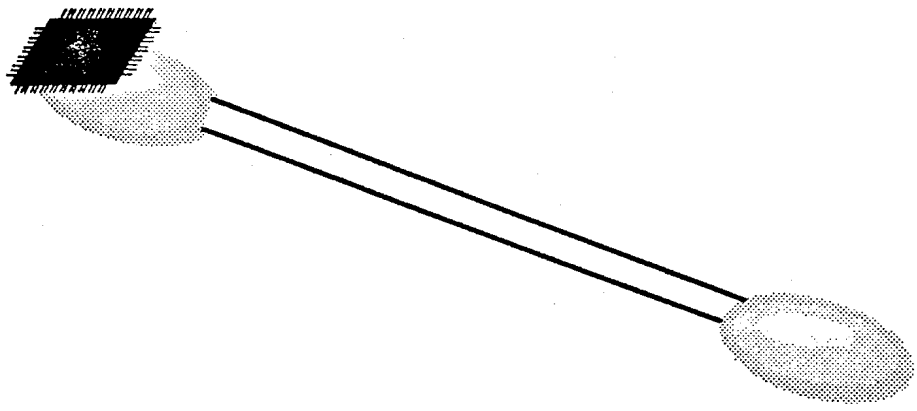
HEWLETT PACKARD

WAX

External Reference Specification

for the Wax I/O ASIC



List of Figures

List of Tables

1 Overview

1.1 Description

This document describes the external interface of the 1FT4-0001 ASIC, Wax. Wax is the internal code name for an ASIC which provides I/O on the GSC. Wax is designed using a standard-cell design methodology utilizing Hewlett-Packard's CMOS 26B libraries. Wax is packaged in a 240 pin MQUAD package.

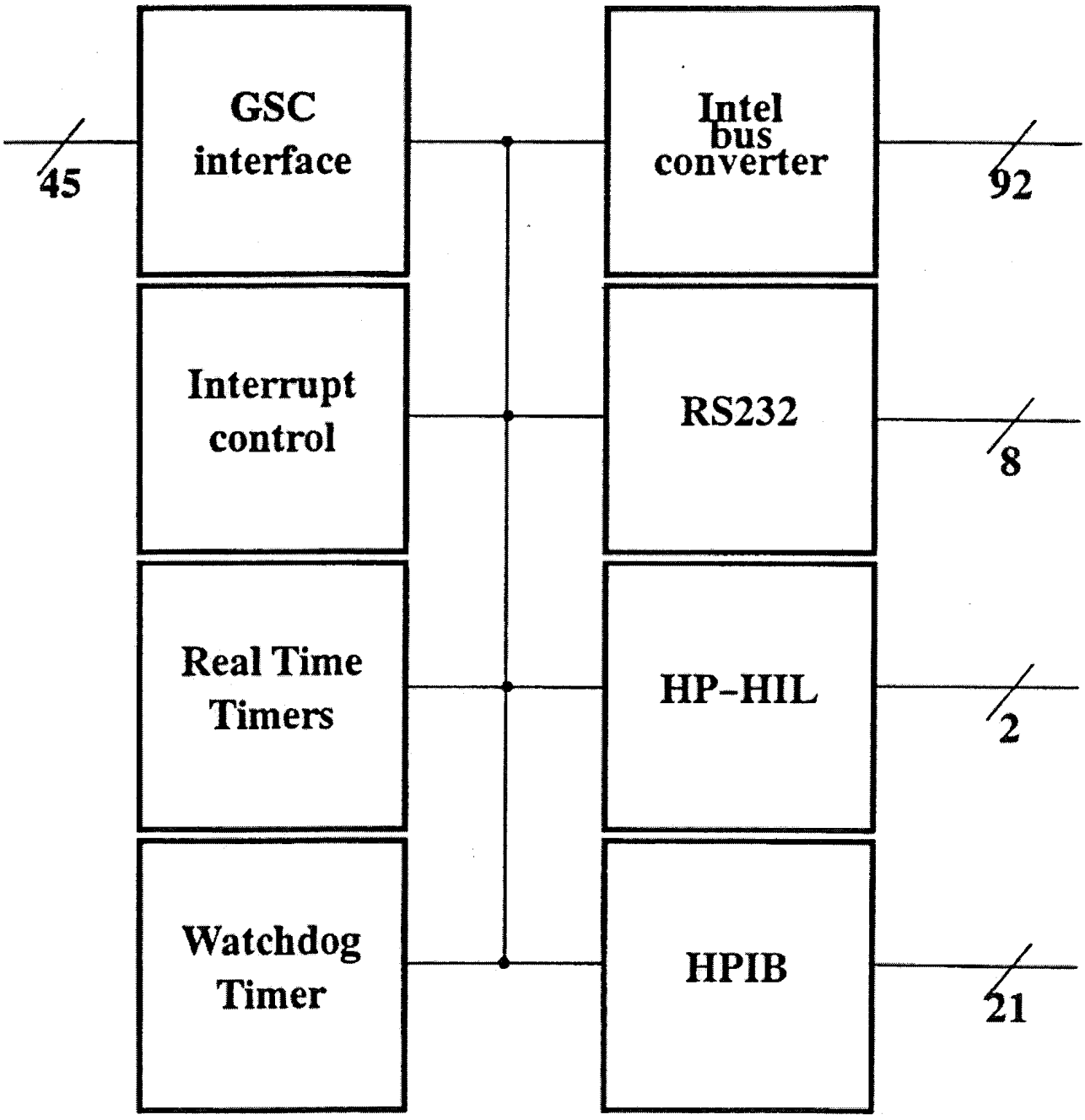


Figure 1 Wax Block Diagram

Wax provides an interface between the GSC and Intel buses. The Intel buses to which Wax connects include the i486, i8086, and a special multiplexed version of the i8086 bus. Also, a serial port, a IEEE 488 compatible HPIB port, HP-HIL human interface port, and a couple of special purpose timers are integrated into Wax.

A test access port (tap) is included in the Wax ASIC. This tap is compatible with the JTAG 1149.1 specification. It allows access to the boundary of the ASIC for board testing. The tap also has access to all internal flip flops through a scan chain. This allows the Wax ASIC to be tested.

Following are descriptions of the circuitry in the Wax ASIC, specifications for the ASIC, connection information for the ASIC, and programming information. Each section of the Wax ASIC is described, and if the circuit requires external connections, a sample circuit is included showing how to use that section. Register definitions are included as are some notes on how to program that specific section.

This is revision 1.0 of the Wax ERS, last modified on May 3, 1993.

1.2 Reference Material

Below is a list of documents which may be used to gain additional understanding of the concepts and devices used in a system with the Wax ASIC:

- Gecko I/O Subsystem ERS
  HP A-A2263-66510-31
- TACT84500 EISA Chip Set Designer's Handbook
  Texas Instruments Inc.
- EISA Bus Specification, Revision 3.12
  BCPR Services Inc., 1400 L Street N.W., Washington D.C. 20005
- NS16550A Data Sheet
  National Semiconductor Corp.
- 8042 Firmware Documentation Revision B
  HP A-1820-4784-2
- Cerberus ERS
  HP A-1RD2-6201-2
- TMS9914A General Purpose Interface Bus Data Manual
  Texas Instruments Inc.
- The Test Access Port and Boundary-Scan Architecture
  IEEE Computer Society Press #2070

2 EISA Conversion Module

2.1 Overview

This module is called an "EISA converter" only for simplicity's sake, and because its primary purpose is to interface a host's GSC (Gecko System Connect) bus to the EISA bus, to give the system a standard expansion I/O bus. But because of EISA's complexity, this module does not directly generate EISA bus signals; it generates an i486-like bus, which the "EBCU" (EISA Bus Control Unit) and "EPCU" (EISA Peripheral Control Unit) of TI's EISA chip set then convert into EISA (and ISA). Thus, in its primary mode of operation, this is an i486 bus conversion module rather than an EISA conversion module.

The first exception to this is the EISA data path. Normally, an EISA system would use the third of TI's EISA chips, the "EDPU" (EISA Data Path Unit), to interface the host data bus to EISA's data bus, and to perform the necessary byte lane copying on EISA's data bus. Wax incorporates this functionality of the EDPU, and therefore can interface directly to the 32-bit EISA data bus. This can save the cost and board space of an extra 160-pin ASIC at the expense of only a small increase in Wax's complexity. But there are electrical limitations to Wax's built-in EDPU; and if Wax is to be used in a system with more than 4 EISA slots, or a system that places Wax more than a few inches from the EISA connectors, an external EDPU should be used (and Wax should be configured to provide an i486-like data bus that is fed into the external EDPU).

The second exception is an alternate mode of operation that can be selected at power-up. This mode causes Wax to generate an 8086-like bus rather than its normal i486-like bus. Primarily differing in the definition of timing and control signals, this mode may allow Wax to directly connect to simple devices that were designed to interface to the ISA bus. The third exception is a variation on the converter's 8086 mode. This variation is designed specifically to let Wax connect directly to TI's TMS380C16 Token Ring controller chip. It multiplexes DMA addresses onto the data lanes, and makes other signal definition changes to help eliminate external glue.

The converter can also be configured to support GSC+ bus transactions on the host side—these include the pending of host or EISA transfers to increase total host bus utilization, and retrying of host transfers to EISA while Wax is busy running another transfer.

So, this converter module is really a GSC/GSC+ to EISA/ISA/i486/8086/Multiplexed 8086 Bus converter; but to save ink and my fingertips, I refer to it simply as the "EISA converter."

2.1.1 Notes on Notation

First, EISA is (to some degree) a superset of ISA; both EISA and ISA boards can be used in an "EISA" system. Wherever this document refers to a device on the EISA bus that could be either an EISA or an ISA board, it will refer to it as an (E)ISA device. "ISA" will refer to ISA-only boards or features, and "EISA" typically refers to boards or features that take advantage of EISA's extensions to the original ISA specification.

Unless otherwise indicated, all numbers in this section of the document are in decimal. Hexadecimal numbers are indicated by a leading "$". Signal names, module names, address and data values, etc. are set in a `typewriter-style_font` to help distinguish them.

2.1.2 Design Objectives

First, this design matches the software interface of the EISA converters on other PA-based controllers and workstations, such as Pace and Scorpio. This includes the addresses at which (E)ISA memory and I/O devices are accessed, how the address map is set up (and used by EISA or ISA devices), and how interrupt acknowledge cycles are run.

Second, this design allows EISA- or ISA-mastered and DMA transfers to run at high speed while using the host bus relatively efficiently. In particular, independent transfers on (E)ISA and the host bus are allowed to run

concurrently. Also, (E)ISA reads from system memory are pre-fetched in 8-word block transfers, so that up to 7 subsequent sequential reads can be satisfied from Wax's internal memory; and up to 8 (E)ISA writes to system memory are buffered in Wax, allowing the (E)ISA cycles to run at full speed even when the host bus cannot immediately be acquired, and allowing more efficient block transfers to be run on the host bus once it is acquired. Additionally, when Wax is in KIOSC mode, it allows the host to "pend" outbound DMA data transfers, letting the KIOSC bus be used for other things while host memory finds the requested data.

Note that host-originated transfers are *not* buffered as (E)ISA master- or DMA-originated ones are. Typically, such transfers will not be large and sequential (as (E)ISA DMA transfers would be), and will be much less frequent (e.g., large blocks of data to or from SCSI or LAN will instead be transferred by DMA), so it makes less sense to buffer them. Even without buffering, CPU accesses to (E)ISA via the GSC bus will run ISA cycles at full speed and EISA cycles at about half of their theoretical top speed. In fact, it is quite probable that the (E)ISA slave will be the bottleneck and will respond much slower than Wax could handle. So for this reason, when Wax is in KIOSC mode, it will "pend" all reads from the host to (E)ISA, releasing the KIOSC bus for other use until the converter actually has data available.

2.1.3 Basic Operation

This conversion module supports the following functions:

- CPU reads from and writes to (E)ISA I/O addresses—also note that address scrambling is performed for accesses to *ISA* I/O space, so that each ISA board gets its own protectable page in the CPU's address space.
- CPU reads from and writes to (E)ISA memory addresses—only 55.5 MB out of (E)ISA's 4 GB address space is accessible to the CPU, but the accessible ranges include subsets of each of ISA 20-address-bit, ISA 24-address-bit, and EISA 32-address-bit memory.
- CPU reads from and writes to the address map RAM—after the CPU sets it up, this mapper lets (E)ISA master and DMA devices access any desired pages in system RAM.
- CPU reads from and writes to the Lock Control Register—this register lets the CPU run a sequence of "locked" (undivided) cycles on the EISA bus.
- CPU reads from and writes to the FIFO Enable Register—this register lets the CPU flush the inbound buffer and clear the outbound buffer, or disable data buffering altogether.
- CPU reads from the Interrupt Acknowledge Register—this is the way the CPU runs interrupt acknowledge cycles to the (E)ISA interrupt controller and obtains the "interrupt vector" number of the highest pending (E)ISA interrupt.
- (E)ISA master and DMA reads from and writes to the mapped address space—these accesses are passed through to the system RAM, after having their addresses translated by the appropriate address map entries. If these accesses are not "locked" on the EISA bus, they can be buffered within Wax: writes do not immediately occur to system RAM, and reads can be satisfied from data which was pre-fetched from system RAM. If these accesses are "locked" by the assertion of EISA's LOCK signal, the lock is propagated all the way back to system RAM, and no other masters can run bus cycles or access system RAM between the locked EISA cycles.
- Interrupt requests and non-maskable interrupts from (E)ISA—after flushing and clearing the data FIFO to ensure that the system sees consistent data, these are simply passed to Wax's interrupt controller where they are dealt with appropriately.

2.1.4 Hardware Organization

On the "outside," this module generates and accepts i486-like control and address signals to communicate with TI's EISA chip set, it accepts data path control signals from the EISA chip set, and it directly connects to

(E)ISA's data bus. In its alternate 8086 mode, it instead generates ISA-like control, address, and data signals that can be directly connected to TI's IBM Token Ring controller chip. On the "inside," the module communicates over Wax's internal-GSC bus, which is very similar to the external GSC (or KIOSC) bus, including multiplexed address and data lines, but there is generally a one-clock delay when signals enter or leave the Wax chip. When the EISA conversion module owns the host bus, it may generate single transfers or 2-, 4-, or 8-word burst transfers; when it is a slave, however, it will respond correctly only to single-word transfers (extra words will be ignored during writes and invalid during reads).

The bulk of the circuitry is synchronous to the host bus clock, which is expected to run at a frequency somewhere in the range of 25 MHz to 37.5 MHz. Unfortunately, the EISA chip set cannot operate the EISA bus at full speed with a "host" clock at most of these frequencies. Rather than take an approximately 10% performance hit on all (E)ISA bus cycles (including those between two (E)ISA devices that don't even involve main system memory), the EISA chip set and the outside end of this conversion module are driven by a separate 33 MHz clock. Thus, all of this module's i486 signals must be re-synchronized to the appropriate clock when they enter or leave the chip. Between the module's data buffering, and the fact that EISA's backplane clock operates at only 1/4 the frequency of the EISA chip set's clock, this re-synchronizing generally has a minimal effect on performance.

Figure 2 shows the complete host-to-EISA conversion path, including the conversion module within Wax, and the two TI EISA chips:

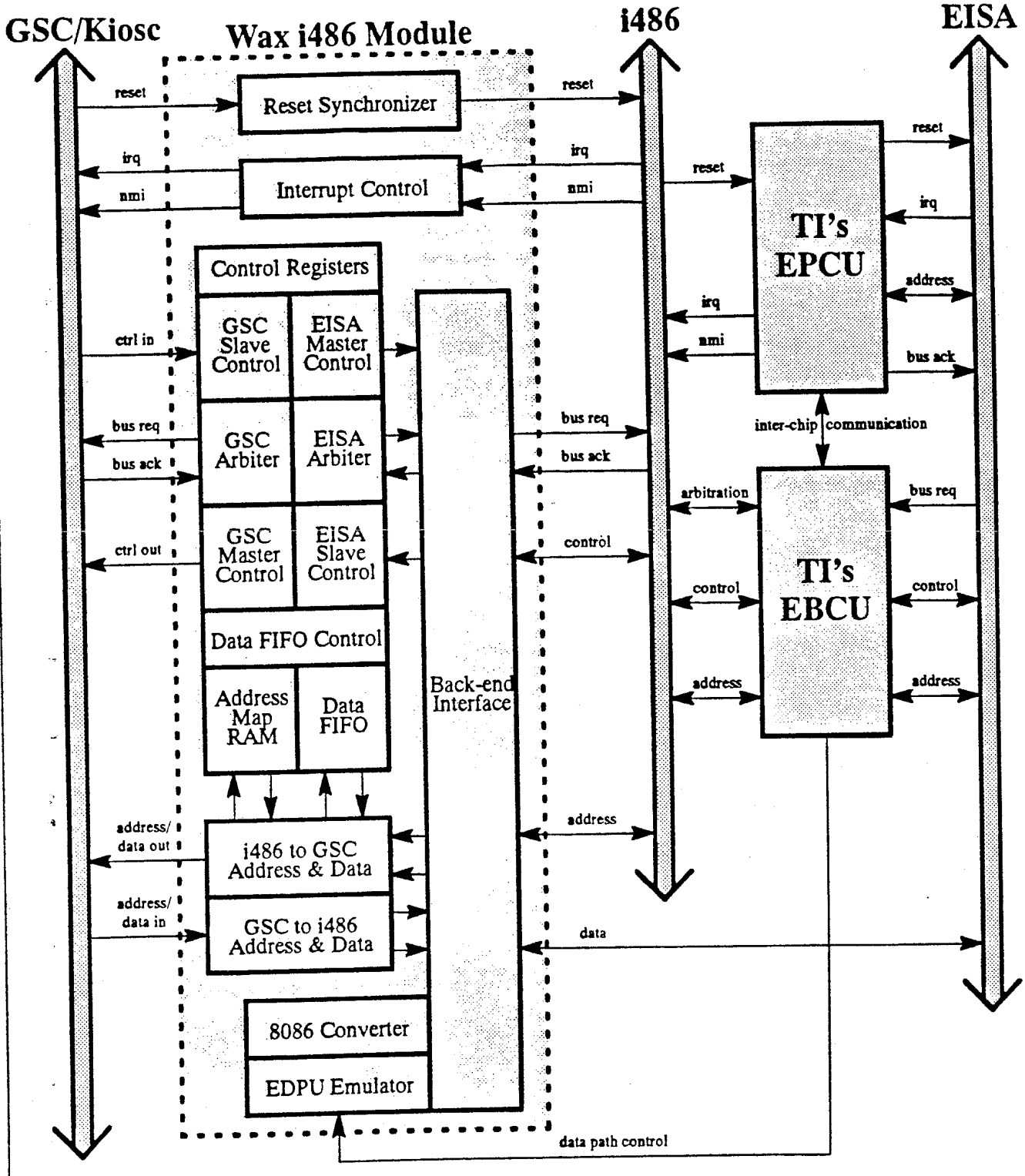


Figure 2 Block Diagram of the Complete GSC-to-EISA Interface

2.2 Functional Description

This section describes how software communicates with (E)ISA expansion boards.

2.2.1 General Information

The central, unavoidable problem with an EISA converter on a PA-based computer is that the CPU is big-endian, whereas the (E)ISA bus is little-endian. In other words, they disagree on the location of the most significant byte within a 16-bit half-word or a 32-bit word.

There are a couple of potential solutions to this problem. Unfortunately, there is no possible solution that would make all software designed for a native-EISA system work without change. The alternative implemented here (and in all other HP workstations) reverses the bytes within the word-wide data path between GSC and the (E)ISA bus:

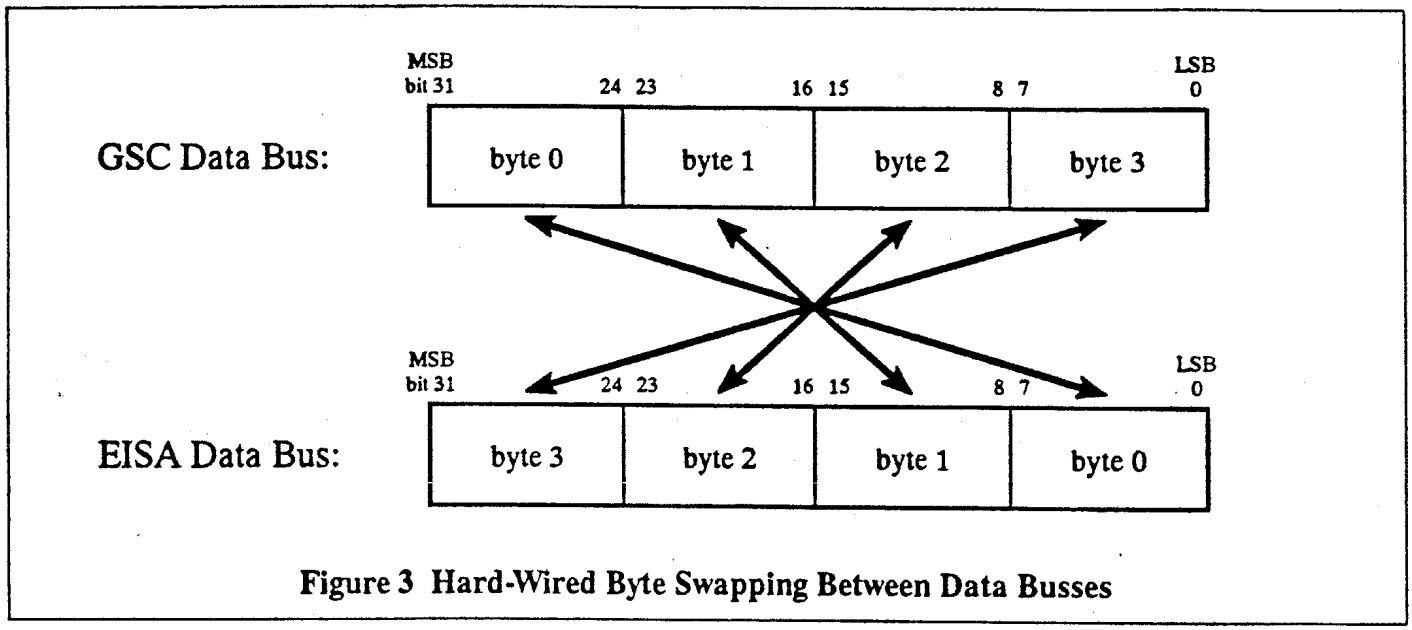


Figure 3 Hard-Wired Byte Swapping Between Data Busses

This choice has the advantages that mass-storage devices written with an (E)ISA I/O controller on this system can be read by a controller on a native-(E)ISA computer (and vice versa) without any data pre- or post-processing, and that all accesses to (E)ISA (byte, half-word, and word) can be run at their "expected" addresses, without having to be modified according to what part of a word is being accessed. This choice's disadvantage is that software must manually swap the bytes within any half-word or word access to (E)ISA:

> To access the word of data $*aabbccdd* in an (E)ISA memory or I/O slave, the CPU would read or write the word $*ddccbbaa*.
> To access the half-word of data $*aabb* in an (E)ISA memory or I/O slave, the CPU would read or write the half-word $*bbaa*.

For example, if you want to write the word `$12345678` to an (E)ISA board, you must actually write the value `$78563412`. If you want to write the half-word `$1234` to an (E)ISA board, you must actually write `$3412`. Reading from an (E)ISA board is similar; you must swap the bytes returned by the read before using the value. Under HP-UX, drivers should make use of the kernel's EISA service routines that read and write half-words and words and perform this byte swapping for you.

Figure 4 shows the converter's address space as seen *by the CPU*:

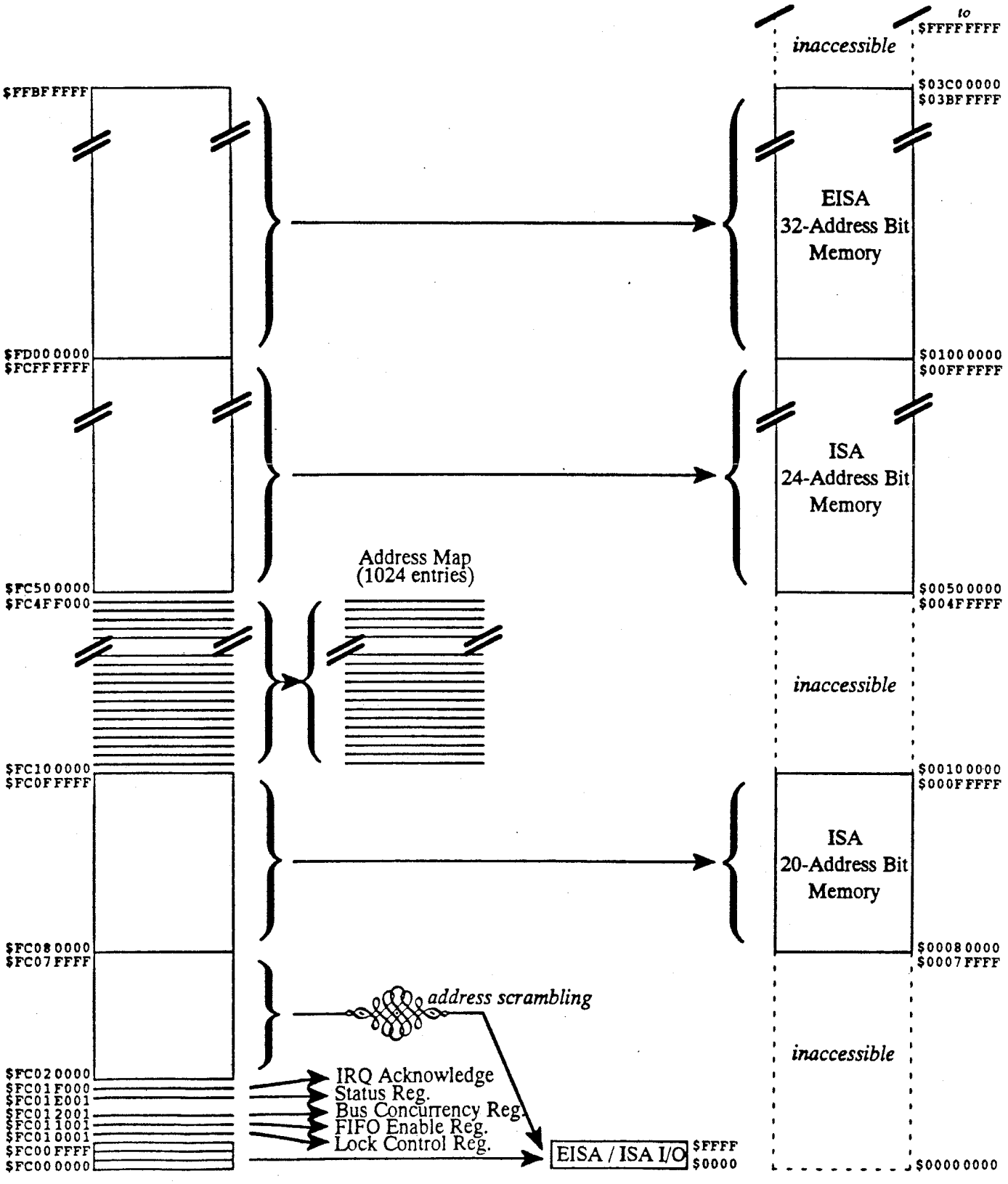


Figure 4 Accesses from the CPU

Figure 5 shows (E)ISA *memory* space as seen *by an EISA or ISA master* (note that the master can access (E)ISA *I/O* space too; the converter doesn't get involved in those accesses at all):

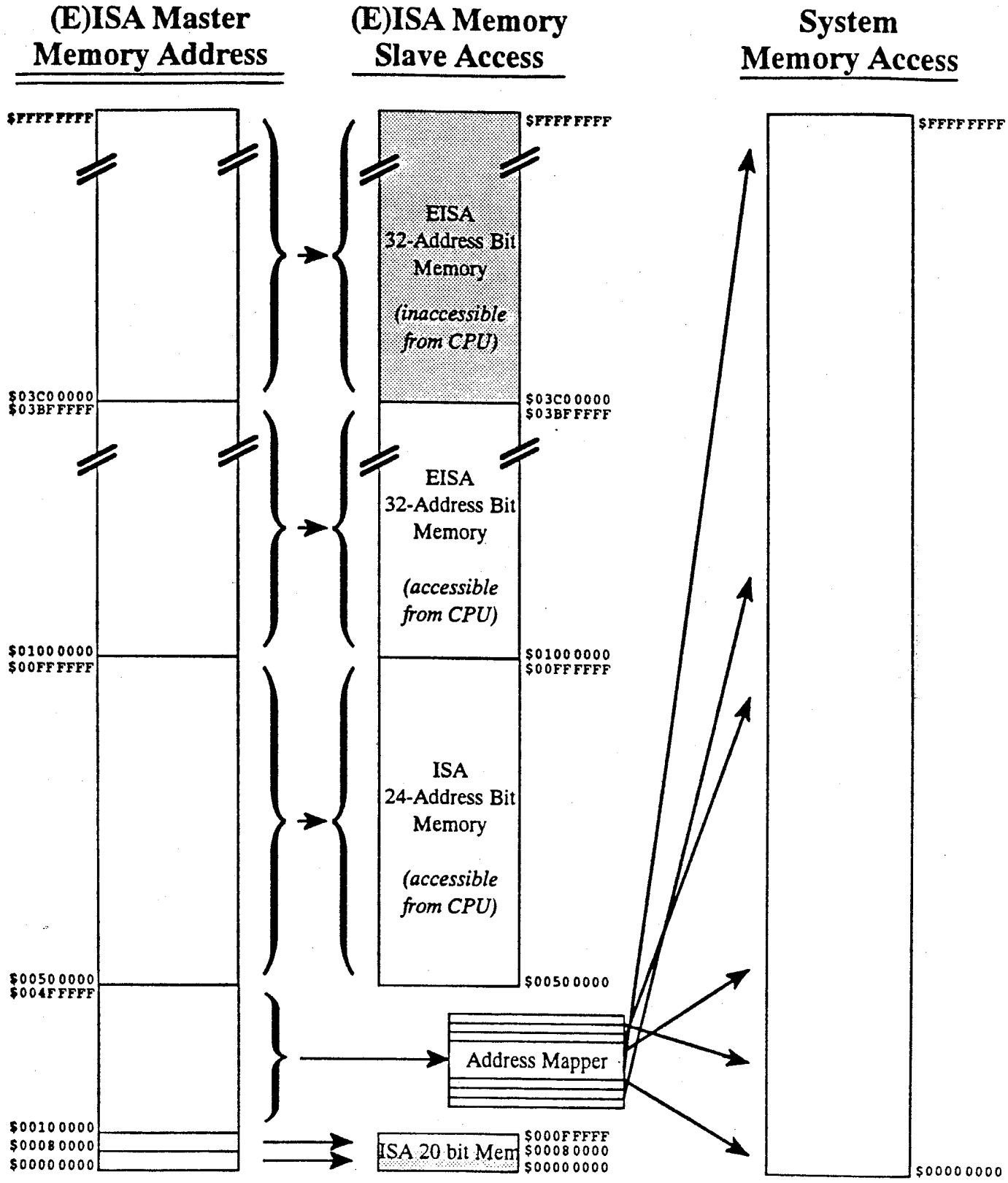


Figure 5 Accesses from an (E)ISA Master

2.2.2 CPU Accesses to (E)ISA Slave Devices

The CPU can access all of (E)ISA's I/O space and much of its memory space as indicated in the above diagram. As a slave, the converter will correctly respond only to single (or partial) word reads and writes; if a multiple word read is issued, the converter will provide only one valid word of data and then supply garbage; if a multiple word write is issued, the converter will accept only the first word and then ignore the rest.

If Wax is used in a KIOSC system that supports pended reads, any read to an (E)ISA slave will be pended, and Wax will release the host bus. As soon as the (E)ISA cycle has completed and data is available, Wax will arbitrate for the KIOSC bus and run a read response cycle to provide the data to the host. While a read is pending, Wax will "retry" *all* accesses to the EISA converter address space.

If a partial-word read or write is issued, the converter may have to perform more work, since GSC allows all possible combinations of byte enables whereas EISA allows only contiguous bytes to be enabled. If a host bus cycle is issued with a set of byte enables that would be invalid on EISA, the host transfer is split into two EISA transfers. GSC also allows transfers to be run with no valid byte enables; EISA disallows such transfers, and so if one is issued, the converter will complete it on the host bus but not pass it on to EISA.

Aside from the above issues and the byte-swapping problems, CPU accesses to (E)ISA *memory* are straightforward. As implied by the above CPU address space diagram, the accessible regions of (E)ISA memory are directly mapped into the CPU's address space:

> To access (E)ISA memory address x (where x is within either of the ranges $00080000-$000FFFFF or $00500000-$03BFFFFF), the CPU reads or writes address $FC000000 + x$.

In this system, (E)ISA memory devices *can* be configured at other address ranges; if they are, they will be inaccessible to the CPU, but they can be used by EISA or ISA master cards for local memory.

CPU accesses to (E)ISA *I/O* devices are not so straightforward. The complications arise from a desire to protect one board from inadvertent or malicious accesses by unauthorized drivers and users. Unlike the (E)ISA memory space, which is divided between boards in large chunks, the I/O space is much more compact. In fact, *all* of ISA's I/O space falls within a 4kB block, the size of one MMU page—without additional hardware protection in Wax's converter, it would be impossible to protect individual ISA I/O boards.

It turns out that, if we extract the *ISA* I/O ranges from (E)ISA I/O space, we are left with built-in I/O and *EISA* slot-specific I/O addresses all naturally aligned on MMU page boundaries, with one board per page. Thus, these parts of the I/O space can be directly mapped from a CPU address:

> To access EISA or built-in I/O address x (where bits 8 and 9 of address x are both 0), the CPU reads or writes address $FC000000 + x$.

Graphically, this part of the converter's address space appears in Figure 6 . This converter will refuse to run CPU accesses at addresses that are marked as invalid; such accesses will be completed on the host bus (i.e., the converter will issue a READY), but no (E)ISA cycle will be run:

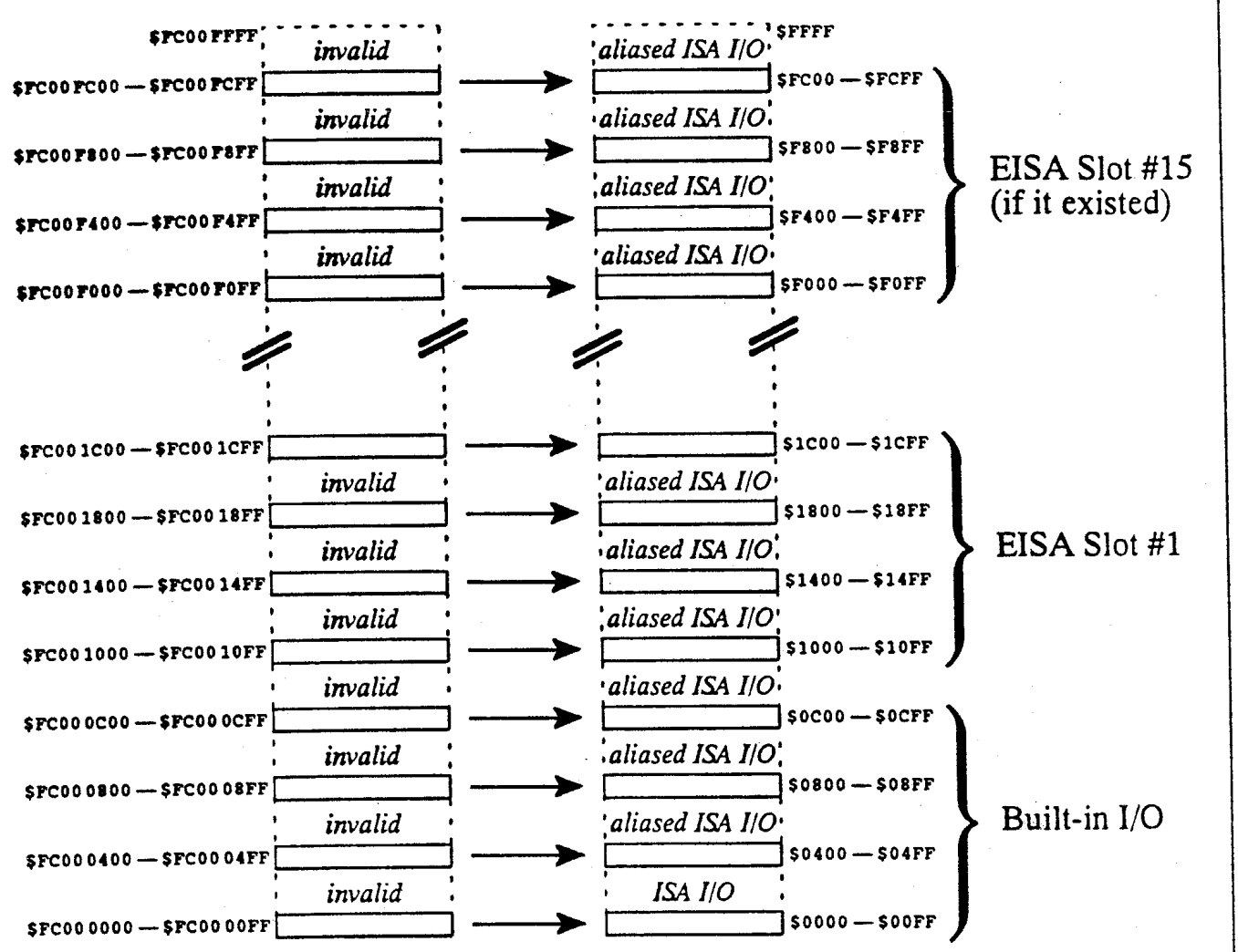


Figure 6 CPU Accesses to EISA & Built-In I/O

Now, Figure 6 shows several blocks of ISA I/O space: the primary ISA I/O range exists at `$100-$3FF`, but the EISA spec also defines aliases for this range every $400 bytes, from `$500-$7FF` through `$FD00-$FFFF`. The problem with this I/O space is that primary ISA I/O space is divided into 8-byte chunks, and each ISA board can use one (or more) of these chunks. Thus, to protect cards from one another, every group of 8 bytes in this region must be mapped into a separate 4kB MMU page. All of the aliased I/O addresses are made accessible to the CPU (just in case some I/O board needs to use them to distinguish between its registers); but because most ISA I/O boards don't even look at address lines higher than `SA[9]`, we can safely assume that each aliased I/O address accesses the same board as its corresponding primary I/O address, and map the aliased I/O address into the same MMU page as the primary address.

Figure 7 shows the address scrambling method chosen by this converter (and the Series 300, 400, and 700 EISA converters) to provide the proper protection. Note that three bits of the CPU address are given as "X"s; these bits are don't-cares in this design, but all software *should* set these bits to 0 for consistency. Also note that one or both of CPU address bits 17 and 18 *must* be set to 1 to access ISA I/O (otherwise the CPU address will point to EISA I/O space or to the converter's internal register space):

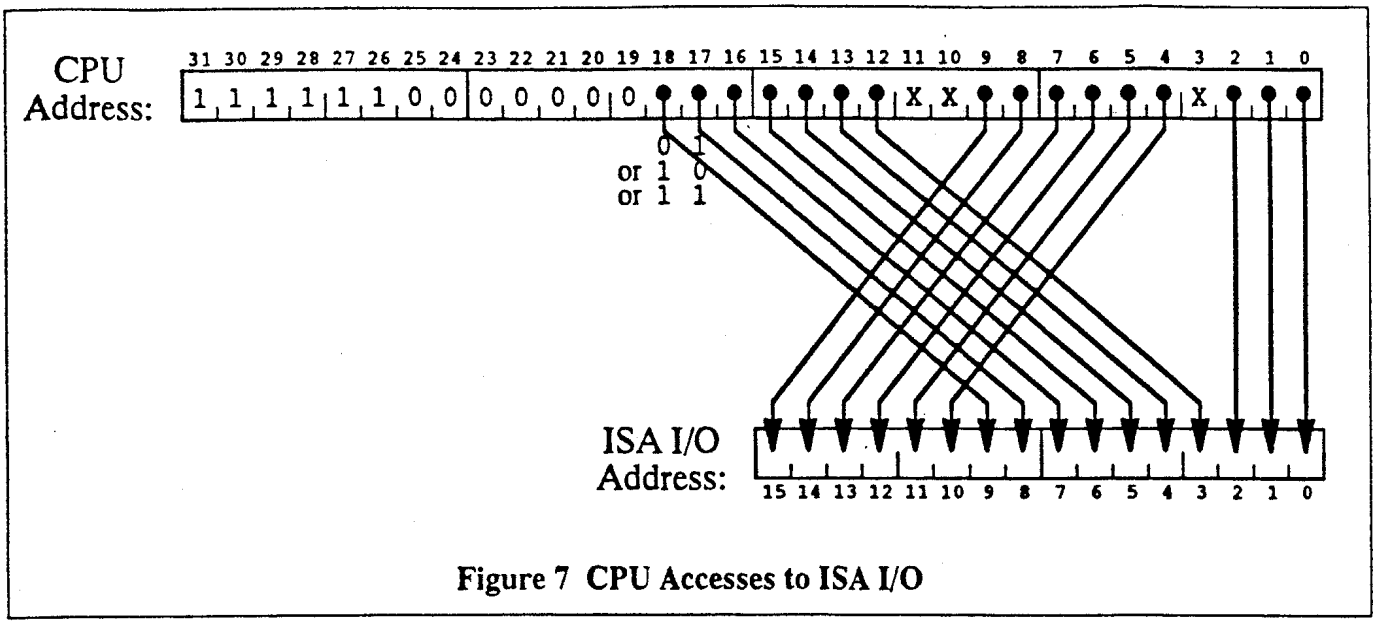


Figure 7 CPU Accesses to ISA I/O

For example,

- the CPU reads or writes address `$FC020000` to access ISA I/O address `$0100`;
- the CPU reads or writes address `$FC0200C0` to access ISA I/O address `$3100`; and
- the CPU reads or writes address `$FC021001` to access ISA I/O address `$0109`.

In general:

> To access ISA I/O address *x* (where bit 8 or bit 9 within address *x* is 1), the CPU reads or writes address
> `$FC000000 + ((x & $FC00) >> 6) + ((x & $03F8) << 9) + (x & $0007)`.

2.2.3 CPU Locked Accesses to EISA Slaves

The CPU can use the converter's internal Lock Control register to run a sequence of locked cycles to the EISA bus. Although the ISA bus has no notion of locked bus transactions, the Lock Control register can also be used during accesses to ISA slaves to ensure that the TI chipset will not take mastership away from the CPU in the middle of a sequence of transfers.

The Lock Control register is accessed by the CPU as the least significant bit (bit 0) at byte address `$FC010001`. When set to 0, this bit will cause Wax to immediately request ownership of the (E)ISA bus. Once Wax acquires the bus, it holds it until the CPU sets the Lock Control register back to 1, and locks all intervening CPU accesses to EISA.

> To run a sequence of indivisible (E)ISA bus cycles, the CPU writes the byte `$00` to address `$FC010001`, performs the (E)ISA reads or writes that should be locked together, then writes the byte `$01` to address `$FC010001`.

Note that Wax allows a write to this register to complete *before* Wax actually acquires the EISA bus. For running locked cycles this is desirable, since EISA doesn't really need to be tied up until the CPU's next access to an EISA device. However, if this bit is used just to keep (E)ISA slaves out of system memory, software needs to know when Wax has actually locked the EISA bus. It can do this by running any cycle for which Wax needs

to own EISA *after* writing to the Lock Control register: when that second cycle completes, Wax is guaranteed to have acquired and locked the EISA bus. A read from Wax's address map RAM (and throwing away whatever data is returned) is a good choice for this second cycle, since it must acquire the EISA bus, but it has no other side-effects.

> To guarantee the CPU exclusive access to system memory that may also be accessed by (E)ISA devices, the CPU writes the byte `$00` to address `$FC010001`, reads the word at address `$FC100000` (or performs any other access to the address map RAM or to an EISA device), then accesses the desired shared system memory, then writes the byte `$01` to address `$FC010001`.

When writing to this register, the value in bits 1-7 is currently ignored; but for future compatibility, these bits should be set to 0. The CPU can also read this register to determine the current state of the Lock Control bit (the values read from the other bits in this register should be treated as undefined and be ignored). At power-up, the Lock Control bit is initialized to 1 (the (E)ISA bus *not* locked by the CPU).

2.2.4 CPU Interrupt Acknowledge Cycles

When any of (E)ISA's interrupt request lines are asserted, the TI chipset will generate an `IRQ` which Wax passes back to the CPU (through Wax's main interrupt controller). In an 80x86-based system, which the TI chipset is designed for, the CPU would then initiate a vectored interrupt acknowledge cycle, in which the chipset would provide a unique interrupt vector for the highest-priority (E)ISA interrupt that is pending. However, the PA processor in this system does not run such vectored interrupt acknowledge cycles; and so Wax provides an alternate means for the CPU to identify the highest-priority pending (E)ISA interrupt.

A CPU read of the byte at address `$FC01F000` will cause Wax to perform a simulated interrupt acknowledge cycle to the TI chipset. The "vector" number returned by the chipset will be passed back to the CPU as the value of this read. Note that the most significant part of this vector number byte must be set by software when it initializes the chipset's interrupt controller.

> To run a simulated vectored interrupt acknowledge cycle, the CPU reads a byte from the address `$FC01F000`; the value returned is the interrupt vector number from (E)ISA's interrupt controller.

Note that the interrupt service routine must also issue an EOI (end of interrupt) command to the TI chipset to clear the interrupt request; the interrupt acknowledge cycle does *not* take its place. Also note that NMIs from the TI chipset are handled separately, and are *not* vectored, so no interrupt acknowledge cycles are run to service them.

2.2.5 CPU Accesses to Address Map RAM

The address map RAM allows (E)ISA master and DMA devices to talk to system memory. The CPU must set up appropriate entries in the address map, to point at selected pages within the system's memory, before enabling an (E)ISA master/DMA transfer to or from system RAM.

Each address map RAM entry translates one 4kB address range of (E)ISA memory into one 4kB page of system memory. The page offset (address bits 0 through 11) is *not* modified by this translation. There are a total of 1024 (`$400`) entries in the address map RAM; thus 1024×4kB = 4MB of (E)ISA memory space can be mapped into system RAM addresses at any time.

The CPU accesses the address map entries as words, from addresses `$FC100000` through `$FC4F0000`, with one entry every `$1000` bytes. The address map is read/writeable from the CPU. Each entry consists of the

page number of a 4kB block of system RAM to be accessible from (E)ISA; in other words, an entry is the desired RAM address shifted right by 12 bits. The upper 12 bits of an address map entry are currently not used; they will read as 0s, and should be written as 0s for future compatibility. The address map *cannot* be directly accessed from (E)ISA; instead, every (E)ISA memory access from $00100000 through $004F0000 *uses* the address map to generate an address into the system's memory.

> To make system RAM addresses $xxxxx000 through $xxxxxFFF accessible to an (E)ISA device, the CPU writes $000xxxxx to an address map entry at $FC100000, $FC101000, . . . , or $FC4FF000.

To prevent data from being read from or written to incorrect system RAM addresses, any write into the address map RAM flushes all inbound data and invalidates all outbound data in the FIFO before taking effect. In a KIOSC system that supports pended reads, a read from the address map RAM will be pended (this is because Wax must arbitrate for the EISA bus before it can safely access the address map RAM, and this process may take a significant number of host bus cycles).

The following section describes in more detail how an (E)ISA device uses this address map to access system RAM.

2.2.6 (E)ISA Master and DMA Device Accesses to System RAM

This converter reserves (E)ISA memory space from $00100000 through $004FFFFF for reads from or writes to system memory. No (E)ISA memory boards should be programmed to respond in this address range. As described above, accesses to this region are translated by the address map before being forwarded to system RAM. In particular:

> After the CPU has written $000xxxxx to the address map entry at $FCyyy000, an (E)ISA device can access system RAM address $xxxxxzzz by performing a memory read or write to (E)ISA address $00yyyzzz.

As an example, suppose the CPU has written $000ABCDE to address $FC321000. Then, an (E)ISA DMA or bus master transfer to (E)ISA address $00321789 will be passed to the system RAM as a transfer to address $ABCDE789, as follows:

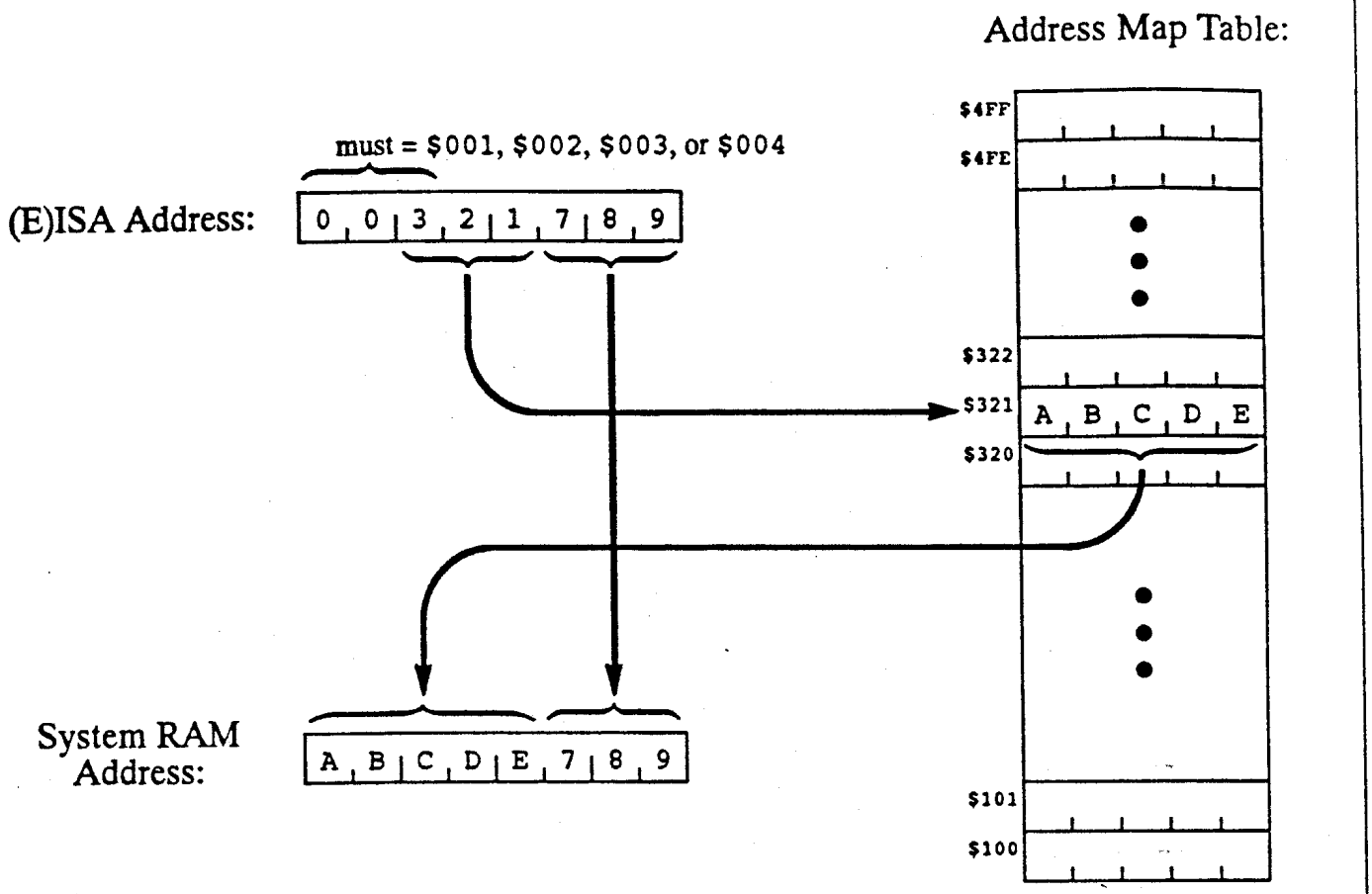


Figure 8 (E)ISA Accesses Through the Address Map

In a KIOSC system that supports pended reads, any (E)ISA-originated read from the host will be pended, freeing the host bus for other use. As soon as host memory is ready to supply the requested data, it rearbitrates for the KIOSC bus and sends the data to Wax. Throughout this time, the (E)ISA slave is stalled, since (E)ISA has no notion of split transfers. Thus, any host transaction to the EISA converter while it is waiting for pended read data must be "retried."

2.2.7 The Data FIFO

(E)ISA-originated transfers to or from system RAM are normally subject to buffering within the converter for efficiency. This converter contains one 8-word data buffer that can be used for either inbound or outbound transfers; it is set up to take advantage of the 2-word, 4-word, and 8-word burst transfer types available on the GSC and KIOSC busses.

When an (E)ISA device first reads from system memory, the converter will issue a burst read on the host bus that includes the requested word through the last word of its 8-word data line. Thus, up to 7 words are prefetched and stored within Wax. Subsequent sequential reads are satisfied from this data FIFO, and no more host bus cycles are issued until the (E)ISA device crosses into the next 8-word data line. Ideally, anyway. In practice, there are actually several events that will cause the data in the FIFO to be invalidated to ensure that the (E)ISA slave doesn't see out-of-date data. The outbound data in the FIFO will be invalidated whenever:

- an (E)ISA read is not sequential (note that the converter allows consecutive 8- and 16-bit reads as well as 32-bit reads to be counted as sequential);

- an (E)ISA read goes beyond the 8 words stored in the FIFO;
- an EISA read from system memory is "locked" on the EISA bus;
- any (E)ISA write to system memory occurs;
- an (E)ISA interrupt (including the non-maskable interrupt) is requested;
- the CPU writes to an (E)ISA slave;
- the CPU writes to the address map; or
- the CPU writes to the converter's FIFO enable bit at address `$FC011001`.

When an (E)ISA device first writes to system memory, the converter will begin to buffer this data in its data FIFO; later, when the FIFO fills up, up to 8 words may be bursted into system memory. However, as in the outbound DMA case, there are actually several events that will cause dirty data in the FIFO to be written into system memory, no matter how many (or how few) words are in the FIFO. The inbound data in the FIFO will be flushed (written to system memory) whenever:

- an (E)ISA write is not sequential (but consecutive 8-, 16-, *or* 32-bit writes are allowed);
- an (E)ISA write goes beyond the 8 word line stored in the FIFO;
- an (E)ISA write to system memory is "locked" on the EISA bus;
- any (E)ISA read from system memory occurs;
- an (E)ISA interrupt (including the non-maskable interrupt) is requested;
- the CPU reads from an (E)ISA slave;
- the CPU writes to the address map; or
- the CPU writes to the converter's FIFO enable bit at address `$FC011001`.

The above rules allow typical (E)ISA DMA type transfers to make most efficient use of the host bus, while ensuring that (E)ISA slaves which poll, or communicate with the CPU via shared memory, or implement semaphores in system memory will work correctly. Inbound transfers will never be reordered, and neither bus will ever detect the end of a transfer while there is inbound data yet to be flushed to system memory.

In unusual situations, the CPU might want to have manual control over the data FIFO. The converter's FIFO enable bit permits such control. Using this register, the CPU can ensure that all inbound data received so far has reached system memory, and that future outbound data will be (re-)read from system memory; in extreme cases, the CPU can disable the data FIFO altogether. Specifically:

> To flush all dirty inbound data (if any) into system memory and invalidate all valid outbound data (if any), the CPU writes the byte `$01` to address `$FC011001`.
> To completely disable the data FIFO, the CPU writes the byte `$00` to address `$FC011001`. It can later write the byte `$01` to that address to re-enable it.

2.2.8 EISA Master Locked Accesses to System RAM

As mentioned above, the converter's data buffering is disabled whenever an EISA master drives the LOCK signal active. Thus, locked EISA transfers are executed to system RAM directly, and without being reordered.

The remaining requirement for correct operation of EISA "locked" accesses is that no *other* transfers to system RAM should be allowed in between the locked ones from the EISA master. This converter satisfies this requirement by asserting the host bus's lock control signal whenever EISA's LOCK is asserted while the it owns

the host bus. Once granted the host bus, the converter will not release it until EISA's LOCK signal is deasserted. Thus, EISA masters' read-modify-write transactions, and any other types of locked cycles, will truly be atomic operations.

The penalty in running such locked cycles is that the EISA master will be tying up the system's high-speed path to memory with its low-speed EISA cycles, for as long as it holds the LOCK signal asserted. Not even CPU cache misses will be able to access system RAM until EISA's LOCK is released. Therefore, locked accesses on the EISA bus must be infrequent and short, or else the whole system will be brought to its knees.

2.2.9 (E)ISA Master and DMA Accesses to Local Memory or I/O

Wax will ignore all (E)ISA I/O accesses, and (E)ISA memory accesses that do not fall within the $00100000-$004FFFFF mapping space. This allows one (E)ISA device to talk to another (E)ISA device without needlessly tying up (or inadvertently accessing) system RAM.

2.2.10 Control and Status Registers

All together, Wax's EISA converter contains four control and/or status registers. All of them are read/write registers (that is, the CPU can read the last value it wrote into them), except that several bits in the Status register are read-only indications of how Wax configured itself at power-up.

The Lock Control register, at $FC010001, is fully described earlier in this section; it allows the CPU to generate locked accesses to slave devices on the EISA bus.

The FIFO Enable register, at $FC011001, has also been previously described (in the subsection about the data FIFO); it allows the CPU to flush or invalidate any data currently in the FIFO, or to disable the FIFO altogether.

The Bus Concurrency register is at $FC012001. It is designed to be used for hardware turn-on and debugging only; changing the power-up state of this register may seriously degrade system performance. There are two control bits in this register. The first determines whether or not Wax can issue multiple "splits" within one GSC cycle: normally, if Wax must split a GSC cycle to the EISA converter, it will run the necessary split transfer and then release GSC's split signal, permitting other GSC activity to proceed; it may eventually need to run several such split cycles before it is finally granted the EISA bus and can complete the original GSC cycle. However, if Wax is used in a system that can get confused when several separate split cycles are run within one GSC transaction, changing this bit will force Wax to keep GSC's split signal active, from the first time it runs a split transfer, until just before it readies the original GSC cycle. This will create a lot of dead bus time in which Wax has the GSC bus locked but is not actually running any GSC cycles; but it guarantees that Wax will split each GSC transfer at most once.

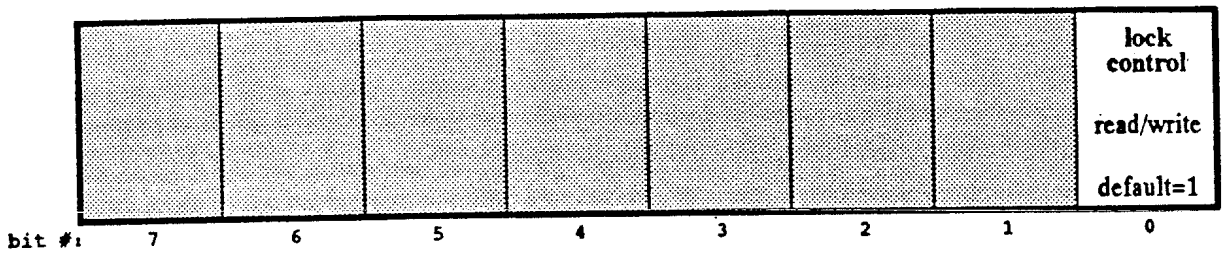
The second bit in the Bus Concurrency register controls whether or not Wax permits concurrent bus operation: that is, whether or not an (E)ISA master device can be granted the bus before Wax owns the GSC bus. By default, such concurrency is allowed, which lets Wax use its data FIFO and the GSC bus efficiently, arbitrating for the GSC bus only when it must run a GSC cycle. If this bit is changed to disable bus concurrency, Wax will arbitrate for the GSC bus whenever an (E)ISA master requests the EISA bus, and will hold onto the GSC bus for as long as the (E)ISA master owns the EISA bus. This can potentially be a long time, and during this time Wax will make very poor use of GSC (since (E)ISA cycles are much slower than GSC cycles). However, it also guarantees that Wax will never need to split a GSC transfer, which would allow Wax to be turned on in a system that doesn't support split GSC transfers.

The EISA converter's Status register is at $FC01E001. This register contains several bits that indicate how the EISA converter configured itself at power-up, and one bit that indicates when a bus error has occurred while the converter was running a GSC cycle. Thus, this register has little use during normal system operation; but it could be useful during system turn-on, in diagnosing hardware problems, and during error recovery after a GSC bus error.

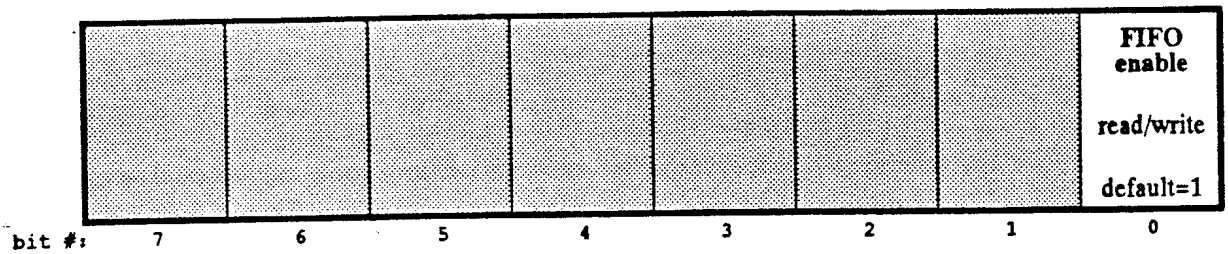
The bus error bit in this register is cleared during system reset. If the EISA converter ever masters a cycle on GSC which bus errors, this bit will be set. If software wants to identify and recover from GSC DMA cycles that bus error, it can use this bit to help decide who ran the cycle that errored. Software can then clear this bit by writing a 1 to it. Writing a 0 to this bit will not change it; and it doesn't matter what values are written into the other bits when the CPU writes into this register to clear the bus error bit, since the others are all read-only bits and cannot be changed from software. The EDPU enabled bit indicates that Wax is internally emulating TI's EDPU (EISA data path unit), rather than providing an i486-like data bus that an external EDPU must convert into the EISA data bus. The 8086 enabled bit indicates that Wax is operating in 8086 mode rather than its usual i486 mode. And the multiplexed mode enabled bit indicates that Wax is operating in its special mode that interfaces to TI's token ring chip. This multiplexed mode is a variation on Wax's 8086 mode in which DMA addresses are multiplexed onto the data pins; if this bit is set, the 8086 enabled bit will also be set.

Following are graphical representations of each of these registers:

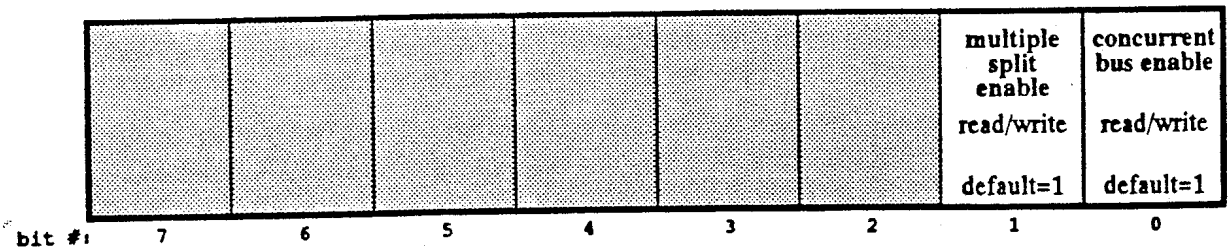
$FC010001: Lock Control Register



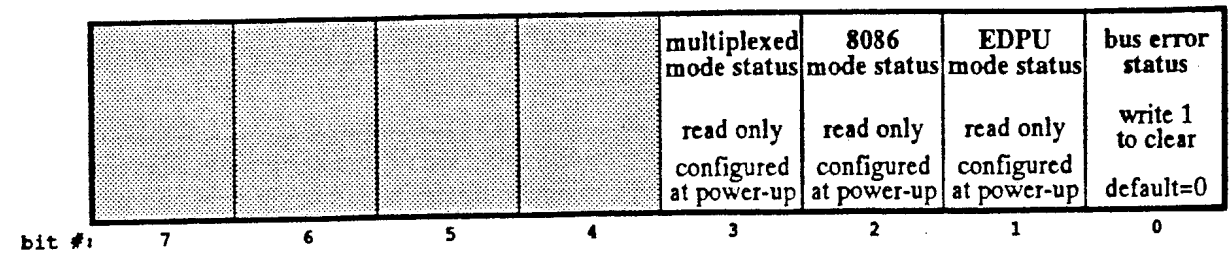
$FC011001: FIFO Enable Register



$FC012001: Bus Concurrency Register



$FC01E001: EISA Converter Status Register



2.3 Hardware Description

2.3.1 Pin Description

The following is a list of the pins on Wax that are dedicated to the i486 conversion module (pins that are of general use to Wax, such as the GSC bus connections and power, are *not* listed here). In addition to the pin names, this list includes the direction (relative to Wax) and a short description of each signal. Signal names ending in "L" are active-low; those ending in "H" are active-high. Unless otherwise indicated, all signal descriptions for "8086 Mode" apply to both normal and multiplexed 8086 modes. Where there is no separate 8086 mode description, the pin performs the same function as it does in i486 mode. The signal name prefixes ("`i486`", "`eisa`", or "`edpu`") indicate where the signals connect (to the intermediate i486 bus, to the final (E)ISA bus, or to the EDPU control points generated by the TI chipset, respectively):

- `i486Hclk` — input
  Master clock for the TI chipset; used by this module for synchronizing i486 bus signals. Can be any speed from 0 to 33.3 MHz; but should be 33.3 MHz to get maximum performance on the EISA bus.
  8086 Mode: `x86Hclk`: Maximum speed is 16.7 MHz.
  Multiplexed Mode: Should be 16 MHz, so that timing to the TI token ring chip will be correct.

- `i486ResetL` — output
  Master reset for the TI chipset; this is a version of Wax's main chip reset that is synchronized to `Hclk`.
  8086 Mode: `x86ResetL`.

- `i486IrqH` — input
  Interrupt request input; the TI chipset asserts this signal (asynchronously) when it detects an (E)ISA interrupt request, and deasserts it when there are no more pending (E)ISA interrupt requests.
  8086 Mode: `x86IrqH`.

- `i486NmiH` — input
  Non-maskable interrupt input; the TI chipset asserts this signal (asynchronously) when it detects a serious fault on the (E)ISA bus (such as bus lock-up or a slave driving (E)ISA's `IOCHK*` signal active), and deasserts it after the CPU resolves or masks the cause of the NMI.
  8086 Mode: `x86NmiH`.

- `i486HholdH` — input
  Host bus hold request; asserted when the TI chipset wants to own the host bus and potentially run cycles to host memory on it. Note that, because Wax permits concurrent bus operation, asserting `Hhold` will *not* necessarily cause Wax to request the GSC bus.
  8086 Mode: `x86HholdH`.

- `i486HhldaH` — output
  Host bus hold acknowledge; driven active by Wax, after it senses a `Hhold`, to give the TI chipset control of the i486 bus. Once driven active, `Hhlda` will not be negated until after `Hhold` is deasserted. Again, note that `Hhlda` may be asserted even when Wax does *not* own the GSC bus.
  8086 Mode: `x86HhldaH`.

  input
  At Power-Up: Sensed to determine whether or not to enable the EISA converter: if this pin is low (i.e., if it has an external 4.7kΩ pull-down resistor on it), it enables the

converter; if high (i.e., if it is tied to VCC directly or through a 1kΩ pull-up resistor), the converter is disabled. When it is disabled, this module will not respond as a GSC or i486 slave at any address, and thus will never master an i486 cycle; but it will still drive its output-only signals (such as `Reset` and `Hhlda`) normally. If Wax is used in a system that permanently disables the EISA converter, all of the i486 input and input/output pins should be grounded or otherwise prevented from floating, so that they don't draw excessive current.

- `i486HbusreqL` — output — Host bus request from Wax; asserted when the CPU runs a cycle to (E)ISA or to the converter that requires Wax to own the i486 bus. This signal causes the TI chipset to include Wax in its next bus arbitration sequence. Wax will assert this signal even if it *already* controls the i486 bus, since doing so shouldn't hurt anything. Note that if an (E)ISA device runs a cycle to system memory after `Hbusreq` is asserted, but before the CPU is granted the bus, Wax must "split" the CPU cycle to avoid deadlock.
  8086 Mode: `x86HbusreqL`.

  input — **At Power-Up:** Sensed to distinguish i486 mode from 8086 mode: if this pin is low (i.e., if it has an external 4.7kΩ pull-down resistor on it), it selects i486 mode; if high (i.e., if it has an external 4.7kΩ pull-up resistor on it), it selects 8086 mode.

- `i486Addr[31:2]` — in / out — The i486 address bus. Driven out from Wax when the CPU has been granted the (E)ISA bus; otherwise an input. When Wax is the i486 bus master, it does not pipeline addresses, so these signals are valid throughout a cycle; but when Wax is the slave, it samples these signals only at the rising `Hclk` edge when `Hads` is asserted.
  8086 Mode: `x86Addr[31:2]`; these signals should be held stable and valid throughout the bus cycle.
  Multiplexed Mode: These signals are output-only from Wax; the (input) addresses for DMA cycles must be provided on the `x86Data[15:0]` signals.

- `i486BeL[3:0]` — in / out — The i486 byte enables that take the place of address bits 0 & 1 plus transfer size bits. Driven out from Wax when the CPU has been granted the (E)ISA bus; otherwise an input. When Wax is the i486 bus master, these signals are valid throughout a cycle; when Wax is the slave, it samples these signals at the rising `Hclk` edge when `Hads` is asserted.
  8086 Mode: `i486BeL[1:0]` become `x86Addr[1:0]`, the two low-order address bits that don't exist on the i486 bus (and, like the rest of the address bus, they should be stable and valid throughout the bus cycle);
  `i486BeL[1:0]` are not used, and should be grounded on the PC board.
  Multiplexed Mode: These signals are output-only from

Wax; the (input) addresses for DMA cycles must be provided on the `x86Data[15:0]` signals.

- **`i486HadsL`** — in / out — Host address strobe, that goes active for one `Hclk` to indicate that the address bus is valid and that an i486 bus cycle is starting. Driven out from Wax when the CPU has been granted the (E)ISA bus; otherwise an input.
  8086 Mode: Replaced by `x86SbheL`, a bidirectional signal that is active when the upper 8 bits of the 16-bit data bus are in use. It is driven out from Wax when the CPU has been granted the 8086 bus; otherwise it is an input.

- **`i486HwnrH`** — in / out — Host read/write signal; high indicates a write operation, and low indicates a read. Driven out from Wax when the CPU has been granted the (E)ISA bus; otherwise an input. When Wax is the i486 bus master, this signal is valid throughout a cycle; when Wax is the slave, it samples this signal on the rising `Hclk` edge when `Hads` is asserted.
  8086 Mode: Replaced by `x86MrdcL`, a bidirectional signal that is active during memory read cycles. It is driven out from Wax when the CPU has been granted the 8086 bus; otherwise it is an input.

- **`i486HmnioH`** — in / out — Host memory/IO signal; high indicates a memory access, and low indicates an I/O access. Wax will not respond as a slave to I/O cycles (i.e., if this signal is low). Driven out from Wax when the CPU has been granted the (E)ISA bus; otherwise an input. When Wax is the i486 bus master, this signal is valid throughout a cycle; when Wax is the slave, it samples this signal on the rising `Hclk` edge when `Hads` is asserted.
  8086 Mode: Replaced by `x86MwtcL`, a bidirectional signal that is active during memory write cycles. It is driven out from Wax when the CPU has been granted the 8086 bus; otherwise it is an input.

- **`i486HdncH`** — in / out — Host data/code signal; high indicates a normal or data access, and low indicates a special or code access. As a master, Wax drives this signal low only during interrupt acknowledge cycles; and as a slave, Wax will not respond to any cycles in which this signal is low. Driven out from Wax when the CPU has been granted the (E)ISA bus; otherwise an input. When Wax is the i486 bus master, this signal is valid throughout a cycle; when Wax is the slave, it samples this signal on the rising `Hclk` edge when `Hads` is asserted.
  8086 Mode: Replaced by `x86IorcL`, an output-only signal that is active during I/O read cycles.

- **`i486HlockL`** — in / out — Host bus lock, indicating that the current bus master wants to run a sequence of indivisible bus cycles to the slave. When the CPU has been granted the (E)ISA bus, this signal is driven out from Wax, and is active when Wax's internal lock control register has been set. Otherwise, this

signal is an input that, when asserted during a cycle that Wax responds to, disables Wax's data buffering and locks the system's busses all the way back to system RAM, to ensure the (E)ISA device indivisible accesses to RAM.
8086 Mode: Replaced by `x86IowcL`, an output-only signal that is active during I/O write cycles.

- `i486HreadyinL` input — Host bus ready input to Wax, driven active for one `Hclk` by the TI chipset to indicate that a bus cycle is complete. This signal's assertion allows Wax to complete a GSC bus cycle that it is running to (E)ISA.
8086 Mode: Replaced by `x86HreadyinL`, a ready indication that—once asserted—should be held active until Wax deasserts its cycle running signal (`Mrdc`, `Mwtc`, `Iorc`, or `Iowc`).
Multiplexed Mode: During *I/O* cycles, this signal does not need to be asserted: Wax will automatically end the cycle after 3 `Hclks`.

- `i486HreadyoutL` output — Host bus ready output from Wax, driven active for one `Hclk` by Wax to indicate that a bus cycle is complete. When the CPU has been granted the (E)ISA bus, this signal is a version of `Hreadyin` delayed by one `Hclk`, since the TI chipset needs this feedback. Otherwise, Wax asserts it when the (E)ISA-originated cycle can complete (that is, after either GSC or Wax's internal data buffer has sourced or accepted the data).
8086 Mode: Replaced by `x86HreadyoutL`, a ready indication that—once asserted—is held active until the cycle running signal (`Mrdc` or `Mwtc`) is deasserted. When Wax owns the 8086 bus, this output is tri-stated; thus, the `x86HreadyinL` and `x86HreadyoutL` pins can be tied together to create a bidirectional ready signal.

- `i486Hlac0L` output — Host bus local access indication; this signal is driven active for one clock after `Hads` during (E)ISA cycles to Wax's mapped address space, to tell the EISA chipset that these cycles are being run on the host bus.
8086 Mode: Not used; should be left unconnected.

- `eisaData[31:0]` in / out — The (E)ISA data bus. Driven out from Wax during CPU writes to (E)ISA or (E)ISA reads from system memory, and also during some (E)ISA assembly and disassembly cycles between different-sized (E)ISA masters and slaves; otherwise an input into Wax.
8086 Mode: `eisaData[15:0]` are `x86Data[15:0]`, the 16 bits of the 8086 data bus;
`eisaData[31:16]` are not used, and should be grounded on the PC board.
Multiplexed Mode: During DMA cycles, these pins also function as address inputs: the high 16 address bits are applied and latched by `Sxal`, then the low 16 address bits are applied and latched by `Sale`.

- `edpuSdileL` input — System data input latch enable, from the EBCU. When low, this signal causes the (E)ISA input data latch to be transparent; when it goes high, the current value of the (E)ISA data bus is latched.
  8086 Mode: Replaced by `x86MuxedL`, an input that determines if the address from the external 8086 device is multiplexed onto the data pins. If it is tied low, addresses are multiplexed onto the data pins, controlled by the following signals, and the address pins are outputs only; this is referred to as "Multiplexed Mode," and must be used if Wax is connected to TI's token ring controller chip. If this signal is tied high, the address pins are bidirectional, as they are when Wax is in its i486 mode.

- `edpuSel[2:0]` input — Select data transaction inputs from the EBCU. These signals encode the source of data and its size during (E)ISA transactions; they help control Wax's input and output (E)ISA data bus byte swapping and multiplexing logic. The possible values are:

sel [2]	sel [1]	sel [0]	Source of Data
0	0	0	8-bit (E)ISA device
0	0	1	16-bit (E)ISA device
0	1	0	32-bit (E)ISA device
0	1	1	*unused*
1	0	0	32-bit host bus
1	0	1	latched (E)ISA input
1	1	0	*unused*
1	1	1	latched (E)ISA input

  8086 Mode: These bits are not used, and should be grounded on the PC board.
  Multiplexed Mode: Bit 2 is replaced by `x86SownL`, an input that is active when the 8086 device owns the bus; bit 1 is replaced by `x86SddirH`, an input that is high when data is being transferred toward system memory; bit 0 is replaced by `x86SdbenL`, an input that is active when the data buffers should be enabled.
  These inputs override Wax's normal output enable logic for `x86Data[15:0]` when Wax is the slave to a DMA transfer.

- `edpuSdoeL[1:0]` input — System data output enables from the EBCU; these work with the `sel[2:0]` inputs and EISA's `beL[3:0]` to determine when data should be driven onto (E)ISA's data bus. When `sdoeL[1]` is active, data from the latched (E)ISA input can be driven out; or when `sdoeL[0]` is active, data from an (E)ISA device or the host is driven.
  8086 Mode: These bits are not used, and should be

grounded on the PC board.
Multiplexed Mode: Bit 1 is replaced by `x86SxalH`, an input that latches the high 16 bits of a DMA address;
bit 0 is replaced by `x86SaleH`, an input that latches the low 16 bits of a DMA address;
both parts of the address are applied to Wax's `x86Data[15:0]` pins.
At Power-Up: Sensed to determine whether or not to enable Wax's internal EDPU. If both of these signals are low (i.e., they are tied or pulled to ground), the internal EDPU is disabled and an external EDPU chip will be used. If at least one of these signals is high (as will be the case if these signals are driven by the EBCU), Wax will use its internal EDPU. If Wax is in 8086 mode, these inputs are ignored during power-up, since an 8086 system does not use the EDPU's functionality.

- `edpuBelatL` input — Byte enable latch enable from the EBCU; it allows Wax to latch EISA's byte enable signals at the start of a cycle and hold them valid throughout the cycle. When low, this signal causes the `eisaBeL[3:0]` inputs to flow into Wax transparently; when it goes high, the byte enable signals are latched.
8086 Mode: Not used; should be grounded on the PC board.

- `eisaBeL[3:0]` input — EISA's byte enable signals. Note that these are different from the `i486BeL[3:0]` signals that control transfers over the i486 bus (they will differ during (E)ISA byte assembly and disassembly cycles). These EISA byte enables are used only by the data path section, and determine when to drive data out onto (E)ISA's data bus.
8086 Mode: Not used; should be grounded on the PC board.

Total: 92 signal pins.

2.4 Basic Schematics for EISA or 8086 Subsystems

The schematics in this section show how the EISA interface of Wax typically will be connected to TI's EISA chip set to create a complete EISA interface, or how the interface in 8086 mode can be connected to TI's Token Ring controller.

Please note that these typical circuits are intended only as guidelines. Any particular implementation will have unique testability, physical layout, and PC technology requirements and limitations; and you should address these issues and make any necessary modifications for your application. In other words, if you just plop these drawings down into a schematic capture program and automatically route it, the board you get back probably won't work or be testable.

Specific areas which usually need special attention are: series damping for all clocks and for EISA's START* and CMD* signals; minimizing HCLK skew between Wax and the TI chipset; and in general keeping traces between Wax and the TI chipset, and between the TI chipset and the EISA connectors, as short as possible. Many signals have rather tight timing margins (even on the "slow" (E)ISA bus), and several signals (especially on (E)ISA) are sampled asynchronously, so long traces may increase propagation delays or cause signal integrity to degrade enough to make the system fail. If Wax must be more than about a foot from the EISA connectors, you should be very careful.

Also, if Wax must be placed more than a few inches from the EISA connectors, or if your system includes more than 4 EISA slots, you should use an EDPU chip to drive the EISA data bus rather than using Wax's built-in EDPU emulator. Wax's data bus drivers are not beefy enough to drive a more heavily-loaded bus.

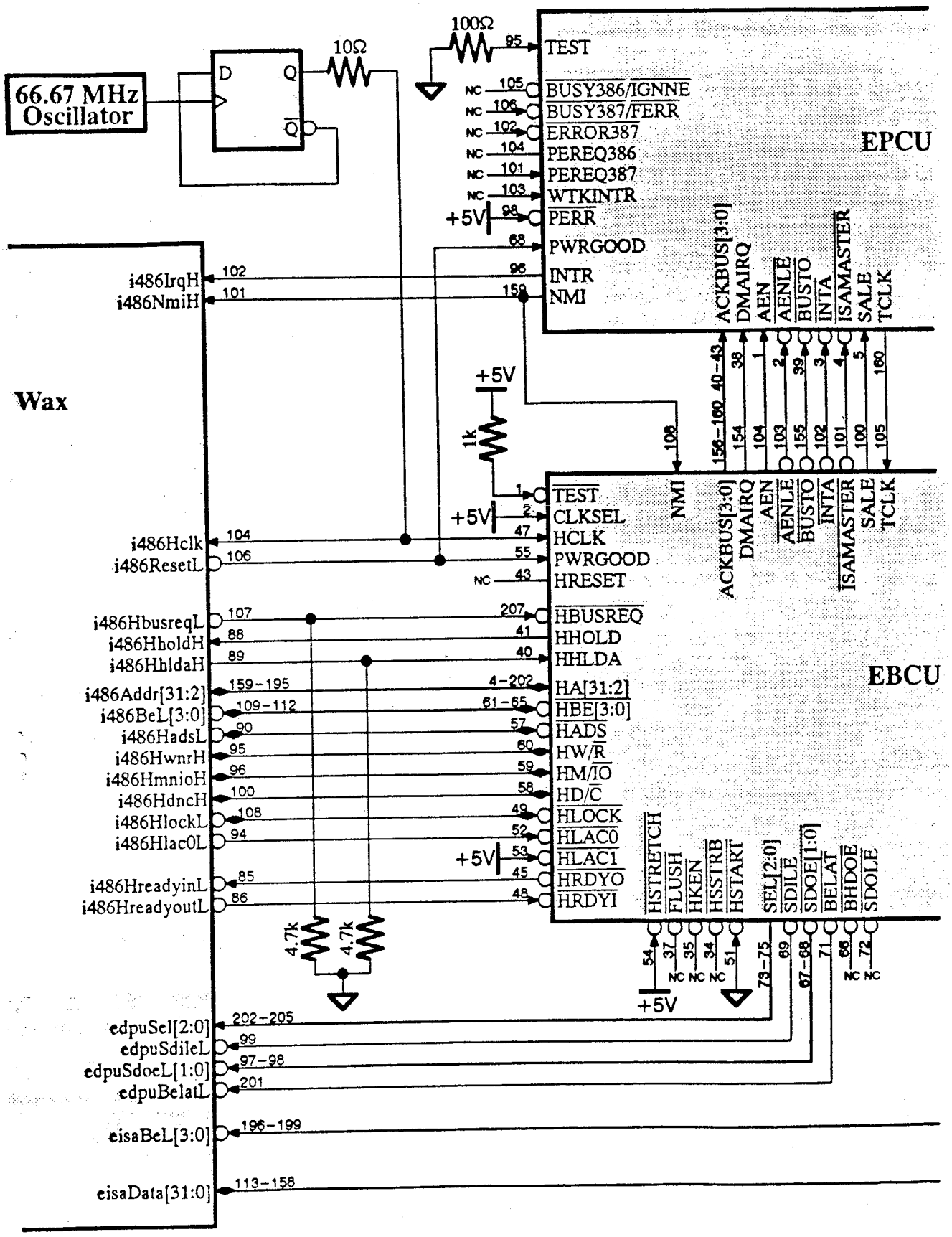


Figure 9 Wax to EISA Schematic, Without External EDPU, Page 1 of 2

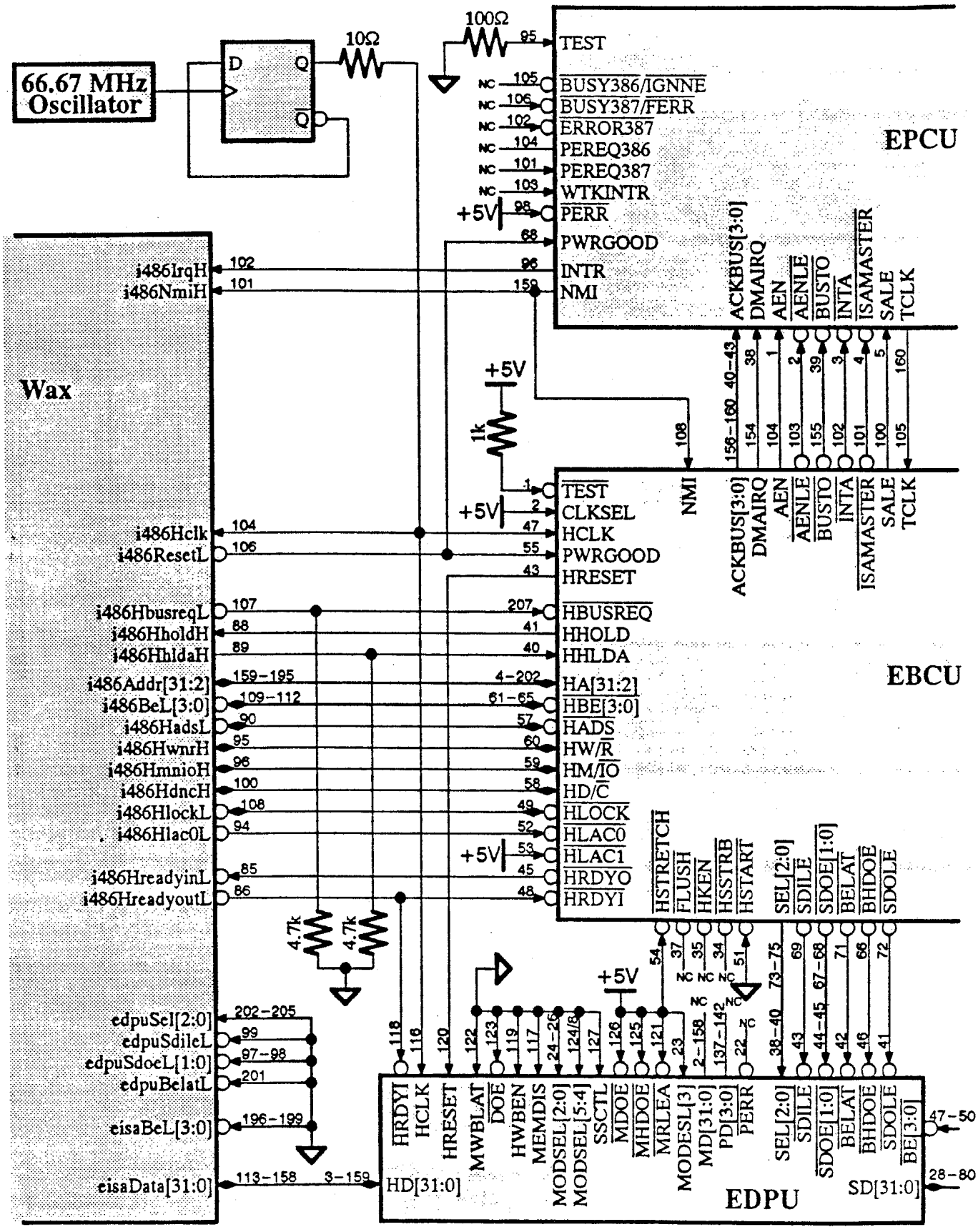


Figure 10 Wax to EISA Schematic, With External EDPU, Page 1 of 2

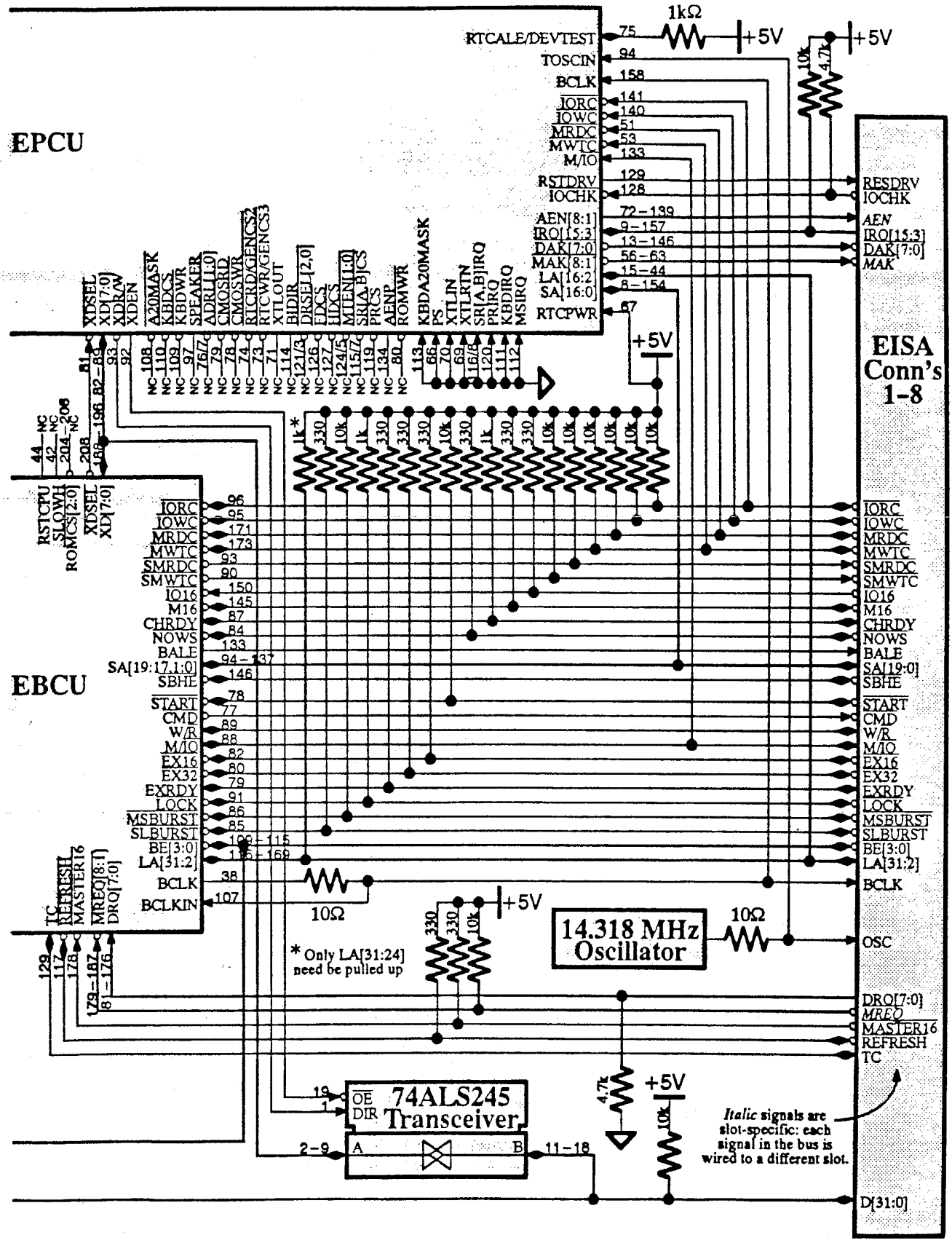


Figure 11 Wax to EISA Schematic, With *or* Without External EDPU, Page 2 of 2

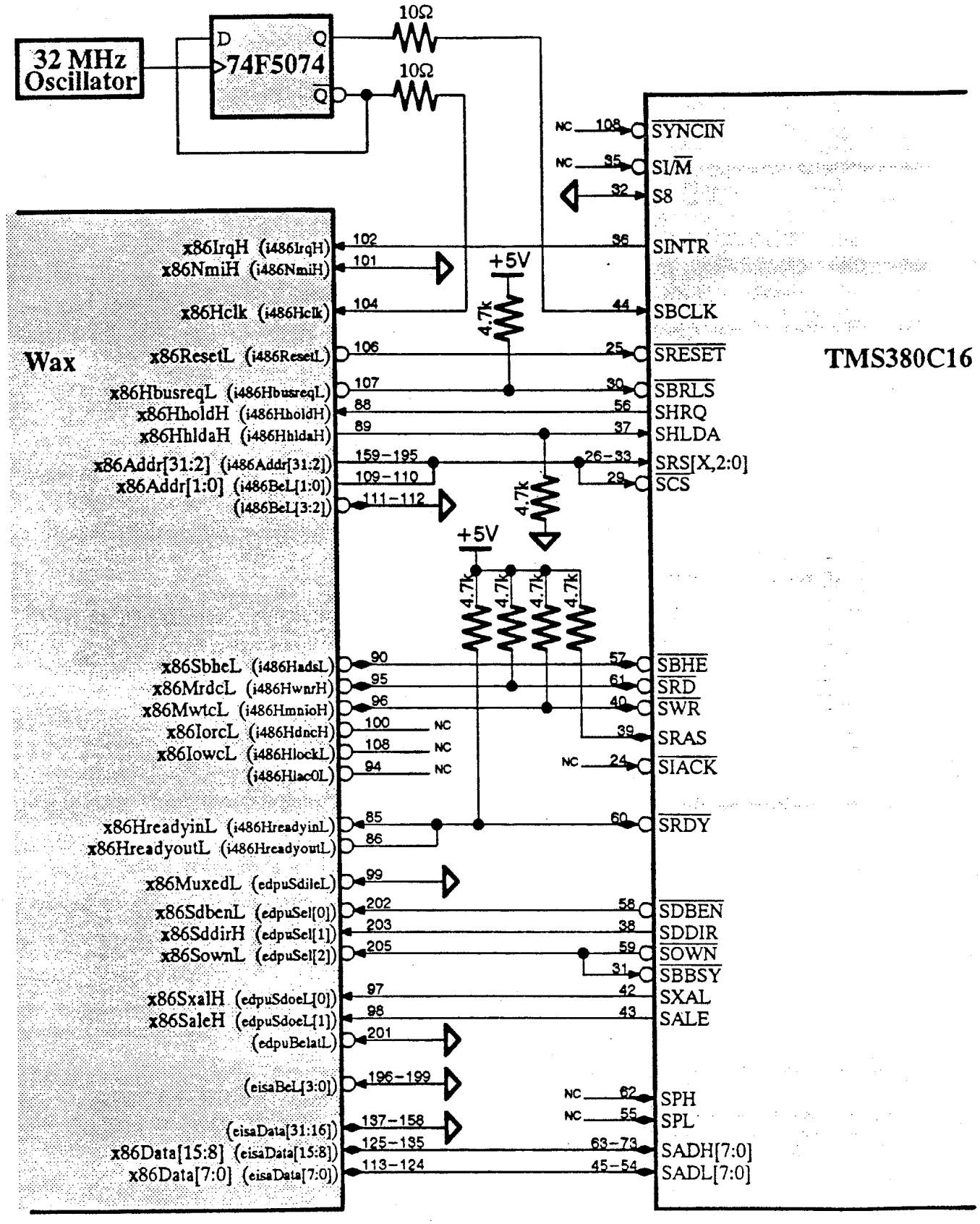


Figure 12 Sample Muxed 8086-Mode Wax to TI Token Ring Controller Schematic, Page 1 of 1

3 GSC interface

3.1 GSC operation

The Wax ASIC is connected to one or more SPUs through the GSC. In a Gecko, this is the on-board bus which interconnects the SPU, I/O, and graphics. The Wax GSC circuitry is similar to the GSC interface in the Lasi ASIC, but Wax is also a GSC+ compatible device. The gscHkioscL signal selects GSC operation when high, GSC+ operation when low. The address space occupied by Wax changes between GSC and GSC+ operation.

3.2 Slave Operation

When Wax is accessed as a slave, it adheres to all GSC(+) protocols. The SPU can access WAX as a slave with any address or transaction type specified although only single word and partial word transfers are supported. All GSC transfers begin with an address phase. Wax checks the parity during the address phase and will ignore the transfer if the parity is in error. GSC+ DMA read return cycles do not have an address phase, so no parity error can occur during these cycles.

During slave writes, Wax checks parity on all data transferred. If a data parity error is detected, Wax asserts gscErrorL signal to indicated the parity error occurred. If a multiple word write is indicated by the transaction type, all words of data are checked for correct parity. Only the first word, or partial word, will actually be written into a register, the remaining words of data are ignored.

During slave reads, Wax generates parity for all words indicated in the transfer byte enables. When Wax is accessed during a slave read, the entire gscAd bus is driven for the first word of the data transfer. If multiple words are indicated by the transaction type, subsequent words are meaningless, but are accompanied by correct parity.

The Intel bus converter will split slave accesses using the gscLsL signal when in either GSC or GSC+ operation, but only if necessary to prevent deadlock. In GSC+ mode, the Intel bus converter interface will allow slave read cycles to be retried or pended. If an error occurs during a pended slave read transfer, the error is ignored.

3.3 Master Operation

When Wax is a GSC(+) master, it adheres to all GSC(+) protocols. Wax requests mastership of the GSC(+) and then waits until it has been granted mastership of the GSC(+). Wax transfers begin with an address phase, then one to eight data phases. Wax generates parity for the address. During mastered transfers, Wax monitors the gscErrorL signal to determine if the transfer terminates with an error condition. Each interface within Wax will indicate if a bus error ocurred during a mastered GSC(+) transaction.

Wax will generate multiple word master write transfers. When Wax writes as a master, each data phase has the data written accompanied by parity. If Wax detects a data parity error during a master write transfer, it will wait for the gscReadyL signal to assert before continuing.

Wax will generate mutiple word master read transfers. When Wax reads as a master, each data phase of the transfer is checked for errors. If an error is detected, Wax will terminate

the transfer. In GSC+ operation, the Intel bus converter allows mastered read cycles to be pended.

3.4 GSC(+) Arbitration

Wax will arbitrate for the GSC(+) whenever one of the sections inside wax requests mastership. Wax will keep the GSC(+) until no more sections are requesting mastership, or the external arbiter negates the gscBgL signal. Wax will split slave accesses to the Intel bus converter when necessary. When mastership has been requested and that requirement is met by splitting a slave transaction to Wax, the gscBrL signal will be negated before mastership is granted. In all other cases, Wax must be granted mastership before it will stop requesting it. Wax will always wait until gscBgL is negated between requests for mastership.

3.5 GSC(+) signals

Below is a description of the system interface on the Wax ASIC:

- gscAd[31:0] — multiplexed **GSC** Address and Data
- gscAddvL — **GSC** Address valid, active low
- gscType[0:3] — **GSC** transaction type and data byte enables
- gscParity — **GSC** parity bit for gscAd[31:0]
- gscReadyL — **GSC** ready bit for data transfers, active low
- gscErrorL — **GSC** error bit for address and data transfers, active low
- gscResetL — **GSC** reset, active low
- gscBrL — **GSC** bus request, active low
- gscBgL — **GSC** bus grant, active low
- gscSplitL — **GSC** split, active low, also called gscLsl
- gscSyncH — **GSC** sync, active high, system clock
- gscSyncL — **GSC** sync, active low, system clock
- gscHkioscL — **GSC** active high, GSC+ (kiosc), active low
- kpendL — GSC+ (kiosc) pend request, active low
- kpackL — GSC+ (kiosc) pend acknowledge, active low
- kretryL — GSC+ (kiosc) retry request, active low
- kdrrL — GSC+ (kiosc) dma read return request, active low
- sysResetL — system reset, active low, output from watchdog timers to reset system
- sysClk40M — system clock input, 40 Mhz

- trstL — tap reset, active low
- tdi — tap data in
- tclk — tap clock
- tms — tap mode select
- tdo — tap data out

4 RS-232 Interface

4.1 Description

The RS-232 interface that is built into the Wax ASIC emulates on of the UARTs in the National NS16550A chip, with the addition of hardware handshaking. The design of this portion of the circuitry was done for the Stiletto ASIC by MCSY in Fort Collins. This circuitry is software compatible with the serial interface in the Lasi ASIC.

4.2 RS-232 Registers

4.2.1 Base Address

The GSC base address for the RS-232 interface is 0xF020 2000. The GSC+ base address for the RS-232 interface is 0xFFE0 2000.This interface identical to the RS-232 port in Gecko, the only difference is the base address.

The RS-232 interface may be disabled entirely by grounding the RS-232Txd output during power up. When this is done, accesses to the RS-232 address space will not generate a gscReadyL. If the rs232Txd signal is not grounded, the RS-232 interface will function as described.

4.2.2 Register Overview

The RS-232 registers are defined as a word port only. When data is written to any register, all thirty-two bits on the data bus are written into the register regardless of the state of the byte enable bits associated with the transfer. When any register is read, all thirty-two bits of the data bus are driven. Unused bits are always read as zeros. The RS-232 registers may be accessed using multi-word transfers, but only the first word of the transfer occurs. During multi-word writes, the second and subsequent words are ignored. During multi-word reads, the first location accessed is driven as the first word of data, but subsequent words are not meaningful data. Correct GSC protocol is followed for multi-word transfers, the data is just meaningless.

Table 1 shows the register assignments and address offsets for the registers of the RS-232 subsystem in the Wax ASIC. The actual address is obtained by adding the address offset for a register to the base address of the Wax ASIC.

Table 1 RS–232 Registers

Description	Offset	R/W	D7	D6	D5	D4	D3	D2	D1	D0
Reset Register	0x000	W	X	X	X	X	X	X	X	X
Undefined	0x001-0x7FF									
Receiver Buffer Register (RBR)	0x800 DLAB = 0	R	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Transmitter Holding Register (THR)	0x800 DLAB = 0	W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Interrupt Enable Register (IER)	0x801	R/W	0	0	0	0	Enable MSI	Enable RLSI	Enable THREI	Enable RDAI
Interrupt Ident Register (IIR)	0x802	R	Fifos Enabled	Fifos Enabled	0	0	Int ID Bit 2	Int ID Bit 1	Int ID Bit 0	Int Not Pending
Fifo Control Register (FCR)	0x802	W	Rx Trig MSB	Rx Trig LSB	X	X	DMA Mode	Tx Fifo Reset	Rx Fifo Reset	Fifo Enable
Line Control Register (LCR)	0x803	R/W	DLAB Bit	Set Break	Stick Parity	Even Parity	Parity Enable	Num of Stop Bits	Wrd Len Bit 1	Wrd Len Bit 0
Modem Control Register (MCR)	0x804	R/W	0	0	0	Loop Back	Unused	*See Note 1*	RTS	DTR
Line Status Register (LSR)	0x805	R	Error In Rx Fifo	Txmitter Empty	Tx Hold Reg Emp	Break Interrupt	Framing Error	Parity Error	Overrun Error	Rx Data Avail
Modem Status Register (MSR)	0x806	R/W	DCD (RLSD)	RI	DSR	CTS	Delta DCD	Trail Edge RI	Delta DSR	Delta CTS
Scratch Register (SCR)	0x807	R/W	Scratch Bit 7	Scratch Bit 6	Scratch Bit 5	Scratch Bit 4	Scratch Bit 3	Scratch Bit 2	Scratch Bit 1	Scratch Bit 0
Divisor Latch Reg LSB (DLL)	0x800 DLAB = 1	R/W	Divisor Bit 7	Divisor Bit 6	Divisor Bit 5	Divisor Bit 4	Divisor Bit 3	Divisor Bit 2	Divisor Bit 1	Divisor Bit 0
Divisor Latch Reg MSB (DLM)	0x801 DLAB = 1	R/W	Divisor Bit 15	Divisor Bit 14	Divisor Bit 13	Divisor Bit 12	Divisor Bit 11	Divisor Bit 10	Divisor Bit 9	Divisor Bit 8
Undefined	0x808-0xFFF									

Note 1 – Bit 2 of the MCR is involved in Hardware Handshaking control. Refer to section NO TAG for more detail.

4.2.3 Register Descriptions

See The National NS16550A data sheet for a detailed description of all of the registers except the Reset Register. A cheat sheet for the Interrupt ID Register is shown below.

4.2.3.1 Hardware Handshake Overview

The upper nibble (bits 7-4) of the IIR register will return 0xC in fifo mode, and 0x0 in non-fifo mode.

The lower nibble (bits 3–0) will return the following interrupt IDs, in priority order.

First Priority	Receiver Line Status	0x6
Second Priority	Character Time-out Indication	0xC
Second Priority	Receiver Data Available	0x4
Third Priority	Transmitter Holding Register Empty	0x2
Fourth Priority	Modem Status	0x0
No Interrupt	No Interrupt	0x1

4.2.3.2 Reset Register

Asserting gscResetL or writing to the Reset Register, 0xF0202800, will reset the Serial Interface. The resetting of the Serial Interface causes all of its registers except the divisor latches to return to their power-up state. Unlike previous serial subsystems, the serial interface in the Wax ASIC is stand alone. Resetting any other serial interface will NOT reset the serial interface in the Wax ASIC. Also, resetting the serial interface in the Wax ASIC will NOT affect any other serial interfaces in the system.

4.3 Hardware Handshaking Control

4.3.1 Hardware Handshake Overview

The basic concept of hardware handshaking as it has been implemented in the 375/380, 720/730/750, and here in the Wax ASIC, is quite simple. Supply a method, in hardware, by which a serial transmission can be suspended if the receiver fifo has reached a set threshold. The modem control lines provide an ideal method of communicating information regarding a serial transmission, and the $\overline{\text{RXRDY}}$ line contains the necessary information regarding the receiver fifo data levels. Combining these two features results in a solid solution. The only necessary feature remaining is the ability to enable and disable the hardware handshaking. All of this is accomplished with the implementations in the machines mentioned, and in the following sections, the exact details of the Wax ASIC implementation will be revealed.

4.3.1.1 Enabling Hardware Handshake

For hardware handshaking to be enabled, two bits in the **Modem Control Register** (MCR) need to be affected. Bit 2 is the hardware handshaking disable bit (normRTS), and should be set to a 0. Consider this bit to be write only. Reading it at power-up or after a directed reset will return the opposite value of what it really is. It powers up a 1 despite what reading the register might tell you. This is an ASP anomaly we duplicated. The RTS bit (bit 1) must also be set to a 1 for hardware handshaking to be enabled. It powers up cleared. A write of 0x02 to the MCR would be the correct way to enable hardware handshaking. To disable hardware handshaking, bit 2 should be set, and bit 1 should be cleared unless there is a need to keep $\overline{\text{RTS}}$ asserted for some other reason.

4.3.1.2 Hardware Gating

For those who want to know what the hardware is really doing to combine $\overline{\text{RXRDY}}$, $\overline{\text{RTS}}$, and normRTS, here's the scoop. $\overline{\text{RXRDY}}$ is an active low signal. It is low when the receiver has reached the trigger level. $\overline{\text{RTS}}$ is active low when used by the hardware outside of the register. For example, writing a 1 to the RTS bit to assert $\overline{\text{RTS}}$ will drive a logic 0 externally. The logic then looks like: !(normRTS | $\overline{\text{RXRDY}}$) | $\overline{\text{RTS}}$. These means that unless normRTS is cleared, $\overline{\text{RXRDY}}$ will be blocked, and unless $\overline{\text{RTS}}$ is asserted, $\overline{\text{RXRDY}}$ qualified by normRTS will be blocked. This is why normRTS must be cleared and RTS must be set to enable hardware handshaking. The final result when hardware handshaking is enabled is that $\overline{\text{RXRDY}}$ will appear inverted on the $\overline{\text{RTS}}$ line.

4.3.1.3 $\overline{\text{RXRDY}}$ Behavior

The National NS16550A data sheet gives a reasonable description of the operation of $\overline{\text{RXRDY}}$, but I will run through one example here. Let's assume we have set up a receiver trigger level of 8 bytes, and have chosen DMA mode 1. To do this, we would have written 0x89 to the **FCR** after having enabled the fifos. Initially, $\overline{\text{RXRDY}}$ will be negated, so with hardware handshaking enabled, $\overline{\text{RTS}}$ will be asserted. This tells the transmitting device that it is okay to send data. $\overline{\text{RXRDY}}$ will stay negated until the trigger level (8 bytes in this case) has been reached in the receive fifo. Once $\overline{\text{RXRDY}}$ asserts and hence $\overline{\text{RTS}}$ negated, $\overline{\text{RXRDY}}$ will stay asserted until the receive fifo is empty. In other words, the transmitting device will continue to be told not to send data until the receive fifo is completely empty.

4.3.1.4 Caveats

The biggest source of confusion has typically been the assumption that the prevention of hardware overflow eliminates the possibility of data overrun. Unfortunately this is not true. Memory buffers can also overflow, so in many cases, Software handshaking (XON/XOFF) is still required. I

so that $\overline{\text{RTS}}$ can temporarily be forced negated to halt the further transmission of data and hence prevent the overflow of Software buffers.

4.4 Software Differences

The Wax ASIC 16550 megacell is intended to function just like the real National NS16550A.

There are a few minor differences, however, between the NS16550A and the Wax ASIC.

Unlike the ASP implementation which uses the WD16C552 externally, there is no need for the software to worry about minimum cycle time requirements with the Wax ASIC. The hardware will guarantee no violation of minimum cycle time specifications for register accesses.

e **Fifo Control Register** in the Wax ASIC behaves a little differently from that of the NS16550A or WD16C552. For the Wax ASIC, bit 0 of the **FCR** must be set before other bits in the **FCR** can be set. What this means is that writing 0xc1 to the **FCR** does not set the receiver trigger level to 14 bytes, unless bit 0 was set by a previous write. The necessary sequence would be a write of 0x01 to the **FCR** followed by a write of 0xc1 to the **FCR**. This may not affect existing software, since the WD data sheet has wording that implies the kind of operation that the Wax ASIC implements.

A bug was recently discovered with the WD16C552 that I also managed to design in. A second transition of a modem control line can be missed if it occurs during a read of the **Modem Status Register**. The level of the modem control input will be latched off of the leading edge of the read strobe to be presented as read data. However, the interrupt resulting from the initial transition of a modem input will be cleared off of the trailing edge of the read strobe. The result is that no new interrupt gets generated, yet the old level of the modem input gets passed on the read.

The Wax ASIC presets the modem control input synchronizers such that if after coming out of reset, any of the modem control inputs are active (0 level external, 1 reflected in the **Modem Status Register**), a modem "delta" for that bit will be indicated. Software should read the **MSR** to clear out modem status interrupts first thing, since the modem lines could glitch during power up, so this difference shouldn't be a problem.

The NS16550A data sheet never really says what happens if a **divisor** value of 0x0000 is loaded for baud rate generation. As it turns out, the NS16550A treats the "0" as if a divisor value of 0x0020 were loaded. The Wax ASIC treats it as if a 0x0001 were loaded, since this is the smallest meaningful value.

The NS16550A data sheet does not say if the **Modem Status Register** is writable, or what should happen if it is written to. The lower 4 bits (3:0) are indeed writable, and they will set or clear the modem "delta" bits (and the interrupt if enabled) just like real changes in the modem control lines. The Wax ASIC has duplicated this operation.

The table in the NS16550A data sheet indicates that bit 4 (loopback) of the **Modem Control Register** will always return a 0 on reads. This is not true. It will return the value that was written into this bit. The Wax ASIC has duplicated this operation.

If the THRE (transmitter holding register empty) interrupt is currently enabled, and there is no THRE interrupt pending, and the THRE interrupt is disabled and re-enabled, the NS16550A will generate a new THRE interrupt. The Wax ASIC has duplicated this operation. However, if the THRE interrupt is already enabled, and the **Interrupt Enable Register** is written so as to keep the THRE interrupt enabled (no change on that bit), the NS16550A will cause a THRE interrupt. The Wax ASIC will not.

If the transmitter fifo and the transmitter buffer are both empty, one would not expect changing to/from fifo mode to cause a THRE interrupt. The UART is transitioning from empty to empty, so there is no edge which should cause an interrupt. However, a THRE interrupt is generated whenever going to/from fifo mode. In addition, clearing the transmitter fifo via bit 2 of the **Fifo Control Register** also causes the THRE interrupt (regardless of whether it was already empty). The Wax ASIC has duplicated this operation.

The **Line Status Register** is writable in the NS16550A for factory testing. The **Line Status Register** in the Wax ASIC is not writable. With scan testing employed at the IC level, this was not necessary.

For the TEMT bit (transmitter empty) in the **Line Status Register**, it is never really defined when the transmitter shift register should be considered empty. The Wax ASIC waits until all of the data bits AND all of the stop bits have been sent out before considering the transmitter empty. This is probably the desired behavior if this bit is being checked to see if it is okay to change baud rates. However, if in loopback mode and data is transmitted, receive data available will be indicated before TEMT, because only the detection of the first stop bit is required for receiving data. This isn't necessarily a problem, but it may be unexpected. I don't really know what the real NS16550A does.

The NS16550A clears the receive fifo registers when it reads from them. The Wax ASIC does not. When the fifo empties, the NS16550A will always return 0's on reads. The Wax ASIC will return the value that was previously in that fifo location. It seems unwise to rely on a fifo value when receive data available (line status) indicates it is an empty fifo location.

The NS16550A data sheet says that a break is defined as received data being 0 for one full character time (start + data + parity + stop bits). However, it really calls something a break if received data is 0 for start + data + parity + 1/2 first stop bit. Basically, the NS16550A indicates all of the receiver line status interrupts (and receive data available) half way through the first stop bit. The Wax ASIC has duplicated this operation.

The NS16550A says that on a framing error, it considers the 0 stop bit to really be a start bit, so it samples it twice and then starts looking for the next data bit. Who knows what they are trying to say. The operation of

the NS16550A is non-deterministic in this case. Normally, the NS16550A will behave as if the 0 stop bit is a start bit, and it will take the next bit as data. However, sometimes it treats the next bit as a start bit rather than data. The Wax ASIC always treats the next bit as data. Additionally, the Wax ASIC does a 2 out of 3 sample on that stop bit anyway, so it shouldn't falsely be seen as a 0.

The NS16550A data sheet lies about the line status interrupts for the receiver and when they are cleared. It says that reading the **Line Status Register** is the only way to clear a break indication, an overrun error, a parity error, or a framing error. This is true in non-fifo mode, but in fifo mode, if the **Line Status Register** is not read, but the receive data is read, the **Line Status Register** will be updated with the info from the next fifo location. If the next location doesn't have any errors, the receiver line status errors have effectively been cleared, while never having read the **Line Status Register**. The Wax ASIC has duplicated this operation.

From looking at the ordering of the interrupts in the table for the **Interrupt ID Register** in the NS16550A data sheet, it would appear that a receiver data available interrupt would have priority over a character time-out interrupt. Both are considered "second" priority, but the receiver data available interrupt is listed first. In reality, the character time-out interrupt has priority. With the fifo trigger level set at 1 byte, an interrupt caused by receiver data being available will turn into an indication of a character time-out interrupt if the receiver is not read soon enough.

4.5 RS-232 Signals

Below is a description of the ports allocated on the Wax ASIC for RS-232 signals:

Table 2 RS-232 Signals

Signal	Pin	Direction	Description
rs232Txd	236	Out	Transmit data
rs232Rxd	235	In	Receive data
rs232Cts	239	In	Clear to send
rs232Rts	240	Out	Request to send
rs232Dcd	1	In	Data carrier detect
rs232Dsr	2	In	Data set ready
rs232Ri	237	In	Ring indicator
rs232Dtr	238	Out	Data terminal ready

Table 3 shows suggested connector pin-out for the RS-232 interface. Both a nine position male D connector, such as those used in PC products, and a twenty-five position female D connector, such as those used in traditional workstations, are shown.

Table 3 RS–232 Connector Pinouts

Signal	Direction	9-Pin	25-Pin
rs232Txd	Out	3	3
rs232Rxd	In	2	2
rs232Cts	In	8	5
rs232Rts	Out	7	4
rs232Dcd	In	1	8
rs232Dsr	In	6	6
rs232Ri	In	9	22
rs232Dtr	Out	4	20
ground		5	7

4.6 Sample Schematic

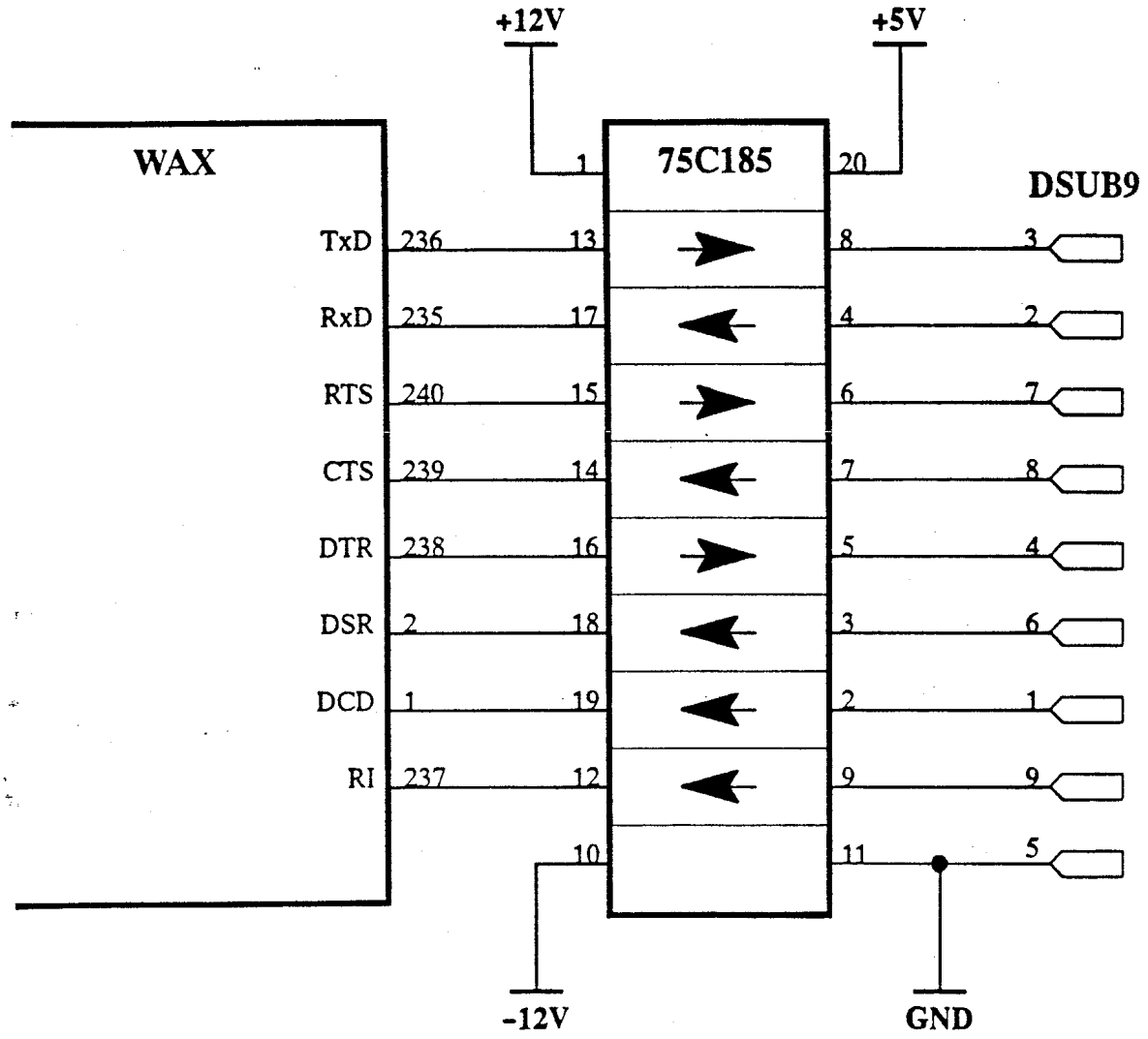


Figure 13 RS-232 Schematic

5 HP-HIL Interface

5.1 Description

The Wax ASIC contains the circuitry to emulate an Intel © UPI-42 microcontroller to provide access to the HPHIL master link controller. The circuitry is compatible with HP P/N 1820-4784, which is used in previous HP9000/7XX workstations. Unlike previous workstations, the Wax ASIC does not contain a real time clock or a sound generator. The design of this portion of the circuitry was leveraged from work done for the Stiletto ASIC by MCSY in Fort Collins.

5.2 HP-HIL Registers

5.2.1 Base Address

The GSC base address for this interface is 0xF020 1000. The GSC+ base address for this interface is 0xFFE0 1000.

5.2.2 Register Overview

Table 4 shows the register assignments and addresses for the registers in the HP-HIL sub-system in the Wax ASIC. This interface looks like a UPI-42 mircocontroller connected to the GSC bus.

Table 4 HP-HIL Registers

Description	Offset	R/W	D7	D6	D5	D4	D3	D2	D1	D0
Assert Reset	0x000	W	X	X	X	X	X	X	X	X
Undefined	0x001-0x7FF									
8042 data	0x800	R/W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
8042 status/control	0x801	R	IStat Bit 7	IStat Bit 6	IStat Bit 5	Istat Bit 4	reserved	NMI reason	IBF	OBF
	0x801	W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Undefined	0x802-0x807									
PS-2 8042 data	0x808	R/W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
PS-2 8042 status/control	0x809	R	IStat Bit 7	IStat Bit 6	IStat Bit 5	IStat Bit 4	reserved	IStat Bit 2	IBF	OBF
	0x809	W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Undefined	0x80A-0xBFF									
Negate Reset	0xC00	W	X	X	X	X	X	X	X	X
Undefined	0xC01-0xFFF									

The HIL registers are defined as a word port only. When data is written to any register, all thirty-two bits on the data bus are written into the register regardless of the state of the byte enable bits associated with the transfer. When any register is read, all thirty-two bits of the data bus are driven. Unused bits are always read as zeros. The HIL registers may be accessed using multi-word transfers, but only the first word of the transfer occurs. During multi-word writes, the second and subsequent words are ignored. During multi-word reads, the first location accessed is driven as the first word of data, but subsequent words are not meaningful data. Correct GSC protocol is followed for multi-word transfers, the data is just meaningless.

5.2.3 Detailed Register Descriptions

It may be helpful to be acquainted with the UPI-42 8042 documentation from Intel and the HP dwg A-1820-4784-2 *Firmware Doc Rev B.* while reading the following description.

5.2.3.1 Assert Reset register

This register puts the 8042 in a reset state until the *Deassert Reset* register is written. The power up value of this register is that reset is **asserted**.

5.2.3.2 8042 Data

This register is used to send and receive data to/from the 8042. Data should not be written unless status register bit *IBF* is a zero; and data should not be read unless status register bit *OBF* is a one.

5.2.3.3 8042 Status

The upper four bits (7:4) are used for communicating interrupting conditions as follows:

Value	Description
0000	Not used
0001	10 ms periodic interrupt
0010	Special purpose timer interrupt
0011	Both a 10ms periodic and special purpose timer interrupt
0100	The *8042 Data* register contains a byte that was requested
0101	The *8042 Data* register contains a STATUS CODE associated with HIL
0110	The *8042 Data* register contains DATA associated with HIL
0111	Power up reset and selftest was completed successfully
1000	The *8042 Data* register contains a key (both shift and control)
1001	The *8042 Data* register contains a key (only control)
1010	The *8042 Data* register contains a key (only shift)
1011	The *8042 Data* register contains a key (no shift or control)
1100	The *8042 Data* register contains a RPG count (both shift and control)
1101	The *8042 Data* register contains a RPG count (only control)
1110	The *8042 Data* register contains a RPG count (only shift)
1111	The *8042 Data* register contains a RPG count (no shift or control)

Bit 3 is **reserved**. It will be set when the last write was to the *8042 control* register; it will be a zero when the last write was to the *8042 data* register.

NMI reason: In the case of an NMI bit 2 will be set if there is a fast-handshake interrupt, else it will be zero to indicate that the RESET key was pressed.

IBF (Input buffer full): This bit will be set when a write to either the *8042 Control* or *Data* occurs and will be cleared when the 8042 reads the data. Writes should occur only when this bit is a zero.

OBF (output buffer full): This bit will be set when the 8042 has valid data for the processor to read from the *8042 Data* register.

5.2.3.4 8042 Control

This register is used to send commands and data to the 8042. The commands are briefly described below. For more detailed information set aforementioned documentation.

LOAD commands: The *8042 Data* register will be loaded with the appropriate value and an interrupt will occur.

0x00-1F: Request a byte of data from internal RAM addresses 0x00-1F
0xF0-FF: Request a byte of data from internal RAM addresses 0xF0-FF

LOAD Timer Output Buffer commands: The timer output buffer (in internal RAM) will be loaded from the cycle interrupt counter, the fast handshake timer, or the real time clock. Use commands 0x13-0x17 to access the data.

0x31: Load the real time
0x36: Load the fast handshake time
0x3B: Load the delay interrupt time
0x3E: Load the cycle interrupt time

Set Interrupt Mask commands: These commands will set individual interrupt masks.
0x40-7F: The lower five bits set the following interrupting conditions masks. If a bit is set, the interrupt is disabled.

Bit 0: Keyboard, RPG and HIL interrupt mask
Bit 1: RESET key NMI mask
Bit 2: Timer interrupt mask
Bit 3: Periodic system interrupt mask
Bit 4: Fast handshake interrupt mask
Bit 5: Reserved, set to 0.

SET-UP commands: These commands are used to set several state variables used by the 8042. Each of these commands should be followed by one or more *8042 Data* register writes.

0xA0: Set repeat delay (1 byte)
0xA2: Set repeat delay (1 byte)
0xA3: Set beep info (not supported in the Wax ASIC)
0xA6: Set RPG interrupt rate (1 byte) in 10ms increments
0xAD: Set the real time 10ms time (not supported in the Wax ASIC)
0xAF: Set the real time day (not supported in the Wax ASIC)
0xB2: Set the fast handshake delay (2 bytes)
0xB4: Set the real time match interrupt value (3 bytes)
0xB7: Set the delay interrupt value (3 bytes)
0xBA: Set the cycle interrupt value (3 bytes)

0xC1: Set the RAM data output pointer associated with the 00 command

0xE0-EF: Write to internal RAM locations 0xF0-FF

Trigger commands: These commands transfer data between the internal RAM buffer and the real time clock (BBRTC) and HIL.

0xC2: Write BBRTC. Not supported in the Wax ASIC

0xC3: Read BBRTC. Not supported in the Wax ASIC

0xC4: Write Beeper. Not supported in the Wax ASIC

0xC5: Write HIL. Data is transferred from the buffer to the HIL MLC

5.3 HIL communication

The HIL interface is controlled by the Master Link Controller (MLC). The 8042 sits between the software and the MLC (usually getting in the way). The two HIL interrupt status codes 0x5X and 0x6X are for reporting data returned from the loop due to an 8042 initiated *poll* command. The contents of the *8042 Data* register hold additional information on who responded to the poll and what data they sent out. See section NO TAG on page NO TAG for more detailed information of the MLC.

5.3.1 Interrupt Status 5X

Interrupt status 5X precedes an interrupt status 6X. The 5X reports header information in the *8042 Data* register as follows:

Status 5X Data

0	X	X	AP	CMD	A2	A1	A0

AP: Indicates that data to follow was returned in response to an auto-poll

CMD: When set, the data that follows is the HIL command

A2-0: Contains the address of the HIL device that sent the following data

5.3.2 Interrupt Status 6X

Interrupt status 6X reports that there is data (or a command) from an HIL device in the *8042 Data* register. This data is in response to an 8042 initiated auto-poll or a user initiated HIL command.

5.4 BBRTC communication

The battery backed-up real time clock is not supported in the Wax ASIC. 8042 commands to write and read from the battery backed-up real time clock may be sent to the 8042 without any ill effects; however, any data read will be invalid.

5.5 Sound Generator communication

The sound generator is not supported in the Wax ASIC. 8042 commands to write and read from the sound generator may be sent to the 8042 without any ill effects; however, any data read will be invalid.

5.6 Configuration and Identification registers

There are many configuration and identification registers that exist in the 8042 internal RAM. These can be read, and sometimes written, with 8042 commands.

5.6.1 Configuration register, R11

R11

0	0	1	KNP	NKR	0	KC1	KC0

KNP: Keyboard is present if zero

NKR: N key rollover is implemented when set

KC1-0: Keyboard code

00	ITF
01	Reserved
10	98203C
11	reserved

5.6.2 Language register, R12

This register holds the keyboard language code for the first keyboard. Refer to the keyboard ERS's for more detailed information.

5.6.3 Nimitz keyboard address map register, R78

For each Nimitz keyboard on the loop the bit corresponding to the keyboards address will be set. A Nimitz keyboard at address 1 will be reflected in bit 0.

5.6.4 "Cooked" keyboard address map register, R79

For each "cooked" keyboard on the loop the bit corresponding to the keyboard's address will be set. A cooked keyboard at address 1 will be reflected in bit 0. Clearing bit 1 will put the keyboard at address 2 into RAW mode.

5.6.5 HIL interface status register, R7A

LPSTAT

RF	X	X	X	RS	LC2	LC1	LC0

RF: If set, loop reconfiguration failed

RS: If set, loop reconfiguration was successful

LC2-0: Count of devices on the loop

5.6.6 HIL interface control register, R7B

LPCTRL

RL	X	X	COOK	X	DRLR	DRLE	AP

RL: If set, loop will be reconfigured

COOK: If set, keyboards will be cooked

DRLR: If set, loop reconfiguration errors will not be reported

DRLE: If set, loop errors (parity, framing, etc.) will not be reported

AP: If set, the 8042 will perform auto-polling at 20ms intervals

5.6.7 HIL loop reconfiguration counter, R7D

This register is incremented each time the loop is reconfigured.

5.6.8 Extended configuration register, R7E

EXCNFG

0	0	0	1	0	0	1	1

5.6.9 Selftest result register, R7F

This register will be set to 0x55 when the selftest has passed.

5.7 Software Hints

5.7.1 Power up reset

Once the *Reset Deassert* register is written software must wait for at least 400ms before accessing the 8042. It is easiest to wait for the first interrupt which will report selftest passed, and read the *8042 Data* register.

5.7.2 Auto-polling and HIL access commands

Software should make certain that HIL auto-polling is disabled before sending commands to the MLC. This can be accomplished by clearing bit 0 of register R7B.

5.8 HP-HIL Master Link Controller

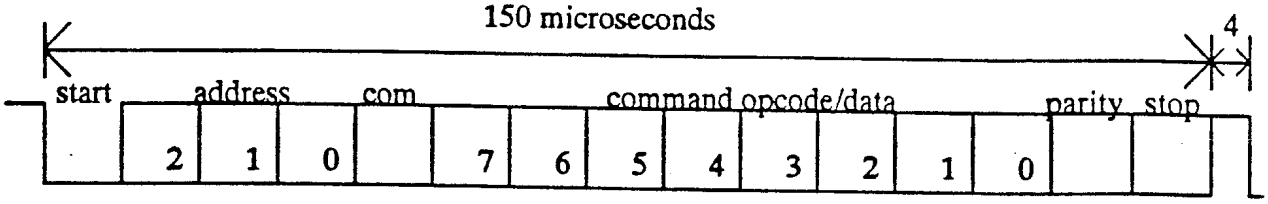
5.8.1 Description of the HP-HIL MLC

The master link controller (MLC) provides the hardware interface between the host system and all the devices connected to the HP-HIL. The MLC accepts commands from the system processor and transmits the information to the HIL devices in the proper format. The MLC also transmits data from the HIL devices back to the system processor. Two eight-bit data buses are used to transfer data to and from the host processor. The MLC also has a 16-frame FIFO to reduce processor interruptions. In the Wax ASIC, the system processor is the 8042 module.

5.8.2 The Least You Need to Know about HIL to "Get By"

The Hewlett Packard Human Interface Link (HP-HIL) is an asynchronous serial communication protocol that allows a computer or terminal to talk to various devices. Devices that speak HIL include keyboards, mice, button boxes, knob boxes, ID modules, graphics tablets and those thingeys that go beep.

Data travels around the Link in a fixed format called a frame. Every frame consists of 15 bits of information including start, device address, command, opcode/device data, parity and stop bits. Each frame is transmitted at a rate of 10 microseconds per bit. A minimum of 4 microseconds of idle time is required between frames. The idle state of the Link is a logic 1. 1 is +5 volts.



FRAME FORMAT

The start bit is 0 indicating the beginning of a frame transmission. Hardware that speak HIL sample this bit three times during the 10 microsecond interval to insure that it is a frame transmission and not a glitch.

HIL has the capacity to address up to seven unique devices. This requires three bits in the frame. The address 0 is the universal address to which all devices should respond. The balance of the addresses, 1 through 7, are assigned to individual devices during the configuration process. Some devices, such as the knob box, require more than one HIL address.

The command bit differentiates device data from a device command. If this bit is one, the next eight bits are a command opcode. If the bit is zero, the next eight bits are device data.

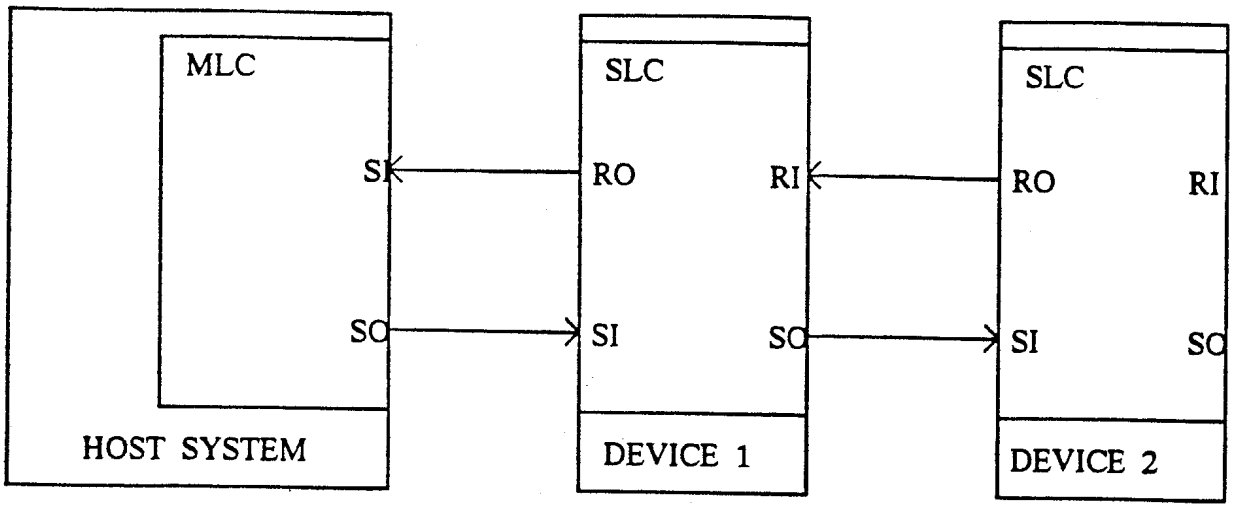
The next field consists of the actual command or data. A device command may be a hard reset or a request for data. Device data can be something like a key from a keyboard or the X and Y coordinates from a mouse.

The parity bit provides error detection for every frame. The sum of all bits, including start and stop, should be odd. HIL hardware understands this and sets the parity bit accordingly before transmission. If a device receives a frame that doesn't follow this protocol, an error is flagged. Parity errors will be discussed later.

The stop bit is 10 microseconds of logic 1. This signals the end of the frame. If the stop bit is not a logic 1, an error is flagged. This is a framing error and will also be discussed later.

Every computer that speaks HIL has one Master Link Controller. Every HIL device has one Slave Link Controller (SLC). There may be 0 to 7 SLCs but only one MLC on the Link. The MLC has two HIL signals: serial in (SI) and serial out (SO). The SLC has four HIL signals. Besides SI and SO, there is also return data in (RI) and return data out (RO). The signal connections for a two-device configuration would look like the following diagram.

Upon power up, all HIL devices are in loop-back mode. In loop-back mode, the serial in signal is internally connected to the return data out. During configuration, each device is assigned an HIL address and has reported on what type of device it is. After configuration, all devices except the last device on the link are in pass-thru mode. In pass-thru mode, the serial in signal is connected to the serial out. The return data in is passed on to the return data out signal. With all the devices but the last one in pass-thru mode, all packets from the MLC will be passed on to the next SLC until it is received by the last HIL device. The last HIL device, in loop-back mode, will send the packet back to the previous device via its return data out signal.



Frames typically originate at the host. The host processor initiates a frame transmission by writing to certain registers in the MLC. The MLC assembles the data into the proper format and transmits the frame. Upon receiving the frame, the SLC can do one of three things:

1. Retransmit the frame — If the address field of the frame is not the universal address (0) or does not match the address of the device, the SLC will retransmit the frame to the next device.
2. Trap an error — If a parity or framing error is detected, the frame is discarded and a command frame with an error opcode is transmitted to the next device.
3. Process the frame — If there is an address match (frame address is 0 or the device address) and no errors, the SLC will interrupt its device processor which will handle the command. When the device processor is finished, it may load the SLC with data to pass on. The SLC will format the frames and retransmit them to the next device.

If data is sent back to the host in response to a command, then the command trails the data frames. This allows the MLC to determine when all of the data related to the last command has been received. It is easy for the MLC to determine when the command has returned by monitoring the command bit of every frame received. Upon receiving the command frame back, the MLC interrupts the host processor.

There can only be one command active on the Link at one time. The host processor typically issues a command to the Link via the MLC and then waits for the MLC to interrupt with data. If the MLC receives a frame without first sending a command frame, it will be a reset command from a device or an error.

HP-HIL has a command set by which all necessary functions to set up and maintain the Link are performed. Basic operation of the Link can be broken down into Link configuration, error recovery, data extraction and device identification.

Link configuration is the process that sets up the Link so it can provide the host with data in an orderly manner. Configuration usually occurs at power up and any time when error recovery calls for a reconfiguration of the Link. The configuration process assigns a unique address to each device on the Link. It also sets each device in the proper mode (pass-thru or loop-back) so that the HIL frames

will be looped back by the last device on the Link. In the process, each device may be requested to identify what type of device it is.

Errors may occur under various conditions on HP-HIL. HP-HIL provides for several levels of error recovery so that if an error occurs, the recovery process will preserve the maximum amount of data and minimize the interaction with the Link. The error recovery is initiated by the host processor but detected by both the device SLCs and the host MLC.

Data extraction commands let the host gather information from the HIL devices on the Link. Character data, pointer position data, status, and device specific data can be communicated back to the host using data extraction commands. When used in conjunction with the device identification commands, the data can be processed by the host system.

Identification commands are used to determine the type of attached devices. They are also used to gather general characteristics of these devices. Device characteristics include resolutions, directional information, and information on how the device reports data.

The previous paragraphs explain the gist of HP-HIL. If one wishes to learn more about HP-HIL to become a local HIL guru or perhaps just to woo babes, obtain a copy of the HP-HIL Technical Reference Manual. The product number on this document is 45918A. The document covers the philosophy behind HP-HIL, the hardware requirements, and the command opcodes. It is a well written document from which much of the previous discussion was plagiarized.

5.8.3 MLC Operation

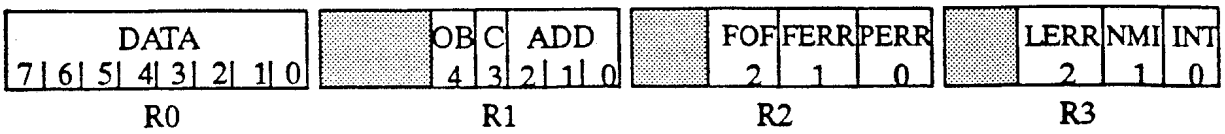
HIL commands or data are sent to HIL devices in two steps. First, the host processor writes the command bit and address to register **W1**. Next, the processor writes the command opcode or device data to **W0**. The MLC computes the parity bit, adds the start and stop bits and transmits the frame.

The MLC stores all the return frames in its FIFO. When the MLC receives the initial command or an error code, it interrupts the processor (int=0). The host processor clears the interrupt by reading register **R3**. The processor then reads the frames stored in the FIFO by alternately reading **R1** (command and address bits) and **R0** (data) until the command frame is read. Each time **R0** is read, the frames inside the FIFO shift up one position. **R0** and **R1** are the top of the FIFO.

5.8.4 Register Definitions

There are four read-only registers (**R0-R3**) and four write-only registers (**W0-W3**). These registers are accessed by either a read or write strobe with the corresponding address bits (A0-A1) asserted. The reading of all registers but **R1** will cause the values in them to change. As expected, writing to any of the write registers changes their value. The read registers can't be written. The write registers can't be read.

OUTPUT REGISTERS



INPUT REGISTERS

5.8.4.1 R0

R0 contains the command opcode or data from the top HIL frame in the FIFO. After a command is sent out to the HIL link, the first frame received will be put in the top of the FIFO. The next frame received will be put in the second position and so on. Reading **R0** will cause all FIFO data to shift up one position and the top data will be overwritten with data in the second position.

5.8.4.2 R1

R1 contains the command bit and address of the frame at the top of the FIFO. This information should correspond to the data in **R0**. Since reading **R0** will destroy the data it contains, frame information should be read by reading **R1** then **R0** for each frame. **R1** should always be read before **R0**. All the frames in the FIFO can be read by alternating reads from **R1** and **R0**.

R1 is also home to the OB (output busy) bit. This bit reflects the operation of the output shifter. If the transmit state machine is transmitting a frame, the OB bit will be set. Software should poll this bit to insure that it is clear before sending out a frame. The reading of **R1** will not destroy its contents.

5.8.4.3 R2

R2 contains the error bits: FOF, FERR, and PERR. FOF is the FIFO overflow bit. It is set when the MLC is receiving more frames than it can store. The HIL specification limits the number of frame transmissions in one HIL transaction to 16. When a frame is received and the FIFO is full, this bit is set and the frame is stored in the input shifter. The input shifter can be accessed by reading the FIFO (**R1** then **R0**) 17 times. When the first frame is read, the input shifter gets copied into the last position in the FIFO. The seventeenth frame read will be the one from the input shifter.

Bit one of **R2** is the framing error bit (FERR). A FERR occurs when the stop bit of an incoming frame is 0. When a framing error occurs, the FERR bit is set and the MLC enters a resynchronization routine. To resynchronize the input, the MLC monitors the serial input high for 150 microseconds. If serial in goes low during that time, the timer is reset and the procedure is repeated. The input state machine is then reset. If the MLC was receiving frames (a command frame had been transmitted but not returned yet) when the framing error occurred, the INT bit will be set with the FERR bit. If the

MLC was not receiving frames when the framing error occurred, it will assume there was no real frame transmission and the FERR and INT bits will hold their current values.

To prevent framing errors caused by false starts (incorrectly indicating a frame reception), the input state machine will sample each bit period three times and determine the sense to be the best-of-three samples. This makes the MLC more resistant to ESD glitches.

Bit 0 of R2 is the parity error bit (PERR). A parity error occurs when the sum of all the bits in a frame, including the start and stop bits, is even. If the MLC detects a parity error when it is receiving frames, a command frame has been sent but hasn't yet returned, the PERR bit and the INT bit are set. If the MLC is not receiving frames when a parity error occurs, the input state machine is reset and no bits are set.

R2 is a destructive register. It will clear all bits when it is read.

5.8.4.4 R3

R3 houses some real riffraff. Bit 2, the LERR bit, is the loop error bit. It is set when the MLC is in autopoll mode receiving frames and a rising edge is detected on the AP pin. This is a serious error. If this error is seen, there is a major flaw in the hardware driving AP or the software. Since the author was working on the perfect project with flawless hardware and infallible software, this bit was not implemented. Also, since the autopoll mode is not supported or needed in the Wax ASIC, there was no need for implementing LERR. There are versions of the MLC that do support autopoll.

Bit 1, NMI, is set when a device on the HIL loop sends the dreaded FB command. FB is a system hard reset sent by an HIL device to signal a non-maskable interrupt or reset. When FB is received, the nmi signal is asserted (low) for 5 microseconds. The reception of the FB command has no affect on the INT bit. The FB command, however, will not be loaded into the FIFO.

Bit 0, INT, is set whenever a command frame is returned and whenever FOF, FERR, PERR, or LERR are set. The interrupt signal is a complement of this bit. When INT is set, the interrupt signal is asserted (low).

Reading R3 will clear the three bits. Therefore, reading R3 clears the interrupt.

5.8.4.5 W0

W0 is the write frame data register. The data or opcode of the HIL frame to be transmitted is written to W0. Whenever W0 is written, an HIL frame is assembled and transmitted. Therefore, developers should always write to W1 before writing to W0 to insure proper data in the HIL frame.

Only one HIL command should be active on the HIL link. When there is a write cycle to W0, the MLC checks the command bit in W1 to see if a command is about to be transmitted. If the command bit is set, the MLC resets the FIFO pointer. If there was any HIL frames in the FIFO, they will be overwritten by the HIL frames returning in response to the command being transmitted.

5.8.4.6 W1

W1 contains the command bit and the HIL device address for the next frame to be transmitted. As mentioned earlier, if the command bit is set, the frame is an HIL command. If it is clear, it is data.

5.8.4.7 W2

The ignore poll frame (IPF) and test bit reside in W2. The ignore poll frame bit is not implemented in this version of the MLC.

As its name implies, the test bit is used for testing the MLC. Setting this bit, puts the MLC in test mode. In this mode, the serial in signal is internally tied to the serial out signal. The external

serial out is pulled high so frame transmissions are not seen by devices on the HIL link. This mode can be used to fill the FIFO with data by writing to **W1** and **W0**, repeatedly. The FIFO can then be emptied by reading **R1** and **R0** repeatedly. By comparing the data read with the data written, one can verify that the MLC is working properly. This type of loop-back test exercises a great portion of the MLC.

5.8.4.8 W3

The autopoll enable bit is the only bit in this register. Since autopoll is not a supported mode in teh Wax ASIC, this register is not used.

5.8.5 FIFO

The MLC has a 16-frame first-in-first-out queue (FIFO) for storing incoming frames. The FIFO has a pointer that points to the next empty location in the FIFO. The FIFO grows from the top down. As mentioned before, **R0** and **R1** reflect the top frame in the FIFO.

As a frame is received, it is placed into the FIFO and the FIFO pointer is incremented. If more than 16 frames are received, the FIFO overflow bit (FOF) is set and incoming frames overwrite the last frame in the input shifter. When the next command frame is received in the input shifter, the MLC will set the INT bit and the host processor will be interrupted. The input shifter can be thought of as the 17th frame in the FIFO. It can be read by reading the FIFO (alternating reads to **R1** and **R0**) 17 times. Subsequent reading of the FIFO will simply repeat the contents of the input shifter.

As frames are read, the FIFO pointer is decremented and all frames shift up one position in the FIFO. Since reading **R0** signals the FIFO that a frame is being read, always read **R1** before reading **R0** or the contents of **R1** will be lost as the next frame shifts up.

If a frame with bad parity or a framing error is received, the data error frame (opcode FC with a universal address (0) is loaded into the FIFO, the parity error bit (PERR) or the framing error bit (FERR) is set, and operation continues as if a command frame were received.

The FIFO will never contain more than one command frame (and its associated data) at one time. If the contents of the FIFO aren't read when when the processor is interrupted, the next command sent out will overwrite the FIFO data when it returns. When the host processor writes data to **W0** (signalling a frame transmission), the FIFO pointer is set to zero pointing to the top of the FIFO. All frames received after this point will overwrite any frames in the FIFO.

Reception of a master hard reset frame (opcode FB) is a special case. When a master hard reset frame is received, the nmi signal is lowered (nmi=0) for 5 microseconds and the NMI bit in **R3** is set. The frame is not shifted into the FIFO. Further frames can be received. The interrupt signal and bit are not affected by a master hard reset.

The FIFO is not cleared nor set at power up. The only way to initialize the FIFO is to send and receive a command. At power up, the FIFO pointer is set to 0.

5.8.6 Software Tips

This section is specifically for folks using the MLC module outside of the Wax ASIC. All HIL transactions inside the Wax ASIC talk to the MLC via the 8042. If you wish to send data out to a device on the HIL, consult the 8042 documentation.

For those using the MLC module without the 8042, make sure that the OB bit is clear before writing to **W0**. If the bit is set and a frame needs to be transmitted, poll the bit until it is clear. Then write to **W0** and the frame will be transmitted. The OB bit will not be set for more than 154 microseconds.

When clearing and interrupt by reading **R3**, always read **R2** immediately after. R2 contains error bits that will help determine if the interrupt was set because a command frame returned or if there was an error.

R3 should not be read within one microsecond of the falling edge of the interrupt signal. If R3 is read within one microsecond of the interrupt, it will not be cleared. If this is a potential problem, the user may want to read **R2** before reading **R3** to allow this time period to lapse.

5.8.7 MLC vs. Cerberus

There are a few differences between the Wax ASIC's MLC and the old Cerberus (1RD2-6201). The differences are transparent to the Wax ASIC users because there is no way to talk directly to the MLC. All transactions with the MLC are through the 8042.

The greatest difference is the absence of the autopoll function. This was not necessary since one of the functions of the 8042 is to initiate autopolling on the HIL. As mentioned before, there are three register bits associated with autopoll that aren't implemented: LERR, APE, and IPF.

The internal clock of Cerberus is twice the frequency of the external crystal. The system clock of the MLC is the same speed as the external clock so transactions are slower. This, again, doesn't matter since all transactions are through the 8042.

Cerberus double buffers its outgoing HIL frames. The MLC doesn't. The 8042 always checks the OB before initiating a frame transmission. If the MLC is busy sending out a frame, the 8042 will wait until it isn't, before initiating another frame transmission.

5.9 HP-HIL Disable

The HP-HIL interface may be disabled entirely by grounding the hilSo output during power up. When this is done, accesses to the HP-HIL address space will not generate a gscReadyL. If the hilSo signal is not grounded, the HP-HIL interface will function as described.

5.10 Sample Schematic

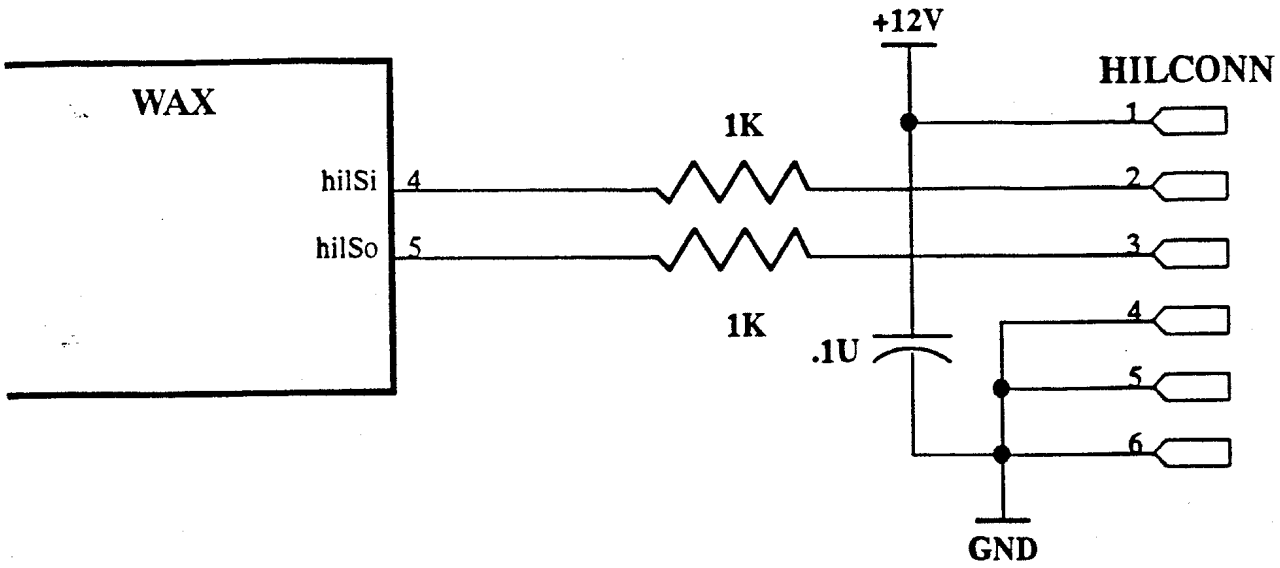


Figure 14 HPHIL Schematic

6 HPIB Interface

6.1 Description of Interface

The HPIB interface that is built into the Wax ASIC is compatible with HPIB interfaces used in Series 300 controllers previously. The design requires three chips external the the Wax ASIC: the TMS9914, SN75163, and SN75162 . This design of this portion of the circuitry was originally designed for the Stiletto ASIC by MCSY in Fort Collins. Below is a table showing the registers used on the HPIB portion of the Wax ASIC:

The HPIB interface was designed with two goals in mind. The first goal was to be as close as possible to be software compatible with the 82335 ISA HPIB card. The second goal was to add DMA to the definition to increase performance and to decrease processor resources need to use the HPIB. To achieve the first goal, the register definition the HPIB interface is a superset of the 82335 ISA HPIB card. In the software hints section, there are descriptions of what the differences are between the 82335 ISA HPIB card and this interface. To achieve the second goal of increasing performance and to decrease processor resources, a 32 byte Fifo with DMA capability was added to the interface. The Fifo transfers byte(8 bit) wide data to the 9914 and word wide(32 bit) data to main memory.

Also added to the interface was a character matching function. The character matching function allows software to set a character value to stop the DMA process. When this feature is enabled, the hardware will stop transferring data when it transfers a data value equal to the special character value. The hardware will then interrupt the software to let it know that the DMA is finished. This is useful to some instruments that do not tag the end of their data with a EOI transfer but instead make the last byte a special character like a <line feed>.

6.2 Register Definitions

The following describes the registers that are part the the HPIB interface. After defining the base address of the interface, there are tables that summarizes the registers. Following the tables, there is a detail description of the registers and their individual bits.

6.2.1 Base Address

The GSC base address for the HPIB interface is 0xF020 5000. The GSC+ base address for the HPIB interface is 0xFFE0 5000.

The HPIB registers are defined as a word port only. When data is written to any register, all thirty-two bits on the data bus are written into the register regardless of the state of the byte enable bits associated with the transfer. When any register is read, all thirty-two bits of the data bus are driven. Unused bits are always read as zeros. The HPIB registers may be accessed using multi-word transfers, but only the first word of the transfer occurs. During multi-word writes, the second and subsequent words are ignored. During multi-word reads, the first location accessed is driven as the first word of data, but subsequent words are not meaningful data. Correct GSC protocol is followed for multi-word transfers, the data is just meaningless.

6.2.2 Summary of HPIB Registers

Table 5 HPIB Registers

Description	Offset	R/W	(MSB)							(LSB)
ID register	0x000–0x003	R	32 bit word – 0x00000001							
Clear Interrupt	0x000–0x003	W	X							
Reset	0x004–0x007	W	X							
Undefined	0x008–0x7FE									
ISA Status	0x7FF	R	Intr pending	Intr enable	?	DMA enable	?	System cntlr	?	?
ISA Control	0x7FF	W	X	X	X	X	System enable	Intr enable	X	DMA enable
DMA Address	0x800–0x803	R/W	32 bit Address							
DMA Count	0x804–0x807	R/W	32 bit Count							
DMA Status	0x808	R	?	empty	full	flush	?	Bus Error	Match enable	DMA direction
DMA Control	0x808	W	X	X	X	flush	X	X	Match enable	DMA direction
DMA Character Match Value	0x809	R/W	8 bit Character Match Value							
FIFO Address	0x80A	R/W	X	X	X	5 bit Address				
FIFO Data	0x80B	R/W	8 bit Data							
Undefined	0x80C–0x80F									
Extended Control	0x810	W	X							
Extended Status	0x811	R	Intr pending	Intr enable	terminal count	char match	?	?	~active cntlr	9914 Intr
Undefined	0x812–0x813									
I/O FIFO Pointer	0x814	R	?	?	?	5 bit Pointer				
Processor FIFO Pointer	0x815	R	?	?	?	5 bit Pointer				
Undefined	0x816–0xFF7									
9914 Registers	0xFF8–0xFFF	R/W	Refer to the 9914 register definition for details							

6.2.3 Detailed HPIB Register Descriptions

6.2.3.1 ID Register

The ID register is the Identification Number of the interface. This is an arbitrary number that the Stiletto team assigned to the interface and is identical in the Wax ASIC. Previous designs(ASP) did not have any

method of identifying an interface. As interfaces become obsolete, the address space that it originally occupied will not be able to be reused because there will be no easy way to identify which interface, new or old, is occupying that address space. The register is a 32 bit register with a 32 value of one(1).

6.2.3.2 Clear Interrupt

The Clear Interrupt register clears the interrupt bit in the **ISA Status register** and the **Extended Status register**. A write to any byte in this address range will clear the interrupt bit.

6.2.3.3 Reset Register

The Reset register resets the interface. A write to any byte in this address range will reset the internal state machines of the interface and will also assert the reset line to the 9914 for 64 GSC clocks. At 37.5 MHz, this signal will be asserted for 1.7 µsecs. At 30MHz, this signal will be asserted for 2.1 µsecs. The 9914 spec for the reset line is a minimum of eight 9914 clocks. In a typical Wax system, the 9914 runs at 5MHz so the minimum reset assert time is 1.6 µsecs.

6.2.3.4 ISA Status register

The ISA Status register is a software compatible register to the 82335 ISA card. It contains the basic status bits for the interface.

Table 6 ISA Status

Description	Offset	R/W	(MSB)							(LSB)
ISA Status	0x7FF	R	Intr pending	Intr enable	?	DMA enable	?	System cntlr	?	?

Intr pending - Interrupt Pending bit. This bit is asserted when either the 9914 has interrupted, the DMA controller has reached terminal count, or the DMA controller has had a character match interrupt. On reset, this bit is set to a 0. It also can be cleared by writing to the **Clear Interrupt register**. This bit will be driven as an interrupt to the processor if IEN is set. If IEN is not set, this bit can still be set but it is not driven as an interrupt to the processor.

Intr enable - Interrupt Enable bit. This bit reflects the value of Intren in the **ISA Control register**.

DMA enable - DMA Enable bit. This bit indicates that DMA is currently enabled. It is enabled by writing a 1 in the **ISA Control register**. It is cleared by several methods. One is to write a 0 in the **ISA Control register**. The second is by reaching terminal count in the **DMA Count register**. The third is by having a character match occur with the character matching enabled. The last is by getting a buserror during a dma transfer.

System cntlr - System Controller bit. This bit reflects the inverse of the value of SYSEN in the **ISA Control register**. This bit controls whether the interface is considered the system controller.

6.2.3.5 ISA Control register

ISA Control register is a software compatible register to the 82335 ISA card. It contains the basic control bits for the interface.

Table 7 ISA Control

Description	Offset	R/W	(MSB)							(LSB)
ISA Control	0x7FF	W	X	X	X	X	System enable	Intr enable	X	DMA enable

System enable - System Controller enable bit. A 0 means that the interface is the system controller. On reset, this bit is set to a 0.

Intr enable - Interrupt Enable bit. A 1 means that interrupts are enabled. On reset, this bit is set to a 0.

DMA enable - DMA Enable bit. A 1 means that the dma is enabled. On reset, this bit is set to a 0.

6.2.3.6 DMA Address

DMA Address register is the 32 bit register that contains the current address that the DMA state machine will use when transferring data to/from memory. When loading the **DMA Address register**, the address MUST be word aligned(lower two address bits must be equal to zero).

6.2.3.7 DMA Count

DMA Count register is a 32 bit register that contains the number of bytes that the DMA state machine still has to transfer before reaching terminal count.

6.2.3.8 DMA status

DMA Status register is a register that contains the status bits specific to the DMA controller.

Table 8 DMA Status

Description	Offset	R/W	(MSB)							(LSB)
DMA Status	0x808	R	?	empty	full	flush	?	Bus Error	Match enable	DMA direction

Empty - Empty bit. This bit indicates when the Fifo is currently empty. This bit should only be used for debugging.

FULL - Full bit. This bit indicates when the Fifo is currently full. This bit should only be used for debugging.

Flush - Flush bit. This bit indicates that the Fifo is currently trying to flush its contents on a INBOUND transfer. This bit can be set several ways:

1. When 9914 interrupt is asserted.
2. When the **DMA Count register** is equal to zero.
3. Manually set by writing a 1 into the FLSH bit location in the **DMA Control register**.

This bit will clear when the 9914 interrupt is cleared and when the Fifo is empty. This bit will also clear if the dmadir bit in the **DMA Control register** is set to OUTBOUND transfer.

Bus Error - Bus error bit. This bit indicates that a bus error occurred during a DMA transfer. A bus error will occur when the DMA state machine attempts to address a memory location that does not exist. This bit will be cleared when the dmaen bit is set to a 1.

Match enable - Character matching enable bit. This bit reflects the current value of the CMEN bit in the **DMA control register**. When enabled, DMA will stop and assert an interrupt if a byte is transferred from the 9914 during an INBOUND DMA transfer that matches the value in the **DMA Character Match Value register**. This feature is designed to be used when an device signals the end of transmission with a certain character value like a line-feed.

DMA direction - DMA direction bit. This bit reflects the current value of the DDIR bit in the **DMA control register**. When the bit is equal to a 0, the transfer is an INBOUND transfer(from 9914 to memory). When the bit is equal to a 1, the transfer is an OUTBOUND transfer(from memory to 9914).

6.2.3.9 DMA Control

DMA Control register is a register that contains the control bits specific to the DMA controller.

Table 9 DMA Control

Description	Offset	R/W	(MSB)							(LSB)
DMA Control	0x808	W	X	X	X	flush	X	X	Match enable	DMA direction

Flush – Flush bit. Setting this bit to a 1 causes the Fifo to flush its contents on a INBOUND transfer. This bit is meant for debug only. The Fifo should flush automatically when the **DMA Count register** is equal to zero or when the 9914 interrupts. On reset, this bit is set to a 0.

Match enable – Character matching enable bit. Setting this bit to a 1 enables the character matching function. When enabled, DMA will stop and assert an interrupt if a byte is transferred from the 9914 during an INBOUND DMA transfer that matches the value in the **DMA Character Match Value register**. This feature is designed to be used when an device signals the end of transmission with a certain character value like a line-feed.

DMA direction – DMA direction bit. When the bit is equal to a 0, the transfer is an INBOUND transfer(from 9914 to memory). When the bit is equal to a 1, the transfer is an OUTBOUND transfer(from memory to 9914).

6.2.3.10 DMA Character Match Value

DMA Character Match Value register contains the character value that is used for comparison when the Match enable bit is set in the **DMA Control register.**

6.2.3.11 FIFO Address

FIFO Address register is an address pointer into the FIFO. Any access to the **FIFO Data register** will be performed on the FIFO address pointed to by the **FIFO Address register**. This register is meant to be used for debug purposes only.

6.2.3.12 FIFO Data

FIFO Data register is a window register into the FIFO. When this register is read from or written to, the FIFO byte pointed to by the **FIFO Address register** is accessed. This register is meant to be used for debug purposes only.

6.2.3.13 Extended Control

Extended Control register is currently architected into the register definition but the bits are not defined.

6.2.3.14 Extended Status

Extended Status register is an attempt to create one register that contains all the possible interrupting conditions so that software will be able to access only one byte to get the information. This register also contains the active controller bit that was not defined in the original **ISA status register.**

Table 10 Extended Status

Description	Offset	R/W	(MSB)							(LSB)
Extended Status	0x811	R	Intr pending	Intr enable	terminal count	char match	?	?	~active cntlr	9914 Intr

Intr pending - Interrupt Pending bit. This bit is asserted when either the 9914 has interrupted, the DMA controller has reached terminal count, or the DMA controller has had a character match interrupt. On reset, this bit is set to a 0. It also can be cleared by writing to the **Clear Interrupt register**. This bit will be driven as an interrupt to the processor if Intr enable is set. If Intr enable is not set, this bit can still be set but it is not driven as an interrupt to the processor. This bit is identical to the Intr pending bit in the **ISA Status register**.

Intr enable - Interrupt Enable bit. This bit reflects the value of Intren in the **ISA Control register**.

terminal count - Terminal Count bit. This bit indicates when the Terminal Count has been reached in the DMA transfer. Terminal Count is when the **DMA Count register** is equal to zero. If interrupts were enabled when the Terminal Count transitions to a one, an interrupt will be generated. It signifies that all the bytes have been placed in memory on INBOUND transactions or all the bytes have been transferred to the 9914 on OUTBOUND transactions.

char match - Character Match bit. This bit is set when the last INBOUND transfer from the 9914 matches the value in the **DMA Character Match Value register**. If the match enable bit is a one in the **DMA Control register**, the DMA direction bit is a 0(INBOUND) in the **DMA Control register**, and interrupts are enabled when the Character Match bit transitions to a one, an interrupt will be generated. When these conditions are meet, the DMA enable bit in the **ISA Control register** will be cleared. DMA can be resumed by writing a one(1) to the DMA enable bit in the **ISA Control register**.

~active cntlr - Not Active Controller bit. This bit reflects the status of the NCONT line on the 9914. It indicates when the 9914 is the Active Controller. A zero(0) means the 9914 is the current Active Controller.

9914 Intr - 9914 Interrupt. This bit reflects the status of the interrupt line from the 9914. A zero(0) means that the 9914 is currently asserting its interrupt line.

6.2.3.15 I/O FIFO pointer

The **I/O FIFO pointer register** is a read only register that indicates the current value of the 9914 or I/O side pointer into the FIFO. This register is meant to be used for debug purposes only.

6.2.3.16 Processor FIFO pointer

The **Processor FIFO pointer register** is a read only register that indicates the current value of the GSC or processor side pointer into the FIFO. This register is meant to be used for debug purposes only.

6.2.4 Summary of 9914 Registers

The 9914 registers are shown in Table 11 . The bits in Table 11 are reference in Motorola bit notation to maintain consistency with the data sheet from TI.

Table 11 9914 Registers

Description	Offset	R/W	D7 (MSB)	D6	D5	D4	D3	D2	D1	D0 (LSB)
9914 Int Status 0	0xFF8	R	INT0	INT1	BI	BO	END	SPAS	RLC	MAC
9914 Int Status 1	0xFF9	R	GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC
9914 Addr Status	0xFFA	R	REM	LLO	ATN	LPAS	TPAS	LADS	TADS	ulpa
9914 Bus Status	0xFFB	R	ATN	DAV	NDAC	NRFD	EOI	SRQ	IFC	REN
9914 Undefined	0xFFC	R								
9914 Undefined	0xFFD	R								
9914 Cmd Pass	0xFFE	R	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
9914 Data In	0xFFF	R	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
9914 Int Status 0	0xFF8	W	X	X	BI	BO	END	SPAS	RLC	MAC
9914 Int Status 1	0xFF9	W	GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC
9914 Undefined	0xFFA	W								
9914 Aux Cmd	0xFFB	W	cs	X	X	f4	f3	f2	f1	f0
9914 Address	0xFFC	W	edpa	dal	dat	A5	A4	A3	A2	A1
9914 Serial Poll	0xFFD	W	S8	rsvl	S6	S5	S4	S3	S2	S1
9914 Parallel Poll	0xFFE	W	PP8	PP7	PP6	PP5	PP4	PP3	PP2	PP1
9914 Data In	0xFFF	W	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

6.2.5 Detailed 9914 Register Descriptions

6.2.5.1 9914 registers

The 9914 registers are expanded in Table NO TAGNO TAG. These registers MUST be accessed in byte mode only. Accessing in half-word(16 bit) or word(32 bit) mode will give unpredictable results. For a complete explanation of the 9914 registers, refer to TI's *TMS9914A General Purpose Interface Bus(GPIB) Controller* Data manual. The following are definitions of most of the bits in the registers

A5–A1 – Primary address of the 9914.

APT – Address Pass Through. This will be a 1 if a secondary command is received. Disable for secondary addressing.

ATN – The attention line is low(true) on the bus.

BI – Byte In. This will be a 1 when a data byte has been received in the Data In register.

BO – Byte Out. This will be a 1 when there is room for a data byte in the Data Out register.

DAV – Device Available. Reflects the current value of the DAV line.

DCAS – Device Clear Active State. This will be a 1 when a DCL or SDC command is received.

END –This will be a 1 when the last byte received was tagged with a EOI.

EOI – End Or Identify. Reflects the current value of the EOI line.

ERR – Error. This will be a 1 if the source handshake becomes active and there are no accepters on the bus.

GET – Group Execute Trigger. This will be a 1 when a Group Execute Trigger command is received.

IFC – Interface Clear. This will be a 1 when the IFC line becomes true.

INT1 – This will be a 1 when an unmasked status bit in **Interrupt Status register** 1 is set to a 1.

INT0 – This will be a 1 when an unmasked status bit in **Interrupt Status register** 0 is set to a 1.

LADS – The device is addressed to listen.

LLO – Local lockout is in operation.

LPAS – The 9914 is in the listener primary addressed state.

MA – My Address. This will be a 1 when teh 9914 recognizes its primary talk or listen address.

MAC – My Address Change. This will be a 1 when the 9914 has received a command which changes the address.

NDAC – Not Data Accepted. Reflects the current value of the NDAC line.

NRFD – Not Ready For Data. Reflects the current value of the NRFD line.

REM – The device is in the remote state.

REN – Remote Enable. Reflects the current value of the REN line.

RLC – Remote/Local Change. This will be a 1 whenever there is a transition between local and remote states.

SPAS – This will be a 1 when 9914 has requested service via rsv1 or rsv2 and has been polled in a serial poll.

SRQ – Service Request. This will be a 1 when the SRQ line becomes true.

TADS – The device is addressed to talk.

TPAS – The 9914 is in the talker primary addressed state.

UNC – Unrecognized Command. This will be a 1 if a command that has no meaning is received.

cs – Clear or set the feature

dal – Disable listener function.

dat – Disable talker function.

edpa – Enable dual primary addressing mode.

f4–f0 – Auxiliary command select.

ulpa – This bit shows the LSB of the last address recognized by the 9914.

6.3 DMA

DMA was added to the HPIB interface to reduce the load on the processor and to increase performance. A FIFO was added to optimize the use of the GSC interface to main memory. The 32 byte size of the FIFO matches the line size of the memory as designed in the Hummingbird ASIC. During normal operation, the DMA state machine will transfer complete 32 byte lines to and from memory. The transfers across the GSC bus are done in eight 32 bit accesses. The transfers to/from the 9914 are done in thirty-two 8 bit accesses.

The FIFO is not a true dual-ported design. When a transfer to/from memory is started, the data is transferred uninterrupted until the the FIFO is either completely empty or completely full. During this time, there will no transfers to/from the 9914. Conversely, once a transfer is started to the 9914, there will be no transfers to/from the memory until the 9914 transfer is complete. This simplified the FIFO design considerably without a major impact on the performance.

6.3.1 Inbound DMA

INBOUND DMA is defined as transferring data from the 9914 to main memory. A typical sequence to program a INBOUND transfer is as follows;

- Program the 9914 per the *TMS9914A General Purpose Interface Bus(GPIB) Controller* manual to place the 9914 in LACS(Listener Active State).
- Write the physical address that the data is destined for in main memory to the **DMA Address register.**
- Write the number of bytes to transfer to the **DMA Count register.**
- Optionally write the character matching value into the **DMA Character Match value register.**
- Write the direction of the dma transfer and (optionally) Match enable into the **DMA Control register.** The direction bit should be set to a zero(0).
- Enable DMA by writing a one(1) to the DMA enable bit in the **ISA Control register.** If interrupts are desired, enable interrupts by writing a one(1) into the intr enable bit at the same time. DMA will begin immediately after this write has completed.

INBOUND DMA can terminate on several conditions. Under normal conditions, INBOUND DMA will terminate when the **DMA Count register** reaches terminal count. Terminal count is defined as when the counter value is equal to zero(0). For INBOUND DMA, the **DMA Count register** is based on the number of bytes transferred from the 9914. Another condition that INBOUND DMA will terminate on is when there is a character match and character matching is enabled. The final condition that will terminate INBOUND DMA is when a bus error occurs during the DMA write to main memory.

There are some peculiarities with the FIFO and INBOUND DMA. Generically, the FIFO will empty into main memory whenever it has become full. There is the special case when the FIFO is emptied into main memory when it is not full. This case is referenced as flushing the FIFO. The FIFO is flushed under a number of circumstances. The first and most normal is when terminal count is reached. In this case, the DMA state machine has transferred the correct number of bytes from the 9914 and therefore needs to transfer all the bytes from the FIFO to main memory even if the FIFO is not full.

The second case is when there is a character match and character matching is enabled. There is no guarantee that the FIFO is full and this is a terminating condition where the DMA state machine will not attempt to get more bytes out of the 9914. Therefore, bytes in the FIFO need to be transferred into main memory. The third case is when the 9914 asserts its interrupt line. When the interrupt line is asserted, DMA is not terminated because there may be many reasons why the 9914 can interrupt that might have no affect on the DMA in progress. However, one of the conditions that the 9914 will assert the interrupt line is when a byte is tagged with EOI(End of transfer). In this case, the 9914 will not supply any more bytes of data so the FIFO will never get full and it needs to be flushed.

The final case is when the flush bit in the **DMA Control register** is set to a one(1). This is a special debug case that will case the FIFO to flush no matter what the current state of the DMA state machine.

6.3.2 Outbound DMA

OUTBOUND DMA is defined as transferring data from main memory to the 9914. A typical sequence to program a OUTBOUND transfer is as follows;

- Program the 9914 per the *TMS9914A General Purpose Interface Bus(GPIB) Controller* manual to place the 9914 in TACS(Talker Active State) or CACS(Controller Active State).
- Write the physical address that the data is from in main memory to the **DMA Address register.**
- Write the number of bytes to transfer to the DMA Count register.

- Write the direction of the dma transfer into the **DMA Control register**. The direction bit should be set to a one(1).
- Enable DMA by writing a one(1) to the DMA enable bit in the **ISA Control register**. If interrupts are desired, enable interrupts by writing a one(1) into the intr enable bit at the same time. DMA will begin immediately after this write has completed.

OUTBOUND DMA will terminate on only two conditions. The first is when terminal count is reached in the **DMA Count register**. Terminal count is defined as when the counter value is equal to zero(0). For OUTBOUND DMA, the **DMA Count register** is based on the number of bytes transferred from main memory. The second condition that will terminate OUTBOUND DMA is when a bus error occurs during the DMA read from main memory.

6.3.3 Character Matching

Character Matching is a new feature added to the HPIB interface for instrument control. There are quite a few instruments that do not tag the last byte with the EOI line asserting but instead tag the end of transfer with a unique value for the last byte. An example would be having the last byte be a line-feed character. In the past, to receive the data from an instrument like this, software has had to handle the transfer instead of hardware. DMA could not be enabled since the number of bytes to transfer was not known and quite often it was detrimental to transfer bytes after the instrument has sent the tag character to signal the end of transfer. For these reasons, software could not enable DMA but instead had to hand transfer every byte and compare the value with the tag character. This was inefficient in both the processor utilization as well as the bus utilization.

When character matching is enabled in this HPIB interface, the hardware will transfer bytes from the instrument until either the DMA count reaches zero or the character received from the 9914 matches the character previous written to the **Character Matching Value Register**. The proposed usage of this feature is to load the maximum number of bytes that the instrument will send into the **DMA Count Register**, the tag character the signals the end of transfer in the **Character Matching Value Register**, and then enable DMA with character matching enabled. If for some reason, software did not want the DMA to stop after receiving the tag character, DMA can be resumed where it left off by simply re-enabling DMA

6.3.4 HPIB Performance

These performance numbers are preliminary and require verification. The numbers assuming that GSC clock is running at 30MHz.

Table 12 Inbound Timing

Description	Source of Delay	Delay	Notes
hpibReq asserted to hpibGr asserted	HPIB interface	260 ns	~6.5 cycles
minimum hpibGr low time	9914 interface	700ns	Measured max rate
Time per byte to transfer from 9914 to FIFO		960ns/byte	
Filling the FIFO from 9914	HPIB/9914 interface	30720 ns	32 bytes x 960ns/byte
GSC arbitration time	Wax/Lasi	400 ns	
Transferring data from FIFO to main memory	HPIB interface	1280 ns	8 words x 160ns/word. Assumes 0 wait state memory
Time per cache line to transfer from 9914 to main memory		32100ns/line or 996Kbytes/sec	

Table 13 Outbound Timing

Description	Source of Delay	Delay	Notes
hpibReq asserted to hpibGr asserted	HPIB interface	260 ns	~6.5 cycles
minimum hpibGr low time	9914 interface	1440ns	Measured max rate
Time per byte to transfer from FIFO to the 9914		1700ns/byte	
Emptying the FIFO from 9914	HPIB/9914 interface	54400 ns	32 bytes x 1700ns/byte
GSC arbitration time	Wax/Lasi	400 ns	
Transferring data from main memory to FIFO	HPIB interface	1600 ns	8 words x 200ns/word. Assumes 0 wait state memory
Time per cache line to transfer from 9914 to main memory		56400ns/line or 567Kbytes/sec	

6.4 Software Hints

6.4.1 Differences with the 82335 interface

- The Clear Interrupt "register" was a write-only register that occupied the address range 0x000 to 0x7F7. On the PA-HPIB interface, this register will only occupy the byte address 0x000.
- An ID register was added at word(32bit) address 0. This is a read-only register that returns the value of 0x00000001.
- The HPIB status/control register occupied the address range 0x7F8 to 0x7FF. On the PA-HPIB interface, this register will only occupy the byte address 0x7FF.
- The PA-HPIB added new registers called Extended status/control registers in the definition. There currently are no bits defined in the control register. The status register has the two bits(~active controller and 9914 interrupt) that were creatively snuck into the 9914 register space on the 82335 as well as duplicate bits to those contained in the regular status and the DMA status register. The reason that the duplicate bits were added was to give software a one register to determine the reason for an HPIB interrupt. The two bits(~active controller and 9914 interrupt) were moved to this register to make a cleaner method to get to these bits.
- The PA-HPIB added a simple DMA interface. An attempt was made to look like the ISA/EISA DMA model. I finally opted for a simple interface that had an address register, count register, status/control register. I have slipped it into an address space in the 82335 definition that seemed unoccupied.

6.4.2 Possible software gotchas

For normal operation, you should not need to access the following:

* HPIB DMA FIFO address
* HPIB DMA FIFO data
* HPIB DMA HPIB side FIFO pointer
* HPIB DMA GSC side FIFO pointer
* flush bit in DMA Control register

These were added for debug purposes case there is a bug with the Fifo.

When the DMA Count register is loaded, the FIFO is cleared. This means that anything that is currently loaded into the FIFO is sent to the big bit bucket in the sky. In other words, do not write to the DMA Count register unless you have finished with the previous DMA.

The flush bit only has meaning in the INBOUND case. It does nothing for the OUTBOUND case. Flushing will cause the contents of FIFO to transfer to memory. This is different then clearing which throws away the data.

When a 9914 interrupt causes a flush to occur, the flush bit stays asserted while the 9914 interrupt remains asserted. The side effect of this is the FIFO becomes a one byte deep Fifo. If any new data is placed in the FIFO, the data will immediately be transferred to memory. As soon as the 9914 interrupt is cleared, the flush bit will clear and the FIFO will resume normal operation.

When a 9914 interrupt occurs, the interface remains in DMA mode. If more data is available, the interface will continue transfer data to/from the 9914.

When a bus error occurs on the GSC bus when the PA-HPIB is a bus master, dmaen will be cleared(stopping any current DMA) and the buserr bit will be set in the HPIB DMA status register. A bus error occurs because either the address generated by the PA-HPIB is not a valid memory address or a data parity error has occurred. The buserr bit will remain set until the dma is re-enabled. DMA is re-enabled when the dmaen bit is set in the HPIB control register.

Don't write a zero(0) into the **DMA Count Register** with DMA enabled. This will cause weird results. An interrupt will occur and the flush bit will get stuck on if the DMA direction is set to INBOUND.

6.5 HPIB Signals

Table 14 is a list of the signals which interface the TI9914 chip to the Wax ASIC:

The HPIB interface may be disabled entirely by grounding the hpibCsL output during power up. When this is done, accesses to the hpib address space will not generate a gscReadyL. If the hpibResetL signal is not grounded, the HPIB interface will function as described.

Table 14 HPIB Signals

Signal	Pin	Direction	Description
hpibData[0]	212	I/O	Data bus
hpibData[1]	213	I/O	Data bus
hpibData[2]	214	I/O	Data bus
hpibData[3]	215	I/O	Data bus
hpibData[4]	216	I/O	Data bus
hpibData[5]	217	I/O	Data bus
hpibData[6]	218	I/O	Data bus
hpibData[7]	220	I/O	Data bus
hpibReset	234	O	TI9914 reset
hpibClk5	227	O	5MHz clock
hpibRs[0]	224	O	TI9914 register select
hpibRs[1]	225	O	TI9914 register select
hpibRs[2]	229	O	TI9914 register select
hpibDbin	221	O	TI9914 data bus direction
hpibWe	223	O	TI9914 write signal
hpibCs	208	I/O	TI9914 chip select
hpibAccgr	210	I	TI9914 DMA grant
hpibAccrq	222	O	TI9914 DMA request
hpibInt	233	I	TI9914 interrupt request
hpibScnt	207	I	Wax system controller bit
hpibContr	231	O	TI9914 controller signal

6.6 Sample Schematic

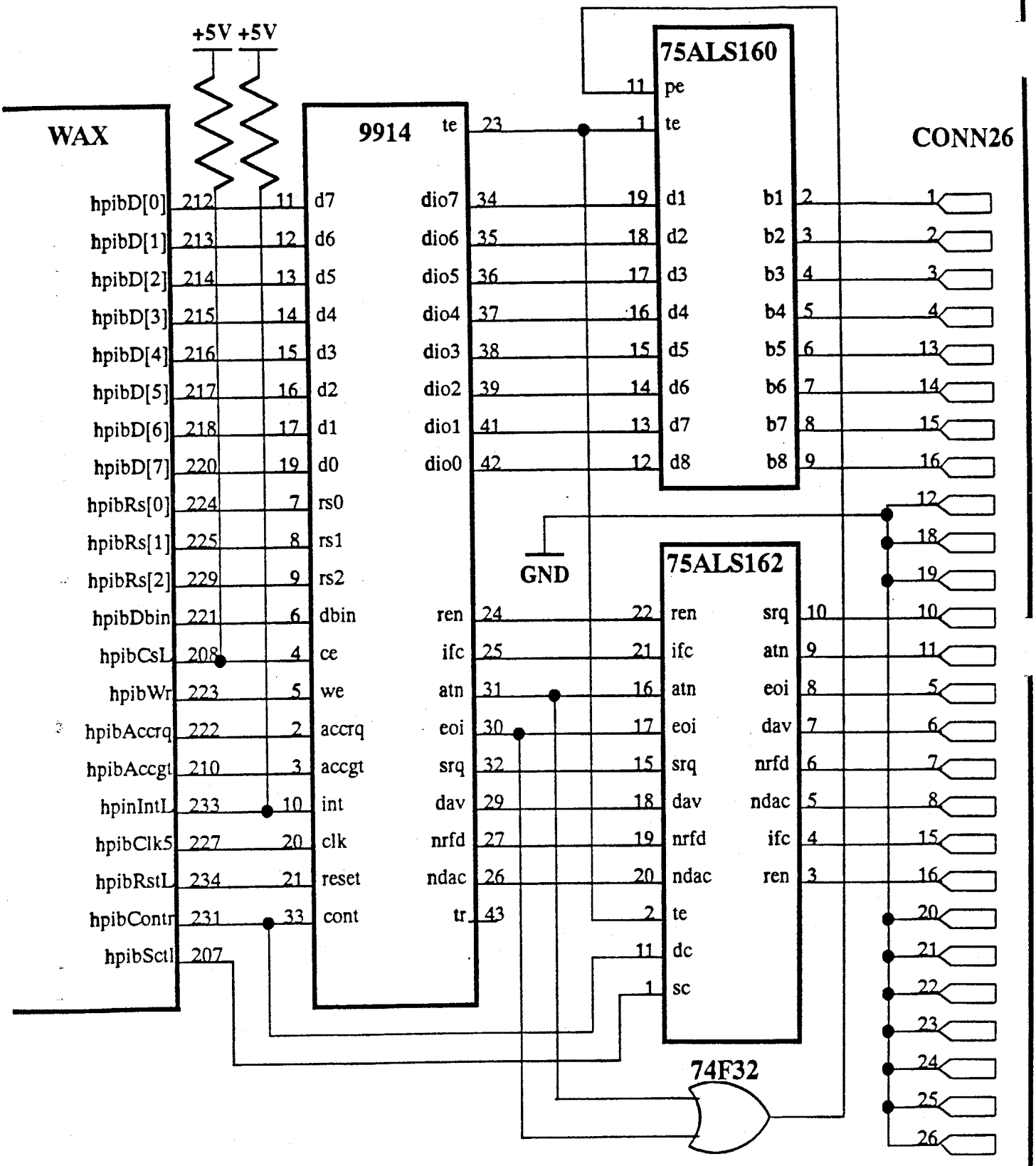


Figure 15 HPIB Schematic

7 Watchdog Timer Interface

The Wax ASIC contains the watchdog timer block. This implementation is compatible with the design on the Stiletto ASIC. The purpose of the watchdog timer is to reset the system if the software gets hung-up. When the watchdog time is enabled, a register in the watchdog timer must be written periodically or the system will be reset. The Watchdog timers drive a reset output pin on the Wax ASIC, this allows the system to be reset by the watchdog timer. The reset output, sysResetL, should be connected to the open-collector reset line from the power supply monitor to the system boards. Since the watchdog timers are disabled when the Wax ASIC is reset, if the watchdog functionality is not required, this block may be ignored.

This register must be written every 256 milliseconds. If the software does not write to the alive register for over 256 milliseconds, Wax will master the GSC bus and a TOC condition will be signalled. The watchdog timer circuitry will request mastership of the GSC and when granted mastership, a TOC interrupt will be written into the address specified in the IAR, then the bus will be released. The software has another 256 milliseconds to write to the alive register before the watchdog timer resets the system.

7.1 Watchdog Timer Registers

Both watchdog registers are defined as a word port only. The byte enable bits are ignored on writes to this register. When data is written to either register, all thirty-two bits on the data bus are written into the register regardless of the state of the byte enable bits associated with the transfer. When either register is read, all thirty-two bits of the data bus are driven. Unused bits are always read as zeros. The watchdog registers may be accessed using multi-word transfers, but only the first word of the transfer occurs. During multi-word writes, the second and subsequent words are ignored. During multi-word reads, the first location accessed is driven as the first word of data, but subsequent words are not meaningful data. Correct GSC protocol is followed for multi-word transfers, the data is just meaningless.

7.1.1 Base Address

The GSC base address for the Watchdog Timer interface is 0xF020 3000. The GSC+ base address for the Watchdog Timer interface is 0xFFE0 3000.

Table 15 Watchdog Timers Registers

Description	Offset	R/W	(MSB) 31	30..9	8	7..1	(LSB) 0
Timer Control	0x000	R/W	X	X	Error	X	Enabled
Timer Alive	0x004	W	X	X	X	X	X
Undefined	0x008-0xFFF						

7.1.2 Detailed Register Definitions

7.1.2.1 Timer Control

The watchdog timer Enabled bit is used to enable and disable the watchdog timer. Writing the least significant bit of the Timer Control register as a one will enable the watchdog timer. Writing the least significant bit as a zero will disable the watch dog timer. This register may be read, and the value of the enable bit will be in the least significant bit position.

If the watchdog timer is bus master when an error occurs, bit eight of the control register will be set. This bit is cleared when any data is written to the control register or when the Wax ASIC is reset.

Errors may be generated by one of three conditions:

1. No register exists at the TOC address (0xFFFBE030)
2. A parity error occurred during the address phase of the transfer
3. A parity error occurred during the data phase of the transfer

7.1.2.2 Timer Alive

If the Watchdog Timer is enabled, the software must periodically write to the Timer Alive register to keep the Watchdog Timer from resetting the system. The data written is irrelevant, the action of performing the write to the register will keep the Watchdog Timer from resetting the system. If the Timer Alive register is read, the data read are meaningless, and the Watchdog Timer is not prevented from resetting the system.

7.1.3 Timing of Events

The Watchdog Timer interface counts GSC clocks to determine when to TOC, when to reset, and how long to assert reset. From the time when the Watchdog Timer is enabled, the Time Alive register must be written every 280 milliseconds to prevent the Watchdog Timer from generating a TOC. If a TOC has been generated, the Timer Alive register must be written within the next 280 milliseconds to prevent the sysResetL signal from asserting.

When a TOC is generated by the Watchdog Timer, a 0x00000005 is written as data to address 0xfffbe030. This information is hard wired into the Wax ASIC.

The interval from enable to TOC, or from TOC to sysResetL assertion is 350 milliseconds when the GSC clock is 30MHz. The interval from enable to TOC, or from TOC to sysResetL assertion is 280 milliseconds when the GSC clock is 37.5MHz. The assertion time of sysResetL is 2.2 milliseconds when the GSC clock is 30MHz. The assertion time of sysResetL is 1.75 milliseconds when the GSC clock is 37.5MHz.

When the system is initially powered up, the watchdog Timer interface may be in such a state that it is asserting it's sysResetL output signal. This situation will resolve in a maximum of 2.2 milliseconds, and then the system will come up normally.

8 Real Time Timer Interface

8.1 Description of the Real Time Timers

The register map for the real time timers is based on the design of the MC6840 timer chip used in previous controllers. This register map is compatible with software which runs on HP9000 workstations and controllers. This design is leveraged from the implementation on the Stiletto ASIC and the design was done by MCSY in Fort Collins.

The Real Time timers consist of two sixteen bit timers and one timer which may be either sixteen bits or thirty-two bits. All three timers have a resolution of one millisecond. The two sixteen bit timers can be programmed to cause an interrupt at intervals from one microsecond up to 65.5 milliseconds. The thirty-two bit timer can be programmed from one microsecond up to 71.5 minutes.

The real time timers have no dedicated signal pins on the Wax ASIC. The RT timers power up disabled and do not affect system operation until they are enabled. There is no way to totally disable the RT timers, they will respond to accesses in their address range.

8.2 General Operation

Typically, the timers are operated by initializing one of the counters and enabling counting and interrupts. The timer will then interrupt when the timer has counted down to 0.

Polling can also be used by reading the CWR repeatedly and checking the INT bit or by reading the counter repeatedly until it reaches 0. The second polling method will not be as accurate but will be within 1 microsecond of the initial time programmed.

8.3 Overview of Software Interface

The timers are initialized by writing the number of microseconds to count down to the appropriate counter. Then the timer must be enabled by setting the GATE bit in the control word register CWR. While writing to the CWR, the interrupt enable bit (IE) can also be set. IE will enable the interrupt signal. If IE is not set, the INT bit in the control word register will set at terminal count (0) but the interrupt signal will not be asserted.

After the timer has interrupted, the interrupt can be cleared by writing a 0 to the INT bit in the control word register.

8.4 Register Definitions

8.4.1 Base Address

The GSC base address for the Real Time Timer interface is 0xf0206000. The GSC+ base address for the Real Time Timer interface is 0xffe06000.

8.4.2 Register Overview

The addressing scheme for the RT timers is as follows:

Table 16 Real TIme Timers Registers

Description	Offset	R/W	(MSB) 31..25	24	23..17	16	15..9	8	7..3	2	1	(LSB) 0
ID Register	0x000	R	0	0	0	0	0	0	0	0	1	0
Control Word 0	0x004	R/W	X	INT	X	GATE	X	IE	X	X	CM1	CM0
Counter 0	0x008	R/W	Current Count									
Holding Reg 0	0x00c	R	Next Count									
Undefined	0x010											
Control Word 1	0x014	R/W	X	INT	X	GATE	X	IE	X	X	CM1	CM0
Counter 1	0x018	R/W	Current Count									
Holding Reg 1	0x01c	R	Next Count									
Undefined	0x020											
Control Word 2	0x024	R/W	X	INT	X	GATE	X	IE	X	TEST	CM1	CM0
Counter 2	0x028	R/W	Current Count									
Holding Reg 2	0x02c	R	Next Count									
Undefined	0x030-0xFFF											

These are addresses for full word transfers. For byte and half-word accesses, the address must be adjusted accordingly. If you wish to read the ID register in byte mode, read address F0206003. If you wish to access counter 0 or counter 1 in half-word mode, use addresses F020600A and F020601A.

8.4.3 ID Register

The ID register is a read-only register that will return the ID number of 0x00000002 when read. This identification number signifies a real time timer module. ID registers are convenient for the Operating System to determine what devices are present in specific address spaces.

8.4.4 Control Word Register

The INT bit is set by the timer when the counter reaches 0. It can be cleared by writing a 0 to bit 24 of the corresponding Control Word Register. It cannot be set by writing the CWR.

GATE enables the counter. When set to 1, the counter will count down. When set to 0, the counter will hold its current value.

The IE bit enables the external interrupt signal. If IE is set and INT gets set by the timer, the timer's interrupt signal will assert. If the corresponding bit in the interrupt block register IMR is set, the processor will receive an interrupt.

TEST is a bit only in CWR2. It is used to divide the 32-bit counter into two 16-bit counters. When set, the upper 16-bits and lower 16-bits count down as two separate numbers. This is used for testing. If the upper 16-bits are not the same as the lower 16-bits and TEST and GATE are set, INT will never get set since the full 32-bit number will never equal 0.

CM0 and CM1 are used to set the Counter mode. At the present time, there are only two modes that the counter can be in: 1 and 0. The extra bit is for future modes. When in mode 0, the counter will interrupt after counting down to zero and await further programming. In mode 1, the counter will count down to 0, interrupt, reload the count value from the holding register and resume counting.

The control bits within the CWRs have been segregated into different bytes. The user can clear an interrupt, change a counter mode, or change the status of the interrupt enable bit without worrying about the current value

of the other bytes. Remember that the addresses given for the CWRs are assuming full word accesses. If you wish to access particular bytes within the CWR, count up from that base address. For example, the byte containing the INT bit in CWR0 could be accessed at F0837007.

8.5 Counter Register

The counters are readable and writeable. They may be read at any time. They may be written at any time except when enabled in mode 1. When a timer is counting in mode 1, counting must be disabled by clearing the gate bit and then writing the new value. The interrupt enable bit should also be cleared in case the timer is at terminal count. The timer may then be enabled by again setting the gate bit.

8.6 Holding Register

Each counter has its own holding register. The holding register maintains the latest value written to its corresponding counter. Whenever a value is written to a counter, that value is put in both the counter and the holding register. For this reason, the holding register is a read-only register. It can be indirectly written by writing a value to the associated counter.

On a byte or half-word write to a counter, only the active data bytes will overwrite the current value in the holding register. Consider timer 1 initially programmed with the value 0xffff. After 10 microseconds, the count will be 0xfff5. Byte writing the value 0x0011 to F083701B will overwrite the least significant byte in the holding register, not the current count. The counter will load 0xff11 and resume counting. The holding registers may be read at any time.

8.7 Software Tips

The method of programming the counters is fairly simple:

- Write the desired delay-1 to the counter. For example, if you wish to set timer 1 up to count off 10 microseconds, write a 0x9 to address 0xF0206018
- Enable the counter and interrupt by writing to the Control Word Register. In our example, we would write a 0x00010100 to address 0xF0206014. This would enable the counter by setting the GATE to 1, enable the interrupt by setting IE to one and put the counter in mode 0.
- After the counter has interrupted, the interrupt can be cleared by again writing 0x00010100 to address 0xF0206014. This will clear the INT bit in the CWR.

Here are a few more useful tips:

- Remember that if the timer is already enabled and counting in mode 1, you must disable the counter before writing a new value to it. After the new value is written to the counter, the timer may be re-enabled.
- Accesses to any register in the RT timers block which is not defined in Table 16 will cause a bus error.
- If the programmer is not using the full 32-bit data bus during reads, mask off all other bits.
- The 16-bit timers are the 16 least significant bits of the data bus when doing a 32-bit access.
- Upon power up or whenever the the Wax ASIC is reset, all RT timer registers will be cleared.

9 Interrupt Control

External interrupts from Wax are always sent to the PCX/L processor via a GSC write transaction to the IO_EIR or an address specified in the IAR. The IO_EIR is a five bit register that is physically located inside the processor. The five bit value written to the IO_EIR indicates which bit of the EIR (CR23) will be set. The Wax chip supports interrupt modes compatible with the needs of both the HP-UX and the HP-RT operating systems.

9.1 Register Definitions

There are four registers in Wax associated with interrupts. Table 17 defines the interrupt control registers. The GSC address for this interface is 0xF020 0000. The GSC+ address for this interface is 0xFFE0 0000.

Table 17 Interrupt Registers

Register	Symbol	Address Offset	R/W	Description
Interrupt Request Register	IRR	000	R	The IRR contains the status of all requesting interrupts. A 1 in an IRR bit indicates that the corresponding interrupt is pending and enabled. When an IRR bit is set it will cause Wax to generate an interrupt transaction.
Interrupt Mask Register	IMR	004	R/W	The IMR is used to mask pending interrupts. A 1 in an IMR bit enables the corresponding pending interrupt to create an interrupt request.
Interrupt Pending Register	IPR	008	R/W	The IPR is used to latch incoming interrupts and indicate them as pending. An active edge on an internal interrupt signal causes the corresponding IPR bit to be set to 1. Writes to this register are intended for diagnostic use only and will cause the entire register to be cleared.
Interrupt Control Register	ICR	00C	R/W	The ICR is used to indicate if the system is running HP-UX or HP-RT. It also indicates if a bus error was detected during a write to the io_eir.
Interrupt Address Register	IAR	010	R/W	The interrupt address register contains two pieces of information: the address to which interrupt transactions are written and the group number written in HP-UX mode.

The interrupt registers appear to be 32-bits and are accessed as such. However, not all of the bits are implemented for each register. The un-implemented bits are not affected by writes and are always read as zeros.

The interrupt registers are defined as a word port only. When data is written to any register, all thirty-two bits on the data bus are written into the register regardless of the state of the byte enable bits associated with the transfer. When any register is read, all thirty-two bits of the data bus are driven. Unused bits are always read as zeros. The interrupt registers may be accessed using multi-word transfers, but only the first word of the transfer occurs. During multi-word writes, the second and subsequent words are ignored. During multi-word reads, the first location accessed is driven as the first word of data, but subsequent words are not meaningful data. Correct GSC protocol is followed for multi-word transfers, the data is just meaningless.

9.2 Interrupt Modes

The interrupt mechanism in Wax must operate in two different modes. The mode is determined by the value of the RT bit set by software in the ICR. Both modes will cause at least one write transaction to the address specified in the IAR when a bit is set in the IRR. Table 18 summarizes the different interrupt modes.

Table 18 Wax Interrupt Modes

icr bit 8	Mode	Interrupt Data
0	HP-UX	Lowest 5 bits of IAR
1	HP-RT	group number of the highest priority bit that is set

If an interrupt source in Wax indicated that it wants to interrupt the processor by setting it's interrupt request signal, the bit in the IPR corresponding to that interrupt will be set to 1. If that interrupt is not masked (IMR bit=1), the same bit of the IRR will be set to 1 and a write to the IO_EIR register will be initiated.

If Wax is in HP-UX mode, the group number programmed into the least significant five bits of the IAR will be written to the IO_EIR. The least significant five bits of the IAR are padded up to the most significant bit with zeros. This group number defaults to 0x00000007 during reset Following an interrupt, the HP-UX operating system will then read the IRR. The IRR will be cleared immediately after the IRR read cycle. The HP-UX operating system will insure that all interrupts indicated in the IRR are serviced.

If Wax is in HP-RT mode, the priority number from the highest priority bit that is set will be written to the address specified in the IAR. For example, if the bit with priority 27 and the bit with priority 15 are both set, 27 will be written to the IO_EIR. See Table 20 for the priority of each bit in the IRR. In HP-RT mode, a write to the IO_EIR will occur for every bit in the IRR that is set. Also, each bit written will be cleared automatically after the write.

In HP-UX modes, the contents of the IRR and unmasked bits (IMR bit=1) of the IPR are cleared on the clock cycle following a read of the IRR. In mode 2, the highest priority bit that is set , will be cleared in the IRR and IPR on the cycle following the GSC write to the IO_EIR.

In either the HP-UX or the HP-RT interrupt modes, the address of the IO-EIR is programmed into the most significant twenty-seven bits of the IAR. This address defaults to 0xFFFBE000 during reset.

9.3 Interrupt Register Bit Assignments

Table 19 shows the implemented bits for the ICR in Wax. Table 20 shows the implemented bits for the IRR, IMR and IPR in Wax.

Table 19 Interrupt Control Register Bit Definition

Bit	Name	Description
0	RT	If set, this bit indicates that the Wax ASIC is operating in HP-RT mode. If reset, this bit indicates that the Wax ASIC is operating in HP-UX mode.
8	Bummed	Indicates that a bus error condition was detected when the Wax ASIC was writing to the IO_EIR. This bit is cleared by any write to this register.

Table 20 IRR, IMR, and IPR Bit Definition

Bit	Interrupt Source	Group[4:0]
0	EISA NMI	30
1	8042 general purpose	12
2	8042 high priority	13
6	RS-232	22
10	EISA	28
13	Woody Audio	8
28	Real Time Timer 1	25
29	Real Time Timer 2	21
30	Real Time Timer 3	18
31	HPIB	16

Table 21 IAR Bit Definition

bit 31 .....bit 5	bit 4 .....bit 0
address of IO-EIR	Group number

The IO-EIR address portion of the IAR is set to correspond to an IO-EIR at address 0xFFFBE000 when the Wax ASIC is reset.. The group number portion of the IAR is set to 0x7 when the Wax ASIC is reset. Either or both of these values may be changed at any time by writing to this register. The current setting of this register may be read at any time.

10 Identification Register and Miscellaneous Control

10.1 ID Register

The Wax identification register is a read only register which allows the Wax ASIC to be identified by software. This register is at address 0xF020 7000 when in GSC mode. This register is at address 0xFFE 7000 when in GSC+ mode. This register returns the value 0x56617800 when read. Writes to this register are ignored.

Table 22 ID Register Bit Definition

bit 31 .....bit 0
0101 0110 0110 0001 0111 1000 0000 0000

11 Test Access Port

11.1 Description

The test access port (TAP) inside Wax controls the boundary scan, the internal flip-flop scan, the clock generation, and some test control signals. The TAP is compatible with JTAG 1149.1 specifications. The TAP in Wax has five pins dedicated to operation of the TAP. The signal trstL initializes the TAP into the test-logic-rest state. The TAP allows normal ASIC operation with trstL grounded, but the intention is that this signal is asserted during the power-up reset of the system.

11.2 TAP Instruction Register

The Wax TAP has an eight bit instruction register which specifies the operation of the TAP. The Wax TAP provides for the use of all public instructions required by the 1149.1 specification. It also includes several private instructions for control of the Wax ASIC.

Table 23 TAP Instructions

Instruction	Binary	Hex	Description
EXTEST	00000000	00	Drive boundary with scan data
SAMPLE	00100000	20	Sample pins into boundary scan
RESERVED	01000000	40	Do not use. Could cause IC to drive unknown values.
HIGHZ	01100000	60	Tristate all pins in boundary
BYPASS	11111111	FF	Select bypass register
FREE_RUN	10000000	80	Allow gscSync into Wax to run
SINGLE_STEP	10000001	81	Force one single pulse of gscSync into Wax
DOUBLE_STEP	10000010	82	Force two pulses of gscSync into Wax
NSTEP	10000011	83	Force n pulses of gscSync into Wax (see detail below)
HALT_HIGH	10000100	10	Halt gscSync into Wax chip in high state
HALT_LOW	10000101	85	Halt gscSync into Wax chip in low state
DR_UPDATE	10000110	86	Update data from scan chain to Q outputs of flip flops
CLR_DR_INH	01000010	42	Clear drive inhibit flip flop
SET_DR_INH	01000011	43	Set drive inhibit flip flop
SELDR0	10100000	A0	Select boundary scan chain
SELDR1	10100001	A1	Select internal scan chain
SELDR2	10100010	A2	Select clock control register
SELDR3	10100011	A3	Select test control register
SELDR4	10100100	A4	Select bypass register

11.3 Boundary Scan Chain

The boundary scan chain is integrated into the pads used in the Wax ASIC. This scan chain is used to place signals on the circuits external to the Wax ASIC, monitor the state of nets connected to the Wax ASIC, drive signals into the core of Wax, and finally, monitor signal generated in the core of Wax. This scan chain is 314 bits long.

11.4 Internal Scan Chain

This is a long data register which allows access to all flip flops in the standard cell portion of the Wax ASIC. The only storage devices not accessible through this scan chain are: The IOMAP RAM, all flip flops within the TAP circuitry, any storage within the boundary (including pad structures), and any latch functions within the ASIC. This scan chain is 2522 bits long.

This scan chain is intended to be used for test purposes. A set of vectors generated from the ATG program can be loaded into this scan chain, the clocks pulsed, and then the state of the scan chain verified. Also, this scan chain is used for testing of the ROM in the HPHIL circuitry.

11.5 Clock Control Register

The clock control register is ten bits long. It is a way to load a counter which is used during the NSTEP command. If the clock control register is loaded with a number, then a NSTEP command is issued, the internal version of the gscSync clock will pulse the number of times corresponding to the number loaded into the clock control register.

11.6 Test Control Register

Below is a table showing the bits in the test control register. The bit labeled 0 is the first bit shifted into the scan input port tdi.

Table 24 Test Control DR

bit	4	3	2	1	0
signal	RTSTMODE	PTEST	IDDQTEST	SCANMODE	MH_CNTL

- RTSTMODE — Enables the receive test mode for the boundary scan chain. Set to 0 by trstL.
- PTEST — Controls the static current draw of the IOMAP RAM. Set to 0 by trstL. When set, the IOMAP RAM should not draw any static current. This bit should be reset for normal operation of the ASIC.
- IDDQTEST — Controls the static current draw of the clock receiver. Set to 0 by trstL. When set, the clock receiver should not draw any static current. This bit should be reset for normal operation of the ASIC.
- SCANMODE — Controls the multiplexing of all flip flop clocks in the ASIC. Set to 0 by trstL. When set, all flip flops receive the output of the gscSync clock receiver. When reset, the flip flops get the appropriate clock for normal system operation. This bit should be set when manipulating the internal scan chain.
- MH_CNTL — Controls the Master Hold bit to the flip flops. This bit is set to 1 by trstL.

11.7 Drive Inhibit Flip Flop

The drive inhibit flip flop will cause all tristateable output pins on the Wax ASIC to go into the high impedance state. This flip flop is cleared by gscResetL or by TAP command; it is set by TAP command only.

This bit is useful after internal ASIC test have completed. Since the ASIC may be in such a state where damage could occur, the drive inhibit flip flop is set until the internal state of the ASIC is consistent with the external environment.

12 Address Map

Table 25 shows the entire address map for the Wax ASIC when it is used in GSC mode.

Table 25 GSC Address Map

Circuitry	Address	Size
Wax Doesn't Respond	0000 0000 - F01F FFFF	4G
Interrupt Control	F020 0000 - F020 0FFF	4K
HP-HIL Interface	F020 1000 - F020 1FFF	4K
RS-232	F020 2000 - F020 2FFF	4K
Watchdog Timer	F020 3000 - F020 3FFF	4K
Wax Doesn't Respond	F020 4000 - F020 4FFF	4K
HPIB	F020 5000 - F020 5FFF	4K
Real Time Timers	F020 6000 - F020 6FFF	4K
Identification Register	F020 7000 - F020 7FFF	4K
Wax doesn't respond	F020 8000 - FBFF FFFF	200M
i486 Bus Control	FC00 0000 - FFCF FFFF	64M
Wax Doesn't Respond	FFD0 0000 - FFFF FFFF	3M

Table 26 shows the entire address map for the Wax ASIC when it is used in GSC+ mode.

Table 26 GSC+ Address Map

Circuitry	Address	Size
Wax Doesn't Respond	0000 0000 - FBFF FFFF	4.2G
i486 Bus Control	FC00 0000 - FFCF FFFF	64M
Wax Doesn't Respond	FFD0 0000 - FFDF FFFF	1M
Interrupt Control	FFE0 0000 - FFE0 0FFF	4K
HP-HIL Interface	FFE0 1000 - FFE0 1FFF	4K
RS-232	FFE0 2000 - FFE0 2FFF	4K
Watchdog Timer	FFE0 3000 - FFE0 3FFF	4K
Wax Doesn't Respond	FFE0 4000 - FFE0 4FFF	4K
HPIB	FFE0 5000 - FFE0 5FFF	4K
Real Time Timers	FFE0 6000 - FFE0 6FFF	4K
Identification Register	FFE0 7000 - FFE0 7FFF	4K
Wax Doesn't Respond	FFE0 8000 - FFFF FFFF	2M

13 Electrical Characteristics

13.1 DC Electrical Characteristics

13.1.1 Absolute Maximum Ratings

Parameter	min	max
V_{DD}	-0.5V	7.0V
V_{DL}	-0.5V	7.0V
DC Input Voltage	-0.5V	7.0V
DC Input Current		+/- 100mA
Power Dissipation		0.75W
Storage Temperature	-40°C	125°C
Ambient Temperature Under Bias	-20°C	85°C

13.1.2 Input Protection

Parameter	min	max
Electrostatic Discharge (between any two pins) through 1500 ohms in series with 100pf		+/- 2.0KV
DC Input Current (for latchup protection)		+/- 100mA

13.1.3 Electrical Characteristics Over Operating Range

Parameter		min	max
V_{DD} -	Supply Voltage	4.5V	5.25V
V_{DL} -	GSC Supply Voltage	3.0V	3.5V
V_{IH} -	High Level Input Voltage	2.0V	
V_{IL} -	Low Level Input Voltage		0.8V
V_{IH} -	High Level Input Voltage (syncH,syncL)	3.83V	
V_{IL} -	Low Level Input Voltage (syncH,syncL)		3.55V
V_{OH} -	High Level Output Voltage (at I_{OH} max)	2.4V	
V_{OL} -	Low Level Output Voltage (at I_{OL} max)		0.4V
I_{IL}/I_{IH} -	Input Leakage Current	-10uA	10uA
I_{OZ} -	Tristate Output Leakage Current	-10uA	10uA
I_{DD} -	Supply Current		150mA
I_{OH} -	High Level Output Current (Group 1 Signals)	-1mA @ 2.4V	
I_{OL} -	Low Level Output Current (Group 1 Signals)	12mA @ 0.4V	
I_{OH} -	High Level Output Current (Group 2 Signals)	-3mA @ 2.4V	
I_{OL} -	Low Level Output Current (Group 2 Signals)	24mA @ 0.4V	
I_{OH} -	High Level Output Current (Group 3 Signals)	-1mA @ 2.4V	
I_{OL} -	Low Level Output Current (Group 3 Signals)	8mA @ 0.4V	
C_{IN} -	Input Capacitance		10pf
T_J -	Operating Junction Temperature	0°C	85°C

Group 1 Signals:

gscAD[31:0], gscType[0:3], gscAdvL, gscReadyL, gscParity, gscErrorL, gscLslL

Group 2 Signals:

eisaD[31:0]

Group 3 Signals:

All other Outputs or I/Os.

13.2 AC Electrical Characteristics

13.2.1 GSC Input Timing

Signal Name	timing relative to falling edge of syncH		spec number
	set up	hold	
gscAD[31:0]	2.5 ns	4.5 ns	1
gscType[0:3]	2.5 ns	4.5 ns	2
gscParity	2.5 ns	4.5 ns	3
gscReadyL	2.5 ns	4.5 ns	4
gscErrorL	2.5 ns	4.5 ns	5
gscResetL	2.5 ns	4.5 ns	6
gscBgL	2.5 ns	4.5 ns	7
kpackL	2.5 ns	4.5 ns	8
kretryL	2.5 ns	4.5 ns	9

13.2.2 GSC Output Timing

Signal Name	syncH falling to signal out delay		load de-rating factor		spec number
	min	max	min	max	
gscAd[31:0]	5 ns (C_L = 20pf)	18 ns (C_L = 100pf)	.02 ns/pf	.07 ns/pf	10
gscType[0:3]	5 ns (C_L = 20pf)	18 ns (C_L = 100pf)	.02 ns/pf	.07 ns/pf	11
gscAddvL	5 ns (C_L = 20pf)	17 ns (C_L = 100pf)	.02 ns/pf	.07 ns/pf	12
gscParity	5 ns (C_L = 20pf)	21 ns (C_L = 100pf)	.02 ns/pf	.07 ns/pf	13
gscReadyL	5 ns (C_L = 20pf)	21 ns (C_L = 100pf)	.02 ns/pf	.07 ns/pf	14
gscErrorL	5 ns (C_L = 20pf)	17 ns (C_L = 100pf)	.02 ns/pf	.07 ns/pf	15
gscBrL	5 ns (C_L = 5pf)	14 ns (C_L = 30pf)	.02 ns/pf	.07 ns/pf	16
gscSplitL	5 ns (C_L = 5pf)	16 ns (C_L = 30pf)	.02 ns/pf	.07 ns/pf	17
kpendL	5 ns (C_L = 5pf)	16 ns (C_L = 30pf)	.02 ns/pf	.07 ns/pf	18
kretryL	5 ns (C_L = 5pf)	16 ns (C_L = 30pf)	.02 ns/pf	.07 ns/pf	19

13.2.3 i486 Interface Input Timing

Signal Name	Parameter	min	max	spec number
i486Hclk	operating frequency	DC	33.33 MHz	20
	duty cycle	40%	60%	21
i486HholdH	signal to i486Hclk rising edge	11 ns setup; 2 ns hold		22
i486Addr[31:2]	signal to i486Hclk rising edge	13 ns setup; 2 ns hold		23
i486BeL[3:0]	signal to i486Hclk rising edge	13 ns setup; 2 ns hold		24
i486HadsL	signal to i486Hclk rising edge	14 ns setup; 2 ns hold		25
i486HwnrH	signal to i486Hclk rising edge	13 ns setup; 2 ns hold		26
i486HmnioH	signal to i486Hclk rising edge	13 ns setup; 2 ns hold		27
i486HdncH	signal to i486Hclk rising edge	13 ns setup; 2 ns hold		28
i486HlockL	signal to i486Hclk rising edge	13 ns setup; 2 ns hold		29
i486HreadyinL	signal to i486Hclk rising edge	11 ns setup; 2 ns hold		30
eisaData[31:0]	signal to i486Hclk rising edge, when EDPU mode is *disabled*	11 ns setup; 2 ns hold		31
i486IrqH	signal to i486Hclk rising edge	[can arrive asynchronously]		32
i486NmiH	signal to i486Hclk rising edge	[can arrive asynchronously]		33

13.2.4 i486 Interface Output Timing

Signal Name	Parameter	min (C_L = 5pf)	max (C_L = 50pf)	spec number
i486ResetL	delay from i486Hclk rising edge	4 ns	14 ns	34
i486HhldaH	delay from i486Hclk rising edge	4 ns	13 ns	35
i486HbusreqL	delay from i486Hclk rising edge	4 ns	13 ns	36
i486Addr[31:2]	delay from i486Hclk rising edge	4 ns	17 ns	37
i486BeL[3:0]	delay from i486Hclk rising edge	4 ns	17 ns	38
i486HadsL	delay from i486Hclk rising edge	4 ns	17 ns	39
i486HwnrH	delay from i486Hclk rising edge	4 ns	17 ns	40
i486HmnioH	delay from i486Hclk rising edge	4 ns	17 ns	41
i486HdncH	delay from i486Hclk rising edge	4 ns	17 ns	42
i486HlockL	delay from i486Hclk rising edge	4 ns	17 ns	43
i486HreadyoutL	delay from i486Hclk rising edge	4 ns	18 ns	44
i486Hlac0L	delay from i486Hclk rising edge	4 ns	14 ns	45
eisaData[31:0]	delay from i486Hclk rising edge, when EDPU mode is *disabled*	7 ns	21 ns	46

13.2.5 i486 Interface EDPU Emulator Timing

Signal Name	Parameter	min (C_L = 5pf)	max (C_L = 120pf)	spec number
eisaData[31:0]	signal to edpuSdileL rising edge	1 ns setup; 4 ns hold		47
eisaData[31:0]	delay from edpuSdileL	5 ns	21 ns	48
eisaData[31:0]	delay from eisaData[31:0]	5 ns	19 ns	49
eisaData[31:0]	delay from edpuSdoeL[1:0]	6 ns	19 ns	50
eisaData[31:0]	delay from edpuSel[2:0]	5 ns	22 ns	51
eisaData[31:0]	delay from edpuBelatL	6 ns	24 ns	52
eisaBeL[3:0]	signal to edpuBelatL rising edge	1 ns setup; 2 ns hold		53

13.2.6 8086 and Multiplexed Mode Input Timing

Signal Name	Parameter	min	max	spec number
x86Hclk	operating frequency	DC	16 MHz	54
	duty cycle	30%	70%	55
x86HholdH	signal to x86Hclk rising edge	11 ns setup; 1 ns hold (to guarantee recognition on this clock edge) [can arrive asynchronously]		56
x86Addr[31:0]	signal to x86Hclk rising edge	14 ns setup; 1 ns hold		57
x86SbheL	signal to x86Hclk rising edge	13 ns setup; 1 ns hold		58
x86MrdcL	signal to x86Hclk rising edge	18 ns setup; 0 ns hold		59
x86MwtcL	signal to x86Hclk rising edge	18 ns setup; 0 ns hold		60
x86HreadyinL	signal to x86Hclk falling edge	12 ns setup; 1 ns hold (to guarantee recognition on this clock edge) [can arrive asynchronously]		61
x86SownL	signal to x86Hclk rising edge, with multiplexed mode *enabled*	16 ns setup; 1 ns hold		62
x86Data[15:0]	signal to x86Hclk rising edge	11 ns setup; 2 ns hold		63
x86Data[15:0]	signal to x86SxalH falling edge, with multiplexed mode *enabled*	2 ns setup; 2 ns hold		64
x86Data[15:0]	signal to x86SaleH falling edge, with multiplexed mode *enabled*	2 ns setup; 2 ns hold		65

13.2.7 8086 and Multiplexed Mode Output Timing

Signal Name	Parameter	min (C_L = 5pf)	max (C_L = 50pf)	spec number
x86ResetL	delay from x86Hclk rising edge	4 ns	14 ns	66
x86HhldaH	delay from x86Hclk rising edge	4 ns	13 ns	67
x86HbusreqL	delay from x86Hclk rising edge	4 ns	13 ns	68
x86Addr[31:0]	delay from x86Hclk rising edge	4 ns	17 ns	69
x86SbheL	delay from x86Hclk rising edge	4 ns	17 ns	70
x86MrdcL	delay from x86Hclk rising edge	4 ns	17 ns	71
x86MwtcL	delay from x86Hclk rising edge	4 ns	17 ns	72
x86IorcL	delay from x86Hclk rising edge	4 ns	17 ns	73
x86IowcL	delay from x86Hclk rising edge	4 ns	17 ns	74
x86HreadyoutL	delay from x86Hclk rising edge	4 ns	18 ns	75
x86Data[15:0]	delay from x86Hclk rising edge	7 ns	22 ns	76
x86Data[15:0]	delay from x86SdbenL, when multiplexed mode is *enabled*	6 ns	16 ns	77
x86Data[15:0]	delay from x86SddirL, when multiplexed mode is *enabled*	6 ns	16 ns	78

13.2.8 Wax Pinout

Pin	Signal	Direction	Notes
1	rs232DcdL	I	Serial data carrier detect
2	rs232DsrL	O	Serial data set ready
3	GND dirty		
4	hilSi	I	Hphil serial data input
5	hilSo	O*	Hphil serial data output
6	kpendL	I	GSC+ pended transaction indicator
7	kpackL	O	GSC+ pended transaction acknowledge
8	kretryL	O	GSC+ transaction retry
9	kdrrL	I	GSC+ DMA read return
10	VDL dirty		
11	gscParity	I/O	Parity for gscAd[31:0]
12	GND dirty		
13	gscAddvL	I/O	gscAd[31:0] has valid address
14	gscReadyL	I/O	GSC(+) transfer acknowledge
15	gscErrorL	I/O	GSC(+) parity error or time-out
16	gscSplitL	O	Connects to gscLsL signal
17	gscHkioscL	I	Select GSC or GSC+ operation/address space
18	gscResetL	I	Power-on reset, GSC clock synchroniziation
19	GND dirty		
20	gscType[0]	I/O	GSC(+) transfer type and byte enable
21	gscType[1]	I/O	GSC(+) transfer type and byte enable
22	gscType[2]	I/O	GSC(+) transfer type and byte enable
23	gscType[3]	I/O	GSC(+) transfer type and byte enable
24	gscBrL	O	Request for GSC(+) bus mastership
25	VDL dirty		
26	sysResetL	O	Watchdog output to reset system on timeout
27	GND dirty		
28	gscBgL	I	GSC(+) bus mastership acknowledge
29	GND clean		
30	VDD clean		
31	GND core		
32	gscSyncL	I	system ECL clock input
33	gscSyncH	I	system ECL clock input
34	VDD core		
35	GND clean		

Pin	Signal	Direction	Notes
36	VDD clean		
37	gscAd[00]	I/O	GSC(+) address/data bus
38	gscAd[01]	I/O	GSC(+) address/data bus
39	GND dirty		
40	gscAd[02]	I/O	GSC(+) address/data bus
41	gscAd[3]	I/O	GSC(+) address/data bus
42	gscAd[4]	I/O	GSC(+) address/data bus
43	gscAd[5]	I/O	GSC(+) address/data bus
44	VDL dirty		
45	gscAd[6]	I/O	GSC(+) address/data bus
46	GND dirty		
47	gscAd[7]	I/O	GSC(+) address/data bus
48	gscAd[8]	I/O	GSC(+) address/data bus
49	gscAd[9]	I/O	GSC(+) address/data bus
50	gscAd[10]	I/O	GSC(+) address/data bus
51	gscAd[11]	I/O	GSC(+) address/data bus
52	gscAd[12]	I/O	GSC(+) address/data bus
53	GND dirty		
54	gscAd[13]	I/O	GSC(+) address/data bus
55	gscAd[14]	I/O	GSC(+) address/data bus
56	gscAd[15]	I/O	GSC(+) address/data bus
57	gscAd[16]	I/O	GSC(+) address/data bus
58	gscAd[17]	I/O	GSC(+) address/data bus
59	VDL dirty		
60	gscAd[18]	I/O	GSC(+) address/data bus
61	GND dirty		
62	gscAd[19]	I/O	GSC(+) address/data bus
63	gscAd[20]	I/O	GSC(+) address/data bus
64	gscAd[21]	I/O	GSC(+) address/data bus
65	gscAd[22]	I/O	GSC(+) address/data bus
66	gscAd[23]	I/O	GSC(+) address/data bus
67	GND dirty		
68	gscAd[24]	I/O	GSC(+) address/data bus
69	VDD clean		
70	gscAd[25]	I/O	GSC(+) address/data bus
71	GND clean		

Pin	Signal	Direction	Notes
72	gscAd[26]	I/O	GSC(+) address/data bus
73	gscAd[27]	I/O	GSC(+) address/data bus
74	gscAd[28]	I/O	GSC(+) address/data bus
75	VDL dirty		
76	gscAd[29]	I/O	GSC(+) address/data bus
77	GND dirty		
78	gscAd[30]	I/O	GSC(+) address/data bus
79	gscAd[31]	I/O	GSC(+) address/data bus
80	trstL	I	JTAG tap reset
81	tdi	I	JTAG tap serial scan data input
82	tclk	I	JTAG tap clock input
83	tms	I	JTAG tap mode select input
84	tdo	O	JTAG tap serial scan data output
85	i486HreadyinL	I	x86HreadyinL if Wax is in 8086 mode
86	i486HreadyoutL	O	x86HreadyoutL if Wax is in 8086 mode
87	GND dirty		
88	i486HholdH	I	x86HholdH if Wax is in 8086 mode
89	i486HhldaH	O	x86HhldaH if Wax is in 8086 mode
90	i486HadsL	I/O	x86SbheL if Wax is in 8086 mode
91	VDD core		
92	clk40M	I	
93	GND core		
94	i486Hlac0L	O	
95	i486HwnrH	I/O	x86MrdcL if Wax is in 8086 mode
96	i486HmnioH	I/O	x86MwtcL if Wax is in 8086 mode
97	edpuSdoeL[0]	I	x86SxalH if Wax is in 8086 multiplexed mode
98	edpuSdoeL[1]	I	x86SaleH if Wax is in 8086 multiplexed mode
99	edpuSdileL	I	x86MuxedL if Wax is in 8086 mode
100	i486HdncH	I/O	x86IorcL if Wax is in 8086 mode
101	i486NmiH	I	x86NmiH if Wax is in 8086 mode
102	i486IrqH	I	x86IrqH if Wax is in 8086 mode
103	VDD dirty		
104	i486Hclk	I	x86Hclk if Wax is in 8086 mode
105	GND dirty		
106	i486ResetL	O*	x86ResetL if Wax is in 8086 mode
107	i486HbusreqL	O*	x86HbusreqL if Wax is in 8086 mode

Pin	Signal	Direction	Notes
108	i486HlockL	I/O	x86IowcL if Wax is in 8086 mode
109	i486BeL[0]	I/O	x86Addr[0] if Wax is in 8086 mode
110	i486BeL[1]	I/O	x86Addr[1] if Wax is in 8086 mode
111	i486BeL[2]	I/O	
112	i486BeL[3]	I/O	
113	eisaData[0]	I/O	x86Data[0] if Wax is in 8086 mode
114	eisaData[1]	I/O	x86Data[1] if Wax is in 8086 mode
115	GND dirty		
116	eisaData[2]	I/O	x86Data[2] if Wax is in 8086 mode
117	VDD dirty		
118	eisaData[3]	I/O	x86Data[3] if Wax is in 8086 mode
119	VDD clean		
120	eisaData[4]	I/O	x86Data[4] if Wax is in 8086 mode
121	GND dirty		
122	eisaData[5]	I/O	x86Data[5] if Wax is in 8086 mode
123	eisaData[6]	I/O	x86Data[6] if Wax is in 8086 mode
124	eisaData[7]	I/O	x86Data[7] if Wax is in 8086 mode
125	eisaData[8]	I/O	x86Data[8] if Wax is in 8086 mode
126	VDD dirty		
127	eisaData[9]	I/O	x86Data[9] if Wax is in 8086 mode
128	GND dirty		
129	eisaData[10]	I/O	x86Data[10] if Wax is in 8086 mode
130	eisaData[11]	I/O	x86Data[11] if Wax is in 8086 mode
131	eisaData[12]	I/O	x86Data[12] if Wax is in 8086 mode
132	eisaData[13]	I/O	x86Data[13] if Wax is in 8086 mode
133	eisaData[14]	I/O	x86Data[14] if Wax is in 8086 mode
134	VDD dirty		
135	eisaData[15]	I/O	x86Data[15] if Wax is in 8086 mode
136	GND dirty		
137	eisaData[16]	I/O	
138	eisaData[17]	I/O	
139	eisaData[18]	I/O	
140	eisaData[19]	I/O	
141	eisaData[20]	I/O	
142	VDD dirty		
143	eisaData[21]	I/O	

Pin	Signal	Direction	Notes
144	GND dirty		
145	eisaData[22]	I/O	
146	eisaData[23]	I/O	
147	eisaData[24]	I/O	
148	eisaData[25]	I/O	
149	VDD core		
150	eisaData[26]	I/O	
151	GND core		
152	eisaData[27]	I/O	
153	GND dirty		
154	eisaData[28]	I/O	
155	VDD dirty		
156	eisaData[29]	I/O	
157	eisaData[30]	I/O	
158	eisaData[31]	I/O	
159	i486Addr[2]	I/O	x86Addr[2] if Wax is in 8086 mode
160	i486Addr[3]	I/O	x86Addr[3] if Wax is in 8086 mode
161	i486Addr[4]	I/O	x86Addr[4] if Wax is in 8086 mode
162	i486Addr[5]	I/O	x86Addr[5] if Wax is in 8086 mode
163	GND dirty		
164	i496Addr[6]	I/O	x86Addr[6] if Wax is in 8086 mode
165	i486Addr[7]	I/O	x86Addr[7] if Wax is in 8086 mode
166	i486Addr[8]	I/O	x86Addr[8] if Wax is in 8086 mode
167	i486Addr[9]	I/O	x86Addr[9] if Wax is in 8086 mode
168	i486Addr[10]	I/O	x86Addr[10] if Wax is in 8086 mode
169	i486Addr[11]	I/O	x86Addr[11] if Wax is in 8086 mode
170	i486Addr[12]	I/O	x86Addr[12] if Wax is in 8086 mode
171	i486Addr[13]	I/O	x86Addr[13] if Wax is in 8086 mode
172	i486Addr[14]	I/O	x86Addr[14] if Wax is in 8086 mode
173	VDD dirty		
174	i486Addr[15]	I/O	x86Addr[15] if Wax is in 8086 mode
175	GND dirty		
176	i486Addr[16]	I/O	x86Addr[16] if Wax is in 8086 mode
177	i486Addr[17]	I/O	x86Addr[17] if Wax is in 8086 mode
178	VDD clean		
179	i486Addr[18]	I/O	x86Addr[18] if Wax is in 8086 mode

Pin	Signal	Direction	Notes
180	GND clean		
181	i486Addr[19]	I/O	x86Addr[19] if Wax is in 8086 mode
182	i486Addr[20]	I/O	x86Addr[20] if Wax is in 8086 mode
183	i486Addr[21]	I/O	x86Addr[21] if Wax is in 8086 mode
184	i486Addr[22]	I/O	x86Addr[22] if Wax is in 8086 mode
185	i486Addr[23]	I/O	x86Addr[23] if Wax is in 8086 mode
186	i486Addr[24]	I/O	x86Addr[24] if Wax is in 8086 mode
187	VDD dirty		
188	i486Addr[25]	I/O	x86Addr[25] if Wax is in 8086 mode
189	GND dirty		
190	i486Addr[26]	I/O	x86Addr[26] if Wax is in 8086 mode
191	i486Addr[27]	I/O	x86Addr[27] if Wax is in 8086 mode
192	i486Addr[28]	I/O	x86Addr[28] if Wax is in 8086 mode
193	i486Addr[29]	I/O	x86Addr[29] if Wax is in 8086 mode
194	i486Addr[30]	I/O	x86Addr[30] if Wax is in 8086 mode
195	i486Addr[31]	I/O	x86Addr[31] if Wax is in 8086 mode
196	eisaBeL[0]	I	
197	eisaBeL[1]	I	
198	eisaBeL[2]	I	
199	eisaBeL[3]	I	
200	GND dirty		
201	edpuBelatL	I	
202	edpuSel[0]	I	x86SdbenL if Wax is in 8086 multiplexed mode
203	edpuSel[1]	I	x86SddirH if Wax is in 8086 multiplexed mode
204	VDD core		
205	edpuSel[2]	I	x86SownL if Wax is in 8086 multiplexed mode
206	GND core		
207	hpibSctl	O	Hpib system controller
208	hpibCsL	O*	Hpib 9914 chip select/hpib enable input
209	VDD dirty		
210	hpibAccgt	O	Hpib 9914 bus mastership acknowledge
211	GND dirty		
212	hpibD[0]	I/O	Hpib 9914 data bus
213	hpibD[1]	I/O	Hpib 9914 data bus
214	hpibD[2]	I/O	Hpib 9914 data bus
215	hpibD[3]	I/O	Hpib 9914 data bus

Pin	Signal	Direction	Notes
216	hpibD[4]	I/O	Hpib 9914 data bus
217	hpibD[5]	I/O	Hpib 9914 data bus
218	hpibD[6]	I/O	Hpib 9914 data bus
219	GND dirty		
220	hpibD[7]	I/O	Hpib 9914 data bus
221	hpibDbin	O	Hpib 9914 data bus direction
222	hpibAccrq	I	Hpib 9914 bus mastership request
223	hpibWr	O	Hpib write to 9914
224	hpibRs[0]	O	Hpib 9914 register select
225	hpibRs[1]	O	Hpib 9914 register select
226	VDD clean		
227	hpibClk5M	O	Hpib 9914 5 MHz clock output
228	GND clean		
229	hpibRs[2]	O	Hpib 9914 register select
230	VDD dirty		
231	hpibContr	O	Hpib system controller
232	GND dirty		
233	hpibIntL	I	Hpib 9914 interrupt input
234	hpibRstL	O	Hpib 9914 reset output
235	rs232RxdL	I	Serial receive data input
236	rs232TxdL	O*	Serial transmit data output
237	rs232RiL	I	Serial ring indication
238	rs232DtrL	I	Serial data terminal ready
239	rs232CtsL	I	Serial clear to send
240	rs232RtsL	O	Serial request to send

* Wax also uses this output as a mode select input during power-up reset.