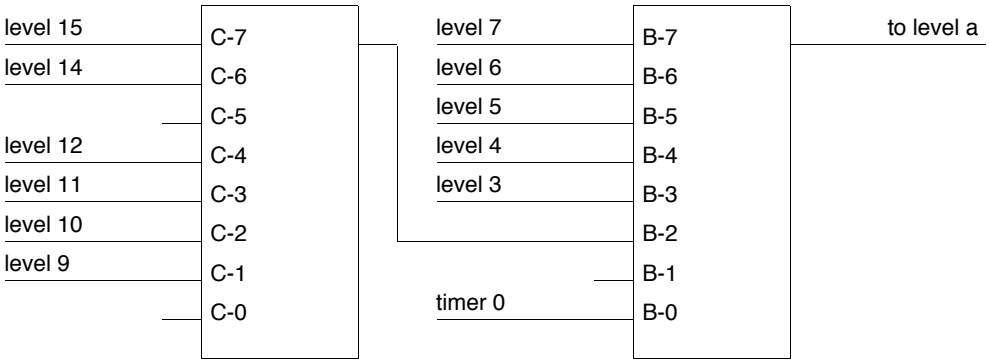
into memory. For historical reasons, the first 4 bytes of the file must contain the size of the image in bytes, and the actual image must begin at offset 512 in the file. This is a botch, and should be fixed.

The network bootstrap load command assumes that a Western Digital WD8013 Ethernet adaptor is plugged into the ISA bus. The ALPHA PC obtains its IP address using BOOTP, and then loads the named file using TFTP. The file is just a raw bits; the a.out header must be stripped off.

Interrupts

ISA Interrupt Assignments at 82380

This picture indicates how the ISA interrupt sources are brought into the B and C levels of the PIC in the Intel 82380. Note that the arrangement of ISA interrupt sources is exactly the same as the arrangement of ISA interrupt sources on the 8259A's of an IBM PC/AT. This may simplify emulation software.



Interrupt Assignments at CPU Chip

This table describes how the various interrupt sources are wired to the 6 interrupt request pins of the CPU chip. Three of the interrupt request pins have multiple sources, but, in all cases, a unique device driver can be associated with each interrupt request.

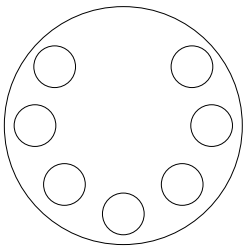
eIrq[n]	Source
0	Interval timer (82C106 IRQR)
1	PIC (82380 INT)
2	Errors (ISA IOCHCHK, XBUS bad data parity)
3	Keyboard and mouse (82C106 IRQK, IRQM)
4	Printer (82C106 IRQP)
5	Serial ports A and B (82C106 IRQA, IRQB)

Connectors

The bus connectors, the keyboard connector, and the power connector on an IBM PC/AT compatible system board are de-facto standardized. Any additional connectors are semi-standardized. We have pinned all of the semi-standardized connectors to be compatible with the Enlight Corporation's low-profile box, as determined by inspection.

Keyboard

The keyboard connector is a 7 pin DIN female. Pins 1 through 5 are electrically and physically compatible with the connector on the end of a standard IBM PC/AT keyboard. Pins 6 and 7 provide access to the IBM PS/2 compatible mouse port in the combo chip.

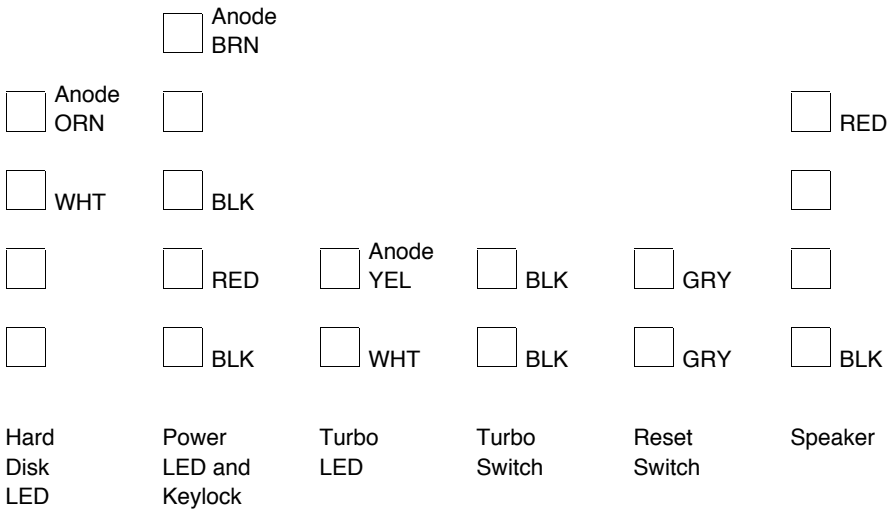


1 - Keyboard Clock
2 - Keyboard Data
3 - NC
4 - Ground
5 - +5
6 - Mouse Clock
7 - Mouse Data

System Board Headers

The switches and LED's on the front panel of the Enlight Corporation's low-profile box are connected to the system board via a handfull of push-on headers. The following picture shows how the wires get pushed onto the headers. The

picture is a copy of the relative arrangement of the header pins on the system board. The unlabelled squares are no-connects.



The cable connected to the RESET pushbutton plugs into a 2 pin header on the system board. The switch is hard-wired into the reset circuit; the ALPHA PC is held in reset when the switch is depressed.

The cable connected to the TURBO switch plugs into a 2 pin header on the system board. The state of the TURBO switch can be read via the P16 input bit of the combo chip. Some of the boxes we have purchased have a 3 pin header, colour coded red-black-green, instead of the 2 pin header, colour coded black-black. The green-black pair on the 3 pin header corresponds to the black-black pair on the 2 pin header.

The cable connected to the small speaker plugs into a 4 pin header on the system board. The speaker is driven by the buffered output of timer 2 on the 82380.

The cable connected to the TURBO LED plugs into a 2 pin header on the system board. The TURBO LED is lit any time the TE bit is set in the ISACR.

The cable connected to the HARD DISK LED plugs into a 4 pin header on the system board. The HARD DISK LED is driven by the P21 output bit of the combo chip.

The cable connected to the POWER LED and the keylock switch plugs into a 5 pin header on the system board. The POWER LED is lit any time power is applied to the system board. The state of the keylock switch can be read via the P17 input bit of the combo chip.

Things That Should Be Fixed

Of course, when you build something, you always discover things that should have been done differently.

1) The DMA channels on the Intel 82380 don't do byte assembly as described in the data sheet. The interface between the memory bus and the XBUS should be fixed to detect cycles in which the entire longword is not being written, and generate a read-merge-write cycle. It is probably still worth optimizing longword writes, since if you are transferring data in demand mode from a device that has a FIFO in it (like many SCSI chips) the controller would assemble the bytes, and the transfer to memory would run a little faster.

2) The ALPHA PC expects software to know which registers on a device are byte wide and which registers are word wide, and to use the appropriate addresses on loads and stores. In fact, it carries this expectation to an extreme, in that there is no way to even read the signals on the ISA bus that indicate if a 16 bit cycle was accepted. However, we have discovered by experience that when the manual for an ISA module says that a register is 16 bit it sometimes means that it is a pair of 8 bit registers (at address N and address N+1), and that it is expecting the hardware to break the 16 bit cycle into 2 8 bit cycles. These signals should be latched on (non refresh) ISA bus cycles so they can be read by software.

3) The speaker isn't loud enough.

4) The ROM is 64K bytes. Although we managed to squeeze everything in, the 64K byte limit caused us to all go into system programming mode earlier than we wanted to. If we changed the 28 pin ROM socket to a 32 pin ROM socket, and sourced the extra bits of the address from the BS field of the ISACR, then we could use the 27C010 (128K x 8) ROM.

5) The box manufacturer uses two different cables for the TURBO switch. The original boxes used a 2 pin header. More recently obtained boxes use a 3 pin header. The system board should be changed to use the 3 pin header, since the 3 pin header is pinned in such a way that the 2 pin header can be plugged onto it.

6) The hole in physical memory in 16 megabyte systems is a pain because the system software is badly designed. I

could add an "alternate address map" mode bit to some control register, and make the bank decode work off address bit 23 instead of address bit 25. Machines with an 8 megabyte bank and a 32 megabyte bank would use the normal mode, and always put the 32 megabyte bank in bank 0 and the 8 megabyte bank in bank 1.

7) We can make a version of the board that can run both an ALPHA and an NVAX, with only changes to the PAL's (that is, with no wires). The idea is that we create a new signal that indicates if a cycle is a memory cycle or an I/O cycle, and use this signal on all the state machines instead of cA[33]. On ALPHA systems this is determined by cA[33]. On NVAX systems this is determined by cA[31..29]. We also need to bring more address bits to the BUSC PAL so that the NVAX address map can be generated; to do this we need to delete the useless cycle that allows arbitrary byte masks on XBUS writes. With these changes you can transform an ALPHA board into an NVAX board by swapping BUSC, CPUL, and whatever PAL ends up generating the signal that indicates an I/O cycle. It would be nice if this could be BUSC.

8) Although this is cosmetic, I figured out why the 16L8's and the 22V10's don't like up. They do line up on the silk. The two packages don't have pin 1 at the same position relative to the edge of the bounding box. Fix the package, and the parts will like up in an attractive way.

Interesting Factoids

In addition, we have learned lots of interesting things about the IBM PC bus in general, and the IBM PC expansion boards that we ave tried to use in particular. In general, the quality of documentation is wretched. Fortunately, most of the board use off-the-shelf parts, and you can get most of the details from the appropriate semiconductor vendor.

1) The Intel 82380 manual (which is huge, but quite good, except for the fact that its description of byte assembly is totally wrong, but that's the designer's fault) makes a passing comment that the 2 IACK cycles generated by the 80386DX are 4 (that is, 8 ticks of the system clock) apart. The straightforward ALPHA program gets the 2 IACK cycles closer together than this, and it matters; sometimes the 82380 hangs. A short delay is needed.

2) When enabled (by setting bit 2 in port 0x03F2), the NS DP8473 floppy disk controller chip generates an internal "reset has gone away" interrupt, which must be cleared by sending a sense interrupt status command. However, if you send the sense interrupt statuus command to quickly after deasserting reset, the floppy disk chip hasn't generated its interrupt yet. The sense interrupt status command is rejected as illegal, and then the chip more-or-less hangs because it has this interrupt pending. A delay is needed.

3) While on the topic of the NS DP8473 floppy disk chip, there is a trap for the unwary in the recalibrate command. Unless you go to extra trouble and put the chip into a special mode, the recalibrate command will not generate more than 77 step pulses (history lesson; 77 is the number of tracks on an 8 inch floppy). This means that the recalibrate command gives up if the heads are positioned in the last 3 tracks of the 80 track IBM PC floppy. I fixed this in my drivers by blindly retrying the recalibrate command if it gets an error.

4) Most of the time when a manual for an IBM PC peripheral says "16 bit" it means "connected to both sides of the data bus, and capable of accepting a 16 bit cycle". But not always. Sometimes "16 bit" only means "connected to both sides of the data bus". Such boards do not assert MEMCS16 or IOCS16 on the ISA bus, and depend on the platform to split the cycle into 2 byte wide cycles. The PIX_DATA register on the ATI 8514 Ultra card has this property.