Firewire for VXIbus, The HP E8491

Lee Atchison
Nate Berg
Greg Hill
Chuck Platz
Andy Purcell
Measurement Systems Division

I. HISTORY OF T&M CONNECTIVITY

IEEE 488, known also as HP-IB or GPIB, started out as one of the first ways to connect a computer to external instrumentation. Developed by Hewlett-Packard, it was designed to provide easy and standard connection between computers, controllers, and instrumentation.

IEEE 488 quickly became used as a computer interface standard as well. It was used to connect to early printers, and hard disks. The CS80 protocol was a hard disk protocol based on IEEE 488. It was in regular usage as late as the early 1990's, in HP-UX and RMB workstations for hard disk connectivity.

However, for the computer industry, the shortcomings of IEEE 488 quickly became apparent. The advent and regular usage of other computer industry standards, such as SCSI, quickly replaced IEEE 488 as a computer peripheral connection method. The computer industry moved on, but the Test & Measurement (T&M) industry still saw value in IEEE 488. IEEE 488 continued to grow in popularity for T&M applications, and is still in regular usage today. Improvements to the IEEE 488 standard, specifically geared for T&M applications, helped continue its life. Industry-wide standards such as IEEE 488.2 and SCPI made it substantially easier to use and more useful as well.

With the growing popularity of Windows-based PCs, the computer industry discovered the need for better, faster, and easier to use I/O standards. At the same time, a similar need arose in the larger, general-purpose consumer electronics market. Both industries felt the emerging demand for the connection of consumer products to computers. Driven by the combined potential of these lucrative markets, general-purpose computer and consumer electronics industry representatives began developing some common I/O standards.

By far, the most support is behind two emerging standards – USB and IEEE 1394. These two standards are positioned to take over most, if not all, of the I/O interconnect market, not just for personal computers, but professional computers, and consumer electronics. Soon everything from televisions, VCRs, stereos, home computers, and business computers will utilize these two standards. USB, positioned for the low-end cost/performance trade-off, will be used for low-cost peripherals such as mouse and keyboards. IEEE 1394, higher on the cost/performance spectrum, will be used for multi-media applications, high-speed computer peripherals such as scanners and printers, and consumer audio-video electronics such as VCRs, TVs, DVDs, and DSS satellite systems.

While all of this has gone on in the computer/electronics industry, the test and measurement industry has been much slower to adopt new standards. The old workhorse, IEEE 488, is still the most used instrumentation connection interface.

In 1987, VXI was introduced as a T&M standard for modular instrumentation, intended to provide higher performance, smaller size, and lower cost solutions than traditional T&M rack & stack instrumentation. However, it quickly became apparent that VXI, while significantly higher performance than IEEE 488 (40 times the theoretical performance of IEEE 488), had a higher price tag. The investment in a single VXI test system's infrastructure (without spending a dime on a piece of measurement equipment), was between $7k and $15k, a large barrier to entry for most test system developers. While VXI had clear advantages, it was also clear that for it to make significant inroads into the rack & stack market, the price of entry would have to be lowered dramatically.

HP took on that challenge. The two products that make up the bulk of the cost of entry into VXI are the mainframe and the computer I/O. By 1996, HP had made magnificent strides in reducing the cost of the VXI mainframe, but the computer I/O connection was still expensive. The time was right for a price/performance breakthrough.

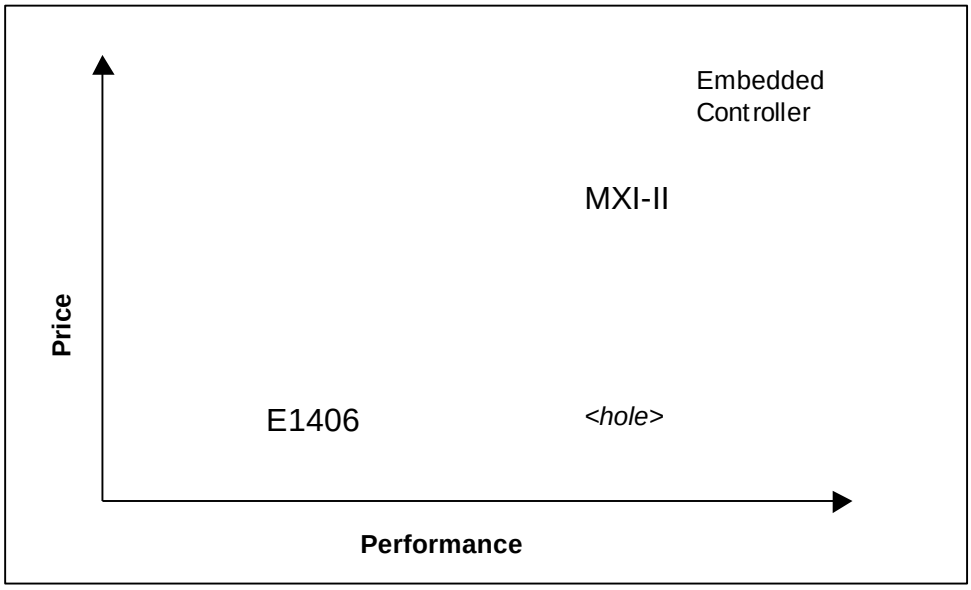
II. MAKING THE CONNECTION

Prior to the introduction of the HP E8491, VXIbus customers have had to choose between low cost and high performance computer I/O. The low cost interfaces, such as the HP E1406B command module, provide very low performance. The higher performance VXI connectivity options, such as MXI-II and embedded controllers, are very expensive.

The specific price and performance comparisons for these products are given in the following table.

Product	Price (including cable and host adapter)	Typical Transfer Rate
HP E1406B	$2,300	<100 KB/sec
MXI-II	$4,500	~10 MB/sec

The following diagram shows graphically the cost/performance tradeoffs of various VXI options:



Our primary objective for the HP E8491 was to fill the "hole". That is, it should have the price of the HP E1406B, but the performance of MXI-II.

In addition, we wanted to address several other VXIbus connectivity problems, most notably:

- Ease of setup – VXI is notoriously hard to setup and configure, with the computer connection being a major culprit.
- Hot connect – Adding, removing, or power cycling a VXI mainframe often results in a need to reboot the computer, or power cycle or reconfigure the entire VXI system.
- Difficult cabling – VXI computer connection cables, such as the MXI-II cables, besides being costly, are very thick and difficult to use.

How could this be accomplished? We needed a mindset leap and something radically different than what had been done before. The leap came from a realization that we weren't going to find the solution using

traditional T&M technologies - we had to use a mainstream computer/electronics technology. This was important for several reasons:

1. It gave us many new and different options to consider.
2. It allowed us to leverage the expertise of the computer industry, drawn from much greater resources than we have in the T&M industry.
3. It allowed us to follow on the coattails of others and use the improvements and changes they made to benefit our solution.

Once we started considering mainstream computer industry technology, the question quickly became, "Which one?" There were several technologies available:

- RS-232
- SCSI
- Enhanced Parallel
- LAN
- FDDI
- ATM
- FiberChannel
- IEEE 1394
- Universal Serial Bus (USB)

Each of these technologies has advantages and disadvantages. Some were not suitable, simply because they did not meet the performance needs (RS-232, Parallel, USB). Some were not attractive, due to their setup/configuration difficulty (SCSI, LAN). From the rest, the question became, which technology would prevail in the mainstream computer industry? Would FiberChannel win out? Would IEEE 1394 win? Or would something else not-thought-of-yet win out?

Three years ago when this decision needed to be made, it was not a clear choice. However, we quickly concluded that IEEE 1394 would eventually win. While the industry has not made its final decision yet, it seems clear that we made the correct choice.

Why IEEE 1394? It has several advantages that make it the most attractive choice:

1. It has the basic performance required of a medium-high speed VXIbus connect.
2. It has a path of continuous performance improvements that has the support of much of the computer industry.
3. Eventually, most new PC should have built-in IEEE 1394 interfaces, just as today's PCs have RS-232 and parallel ports. While this has not yet happened, the first computers having built-in IEEE 1394 are starting to appear. It seems that eventually, almost all computers will have this capability.
4. It has an inexpensive and easy to use cable and cabling system.
5. It is designed to be "plug & play", allowing easy and convenient setup and configuration.
6. It is designed to allow "hot plugging" and power-cycling of devices, without negatively impacting the entire system.
7. It is relatively inexpensive, and becoming more so with the passing of time.

Bottom line, IEEE 1394 has everything we needed in our solution. However, we had several hurdles to get over:

1. IEEE 1394 was a very new technology that few people had experience with.
2. Industry's ultimate adoption of IEEE 1394 was expected, but not known for certain. We faced the risk of adopting another BETA (of VCR fame), the best technology, but not the winner in the marketplace.
3. There were many new and unknown technical details that would have to be worked out for the design to work smoothly within VXI.
4. IEEE 1394 was a moving target – the standard was changing and improving, as we were moving forward with our project.

These hurdles became major risks for our project, risks that we decided to take on and manage in order to move forward.

III. IEEE 1394 TECHNICAL DETAILS

IEEE 1394 is a high performance serial bus - data can be transferred at 400 Mb/s. Addressing is based on node ID's that are automatically assigned after each bus reset. There are no address switches or bus terminators. There are provisions for asynchronous and isochronous packets.

Isochronous packets are used to deliver data at a guaranteed, constant rate. An IEEE 1394 system may utilize up to 64 active isochronous channels. Each channel may transfer one isochronous packet every 125 µs. Isochronous packets are not acknowledged by the receiver. The isochronous bandwidth is managed such that the total traffic on all active channels takes no more than 100 µs per 125 µs period. Isochronous packets are used in multimedia applications to carry audio and video data. The isochronous packet format is shown in Figure 1.

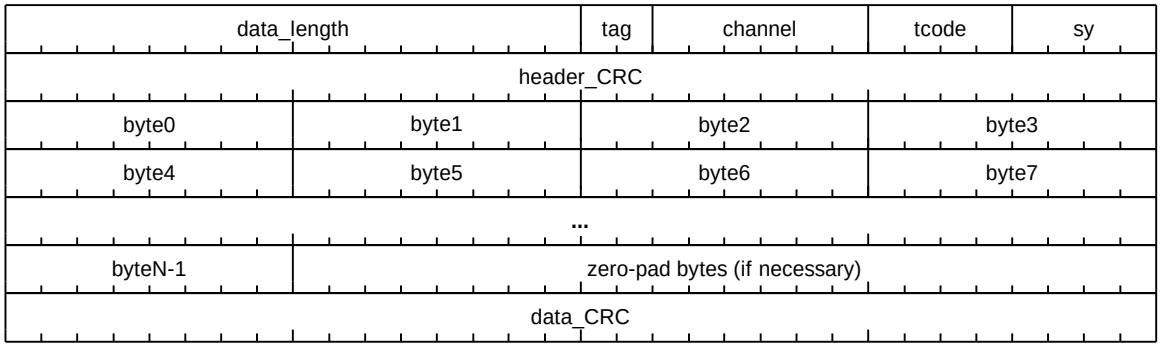


Figure 1 Isochronous Packet Format

Asynchronous packets are used for communications that are not needed on a fixed, regular basis. Asynchronous packets are different than isochronous packets, in that each asynchronous is acknowledged by the receiving node. This lets the sender know whether or not the packet was received and processed successfully. Asynchronous packets thus offer a guaranteed delivery mechanism. An example of an asynchronous write packet is shown in Figure 2.

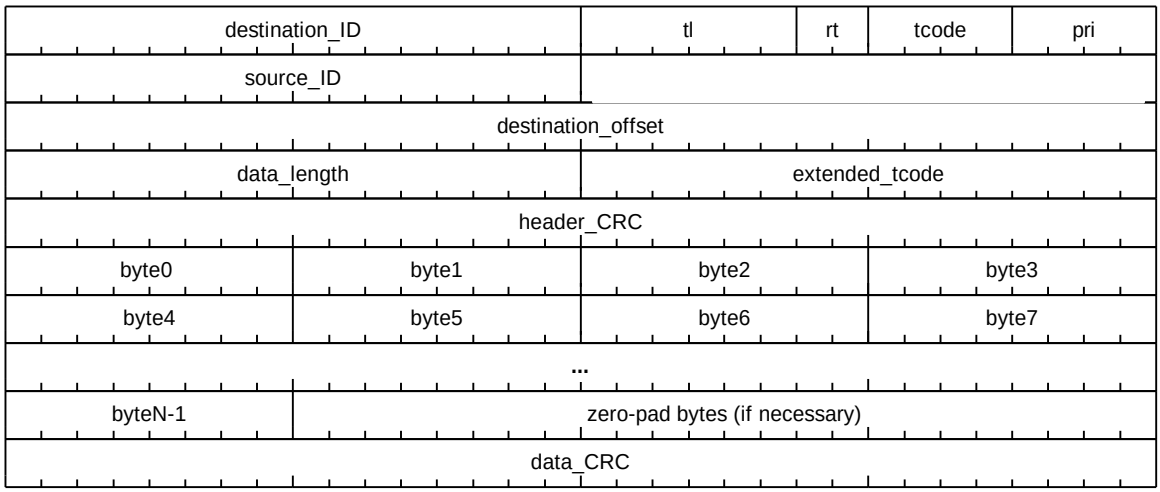


Figure 2 Asynchronous Packet Format - Block Write Request

The transaction code (tcode) indicates whether the packet is a read, write, or lock request or response. The destination offset is used to determine what data is to be accessed. Every IEEE 1394 node has a huge, 48 bit address space. One part of this space is a configuration ROM that is used by software to determine protocols the node understands. Another part contains mandatory Control and Status Registers (CSR's) used by a 1394 bus manager. The lower part of this 48-bit space may be mapped to physical memory on Open Host Controller Interface (OHCI) link implementations, providing for very efficient processing of data.

The high performance features of 1394 are attributable to the cabling and the hardware implementations for the physical (phy) layer and link layer. See

Figure 3.

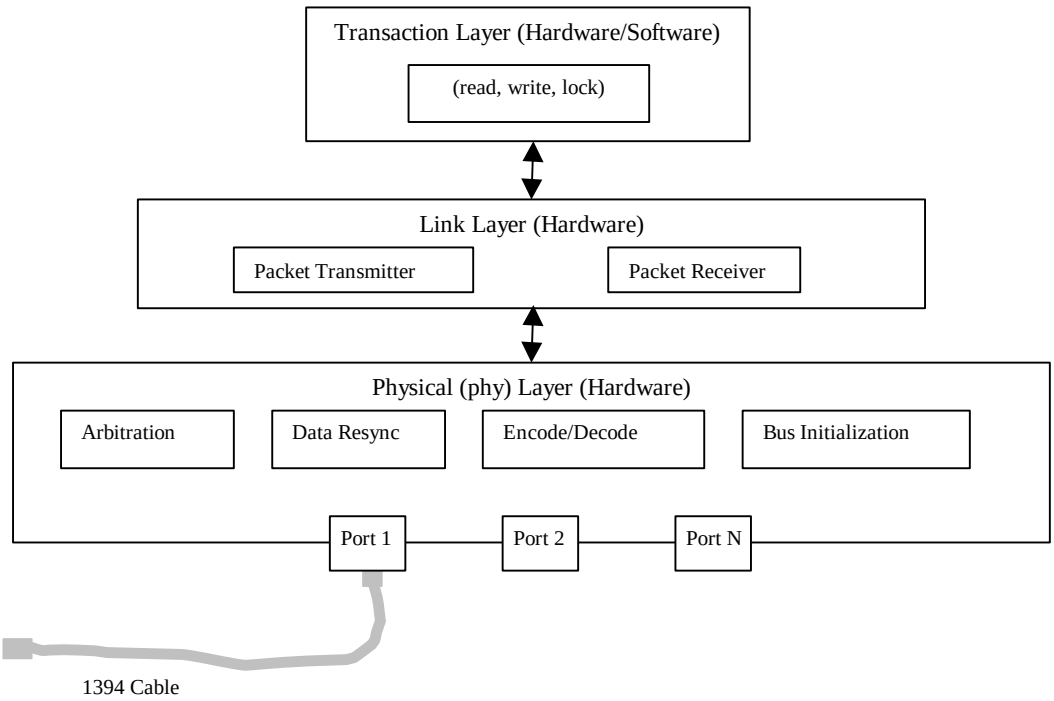


Figure 3 1394 Layering

IEEE 1394 busses can have up to 63 devices connected in a tiered star topology. For best performance, the number of "hops", or cable segments, between any 2 devices should be minimized. Loops are not permitted, and the maximum number of cable hops is 16.

Figure 4 shows two possible 1394 topologies using nodes equipped with 3 ports.

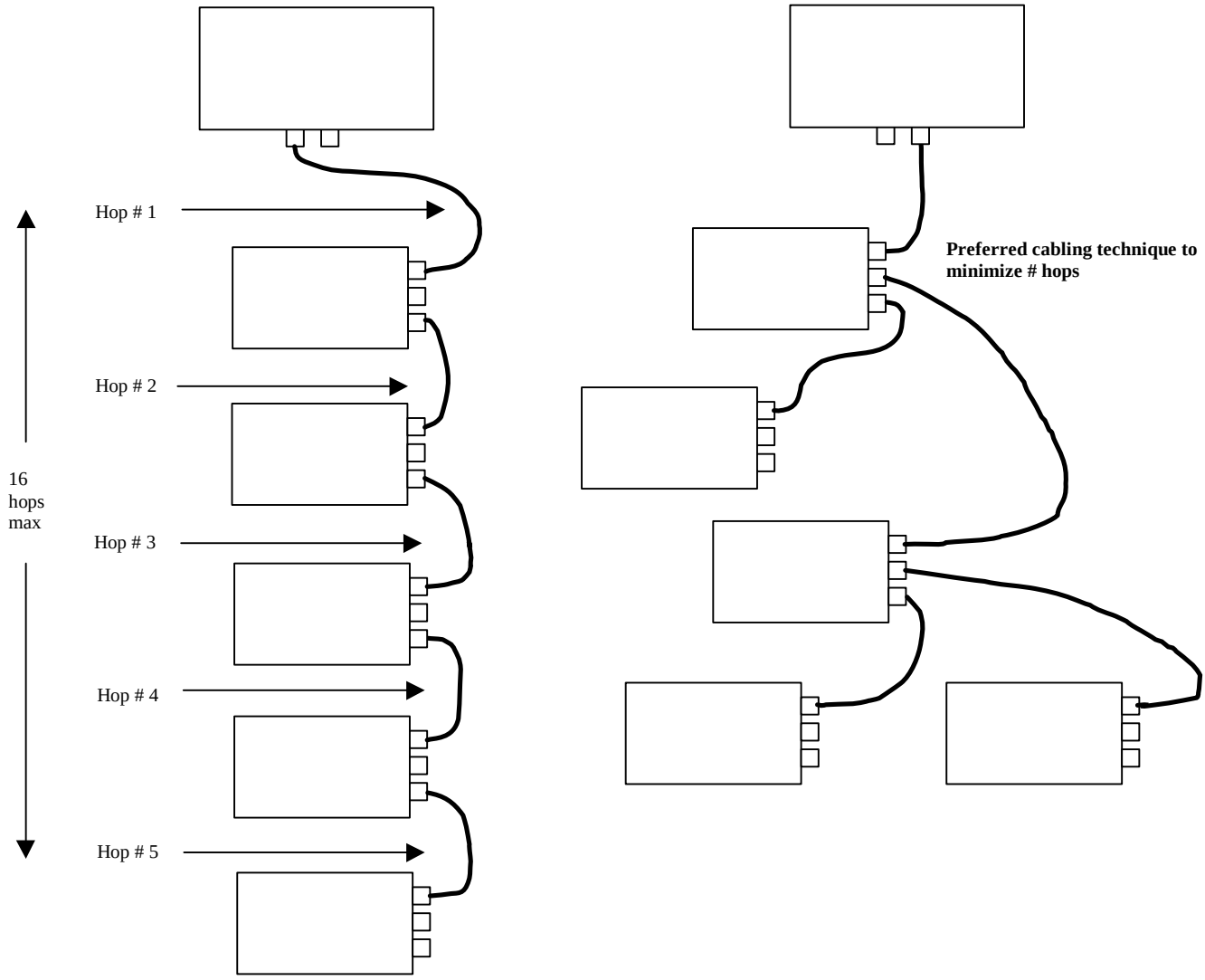


Figure 4: Example IEEE 1394 topologies for a two-port computer and six three-port peripherals.

1394 Cables

The cable consists of 6 wires – two differential pair signals TPA, TPA' and TPB, TPB', and a pair for power and ground. See Figure 5.

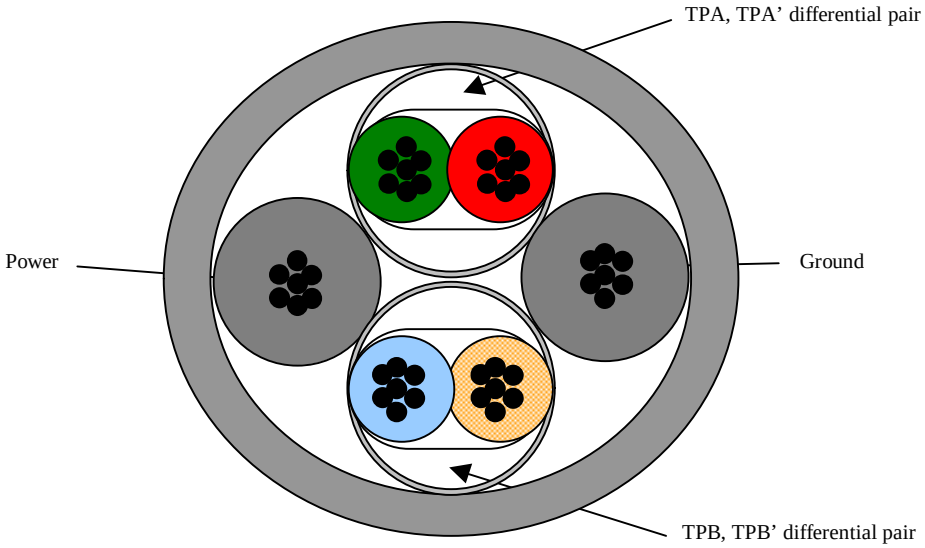


Figure 5 1394 Cable Cross Section

TPA, TPA' are used to transmit strobe, and TPB, TPB' are used to transmit data. Data is transmitted using NRZ (non-return to zero) coding. Strobe changes state whenever 2 consecutive data bits are the same. This allows reconstruction of a clock at the receiving node.

Power and ground allow IEEE 1394 nodes to be powered from the cable. Many portable IEEE 1394 cameras are made to run off of the cable power. The voltage range is from 8 Vdc to 40 Vdc, and the maximum current per port is 1.5 amps. It is possible to use the cable power to energize the IEEE 1394 physical layer when the node's main power source is turned off. This allows a powered-down device to act as a bus repeater.

The maximum cable length is 4.5 meters.

Note that the IEEE 1394 specification also allows for backplane environments, in which the signals are not routed via a cable but rather via a backplane printed circuit card.

Physical (phy) Layer

The cable signals are routed to a 1394 physical layer "phy" chip. The phy chip is responsible for:

- bus reset activities
- receiving and transmitting data
- resynchronizing the bit stream to a local clock and transmitting the data out other phy ports
- arbitrating for the bus
- communications to/from the link chip

Bus Reset Activities

The bus reset activities are described in IEEE 1394 4.4.2.1. The phy participates in tree identification, sending of self ID packets, and reception of phy configuration packets. The physical layer is where all of the magic happens in determining the node ID.

Arbitration

The phy device participates in bus arbitration when it needs to send a data packet. For asynchronous packets, the bus must be idle for a "subaction gap" time before arbitration can start. See Figure 6.

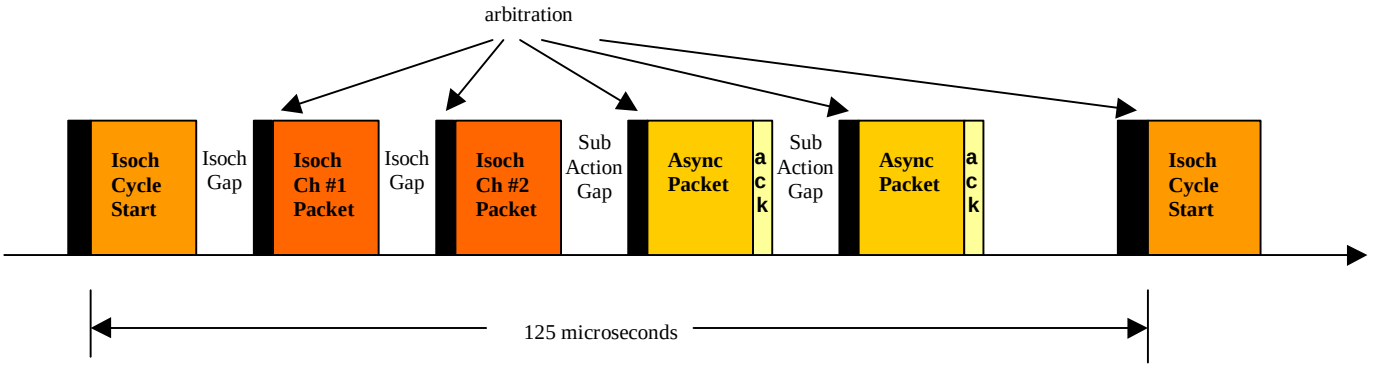


Figure 6 Isochronous and Asynchronous Arbitration

Note that isochronous channel packets must finish before the asynchronous packets can be sent. This is because the subaction gap required before asynchronous packets can be sent is slightly larger than the isochronous gap. If an asynchronous packet is being transmitted when a cycle start is scheduled, the cycle start is held off until after the asynchronous packet is completely transmitted.

Even if there are 2 isochronous channels, each generating 16 MB/sec, there is enough bandwidth left to allow roughly 10 MB/sec of asynchronous traffic.

Link Chip Communication

The communication to the link chip occurs via control lines and a variable number of data lines. The number of data lines in use depends on the speed being used. All of these signals should be electrically isolated, as described in A.4 of IEEE 1394.

The phy has registers that can be read to obtain the node ID, the gap count, and the number of ports. There are phy registers that can be written to initiate a bus reset and to try to become root. These registers are only accessible by programming the link chip. There is no microprocessor interface.

Link Layer

The link layer is responsible for interfacing to the physical layer below and the transaction layer above. The link receives packets if the destination node ID matches the node ID. The link hardware automatically sends an “ack pending” after an asynchronous packet is received. While the packet is being received, the link hardware calculates CRC’s (cyclic redundancy check, to detect data transmission errors) for both the packet header and for the data payload.

The link does have a microprocessor interface and a plethora of read/write registers. These registers are used for

- controlling interrupts
- providing information to the interrupt service routine
- setting up automatic acknowledgments
- setting up packet speeds
- setting up packet headers and then sending packets

Most link chips make use of on-chip FIFO (first in, first out) data buffers between the IEEE 1394 bus and external memory. Some link chips have a packet data port that sends the data payload directly to external memory.

IV. THE HARDWARE

A block diagram of the HP E8491 hardware is shown in Figure H. Six major functions are shown:

- The IEEE 1394 Interface provides the physical and link layer functions necessary for communicating over the IEEE 1394 cable.
- The Data Buffer provides temporary storage of data moving between the IEEE 1394 and VXIbus interfaces.
- The VXI Interface includes all the hardware necessary for communicating over the VXIbus backplane.
- The Processor section includes a microprocessor, memory, and miscellaneous support logic necessary to control the operation of the rest of the device.
- The Clock Routing circuit drives the backplane 10 MHz CLK10 signal, and the Clk Out signal on the front panel, from either the external Clk In signal or an internal oscillator.
- The Trigger Management section allows the processor to drive and sense trigger events. It also provides for the routing of trigger signals between the front panel and the backplane trigger lines.

The main goal of the hardware of the hardware design is to move data very efficiently between the IEEE 1394 serial bus and the VXIbus backplane. This data can be simple register operations, block data moves, or messages. Simple register operations are single-word reads or writes of registers or memory locations. Block data moves are multiple reads or writes of contiguous memory locations or FIFO registers. Messages are character strings transferred to and from VXIbus message based devices via the VXIbus byte transfer protocol. The hardware handles each of these operations differently.

IEEE 1394 Interface

The HP E8491 has a full-featured, 400 Mb/s IEEE 1394 interface. It includes 3 connectors, allowing tree-like system topologies. Thy Phy (physical layer) interface is DC isolated from the system ground, as required by the IEEE 1394 specification. When the HP E8491 is powered, it supplies power to the Phy interface. When the E8491 is powered down, the Phy interface draws power from the IEEE 1394 bus, maintaining the capability to pass data between its 3 ports.

The Link device handles IEEE 1394 packet headers separately from the packet data. When an IEEE 1394 data packet is sent to the HP E8491, the header is stored in the Link device's internal registers. The data is sent through a 32-bit packet data port to the HP E8491's Data Buffer circuit. Similarly, when the E8491 sends a data packet, the header is sent from the Link device's internal registers. The data comes into the Link device's packet data port from the Data Buffer circuit.

Data Buffer

In order to accommodate the speed mismatch between the synchronous IEEE 1394 bus and asynchronous VXIbus devices, the E8491 has an intermediate data buffer. This buffer has 3 FIFO data ports, one dedicated to the Link device, one for the VXIbus interface, and one for the E8491's processor. The buffer can store 64 IEEE 1394 data packets (up to 2 kB each), and can support simultaneous, full speed transfers at all 3 ports. Thus IEEE 1394 transactions can be overlapped with VXIbus transactions, allowing maximum utilization of both buses.

VXIbus Interface

The final component of the high-speed data path, the VXIbus interface moves data between the E8491's Data Buffer and other VXIbus devices. It supports 8, 16, 32, and 64 bit data transfers on the VXIbus. It includes 128 kb of shared RAM, and has all the features required of a Slot 0, VXIbus commander. Its state machines can perform single read/writes, multiple read/writes, BLT (VMEbus Block Transfer mode) read/writes, and IRQ Status/ID reads. During multiple or BLT read/writes, the VXIbus addresses may either increment (for normal memory) or remain constant (for FIFO registers).

An important component of the VXIbus interface is the Data Mux. Its job is to multiplex data from the fixed, 32 bit wide Data Buffer bus to the variable 8-64 bit wide VXIbus. In addition, it performs the classic byte swapping between big-endian and little-endian data formats. Figure H.2 illustrates the organization of data bytes in the 32-bit data buffer and on the VXIbus backplane for various transfer sizes and byte swapping modes. To get every byte into its right place, the Data Mux has a byte oriented, full crosspoint, bi-directional multiplexer. In other words, it can move a byte of data from any byte location on either side to any byte location on its other side. In addition, it has time multiplexing capabilities. It can match a single Data Buffer quadlet to multiple VXIbus byte or doublet data transfer cycles, or a pair of Data Buffer quadlets to a VXIbus octet cycle.

The Processor

The processor circuit is responsible for coordinating the various E8491 functions. It is based on a TMS320F206 DSP, having both internal (for fastest access and execution) and external ROM and RAM. The processor manages the IEEE 1394 interface, both interpreting incoming packets, and building outgoing packets. It also programs the VXIbus interface to execute various backplane operations. It executes the word serial protocol for communication with VXIbus message based devices. It is responsible for programming the FPGAs (field programmable gate arrays) in the data buffer, VXIbus interface, and trigger management circuits. All program and FPGA code is stored in flash ROM so that it can be updated over the IEEE 1394 bus.

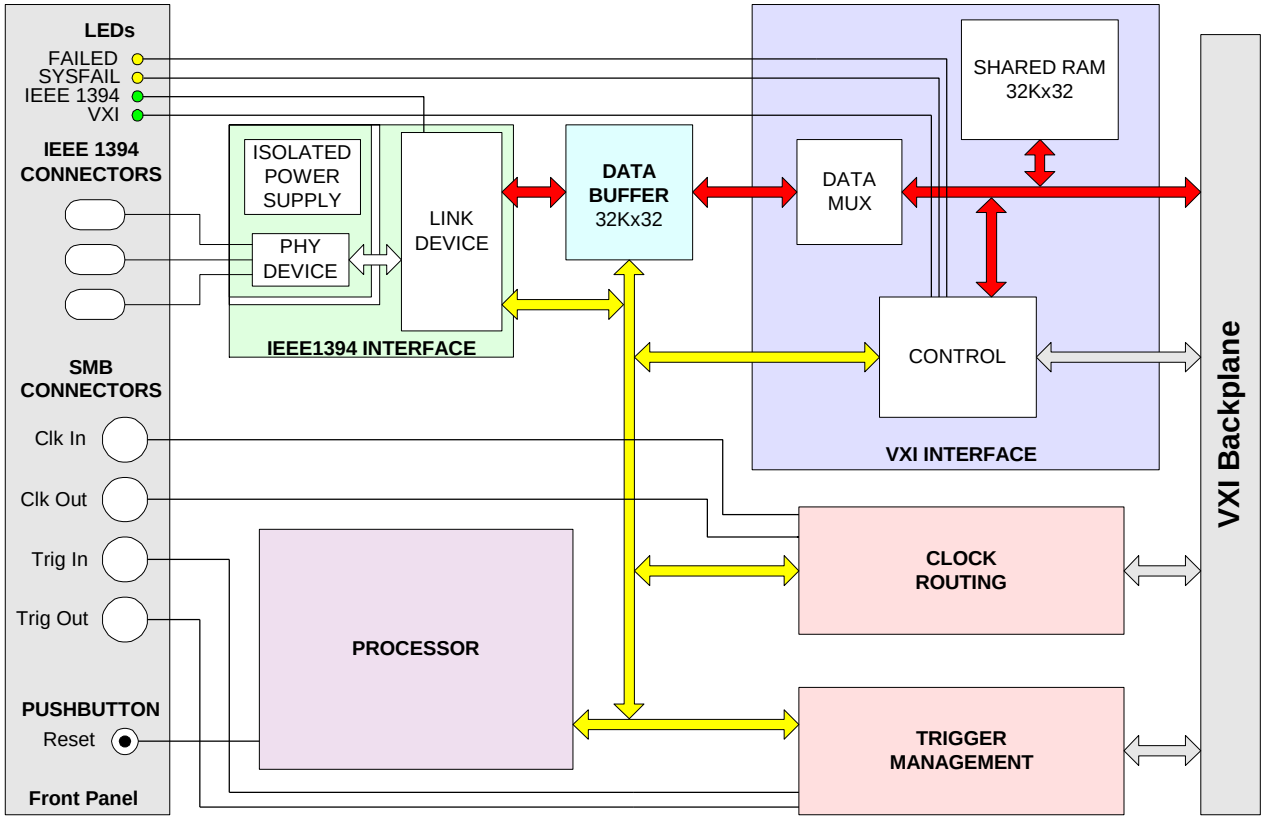


Figure 7: E8491 Hardware Block Diagram.

Data Buffer Byte Organization:

No Byte Swapping				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	0	1	2	3
Word 1	4	5	6	7
…	8	9	…	

Byte-swapped Doublets				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	1	0	3	2
Word 1	7	6	5	4
…	9	8	…	

Byte-swapped Quadlets				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	3	2	1	0
Word 1	7	6	5	4
…	11	10	…	

Byte-swapped Octets				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	7	6	5	4
Word 1	3	2	1	0
…	15	14	…	

Doublet-swapped Quadlets				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	2	3	0	1
Word 1	6	7	4	5
…	10	11	…	

Doublet-swapped Octets				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	6	7	4	5
Word 1	2	3	0	1
…	14	15	…	

Quadlet-swapped Octets				
	B31-25	*B24-16*	*B15-8*	*B7-0*
Word 0	4	5	6	7
Word 1	0	1	2	3
…	12	13	…	

VXIbus Byte Organization:

8 Bit Cycles: Even Starting Addresses				
	D31-25	*D24-16*	*D15-8*	*D7-0*
Cycle 0			0	
Cycle 1				1
Cycle 2			2	
…			…	

8 Bit Cycles: Odd Starting Address				
	D31-25	*D24-16*	*D15-8*	*D7-0*
Cycle 0				0
Cycle 1			1	
Cycle 2				2
…			…	

8 Bit Cycles: Odd-Only Addresses				
	D31-25	*D24-16*	*D15-8*	*D7-0*
Cycle 0				0
Cycle 1				1
Cycle 2				2
…				…

16 Bit Cycles				
	D31-25	*D24-16*	*D15-8*	*D7-0*
Cycle 0			0	1
Cycle 1			2	3
Cycle 2			4	5
…			…	

32 Bit Cycles				
	D31-25	*D24-16*	*D15-8*	*D7-0*
Cycle 0	0	1	2	3
Cycle 1	4	5	6	7
Cycle 2	8	9	10	11
…	12	13	…	

32 Bit Cycles								
	A31-25	*A24-16*	*A15-8*	*A7-0*	*D31-25*	*D24-16*	*D15-8*	*D7-0*
Cycle 0	0	1	2	3	4	5	6	7
Cycle 1	8	9	10	11	23	13	14	15
Cycle 2	16	17	18	19	20	21	22	23
…	24	25	26	27	28	29	…	

Figure 8: Data byte organization in the HP E8491 Data Buffer for various byte swapping modes and on the VXIbus backplane for various data transfer modes. Byte 0 is the data in the lowest addressed byte location of a given data transfer.

V. A LIGHTWEIGHT PROTOCOL

A customized, lightweight protocol was created for the HP E8491. This was done because of the efficiency needed for communicating to VXI register based instrument cards and the requirement to keep the firmware simple. Another factor was that other IEEE 1394 protocols (SBP-2, AV/C, DPP, etc) were in the development stages.

The protocol for the HP E8491 makes use of asynchronous packets. This is because 1394 asynchronous packets are acknowledged, so the requester node can always determine if the responder node received the packet. 1394 isochronous transactions are not acknowledged. With asynchronous packets, there is also a response packet that is sent for each asynchronous packet type, so the requester knows if the responder node successfully dealt with the packet.

The protocol uses 1394 split transactions exclusively. See Figure 9. Split transactions allow a 1394 node to acknowledge with an "ack pending". This is done automatically by the link chip. The 1394 node can then take time (but not too much time) to perform the request and then formulate and send a response packet. The alternative to split transactions would be unified transactions, but unified transactions require very fast link and transaction layers.

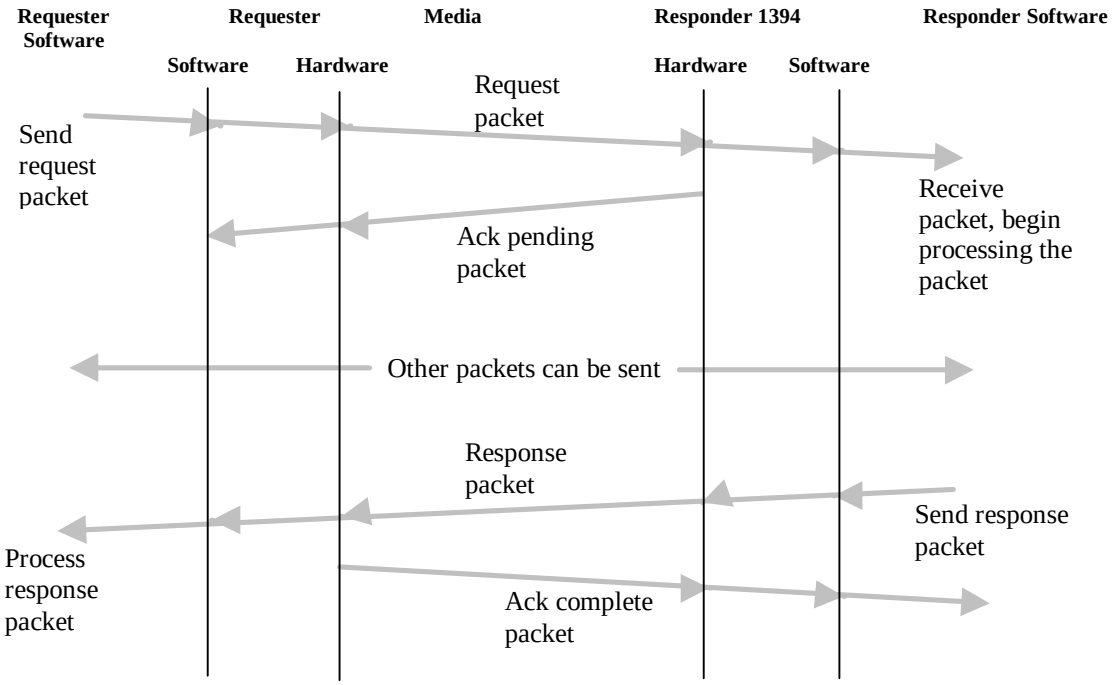


Figure 9 1394 Split Transactions

Several design philosophies guided the protocol development.

1. Software is easier to write/debug/maintain than Firmware.
2. Software runs on fast CPU, Firmware on slow CPU.
3. Software has lots of available memory. Firmware does not.
4. Firmware should be as stateless as possible.
5. The host must be capable of controlling multiple HP E8491 devices.
6. Need to allow as much concurrency of commands as possible.
7. The Firmware should never block indefinitely.

The lightweight protocol used in the HP E8491 makes use of asynchronous quadlet reads, quadlet writes, and block writes.

Quadlet reads are used when an application needs to read (peek) a VXI memory location. This operation is performed many times for register based VXI instruments cards. The destination address in the header determines the VXI memory location, the address space, and the data width for a particular peek. This is done by reserving 6 address ranges in the IEEE 1394 address space of the HP E8491. One is for A16/D08, another for A16/D16, etc. Any peeks that do not fit into one of these 6 categories are done via a block write request packet instead of a quadlet read packet. Figure 10 shows the structure of a quadlet read packet.

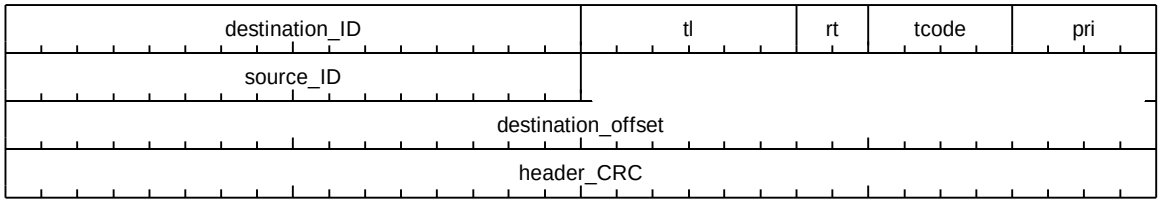


Figure 10 1394 Quadlet Read Packet

Quadlet writes are used when an application needs to write (poke) a VXI memory location. The destination address again determines the VXI memory location that is accessed, the address space, and the data width for this transfer. The same 6 address ranges described above are used for quadlet write requests to poke data. The quadlet data contains the data that is written. Figure 11 shows the structure of a quadlet write packet.

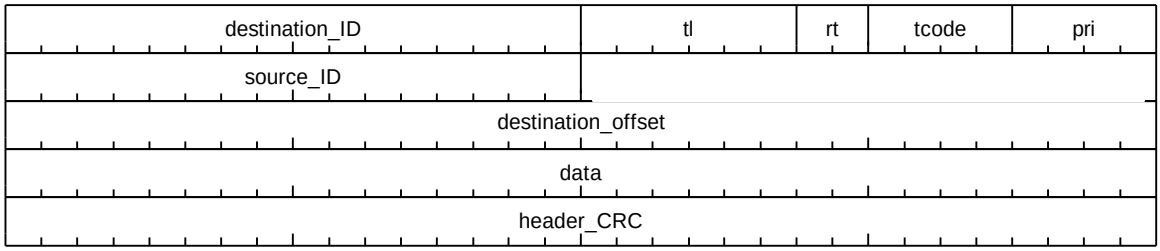


Figure 11 1394 Quadlet Write Packet

The protocol makes extensive use of block writes. Block writes are used to send protocol commands other than peek and poke operations. Figure 12 shows a block write.

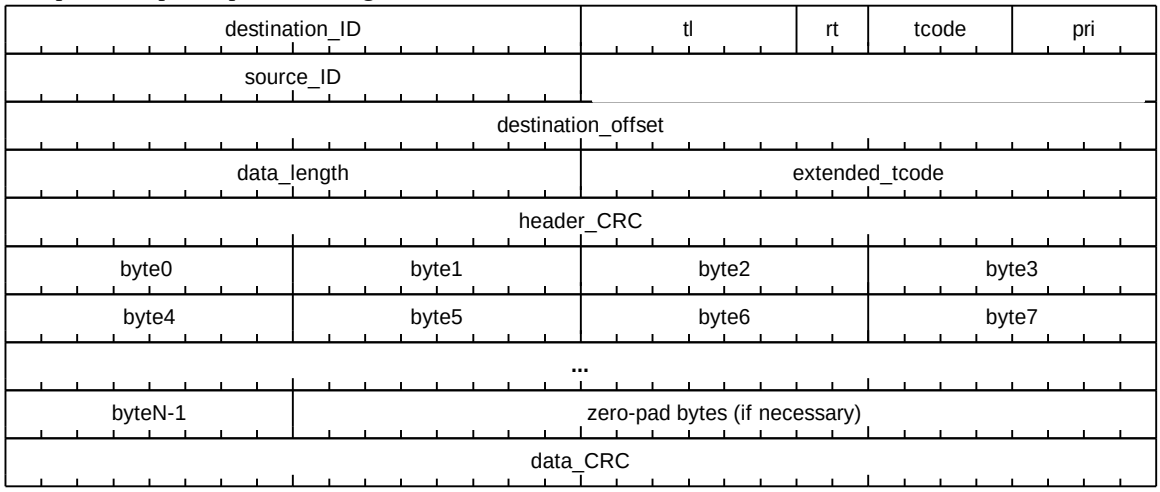


Figure 12 1394 Block Write Packet

In general, the protocol utilizes block write packets in pairs. First, the host sends a 'send to the HP E8491. Then the HP E8491 sends a 'return packet' back to the host. There are a couple of exceptions to this general rule, one being large block data transfers and another being firmware-initiated packets. Each of these will be discussed in more detail.

For a typical situation, a user application on the host computer makes a request to our driver software. The host software then formulates and sends a 'send packet' to the firmware based on this request. Included in the packet data is a command number, indicating to the firmware what operation is requested. It also contains a 1394 return address, which is the address to which the firmware should send its 'return packet'. Any other data required for this particular command is sent at the same time. The firmware, upon receiving the packet, parses the command number, executes the request, and sends a block write packet back to the host to the address specified in the original request. Included in this return packet is typically an error code and any other data required by the host software.

An example of a complete transaction is a user request to assert a particular VXI trigger line. The host software processes this request by forming a packet to be sent to the firmware. In this packet is data describing what command to execute (assert a trigger line), and any data needed for this command (which trigger line, where to send the return packet). This packet is sent using the split-transaction mechanism described above. At this time, the host software waits for the firmware to write a response packet to the indicated IEEE 1394 address. Meanwhile, the firmware unpacks the data in the packet, sees that it is an 'assert trigger' command, determines which line to assert, and then actually asserts the line. It then formulates a return packet and sends it to the host.. The firmware's job is done, and the HP E8491 waits for the next request. The host software, however, wakes up upon receiving the return packet, unpacks this packet to see if the operation was successful, and returns an error code back to the user’s application.

Another interesting scenario is the handling of firmware-initiated packets. This situation arises when the firmware detects a VXI interrupt that must be sent to the host computer. Prior to the interrupt, the HP E8491 must be configured to enable interrupts. First, the host software creates a separate thread which is 'sleeping' on a particular 1394 address (a different address than is used for normal return packets). Second, the host sends an 'interrupt enable' packet to the firmware, indicating the target IEEE 1394 address for the interrupt packets and which interrupt sources to monitor. When an enabled interrupt event occurs, the firmware formulates a packet that includes all the relevant information for this interrupt, and sends it to the interrupt packet address. This wakes up the interrupt thread in the host software, which then unpacks the packet, parses the information, and does whatever processing is necessary to handle this interrupt. Note that a normal 'return packet' is not used in this situation.

There is, however, a race condition that must be handled. If the firmware detects two interrupt conditions in a row, and sends two packets, one right after the other, the host may not have time to read the data from the first packet before it is overwritten by the second. To handle this, we created a simple mechanism of disabling/enabling interrupt packets. Whenever the firmware sends an interrupt packet, it disables the sending of further interrupt packets. When the host software completes processing of an interrupt, it sends an “ok to send” packet to the HP E8491, indicating to the firmware that it can once again send interrupt packets.

The transfer large blocks of data presents additional issues, since an IEEE 1394 packet sent at 400Mb/s can hold at most 2048 bytes of data. In the case of writing blocks of data to VXI, the mechanism works much like the "assert a trigger line" example above. First, a command packet is sent to the HP E8491 telling it to be ready for a large block of data. Next, data-only packets are sent to a different 1394 address on the HP E8491, one after the other until all data has been sent. At this point, the host software sleeps on the 'return packet' address, waiting for the HP E8491 firmware to complete the write, formulate and send the return packet back to the host. Note that while sending the data packets, it is possible to fill up the packet queue in the firmware. In this situation, the firmware will return a 'conflict error' back to the host, indicating that the host should sleep a bit to allow the firmware to catch up, then start sending again.

The protocol handles large block reads in a manner similar to large block writes. First, the host sends a 'command packet' to the firmware telling it that it wants to read a block of data. Included in this packet is

information about where to write the data (in 1394 space) as well as where to write the normal 'return packet'. The firmware starts reading from VXI space, and sending packets to the data area. This area is configured (by the host) to map directly into the user's buffer, so no host software (other than the PCI card driver software) is involved with the transaction at this time. Once the firmware completes writing all of the data, it formulates and sends a 'return packet' to the address on which the host is sleeping, waking it up.

VI. FIRMWARE FOR THE HP E8491

Firmware is required to interact with the 1394 link layer chip when receiving and transmitting 1394 packets and for setting up the hardware to perform VXI operations.

The firmware is simple enough to not require the use of an embedded operating system. A single task main() loop is sufficient. All of the firmware fits in the available 64 kB of fast on-chip ROM. Interrupt service routines (ISRs) are used to handle asynchronous interrupts from the 1394 link layer chip and interrupts from the VXI hardware.

1.1 Booting

The firmware boots from internal flash ROM, creates a ‘C’ environment, and then calls a ‘C’ routine that performs a ROM checksum and RAM test. The firmware then calls the normal ‘C’ startup routine that initializes global and static variables and ends by calling main().

The main() function calls more self-test routines and a routine to program the FPGA’s. The FPGA’s are programmed with a very long serial bit stream, so the programming routine is optimized to speed the boot process. After the FPGA’s are programmed, more self-tests are executed, and finally the interrupts are setup and enabled. Finally, the main() function enters an infinite loop, checking for work to do, based on flags set by the ISRs.

1.2 Firmware for the IEEE 1394 Interface

The 1394 firmware gets involved whenever there is an interrupt from the 1394 link chip.

When a packet is received, the link hardware sends an immediate acknowledge of ‘ack pending’, and the firmware ISR is called. If there is a data payload, it is important to wait until all of the data has been received before the ISR is called. The ISR checks for CRC errors and then reads the packet header from registers on the link chip. The packet header information is examined to find the transaction code and destination address.

If the packet is a quadlet write or read, the ISR immediately performs the operation. The destination address in the packet header determines the VXI address to poke (quadlet write) or peek (quadlet read). When the operation is done, the response packet is formulated and queued. The response state machine, implemented in firmware, will try to send the response immediately if it can. The ISR then exits. The host should receive the response packet and reply by sending ‘ack complete’. The link ISR is again called when the acknowledge is received. If the host sends an ‘ack busy’, the response state machine retries sending the response packet later.

When a block write packet is received, the destination address is checked to see if it is an HP E8491 protocol command. If so, the protocol command is embedded in the data payload. The protocol command is read from memory and the appropriate data structures are modified so the main() loop processes the command after the ISR exits. A write response, with response code of ‘response complete’ is formulated and queued. The ISR then exits.

If the block write destination address indicates the packet was a pure data packet, the ISR again updates data structures and queues a write response with response code of ‘response complete’. The ISR then exits.

The protocol calls for the HP E8491 to initiate the sending of write blocks to the host when done with certain commands. This is done by queuing the transaction request. The transaction request state machine, implemented in firmware, is used to send transactions from the HP E8491 to the host. The link ISR is called when the command is sent and an acknowledgement is returned. Normally the host will send an 'ack pending'. The transaction request state machine then starts a split transaction timer, which acts as a watchdog. The host should send back a response within the split transaction timeout interval. The ISR exits. The host later sends a response and the HP E8491 link chip automatically sends an 'ack complete'. The link ISR is called again. The response code is checked and the transaction request is finished. Any memory associated with this request is freed.

1.2.1 IEEE 1394 Bus Resets & Configuration ROM

After a bus reset, the first packets received are self-ID packets. Since the HP E8491 is not a bus manager, the self-ID packets are ignored. Next, the bus manager attempts to enumerate the bus to find what devices are present. The bus manager does this by issuing a series of quadlet read packets to get the configuration ROM information from each node. The quadlet reads should end when the limit of the configuration ROM size is reached, but in practice, the firmware designer should be prepared to send a response indicating an address error if the destination address goes beyond the size of the configuration ROM.

The configuration ROM contains the 64 bit unique identifier required of all 1394 nodes. It contains strings indicating the manufacturer and the model identification for the device. It also contains numbers that identify the protocol to be used when communicating to the device.

The structure of the configuration ROM is shown at URL:
http://www.microsoft.com/hwdev/respec/pnpspecs.htm#1394

1.3 VXI Firmware

1.3.1 VXI Interface

The VXI interface provides two paths for transferring data to and from the VXI back plane. The first path uses a data register to peek or poke a single 8-bit byte or 16-bit word. This path is used in conjunction with quadlet read requests for peeks and quadlet write requests for pokes. All of the information required to configure the VXI interface is contained in the packet. The second path transfers data between the VXI back plane and the HP E8491 data buffer. This path is used for single 32 and 64 bit transfers and for all multiple data element transfers. All the data contained in a single packet is transferred between the VXI back plane and the buffer in a single operation of the VXI interface. The VXI interface interrupts the DSP at the end of the operation. This allows the DSP firmware to perform other tasks such as sending and receiving other packets while the operation is taking place.

The VXI Interface also controls and monitors the VME interrupt lines. The assertion of any interrupt line (handled by the E8491) will interrupt the DSP. The VXI Interface is then used to generate an interrupt acknowledge (IACK) cycle. The results of the IACK cycle are placed in a packet and sent to the host controller. If more than 1 device is interrupting at the same time, multiple IACK cycles will be generated until no line is asserted. All of the resulting Status/ID words along with the corresponding VME interrupt line numbers are placed in the data buffer and then sent to the host controller as an SRQ packet. VXI signals are processed in much the same way. The VXI Interface will interrupt the DSP when Status/ID data is written to the HP E8491 signal register FIFO. All of the data will be placed in the data buffer and sent as an SRQ packet to the host controller.

1.3.2 Message Based Word Serial Commands and Message Transfers

HP E8491 firmware processes all word Serial commands and message transfer request packets. For message transfers to a VXIbus message based device, the firmware extracts the data from the request packet and sends it to the message-based device, one byte at a time. In the case of transfers from a message-based device, the firmware gets the data from the device and places it in a return packet. The firmware paces the word serial message transfer by polling the state of the Write Ready, Read Ready, DIR, and DOR bits in the message based device's response register.

1.3.3 Triggers

The HP E8491 supports VXIbus TTL triggers 0 through 7, VXIbus ECL triggers 0 and 1, and External (front panel) triggers In and Out. The HP E8491 supports the routing of trigger inputs of one type to trigger outputs of another type. For example, any TTL trigger can be routed to any ECL trigger and to the EXTERNAL trigger out connector. Likewise, the EXTERNAL trigger In can be routed to any TTL or ECL trigger. The triggers can be enabled to interrupt the DSP when the trigger is asserted. When a trigger interrupt occurs, the logic state of all trigger lines is read and reported to the host controller in a trigger SRQ packet. Furthermore, the logic state of any trigger can be controlled and monitored by the host controller.

1.4 Dual Boot and Downloading

The HP E8491 has a dual boot capability. The firmware can boot from the internal flash ROM or from external flash ROM. Dual booting allows the HP E8491 to download code even if one of the ROM's is corrupt. This feature is almost free since a large external ROM is required to store the FPGA configuration data.

Since the FPGA configuration data is stored in the external flash ROM, and downloads must work when either flash ROM bank is corrupt, the downloads must work when the FPGAs are unprogrammed. When the FPGAs are unprogrammed, the HP E8491 is not capable of sending or receiving block write packets. Thus the download protocol uses only quadlet reads and writes. Quadlet writes are processed entirely within in the link chip registers, so a minimal hardware set is required for a successful download.

VII. HISTORY AND FUTURE OF THE IEEE 1394 STANDARD

Work on defining a fast serial bus began at Apple Computer. Official work on the IEEE 1394 standard began in 1986. The IEEE 1394 standard says "This standards effort started in 1986 at the request of the membership of the IEEE Microcomputer Standards Committee as an attempt to unify the different serial busses originally proposed as parts of the IEEE 1014 VME, IEEE 1296 Multibus II, and IEEE896 FutureBus+ efforts." Various improvements and new features were added. Isoochronous transport was added to satisfy multimedia needs. The standard was approved in 1995 – hence the name IEEE 1394-1995.

The IEEE 1394 standard makes use of IEEE 1212 for the addressing and definition of Control and Status Registers (CSR's).

Many follow-on 1394 related standards groups have formed to address needs not met in the original standard. The table below summarizes some of these efforts.

Standard Name	Purpose
P1394a	Seeks to add some features to the original standard while maintaining backward compatibility. Seeks to allow better utilization of the available bandwidth. Expected approval date: Q3, 1999.
OHCI – Open Host Controller Interface	Define an implementation of the link layer. The OHCI standard provides mechanisms for DMA and defines a common set of programmable registers.
1394b	Seeks to extend speeds to 3200 Mbps. Expected approval date: Q4, 1999.
SBP-2	Defines a protocol for sending commands and data over 1394, useful for storage and other peripherals.
IPV4	TCP/IP over 1394
AV/C	Specify a protocol and command set to control consumer audio-video devices.
IICP, IICP488	Instrumentation and Industrial Control Protocol. Provides efficient, flow-controlled communications to instruments and industrial devices. IICP488 defines how IICP is used for GPIB-like communications. Expected approval date: Q3, 1999.
DPP	Direct print protocol.

The IICP and IICP488 protocols are being developed within the 1394 Trade Association Instrumentation and Industrial Control working group, II-WG. Hewlett-Packard is a leading contributor to this effort.

VIII. FUTURE OF IEEE 1394 IN TEST & MEASUREMENT

The future of IEEE 1394 in T&M looks very promising. This is due to the multiple customer benefits 1394 offers – ease of use, low cost, and high speed. The only missing ingredient is widespread industry adoption. The IEEE 1394 market is small today, but growing - largely due to consumer audio/video devices. For these devices, which must be made easy for consumers to use, a single IEEE 1394 cable can replace the audio, video, and other control cables that are usually needed. Other indicators of a growing market are:

- There are a growing number of vendors of 1394 host bus adapters.
- There are PC's being introduced with 1394 built in or as an option. We will see many more as PC manufacturers strive to meet the design guidelines for future PC's.
- Microsoft has IEEE 1394 support built into Windows 98 and the next release of Windows NT.

We expect a slow migration away from legacy I/O. In this period, multiple I/O interfaces will be available on T&M products. HP-IB has been around for 20+ years and will not be displaced overnight.

As more instruments are equipped with IEEE 1394, and as IEEE 1394 becomes standard I/O in the computer, test system designers will switch to IEEE 1394. Test system designers looking for higher transfer rates will be first to switch from legacy I/O.

To continue using instruments with legacy I/O, test system designers will make use of IEEE 1394 to HP-IB converter boxes. These converter boxes will convert IEEE 1394 I/O traffic on one side to legacy HP-IB I/O traffic on the other side.

IEEE 1394 has many more capabilities than HP-IB, opening exciting new opportunities. For example, the peer to peer nature of the IEEE 1394 bus allows for a paradigm shift away from a computer controlling all communications in a test system to a paradigm where smart instruments can communicate to other 1394 peripherals. An instrument that produces data could send data directly to a peripheral that could store, display, or process the data. The future is open and the possibilities are exciting.