PMAC, PCS, INDUSTRIAL AUTOMATION SYSTEMS, AND OPEN CONNECTIVITY

A white paper on the development of 3U Turbo PMAC2 and the UMAC (Universal Motion and Automation Controller).

This paper is written for the purpose of informing all concerned about Delta Tau's point of view on what flexibility and capability will be provided by the latest evolution in communication software (OPCs) and the advent of the new busses, USB and FireWire (yes, they are busses!!, although they are "long and skinny" instead of short and fat!!) Also, the resurgent interest in Ethernet and TCP/IP is examined.

Brief descriptions are provided of each of these, and their interrelationships with PMAC, MACRO, PC/104, and I/O are discussed.

This paper takes on added significance because of Delta Tau's latest Motion System development, the UMAC, which is based upon Turbo PMAC technology and provides a 3U package with easy internal interconnectivity by using the open architecture UBUS backplane and external (World & Machine) connectivity via a multitude of channels such as PC/104, a variety of field busses such as DeviceNet, ControlNet, ProfiBus, CanBus etc., USB, Ethernet, FireWire, and a great number of axes, digital and analog I/O accessories, including custom designed boards when desired.

Although this paper was written about three years ago, the analysis remains pertinent.

Delta Tau's alternate choice uses to provide the high-speed parallel PCI bus on some of its products, as well as USB2 and Ethernet 100 as primary communications on ALL of its products. Additionally, complete field bus support is provided by using the Hilscher product line.

Table of Contents

Glossary and Background Information

This document contains a lot of information that may not be familiar to the reader. This glossary defines some of the terms and where to obtain additional information on the new technologies. It's placed at the front so you know it's not at the back. The suggested websites provide links to many other sites containing a wealth of product, vendor, technical, and support information. There's a lot of new technology available, and this is an attempt to collate some of the most useful and pertinent information in a convenient manner.

OPC - OLE for Process Control

OPC is a universal device driver concept based on Microsoft's OLE agreed to by many if not most vendors and developers of automation hardware and software. OLE or Object Linking and Embedding is a set of standard software APIs that allow one application to use capabilities provided by another application. For example, the embedding of an MS PowerPoint figure in an MS Word document. OPC was defined and is maintained by a nonprofit organization called the OPC Foundation. You can access the entire specification and related information at www.opcfoundation.org. Both OLE and OPC are based on Microsoft COM or Component Object Model.

OPC Broker

OPC defines a set of interfaces that allow clients to access any automation device with an OPC Server. OPC Broker is a piece of software that connects multiple OPC Servers together transparently copying data from one to another. It is a broker moderating the flow of data between these servers, hence the name. In today's verbiage this is referred to as "middleware".

COM – Component Object Model

COM is a specification and a set of services that allows you to create modular object oriented, customizable, and upgradeable distributed applications using a number of languages. COM is an integral part all MS operating systems beginning with MS Windows 95. There are many excellent but highly technical books on COM available. A dry, but excellent starting place is the book "Inside COM" by Dale Rogerson from Microsoft Press. For those less technically inclined, the Microsoft Press book "Understanding ActiveX and OLE" by Chappell is a good starting place. You can also access the Microsoft web site at www.microsoft.com or msdn.microsoft.com.

DCOM – Distributed COM

During the evolution of COM a distributed network version of the COM specification was developed that allowed applications to access services on remote computers connected together by the Internet. Recently DCOM was absorbed into a single specification of COM. In 1998 it was submitted to the Internet Engineering Task Force for consideration as a standard part of the TCP/IP protocol stack.

COM Server

A server is a software program residing on a networked computer that provides a service. That service can range from a web server to a file server. Different industry segments such as the financial and healthcare industries are defining standards for servers that provide account information and the like. The automation industry has defined a standard known as OPC that includes interfaces for data access, logging, alarming, and event services.

COM Client

A client is a software program that uses a networked server to provide a specific service. For example, a client application might use the services of a Microsoft SQL Server for database support and a PMAC OPC Server to gather data from PMAC to store in the database.

COM Interface

An interface is a published API defining the services provided by a COM server. The interface can be accessed by a COM Client using many different compilers and programming languages. Accomplishing this is easier in some languages such as Borland Delphi and MS Visual Basic or Java.

In-Process Server
An in process server is a COM Server operating as a classical dynamic link library (DLL).

Local Server
A local server is a COM server operating as a separate operating system process physically located on the same computer as the client.

Remote Server
A COM server executing on another networked computer.

SCADA – Supervisory Control and Data Acquisition
This is another name for a Graphical User Interface (GUI), Man Machine Interface (MMI), Human Machine Interface (HMI), or Human Computer Interface (HCI).

PC104
PC104 is essentially an ISA bus designed for embedded applications. It exists in two sizes. Core modules are 4"x4" and little boards are 6"x4". Information on PC104 products and vendors is available at www.ampro.com.

Fieldbus
A fieldbus is a network explicitly designed to meet the needs and requirements of a distributed automation environment. There are at least 10 different fieldbus specifications used by the various PLC and automation vendors. Information on these can be obtained from the vendors' website or its equivalent "open" standards consortium. The appendix at the back of this white paper details the characteristics of many of the automation field busses. It is available from www.synergetic.com. It's worthwhile to take a look at this information.

Ethernet
Ethernet is the defacto enterprise network. It was developed by Xerox in the early 1980's and defines the physical layer of a multiple access network. It can be found on nearly all computers today. The primary protocols built on top of Ethernet are Microsoft's NetBEUI, Novell's Netware, and TCP/IP.

TCP/IP – Transport Control Protocol/Internet Protocol
T CP/IP is really two network protocols developed by the Defense Advanced Research Project Agency in the 1970's. TCP is the Transport Control Protocol and IP is the Internet Protocol. IP runs on top of Ethernet, a serial port or modem connection (Serial Line Internet Protocol – SLIP), parallel port, or ATM (Asynchronous Transfer Mode). It defines how a network of routers locate host computers and send messages using the infamous xxx.xxx.xxx.xxx Internet address. Hence, it is the internet protocol that connect multiple networks together. TCP operates on top of IP and defines how large packets of data are broken up and that the individual packets are properly reassembled when they are received. Furthermore, it is responsible for sending acknowledgements back to the sender.

Microsoft Windows
Microsoft Windows is known to every computer user. Different flavors of Windows provide different capabilities and resources. These differences are important to the automation integrator for security and remote OPC connection capabilities. COM, therefore OPC, has restrictions on which groups and users are allowed to access OPC servers. Furthermore, Windows 95 and 98 do not allow an OPC Server to be started from a remote computer – it must be running prior to connection. Windows NT does not have this restriction and remote users can start the server and connect remotely.

DDE – Dynamic Data Exchange
DDE is an early Microsoft approach to allowing data exchanges between different application processes on a computer. It originally allowed data to be shared between the various pieces of the Microsoft Office suite.

It was adopted by a number of automation vendors to allow the sharing of data between different automation applications. Later development of DDE resulted in NetDDE.

USB – Universal Serial Bus

The USB is a hardware and software specification defining all aspects of high-speed serial bus for PCs designed to replace the mouse, keyboard, parallel port, and serial ports. Its goal is to provide a versatile and capable cable system that can connect all standard PC peripherals with a single cable. The Specification defines a sophisticated bus protocol and device driver architecture that allows the user to connect or remove any device to the bus without cycling power and have that action immediately recognized by the operating system. You can obtain the specifications at www.intel.com.

UBUS

The UBUS is the backplane with which the UMAC controls communications between boards plugged into the 3U Rack. The UBUS provides the 3U Turbo PMAC2 CPU card, as well as the MACRO CPU card the ability to control a multitude and variety of analog and digital I/O cards via the existing, standard PMAC "JEXP" and "JTHUMB" protocols that are presently used to control the existing peripheral I/O of the PMAC. Significantly, the UBUS adds the ability to the CPU cards to support external dual ported RAM (DP RAM), instead of only the CPU's on-board DP RAM. This new feature greatly enhances the UMAC's ability to communicate at high speed with a great many new communication and field bus I/O cards. Complete specifications for the UBUS are available for allowing custom design 3U boards to be designed and used.

UMAC

The UMAC is a 3U sized motion and I/O control system which is based upon the 3U PMAC2 Turbo CPU or the MACRO CPU. The Turbo CPU supports the PC-104 and therefore allows the UMAC to operate with an on-board PC/104 within its 3U chassis and the UBUS backplane. The UMAC allows both CPU's to communicate with a great variety of I/O and communication devices including various field busses and provides a highly capable, and adaptable automation platform.

IEEE-1394 – Fire Wire

IEEE 1394 is an IEEE standard for a very high-speed serial bus designed as an adjunct to the existing suite of parallel back plane busses. It allows for connection of video devices, mass storage, and processor-to-processor communication. It is an integral part of the Windows 2000 specification and is prolific in the desktop video market. You can find the specification and other related information at www.ieee.org or www.1394ta.org.

MACRO – Motion and Control Ring Optical

MACRO is an open optical network standard developed by Delta Tau designed to provide very high speed synchronous communication specifically for distributed servo and I/O applications. You can find more information on the Delta Tau website at www.deltatau.com.

PMAC

Delta Tau's Programmable Multi Axis Controller controls 2 to 32 axes of servo axis along with thousands of I/O points.

Abstract

System integration has always proven to be an annoyance at best and a very difficult problem at worst. Automation equipment vendors develop unique hardware and software solutions to solve specific problems that often leave the user with an incompatible set of communication cables, protocols, bus structures, and software drivers. As system requirements become more demanding, integrators and developers are increasingly forced to abandon sole-source or one-stop solutions to meet performance, growth, and price constraints and select subsystems from a variety of vendors. As a result, the integration problem has grown more pervasive and critical.

While vendors boldly claim that their approach is "Open" the difficulties associated with connecting these disparate systems together is still with us. At last count one could safely say there were 13 different automation communication systems each with a different physical medium, data rate, and widely varying protocols and capabilities. The problem also exists in the non-automation arena. Today's PCs have so many different interfaces that its hard to determine which ones are appropriate - IDE, EIDE, ATA, ULTRA DMA, SCSI (I, II, and III), RS232, IEEE-1284, EPP, PCI, EISA, ISA, AGP, VGA, SVGA, ... and on and on. Fortunately, TCP/IP is the defacto standard for computer networks and Microsoft Windows is a standard for the enterprise and on the factory floor.

The insertion of PMAC into this mélange of possibilities is daunting. Whose PLC should I use? Which processor bus or automation network should I use? Which communication, software, or hardware vendor should I use? When I get the pieces, what SW do I need? How do I connect this piece of equipment to that one? What does the future hold? And, most importantly, who is responsible when it doesn't work like they told me it would?

Delta Tau understands these issues and is aggressively taking steps to provide a clear, well-defined path that will allow you to insert increasingly powerful PMACs into a wide array of automation environments. In doing so you can add PMAC's cost effective and state-of-the-art motion capabilities into existing automation environments or design new automation systems around PMAC and select the logic controller and communication approach that meets your specific requirements.

In this white paper we examine tomorrow's hardware, software, communication, and automation technology and describe how Delta Tau is preparing PMAC to take advantage of tomorrow's promises. This is done by bringing together several hardware and software specifications on which tomorrow's automation systems will be based. Most importantly, we define precisely how Delta Tau is modifying PMAC's hardware and software architecture to capitalize on these developments so that you, the user and integrator, can plan for these impending changes.

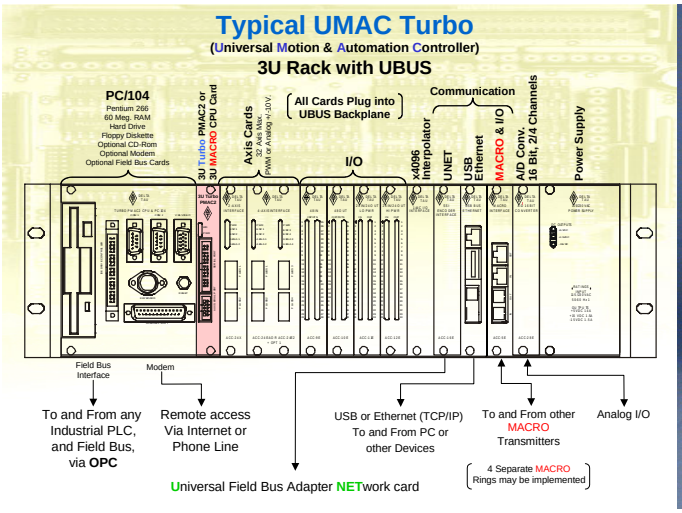
For the first time in fifteen years, the PMAC's form factor will no longer be dictated by its insertion into a STD, VME, ISA, or PCI bus. Its user environment will not be restricted by custom device drivers and application software that requires constant maintenance. Instead, PC standard high speed serial busses such as USB (replacing ISA) and FireWire (replacing PCI), both of which are the new, "long skinny busses," will allow communication connectivity to free form the PMAC, using inexpensive 6 pin connectors and simple cables, thus freeing the PMAC from having to be within the PC enclosure. Universal device drivers based on OLE for Process Control, OPC, will allow connectivity between OPC compliant devices, application software, and PMAC. What remains to be done is to choose a dependable, ruggedized packaging scheme capable of supporting this free form environment.

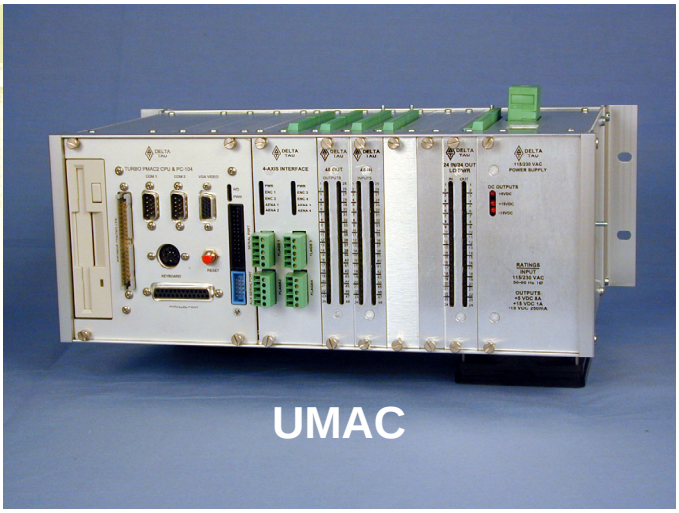
An example is our new 3U Turbo PMAC2 CPU, which is made to fit in a 3U chassis, the UMAC (Universal Motion and Automation Controller) rack, with the UBUS (Universal Bus). It combines the best of PMAC with an embedded PC/104 bus interface, which allows running MS Windows in a rugged stand-alone chassis. Integrated communication support using OPC allows connectivity to a variety of automation field busses and the enterprise using Ethernet, USB, or FireWire via the PC/104, or the same connectivity via the UBUS and PMAC itself by means of UNET card (Universal Field Bus Adapter Network card). As we describe PMAC's future in this white paper, keep the picture of the 3U chassis shown below in your mind.

The 3U design for the Turbo PMAC2 CPU can be configured either in the enclosed typical "UMAC" configuration (shown) or a compact, cost-effective "stack" configuration. The PC/104 interface is available in either configuration.

Incidentally, it is important to note that the PC/104 bus used here is nothing more than a physically restructured ISA bus which can readily be replaced by the USB now and/or FireWire in the future.

As with any new event in the realm of technology, the new "Skinny Serial Busses" will not immediately replace the old parallel busses but will cause a gradual shift. Also, serial in general is never as fast as parallel. A glaring deficiency being real time interrupts. Nevertheless, the great packaging convenience free of "PC form factors" and cost reduction will have a lasting effect on PC/Peripheral connectivity and the PMAC with its peripherals will be no exception.





Delta Tau's new 3U Turbo PMAC2 and chassis (UMAC), now includes PC/104 bus with Windows operating system with Ethernet, SVGA, and a 3.2GB hard disk. Servicing and installation can be done using the local CPU or a laptop using the integrated USB connection. When coupled with available Field bus cards and PMAC's OPC server you can now connect to most classical PLCs over any field bus and have the full power of PMAC available at the 3U rack. A similar connectivity to field busses can be achieved by using the UNET (Universal Field Bus Adapter Network card) card which plugs directly into the UBUS backplane.

We begin by describing how PMAC will address three very different aspects of the integration problem.

The first topic is universal software support under MS Windows using OPC. OPC has emerged as the preferred way to connect automation systems to each other and to large scale manufacturing software systems. We will describe the architecture in detail and show how PMAC supports it.

Next, using OPC we will describe the steps we are taking to provide connectivity between PMAC and a variety of automation controllers using vendor specific field busses.

Finally, we will discuss USB and FireWire connectivity between PMAC and PC palmtops, laptops, desktops, and servers.

Delta Tau's serial connectivity to these machines exploits the OPC interface and will provide a wide range of high-speed communication with the click of a switch. As we cover each topic we will often refer to the public-domain specifications for OPC, USB, and FireWire. In doing this we intend to dispel popular misconceptions and give the reader a concrete understanding of exactly why these new technologies were developed, how they operate, and what benefits you can expect from them.

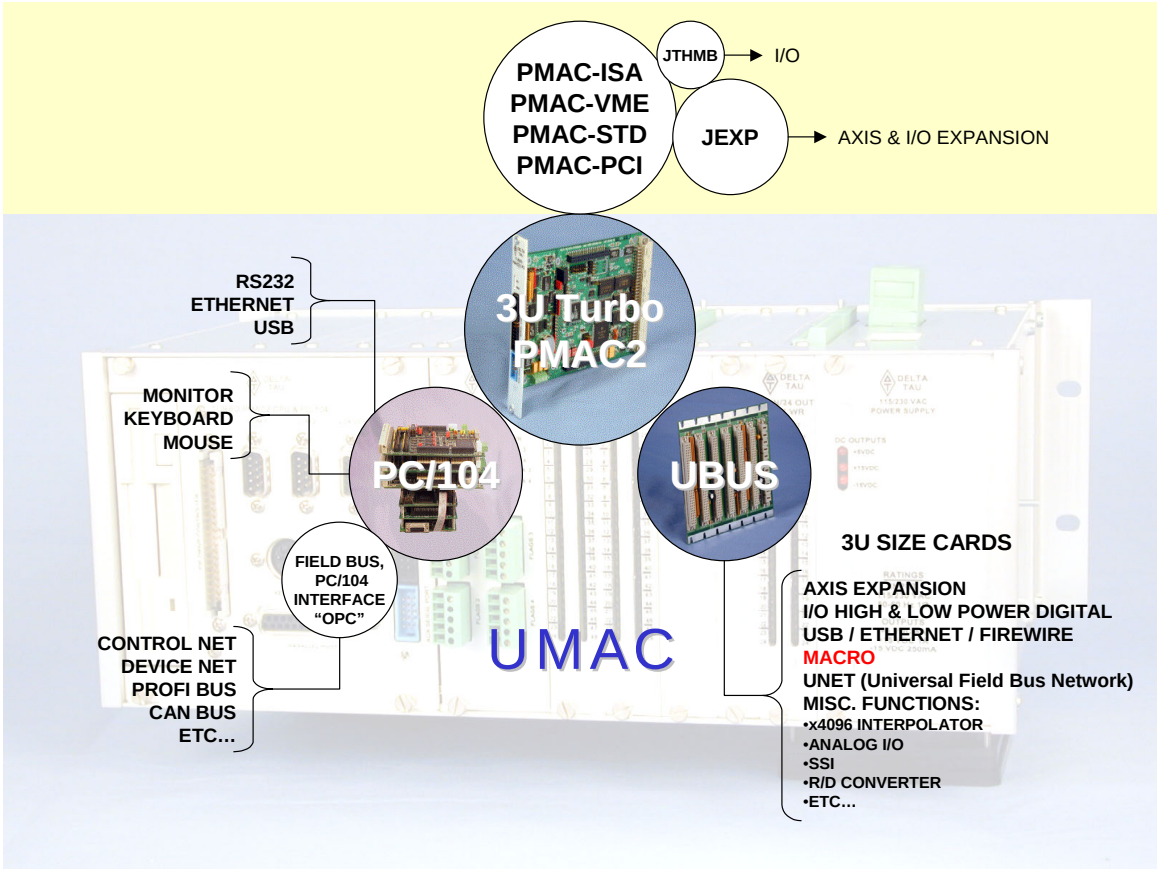
In addition to Field Bus connectivity via the PC/104 and OPC software, the UMAC is able to control a variety of Field Busses via the "UNET" card (Universal Field Buss Network Card) which plugs directly into the UBUS and via selected "Hilscher" cards, can communicate to I/O devices on many of the popular Field Busses such as Device Net, Control Net, Profibus, Interbus, Can Open, etc. Also, the UMAC can communicate with the USB and Ethernet, directly through the UBUS and special communications cards which allow the PMAC to connect to a remote PC, as if it was plugged into the local PC Bus.

The MACRO bus will not be discussed in this paper. Its primary focus is high speed, synchronous real-time communications between the motion controller and its drives and I/O devices. While no other bus can match its capabilities in this regard, its primary focus is outside the scope of this paper. The UMAC supports up to 4 independent MACRO rings.

The combination of OPCs, USB, FireWire, PC/104 PMAC and MACRO provides a truly unique solution to almost any motion application.

PMAC's New Connectivity Architecture

PMAC's architecture has historically been driven by the need to plug the controller and its accessories into a STD, ISA or VME based PC chassis. Although PMAC functions as a standalone motion computer, the PC chassis supplied power, a powerful and familiar CPU, useful support software and disk support, and an environment in which you could develop user interfaces. This required an open chassis slot, proper configuration of the device drivers, development of custom interfaces and application software, and maintenance of that ever-troublesome PC environment. Delta Tau recently developed the newest member of the PMAC family, the 3U Turbo PMAC2, with a 3U expansion chassis and the UBUS backplane, the UMAC, that allows you to develop a standalone system with that familiar, rugged, PLC feel. Using this modular architecture there are plenty of extra slots for axis control, expansion I/O and PMAC accessories. Beyond the chassis are two very important developments that will revolutionize PMAC's capabilities as illustrated in the figure below, and explained in subsequent paragraphs.



Delta Tau's 3U Turbo PMAC2 provides two major additions to its suite of capabilities. An embedded PC/104 connector allows you to use any industry standard PC/104 card with PMAC. The PC/104 in the figure has a 233 Pentium CPU with SVGA, Ethernet, and a 4.0 GB hard disk. A new connectivity board hosted on UMAC's expansion bus (UBUS) provides Axis, I/O, MACRO, USB, and FireWire as well as Field Bus connectivity and other general purpose system functions in a single rack.

Embedded MS Windows and PC/104 Fieldbus Connectivity

The first innovation is an integrated PC/104 connector on the backside 3U Turbo PMAC2 controller that allows you to use any industry standard PC/104 expansion board. Using this very popular format, Delta Tau has embedded the PC in the 3U chassis. Our current offering includes a Pentium 133 or 233 class CPU with 16, 32, and 64 MB of memory, integrated SVGA or Flat Panel, a 4.0 GB hard disk, and Ethernet running the MS Windows operating system of your choice. As part of this suite we include a robust and industry standard OPC Server for PMAC. Using this configuration your PMAC can now be configured, controlled, and monitored from the in-chassis PC running your favorite PMAC application or, more importantly, remotely from any MS Windows based PC connected to your network by Ethernet or a dial-in connection.

Note that while the PC/104 bus does not offer any fundamentally new capabilities, since it is really just a physical re-configuration of the ISA bus, its access to compact, inexpensive, embedded computers does provide effective new capabilities.

The combination of the 3U's PC/104 connector and industry standard OPC Servers immediately allows the 3U to utilize a variety of third party automation network cards to couple PMAC to the existing world of industrial logic controllers by using the newly developed PMAC OPC Server. Below are offerings from two vendors with PC/104 automation field bus cards. Many, but not all, cards are currently OPC compliant. However, the vendors have indicated that they are definitely moving in that direction.

Hilscher located in Waterloo, Ontario supports:

- ControlNet
- DeviceNet
- Data Highway
- Profibus

Synergetics in Downers Grove, IL supports:

- ControlNet
- DeviceNet
- Profibus
- Interbus
- PAMUX
- SDS
- CANOpen

Using this approach, you can take existing remote I/O devices or ladder programs running on a PLC and share parametric, control, and status data with PMAC thereby achieving a high level of integration with no network specific SW development. To accomplish this Delta Tau is developing an integrated suite of OPC tools specifically tailored to PMAC that will allow you to transparently transfer both I/O and motion data back and forth between the automation controller and PMAC. In doing this you can have the PLC wait for PMAC to complete a complex motion, send target trajectories from the PLC to PMAC, or use PMAC specific I/O in your ladder program.

PMAC, MACRO, USB, and FireWire Connectivity

The second major innovation besides the PC/104, utilizes PMAC's JEXP 3U expansion bus now referred to as UBUS (Universal Bus) in backplane form, as the local 3U backplane, to provide not only a large number of PMAC axis and I/O accessories such as optically coupled I/O, SSI encoder inputs, x256 and x4096 sinusoidal encoder interpolators, with up to 32 axis of either analog +/- 10V, or direct digital PWM, or Stepper Motor control but also, MACRO, USB, and IEEE-1394 (FireWire) connectivity. The USB and 1394 serial busses provide a wide range of high-speed isochronous and asynchronous data communication between the 3U chassis, another 3U chassis, or a remote PC using the "Long Skinny Bus". These developments are already revolutionizing the PC world by replacing the COM, LPT, keyboard, and mouse ports with one-size-fits-all hot plug-and-play connection. You can readily purchase modems, desktop video systems, printers, scanners, and even digital speakers for USB. The adventuresome hacker will have also noticed the proliferation of high quality camcorders and digital video recorders already equipped with FireWire connectors. It won't be long before all computers are equipped with this low cost port enabling synchronous communication at speeds ranging from 100 to 800 Mbit/s. Already, most Sony PCs come equipped with both USB and 1394 connectors, and the Windows 2000 operating system, due for release in September of '99, calls for these two busses to replace all other external interfaces on the mother board.

Using these two very useful commercial busses Delta Tau anticipates a flood of low cost USB I/O devices and chassis to chassis communication systems based on 1394 FireWire. System integrators and technicians will be able to walk into PMAC equipped plants with a palm top PC, plug into a USB or 1394 port, and begin work immediately. As customer requirements grow and they order more PMACs, using OPC and the standard enterprise network, you'll be able to dial into the plant network, diagnose problems remotely, update system and application software, and provide support from the comfort of your office half a world away. Concurrently, using the MACRO bus capability, can provide you with unparalleled performance for further, real time, local, fully synchronous distributed control with PMAC's centralized brain located in the 3U rack. Although USB and 1394 may appear to be suitable for real time motion control, they are in fact best suited for communications and data transmission purposes, not high-speed synchronous control of multiple servo axes distributed over a wide work area. In summary, therefore, the new 3U Turbo PMAC2 uses the PC/104 when needed for global (network) OPC based communications, and through its local

expansion port in the 3U rack, can use the USB and 1394 for PC communications and possibly in the future, for non-time critical I/O control. The MACRO interface (if needed), can be used for distributed servo control, MACRO's attribute is that it provides a centralized PMAC brain where all computations take place for up to 32 or more axes of control in the local 3U rack (UMAC). The simplistic, unintelligent drives are distributed and result in lower system cost, and because of the centralized brain, a much simpler and effective programmability of the system's motion and I/O refinements.

This is not a simple PMAC anymore. As you can see, these new developments move PMAC from a standalone high performance motion solution to an integral part of the growing enterprise solution. In the remaining sections of this white paper we describe the four primary technologies introduced above. We do this in some detail because there is a lot of misunderstanding surrounding the different technologies.

The UMAC, because of its power and versatility, is rapidly becoming Delta Tau's flag ship product, the latest seal of approval for the UMAC is that is has now become the standard adjunct to Delta Tau's Advantage 810 NC controller, replacing the older "PC Pack." The UMAC offers higher performance, much greater machine connectivity, nearly unlimited power and flexibility all at a reduced cost.

OLE For Process Control - OPC

In this section we examine the structure of OPC, the PMAC OPC Server, and introduce Delta Tau's OPC Broker. The OPC Broker is a middleware component that allows you to connect multiple OPC Servers together and route data from one device to another without writing any code.

Current Client Application Architecture

Most client applications require data from a data source or device and access that data by developing a suite of "Drivers" that specifically meet the software developer's view of the world. This leads to the following familiar problems:

- Duplication of effort - a different driver for each piece of hardware
- Inconsistencies between drivers – different interfaces for each device type and vendor
- Changing support for hardware changes – ever changing drivers to support new capabilities
- Access Conflicts - multiple packages cannot simultaneously access the same device

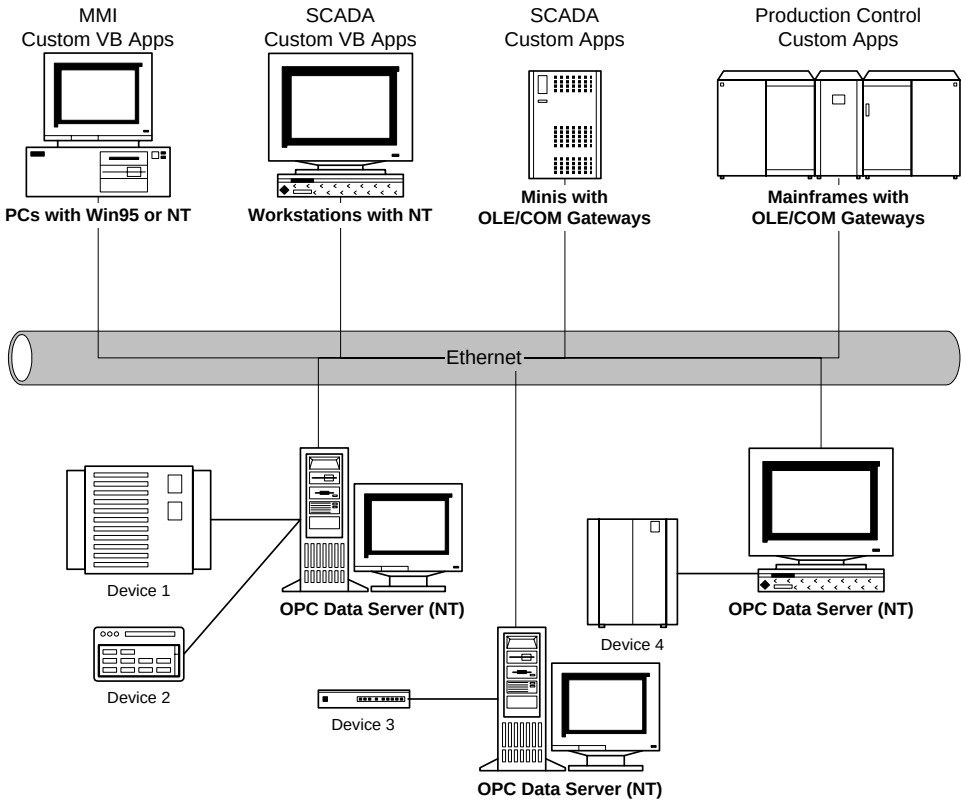
This leaves the integrator and developer in the difficult position of constantly verifying whether a given driver exists for a specific software system and hardware device. Worse yet, the developer gets involved diagnosing driver problems, writing custom workarounds, and dealing with large numbers of large unreadable manuals.

OPC is about Standard Interfaces

OLE for Process Control, hereafter referred to as OPC, draws a line between hardware providers and software developers. It defines interfaces that allow client applications access to automation device data in a consistent manner. Vendors such as Delta Tau can now develop a reusable, highly optimized server to communicate and efficiently maintain access to device dependent data in a manner that all vendors have agreed to. With wide industry acceptance OPC provides many benefits:

- Hardware manufacturers such as Delta Tau write one set of reusable software components to provide access to their devices
- Software developers no longer have to write drivers or rewrite them because of changes in a new hardware release
- Customers have more choices with which to develop integrated manufacturing systems

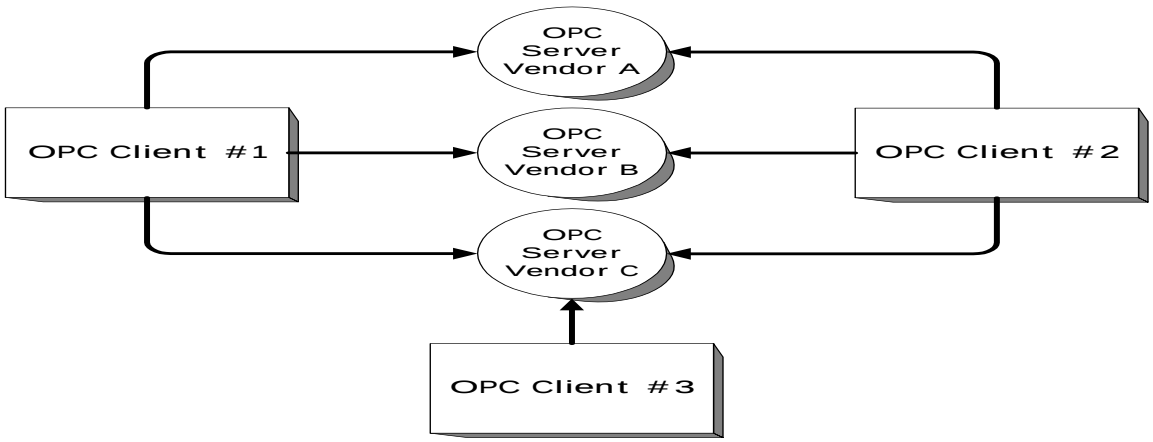
Leveraging Microsoft's OLE and COM standards in the automation environment, OPC greatly simplifies system integration in heterogeneous computing environments. To date, integrating multiple automation subsystems with each other and to the enterprise software applications, required tremendous amounts of custom SW development or casting your future with a single provider. OPC changes the situation by using OPC data servers to standardize access to any device's data. The PMAC 3U with embedded PC-104 MS Windows carries this further by combining PMAC with an OPC Data Server in a single chassis. In doing so, PMAC users can directly insert PMAC into this emerging enterprise architecture with the click of an Ethernet cable.



The original purpose of OPC was to standardize the interchange of data between vastly different devices and make that data available to the enterprise. 3U Turbo PMAC2 with its PC/104 piggyback incorporates the OPC Data Server in the 3U chassis thus greatly increasing PMAC's utility and connectivity. Delta Tau and many other vendors are capitalizing on OPC by applying it in many unique and innovative ways.

OPC Fundamentals

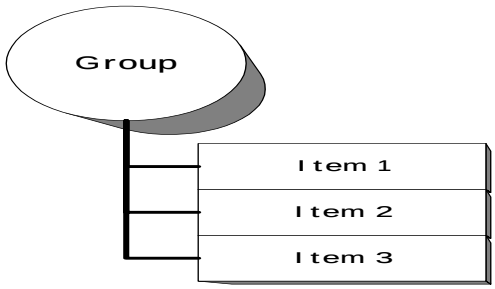
OPC Servers are universal device drivers provided by each system vendor. The server is tailored for each device and hides how the server physically accesses its data. To access data, the server maintains a list of data or tag names. All that is required is knowledge of the device's naming convention. This allows client applications to access completely different devices without regard to device specifics. The figure shown below illustrates three servers connected to three different clients. Each client can access server data independent of the other clients while using the exact same OPC interfaces.



The OPC interface specification allows any OPC client to access any OPC server regardless of vendor and access data on the target device. This is true even if the device is a database, automation network, PLC, PMAC, or intelligent switch.

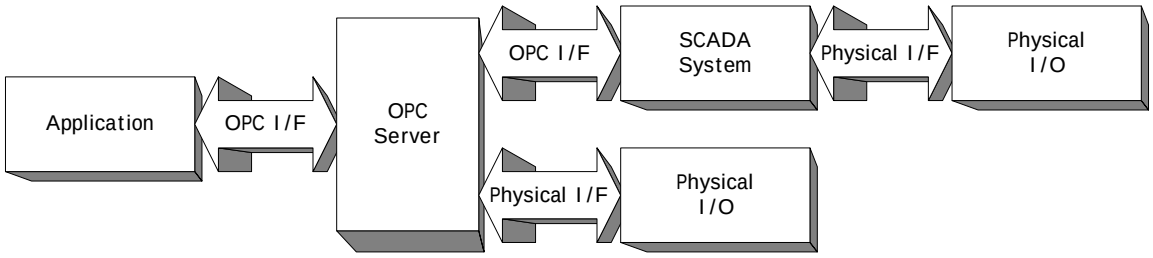
At its simplest level an OPC server is comprised of three types of objects: an OPC Server object, one or more OPC Group objects, and one or more OPC Item objects. The OPC Server maintains information about the server and functions as a container for collections of OPC Groups. Client applications connect to the server to query server status and define OPC Groups. Groups provide a container for logically organizing OPC Items, specifying item sampling rates, read/write access privileges, and other item characteristics. For example, a group might contain sets of items for a particular operator display, production report, or motion program. Groups may be public or local (private). Public groups can share items with multiple clients, whereas local groups maintain items for a specific client.

Within each Group the client defines one or more OPC Items. OPC Items are named connections to data within the server. Access to a specific OPC Item is provided by the OPC Group that "contains" the OPC Item. Associated with each item is a data value in the form of a VARIANT, a Quality flag indicating the 'goodness' of the data, and a Time Stamp indicating the time the data was collected. OPC Items should be thought of as paths specifying the address of the data, not the physical source of the data that the address references.



Where OPC Fits in the Software Architecture

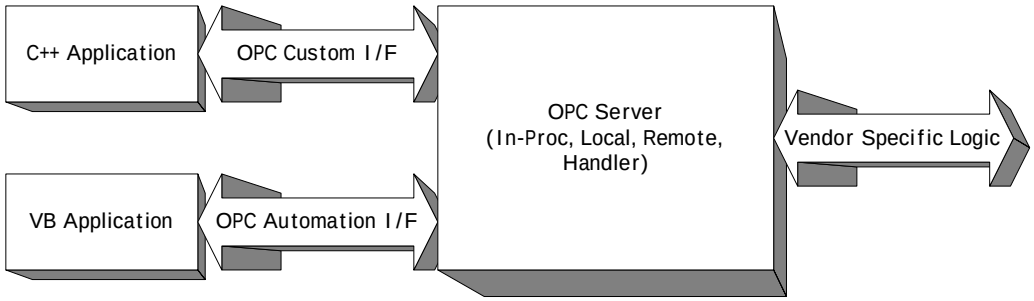
Although OPC was originally designed for accessing data from networked data servers, OPC interfaces are being used in many places in today's software environment. At the lowest level OPC servers are used to exchange data between a physical device and an application. The architecture and design also allow the construction of OPC Servers that access data from different OPC Servers from different devices running on different nodes. This includes application to application interfaces. Current enterprise software systems are being fielded that rely on OPC for database access, maintenance, alarming, MMI/SCADA, and general system integration at the level of ladder and program development and control.



Current developments at Delta Tau are underway to move PMAC-NC and PMAC Executive in this direction so that rather than directly talking to PMAC using PComm32 on the host computer the PMAC OPC Server will be used. In doing so a single PMAC-NC or PMAC Executive application will be able to communicate with other controllers or control multiple PMACs running in physically separate locations of different computers. Further enhancements to these Delta Tau tools will decompose them into COM compliant objects that will allow the developer to access specific capabilities from their own custom software.

OPC Components and Microsoft Windows COM Support

OPC is a specification for two sets of interfaces based on Microsoft's Component Object Method or COM technology. The specification defines how OPC Servers are expected to behave while providing data access for client applications. The vendor is free to implement the requirements in the most appropriate and efficient manner as dictated by their device's requirements. The definitions are based on an architecture that all participating vendors have agreed is generally appropriate for the automation environment. The figure below illustrates custom Visual C++ and Visual Basic applications accessing an OPC Server using the two interfaces to communicate with the device. Missing from the figure are Visual Java applications.



Aside from the obvious advantage of having universally defined interfaces for all OPC compliant devices, Microsoft's OLE/COM architecture provides tremendous operating system support for features like persistence, security, and remote access. OPC Servers, like all COM servers, can exist in three different flavors: In-Proc, Local, or Remote. This architecture allows client applications to access an OPC server as though it were:

- In-Proc - A simple DLL based driver dedicated to the client running on the local machine
- Local - A separate process executing on the local machine with all of the benefits of multiple client access, security, and memory protection
- Remote - A separate process executing on a remote machine accessed via TCP/IP

Even though these three flavors are hugely different in terms of the operating system support required to implement them, neither the OPC Server nor the OLC Client are required to know much about how this is done. Using Microsoft's OLE/COM architecture, OPC allows you tremendous flexibility in your system design without regard to the specifics of your system configuration. You can easily place different portions of your system architecture on physically different computers for any reason you can dream up and connect the entire system together using nothing more than your existing Ethernet.

So What?

This is where things get interesting. Lets say that your PMAC uses 3 coordinate systems comprised of 8 motors, 20 motion programs, and 10 PLC programs, each requiring a number of P, Q, and M variables to define specific motions, program characteristics, and machine operation. With PMAC's OPC Server you can define a group for each motion program and PLC so that you can set and query the program and machine parameters, a group for each coordinate system, so you can monitor its status and control its execution, and a group for each motor so that you can monitor and jog each motor individually. You might also define a few groups for specific sets of PMAC I/O that you wish to monitor or control from your application. Once the groups and items are created and registered the client application will automatically receive new item data from the server. This is true even if there are multiple PMACs scattered all over the plant.

PMAC OPC Command Line Interface

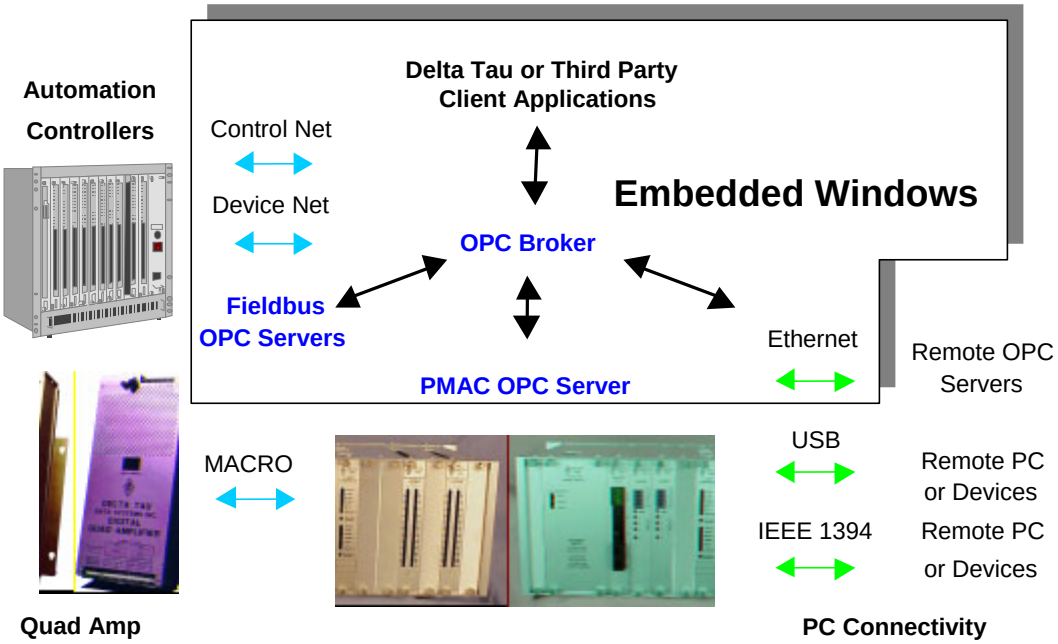
The flexibility of OPC allows us to define and utilize item data for any purpose. An example of this is a special PMAC OPC Server access path that allows you to send and receive command strings with the behavior you would expect from PMAC's ASCII command buffer capability. Using this particular item you can send a command string to PMAC and cause any PMAC command to be executed.

PMAC and Third Party Automation Software

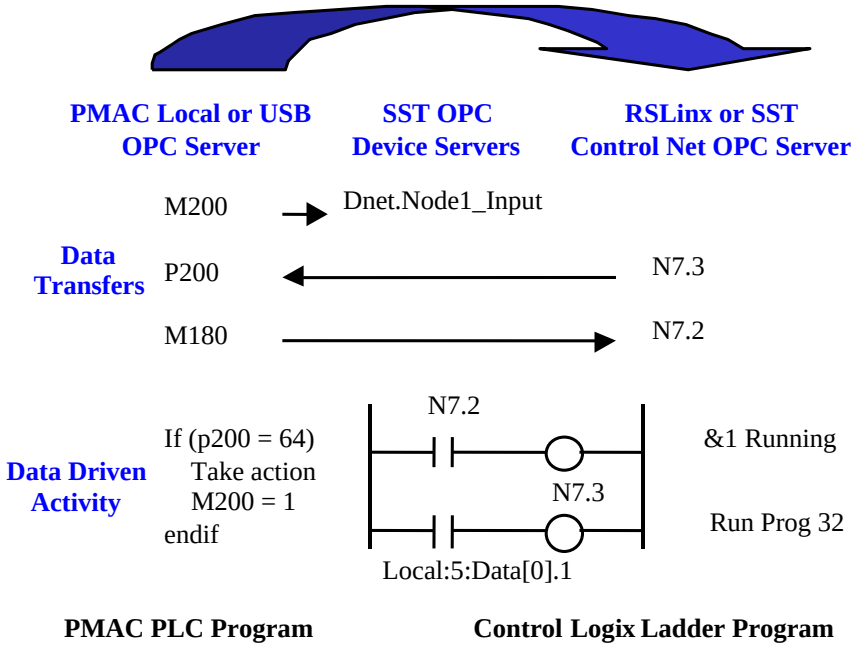
One of the driving forces behind Delta Tau's adoption of OPC was the need to couple PMAC into the existing base of enterprise and automation software systems from companies such as Intellution, Object Automation, National Instruments, Taylor-Waltz, Wonderware, Iconics. These systems provide a wide variety of logging, control, and MMI/SCADA capabilities that are based on OPC. Using PMAC OPC Server you can easily integrate PMAC into enterprises served by these systems.

PMAC OPC Broker and Automation Connectivity

One of Delta Tau's first and most important applications of the PMAC OPC Server is the development of an OPC Broker that allows OPC servers from multiple vendors to transfer data between one another. Using this approach, third party industrial network cards with OPC servers from companies such as SST and Synergetics can be configured to transfer data between an automation network and PMAC. This provides another approach for integrating PMAC directly into the plant controller environment using existing controller networks. For example, it is now easy to purchase a Control Net or Data Highway card from SST, place it in your PMAC 3U chassis, configure the SST OPC Server to service specific tags on an Allen Bradley Control Logix or PLC 5, and share data between the PLC and PMAC. You can now generate motion data in a PLC, send it to PMAC, start a motion program, and send PMAC I/O data back to the controller. The following figure demonstrates how this is accomplished. The wonderful truth is that this can be done using Profibus, Ethernet, DeviceNet, or any other supported network system.



The figure below illustrates the sharing of tags defined on a PMAC, Control Logix processor, and a Device Net node. This example, taken from a real implementation, demonstrates the flexibility of the OPC Broker approach and uses a combination of Device Net, Control Net, and Ethernet connecting several different systems. The DeviceNet and ControlNet communication interface cards were supplied by SST. The programs on each system are written in their native form - Ladder on the Control Logix and a PMAC PLC program. The data exchange between the two systems is transparent once the OPC Broker is configured.



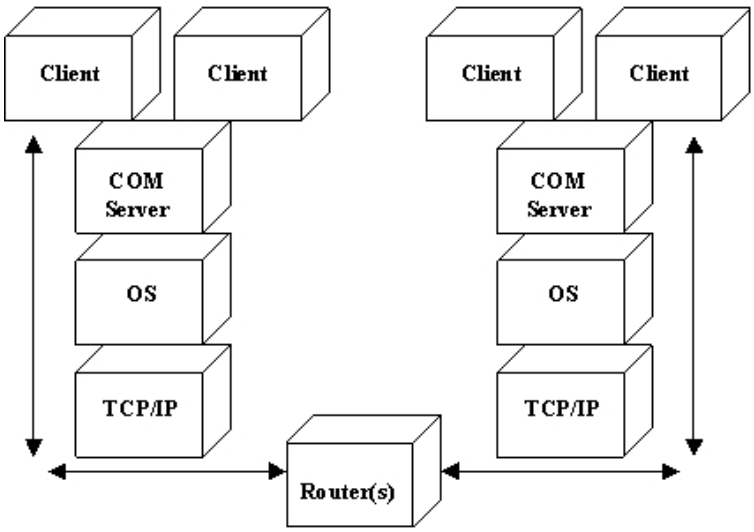
Performance of COM Servers and OPC

A common concern expressed by those newly initiated into the world of COM and OPC is: "What kind of throughput can I expect?" The answer to this depends on the COM architecture, your specific network and computer architecture, and the automation environment you are operating in. To address this question the user needs to understand a few things about the MS Windows operating systems, TCP/IP and internet, the COM architecture, and OPC. This does not include the design of a particular OPC Server or the device handled by the server. In this section we will discuss the architectural issues so that you can understand what factors impact performance and can ask the right questions of a systems' vendor. At the end of the section we provide some concrete performance numbers from a variety of sources including our own preliminary measurements. Keep in mind that there are extraordinary changes and enhancements on the horizon for some of the components listed. We will also outline these so that you can spot improvements as they are announced in the open literature.

Subsystems Involved in Distributed Components

The figure shown below illustrates the major pieces involved in any COM server architecture. The Client application or applications run as separate processes on the operating system and are each given an allotted amount of execution time to perform their tasks. COM clients use a set of OS provided capabilities to access a variety of COM Servers of which OPC Servers are an example. As previously noted, the COM Server can execute in one of three flavors: as an in-process DLL, a local server, or a remote server. Before discussing the remote case we'll consider the operation of in-process and local servers.

The COM architecture separates the client interface from the implementation of the service. In-process servers can in many ways be treated as classical DLLs. In this situation the only performance issue is that of implementing any multi-threading approaches. The situation gets a little more complex when the COM Server is a local server. In this instance the server receives execution time like any other process. When the client places a call to the local server the operating system packs all of the parameter associated with the call into a package in a process called marshalling and passes it to the server process. The server unmarshalls the package and then implements the service. Fortunately for all involved the marshalling process is transparent to the server and to the client. There is a slight performance penalty in this process but it is not substantial. The primary issue associated with this interprocess communication are the priorities and scheduling of the COM server process and the client. If the CPU is busy with lots of database processes, network routing, MMI software, file accesses, and other time consuming compute processes the time required to implement the client request can be substantial. It will also be non-deterministic. You cannot guarantee how fast the request will be serviced.



Things are more interesting if the server is a remote server. The package built during the marshalling process is passed by the operating system to its TCP/IP protocol stack and sent via the attached network to the receiving computer where it is routed to the proper COM server stubs, unmarshalled, and the service implemented. There are now some other factors that influence the time required for a server to implement the request of a client. The operating system must now schedule the TCP/IP stack to send the marshalled package. Depending on the traffic on the network, firewalls, and routers you have in your network this transmission can be on the order of a few hundred milliseconds. The same is true of the receiving computer, the message needs to be reassembled, routed to the proper server, unmarshalled, and sent to the client.

Don't get worried by all of this handling – it is all-transparent to the client and to the server. Furthermore, it's fast and getting much faster. Microsoft is constantly improving the OS components involved in handling the TCP/IP stack, COM servers, and the scheduling of processes. Windows CE is a perfect example of an OS improvement designed to move toward real-time system performance. Network configurations can also be modified to improve performance. Network traffic can be reduced by proper use of subnets, minimization of routers in the network path, the use of higher speed networks, and the use of readily available Ethernet switches. You also need to watch the development of TCP/IP. IP version six or IPv6 is designed to handle the demanding real-time requirements of voice and video over TCP/IP. One of its enhancements is RSVP – a bandwidth reservation protocol that allows a TCP/IP connection to reserve a specified amount of bandwidth over the network through the routers thereby guaranteeing the channel rate.

When considering the performance of COM remember that most of the MS Windows operating system is based on it. Everything from MS Word to Internet Explorer, from Real Audio to SQL Server. The push for enhanced performance is driven not by the needs of the industrial automation world but by everything from your future home entertainment system to your stockbroker's order entry system.

Early Generic COM Performance According to Microsoft

If you search the Microsoft Developer's Network website you can locate a paper detailing the performance of a generic COM application (msdn.microsoft.com/library/conf/html/SQ472.htm). The computer used to produce these numbers was a Dell 120 MHz Pentium running Windows NT 4.0 connected to each other over normally loaded 10 Mbps corporate network. The table below illustrates this published performance.

Parameter Size	4 bytes		50 bytes	
	Calls/sec	MS/call	Calls/sec	mS/call
In-process Server	3,224,816	0.00031	3,277,973	0.00031
Local Server	2,377	0.42	2,023	0.49
Remote Server	376	2.7	306	3.27

OPC Performance According to Rockwell Software

A more telling demonstration of OPC performance was released to the press on August 7, 1998 (www.software.rockwell.com/press/OPCrelease.cfm). The next few paragraphs presented here are taken verbatim from this release.

The results indicate that the amount of data that a server can serve to client applications whether the client is local or remote (using DCOM) exceeds the amount of data that realistically could be processed by a typical client application. In real world scenarios, using exception based notification; the amount of data that actually changes is typically a small percentage of the data that is being monitored.

From the start of OPC, there were those skeptics that doubted that a standard interface could be created and provide the level of performance of existing proprietary data exchange mechanisms.

The testing focused on making sure that the performance and throughput were deterministic (sustained and maintained over a period of time).

Tests were run where the client and the server were running on the same computer (called local), and tests were run with the client(s) and servers residing on different physical computers (called distributed or remote). We tested where the data type of an item was a single word, as well as an array of words.

The following section includes some examples of the throughput performance tests that were run to characterize the performance and throughput of OPC. Results of additional performance tests are available in an excel spreadsheet (RSOPCSUBPERF.xls), from Rockwell Software, or from the site you obtained this white paper.

To evaluate the performance of OPC in the distributed environment, we used six Pentium 266 computers, five computers running the client application, and one computer running the server application.

Each client application defined and added 10,000 items (AKA tags) to the server, and requested the server to update the client application at a rate of 250 milliseconds per item. We deliberately programmed the data source such that all the data was constantly changing (worst case scenario).

The result (see table below) was that the server was able to update two hundred thousand (200,000) items per second continuously.

	Items	**Changes / second**	**Items / second**
Client 1	10000	4	40,000
Client 2	10000	4	40,000
Client 3	10000	4	40,000
Client 4	10000	4	40,000
Client 5	10000	4	40,000
Server Total			200,000

For this test, each client application defined and added 100 items (AKA tags) to the server, with each item being 200 words of data. The data type for the item was safe array of 4-byte signed integers (VT_I4 || VT_ARRAY).

The client requested the server to update the client application at a rate of 70 milliseconds per item.

The result (see table below) was that the server was able to update one million four hundred thousand (1,400,000) words per second continuously.

	Items	**Words / Item**	**Changes / second**	**Items /second**	**Words / second**
Client 1	100	200	14	1400	280,000
Client 2	100	200	14	1400	280,000
Client 3	100	200	14	1400	280,000
Client 4	100	200	14	1400	280,000
Client 5	100	200	14	1400	280,000
Server Total				7000	1,400,000

To evaluate the performance of OPC on a single (local) computer we used a Pentium 266 computer, running the both the client application, and the server application.

The client application defined and added 10,000 items (AKA tags) to the server. The datatype of an item in this case was a 4byte-signed integer (VT_I4)

The client requested the server to update the client application at a rate of 200 milliseconds per item.

The result (see table) was that the server was able to update fifty thousand (50,000) items per second continuously.

	Items	**Changes / second**	**Items / second**
Client 1	10000	5	50,000

For this test, the client application defined and added 100 items (AKA tags) to the server, with each item being 500 words of data. The data type for the item was safe array of 4-byte signed integers (VT_I4 || VT_ARRAY).

The client requested the server to update the client application at a rate of 60 milliseconds per item.

The result (see table below) was that the server was able to update eight hundred thousand (800,000) words per second continuously.

	Items	Words / Item	Changes / second	Items /second	Words / second
Client	100	500	16	1600	800,000
Server Total					

OPC Performance According to Intellution and Fisher-Rosemont

Rather than belabor the point with more dry data we've located two other primary sources of public OPC performance data that you can check for yourself. One from Intellution, www.intellution.com/opchub/performance.asp, and one from Fisher-Rosemont www.easydeltav/public/product/white/opcperf.htm Both sources provide detailed data on configurations, conditions, and performance for a variety of servers and devices.

Early Stages of Delta Tau OPC Broker Performance

As noted in an earlier section Delta Tau is actively developing an OPC Broker whose primary purpose is to allow the system integrator to route messages from one OPC Server to another. This work also forms the basis for a multi-server client interface toolkit designed specifically for exciting new PMAC software products. In doing so you can develop systems that transparently transfer data between a PLC or other device on a proprietary automation fieldbus and PMAC. In the current configuration with the PMAC OPC Server and an SST Control Net fieldbus card talking with an Allen-Bradley Logix 5500 we routinely see single item transfers taking less than 1 mS with occasional burps due to network traffic and CPU usage that push this up to 10-20 mS. This configuration has been run continuously for weeks transferring tens of millions of individual data items between the PLC and PMAC. These items include motion program start commands, PMAC status, and I/O connected to the Logix 5500 that is used by PMAC to control program execution.

Universal Host Connectivity using USB and FireWire

OPC simplifies the development of device independent applications and provides remote connectivity via a variety of field busses and Ethernet. Delta Tau is extending the open connectivity notion further by adding this capability to two new PC connectivity standards: USB and FireWire. These two standards are revolutionizing the computer world by allowing hot-swap, plug and play, connectivity between a variety of devices. This is being done by natively hosting the USB and FireWire interface controllers on UMAC's UBUS. To utilize UMAC with an external PC you will no longer need to find a free I/O address, IRQ, or bus slot. These busses will allow real-time asynchronous and synchronous communication with UMAC at data rates ranging between 500 Kbyte/Sec and 40 Mbyte/Sec. You, the user, can anticipate new form factors for all UMAC hardware and software products based on their rugged 6 pin connectors and inexpensive cables ranging from 4 to 10 meters in length.

In this section we discuss UMAC's host software architecture and its implementation of USB and Ethernet. This new software architecture builds on the existing architecture by adding the new serial bus interfaces so your existing PComm32 applications not only work like they did before but also allow you to use the new serial busses. Furthermore, OPC client applications can use the new busses as well as your existing RS232, bus, or DPR enabled UMAC. UMAC's implementation of the USB, FireWire and Ethernet interfaces rely on its proven DPRAM architecture and a few intelligent interface devices. These devices handle all bus protocols while interpreting and generating host communication packets.

In sections 5 and 6 we will discuss the architecture of both USB and FireWire in more detail. In doing so we hope to dispel any misconceptions about their characteristics or their performance. You should view these two technologies as buses, rather than networks ("Long Skinny Busses" if you like!) where USB is equivalent to the present day ISA bus and Fire Wire is the equivalent of PCI. They both allow the user to attach or remove a device at any time with no configuration of the new device or currently connected devices. Furthermore, the performance of USB approaches that of the present day ISA bus and FireWire approaches that of PCI. Numerous support nodes and hubs exist to enable expansive system development. Several vendors, such as National Instruments, already supply a variety of remote I/O modules for USB.

There are also efforts to replace GPIB with FireWire thereby allowing access to tremendous numbers of laboratory quality measurement devices.

PMAC's Client Software Architecture

Existing host access to UMAC is provided by Delta Tau's 32 bit PComm32 device driver. It is the basis for PEWin32, PMAC-NC, PMACPlot, PMACPanel, and a host of third party applications. The driver is split into two pieces, a Ring 0 portion for accessing PMAC via the serial port, as an I/O device on the bus, or using PMAC's DPR, and a Ring 3 interface accessible by existing client applications. PMAC OPC server utilizes the Ring 0 portion of the existing driver to access in-chassis PMACs and provides a COM compliant Ring 3 interface that exposes the OPC interfaces to the client environment.

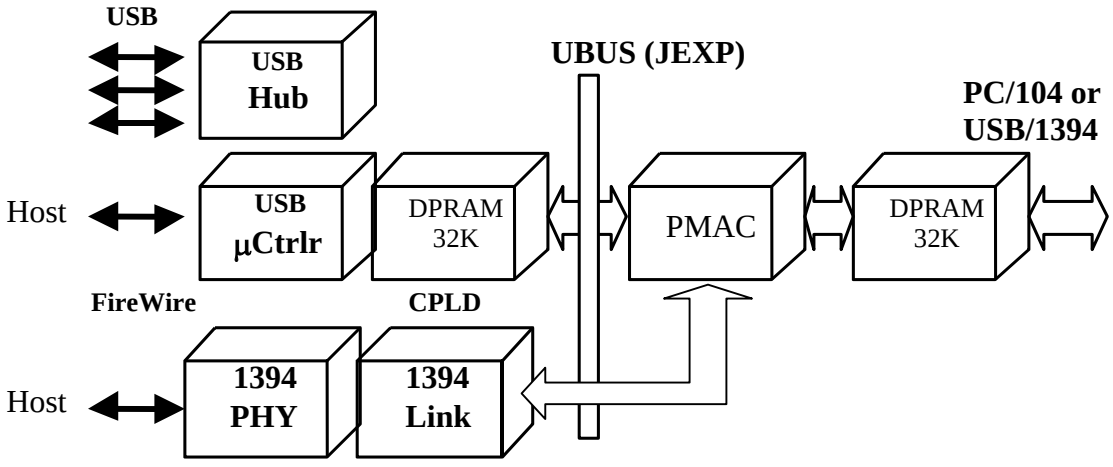
To implement USB connectivity Delta Tau has developed another Ring 0 component to provide connectivity to the PCs standard USB hub. This component handles all USB plug and play requirements and processes PMAC specific USB packets. A similar Ring 0 component is under development to support FireWire. Using this new Ring 0 component existing PComm32 applications can add USB to their list of communication mechanisms. Hence, PEWin can use the three existing communication modes of serial, bus, or DPR and now USB!

The implementation of PMAC OPC using USB is slightly different. To accomplish this, a separate OPC Server specifically for USB has been implemented. Because all OPC compliant interfaces must be identical OPC client's can use the USB specific version or the in-chassis route for PMAC's located in the host chassis.

Again, it is imperative that the reader understands that these new connectivity solutions allow tremendous flexibility. You can easily put one PMAC in the 3U chassis, connect another to the chassis using USB, and talk to both via Ethernet.

PMAC's Implementation of the USB and FireWire Interface

Silicon vendors are making huge strides developing commodity devices for both USB and FireWire. Delta Tau is taking full advantage of their investment by taking advantage of powerful yet low cost interface devices and coupling them to PMAC's proven DPRAM architecture. This minimizes critical timing issues and makes full use of firmware support for DPRAM. In doing this your existing PMAC programs can utilize USB and FireWire to replace the limitations of the ISA bus. The figure shown below illustrates this.



UMAC's interface to PC/104, USB, and FireWire make use of its proven DPRAM architecture. This minimizes changes to the PMAC firmware and your PMAC programs.

On the right is the standard PMAC to ISA bus DPRAM interface. PC/104 is essentially ISA on a new connector and as such does not require significant modification. On the left are the new additions required to handle USB and FireWire. The interface to the daughter card is done using UBUS (JEXP) and provides another page of DPRAM that mirrors existing DPRAM. The heart of the USB interface is an integrated

8251 micro-controller that has a USB port. The firmware for the interface is responsible for handling all USB plug-and-play requirements and is actually downloaded from the host when the device is inserted on the bus. The controller then receives requests and commands from the host and executes the required operations using DPRAM as the interface to PMAC. The addition of a multiport USB hub is a simple matter of using another device to provide 3 additional USB ports.

Our plans for FireWire are based on a similar architecture and will use the USB controller to handle the developing FireWire communication protocol. The FireWire devices are divided into physical (PHY) and link (LNK) components. A CPLD capable of handling the high-speed interface to DPRAM is being designed now.

UMAC Implementation

It is also possible to contemplate the possibility of using a complete PC-based solution for motion, with only splines being executed at the UMAC rack. This is a low cost approach to motion control and the communication interface card can receive trajectory points from the PC via USB, FireWire or Ethernet and using a low cost DSP, generate the necessary signals to the UBUS which can transform the UMAC rack into simple, lower cost motion platform. The UMAC rack can thus be used as a powerful stand-alone controller when the “3U Turbo PMAC2” is used, a MACRO Pack if the “MACRO CPU” is used, and finally, a PC driven motion platform to generate splines with data generated at the PC and sent via the USB, FireWire or Ethernet.

The user can make the UMAC as powerful as he wishes to make it and conversely; it can be the simplest of controllers being remotely operated by the PC.

The UMAC rack, because of its 3U format, is ideally suited for the CE environment.

USB

The incorporation of USB directly into PMAC makes it extremely easy to plug a remote PMAC into PC, laptop, or palmtop with a widely available $12 USB cable. You can now run PEWin on a laptop and plug in half a dozen PMAC's at anytime and begin working with them. Using USB for host connectivity and MACRO for distributed servo control your system can be configured to operate in extremely demanding applications.

In this section we discuss the characteristics of USB as defined by its specification. This will allow you to understand how it operates and what its capabilities are. At the end of the section we will describe PMAC's USB hub implementation.

General

The lack of flexibility in reconfiguring the PC has been acknowledged as the Achilles' heel to its further deployment. The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the addition of external peripherals continues to be constrained by port availability and difficulty configuring software services. The lack of a bi-directional, low-cost, low-to-mid speed peripheral bus has held back the creative proliferation of peripherals in all areas including automation. Existing interconnects are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need. For example, a Mouse port, a Keyboard port, a printer port, and on and on. The USB is one answer to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface that is consistent with the requirements of the PC platform of today and tomorrow.

Taxonomy of Application Space

Figure 3-1 is taxonomy showing the range of data traffic loads that can be serviced over a USB. As can be seen, a 12Mb/s bus spans the mid-speed and low-speed data ranges. Typically, mid-speed data types are isochronous, while low-speed data typically comes from interactive or I/O devices. The software architecture allows for future extension of the USB by providing support for multiple USB Host Controllers.

Bus Topology

The USB connects USB devices with the USB host. The USB physical interconnect is a tiered star topology. A hub is at the center of each star. Each wire segment is a point-to-point connection between the host and a hub or function, or a hub connected to another hub or function.

Bus Protocol

The USB is a polled bus. The Host Controller initiates all data transfers. All bus transactions involve the transmission of up to three packets. Each transaction begins when the Host Controller, on a scheduled basis, sends a USB packet describing the type and direction of transaction, the USB device address, and endpoint number. This packet is referred to as the "token packet." The USB device that is addressed selects itself by decoding the appropriate address fields. In a given transaction, data is transferred either from the host to a device or from a device to the host. The direction of data transfer is specified in the token packet. The source of the transaction then sends a data packet or indicates it has no data to transfer. The destination, in general, responds with a handshake packet indicating whether the transfer was successful. The USB data transfer model between a source or destination on the host and an endpoint on a device is referred to as a pipe.

There are two types of pipes: stream and message. Stream data has no USB-defined structure, while message data does. Additionally, pipes have associations of data bandwidth, transfer service type, and endpoint characteristics like directionality and buffer sizes. Most pipes come into existence when a USB device is configured. One message pipe, the Default Control Pipe, always exists once a device is powered, in order to provide access to the device's configuration, status, and control information. The transaction schedule allows flow control for some stream pipes. At the hardware level, this prevents buffers from underrun or overrun situations by using a NAK handshake to throttle the data rate. When NAKed, a

transaction is retried when bus time is available. The flow control mechanism permits the construction of flexible schedules that accommodate concurrent servicing of a heterogeneous mix of stream pipes. Thus, multiple stream pipes can be serviced at different intervals and with packets of different sizes.

System Configuration

The USB supports USB devices attaching to and detaching from the USB at any time. Consequently, system software must accommodate dynamic changes in the physical bus topology. All USB devices attach to the USB through ports on specialized USB devices known as hubs. Hubs have status indicators that indicate the attachment or removal of a USB device on one of its ports. The host queries the hub to retrieve these indicators. In the case of an attachment, the host enables the port and addresses the USB device through the device's control pipe at the default address. The host assigns a unique USB address the to the device and then determines if the newly attached USB device is a hub or a function The host establishes its end of the control pipe for the USB device using the assigned USB address and endpoint number zero. If the attached USB device is a hub and USB devices are attached to its ports, then the above procedure is followed for each of the attached USB devices. If the attached USB device is a function, then attachment notifications will be handled by host software that is appropriate for the function.

Data Flow Types

The USB supports functional data and control exchange between the USB host and a USB device as a set of either uni-directional or bi-directional pipes. USB data transfers take place between host software and a particular endpoint on a USB device. Such associations between the host software and a USB device endpoint are called pipes. In general, data movement though one pipe is independent from the data flow in any other pipe. A given USB device may have many pipes. As an example, a given USB device could have an endpoint that supports a pipe for transporting data to the USB device and another endpoint that supports a pipe for transporting data from the USB device.

The USB architecture uses four basic types of data transfers on a pipe:

Control Transfers: Control data is used by the USB System Software to configure devices when they are first attached. Other driver software can choose to use control transfers in implementation-specific ways. Data delivery is lossless.

Bulk Data Transfers: Bulk data typically consists of larger amounts of data, such as that used for printers or scanners. Bulk data is sequential. Reliable exchange of data is ensured at the hardware level by using error detection in hardware and invoking a limited number of retries in hardware. Also, the bandwidth taken up by bulk data can vary, depending on other bus activities.

Interrupt Data Transfers: A small, limited-latency transfer to or from a device is referred to as interrupt data. Such data may be presented for transfer by a device at any time and is delivered by the USB at a rate no slower than is specified by the device. Interrupt data typically consists of event notification, characters, or coordinates that are organized as one or more bytes. An example of interrupt data is the coordinates from a pointing device. Although an explicit timing rate is not required, interactive data may have response time bounds that the USB must support.

Isochronous Data Transfers: Isochronous data (Also called streaming real time transfers) is continuous and real-time in creation, delivery, and consumption. Timing-related information is implied by the steady rate at which isochronous data is received and transferred. Isochronous data must be delivered at the rate received to maintain its timing. In addition to delivery rate, isochronous data may also be sensitive to delivery delays. For isochronous pipes, the bandwidth required is typically based upon the sampling characteristics of the associated function. The latency required is related to the buffering available at each endpoint. A typical example of isochronous data is voice. If the delivery rate of these data streams is not maintained, drop-outs in the data stream will occur due to buffer or frame underruns or overruns. Even if data is delivered at the appropriate rate by USB hardware, delivery delays introduced by software may degrade applications requiring real-time turn-around, such as telephony-based audio conferencing. The

timely delivery of isochronous data is ensured at the expense of potential transient losses in the data stream. In other words, any error in electrical transmission is not corrected by hardware mechanisms such as retries. In practice, the core bit error rate of the USB is expected to be small enough not to be an issue. USB isochronous data streams are allocated a dedicated portion of USB bandwidth to ensure that data can be delivered at the desired rate. The USB is also designed for minimal delay of isochronous data transfers

Allocating USB Bandwidth

USB bandwidth is allocated among pipes. The USB allocates bandwidth for some pipes when a pipe is established. USB devices are required to provide some buffering of data. It is assumed that USB devices requiring more bandwidth are capable of providing larger buffers. The goal for the USB architecture is to ensure that buffering-induced hardware delay is bounded to within a few milliseconds.

The USB's bandwidth capacity can be allocated among many different data streams. This allows a wide range of devices to be attached to the USB. For example, telephony devices ranging from 1B+D all the way up to T1 capacity can be accommodated. Further, different device bit rates, with a wide dynamic range, can be concurrently supported. The USB Specification defines the rules for how each transfer type is allowed access to the bus.

PMAC's USB Hub Implementation

USB Hubs are commercially available and inexpensive. Most Hubs have up to 3 each A type receptacles.

Allowable length between Hubs is a maximum of 6 meters (18 ft) and up to 6 Hubs can be daisy chained for a maximum distance of 30 meters (5 each, 6 meter segments).

FireWire IEEE-1394 High Speed Serial Bus

In this section, we discuss the characteristics of FireWire as defined by its public specification. This will allow you to understand how it operates and what its capabilities are.

General

IEEE-1394 (or FireWire) is a high-speed, low-cost serial bus suitable for use as a peripheral bus or a backup to parallel back-plane bus. At a basic level it can be viewed as a very high speed USB capable of delivering data at rates suitable for digital video. Highlights of the serial bus include:

Automatic assignment of node addresses - no need for address switches.

Variable speed data transmission based on ISDN-compatible 1 bit rates from 24.576 Mbit/s for TTL backplanes to 49.152 Mbit/s for BTL backplanes to 98.304, 196.608, and 393.216 Mbit/s for the cable medium.

The cable medium allows up to 16 physical connections (cable hops) each of up to 4.5 meters, giving a total cable distance of 72 meters between any two devices. Bus management recognizes smaller configurations to optimize performance.

Bus transactions that include both block and single quadlet reads and writes, as well as an "isochronous" mode which provides a low-overhead guaranteed bandwidth service.

A fair bus access mechanism that guarantees all nodes equal access. The backplane environment adds a priority mechanism, but one that ensures that nodes using the fair protocol are still guaranteed at least partial access.

Serial Bus Applications

There are four main reasons for providing a serial bus on a system that already has a parallel bus:

The many modules that make up a system might operate on different backplane bus standards, yet they need to work together.

Although located within the same enclosure, the system is too large or physically disperse to use a single backplane, yet modules in the different backplanes must communicate.

One or more modules of a system are located neither on the same backplane nor within the same enclosure.

Many system modules are price-sensitive and do not need the full bandwidth of a parallel bus.

The Serial Bus can also be used as a powerful and low cost peripheral interconnect. The compact Serial Bus cable and connector allow bandwidths comparable with existing I/O interconnect standards. The Serial Bus has the added advantage of architectural compatibility with parallel computer buses; this leads to lower communications overhead than limited-function dedicated I/O interconnects.

Bus Bridge

The Serial Bus architecture limits the number of nodes on any bus to 63, but supports multiple bus systems via bus bridges. The CSR Architecture defines the Serial Bus addressing structure, which allows almost 2 16 nodes. A bus bridge normally eavesdrops on the bus, ignoring all transactions between local addresses but listening carefully for remote transactions. When the bridge receives a packet for a remote address, it forwards the packet to an adjacent bus. After initialization, the bus bridges are transparent to the system.

Although the Serial Bus may be used in many bus configurations, when used to bridge CSR-Architecture-compliant busses it is expected to be used mostly in a hierarchical bus fashion, as illustrated in figure 1-1 below, where bus #5 is a Serial Bus and bridges together other CSR-Architecture-compliant buses 1 through 4; #1 could be IEEE Std 896 Future-Bus+ 5 , #2 could be IEEE Std 1596 Scalable Coherent Interface, and so on.

Service Model

This specification uses a protocol model with multiple layers. Each layer provides services to communicate with the next higher layer and with the Serial Bus management layer. These services are abstractions of a possible implementation; an actual implementation may be significantly different and still meet all the requirements. The method by which these services are communicated between the layers is not defined by this specification. Four types of service are defined by this standard:

- Request service. A request service is a communication from the higher layer to the lower layer to request the lower layer to perform some action. A request may also communicate parameters that may or may not be associated with an action. A request may or may not be confirmed by a confirmation. A data transfer request usually will trigger a corresponding indication on a peer node(s). (Since broadcast addressing is supported on the Serial Bus, it is possible for the request to trigger a corresponding indication on multiple nodes.)
- Indication service. An indication service is a communication from the lower layer to the upper layer to indicate a change of state or other event detected by the lower layer. An indication may also communicate parameters that are associated with the change of state or event. An indication may or may not be responded to by a response. A Data transfer indication is originally caused by corresponding requests on a peer node.
- Response service. A response service is a communication from the higher layer to the lower layer to respond to an indication. A response may also communicate parameters that indicate the type of response to the indication. A response is always associated with an indication. A data transfer response usually will trigger a corresponding confirmation on a peer node.

- Confirmation service. A confirmation service is a communication from the lower layer to the upper layer to confirm a request service. A confirmation may also communicate parameters that indicate the completion status of the request, or any other status. A confirmation is always associated with a request. For data transfer requests, the confirmation may be caused by a corresponding response on a peer node. If all four service types exist, they are related as shown by the following figure:

What IEEE-1394 Means to PMAC

The ability to provide high-speed connectivity between a CPU and peripheral device of any type has only been achieved in the past using a local parallel bus located on the backplane. There are usually more devices than slot, in general you cannot plug a device into the bus while the system is operating or hot, and resolution of address conflicts can be troublesome. Furthermore, simple changes to the architecture such as adding a new device can take days and require numerous software changes.

Using FireWire most of these problems are solved. To date, high-speed connectivity to a remote computer required a network with all of its associated software protocols, network delays, and configuration problems. FireWire is a bus - not a network. PMACWire places PMAC's DPR on the bus.

Node and Module Architectures

The Serial Bus architecture is defined in terms of nodes. A node is an addressable entity, which can be independently reset and identified. More than one node may reside on a single module, and more than one unit may reside in a single node, as illustrated in figure 3-1.

Each module consists of one or more nodes, which are independently initialized and configured. Note that modules are a physical packaging concept and nodes are a logical addressing concept. A module is a physical device, consisting of one or more nodes that share a physical interface. In normal operation, a module is not visible to software. Of course, this is not true when the module is replaced, when the shared bus interface fails, or when specialized module-specific diagnostic software is invoked. A node is a logical entity with a unique address. It provides an identification ROM and a standardized set of control registers and it can be reset independently. The address space provided by a node can be directly mapped to one or more units. A unit is a logical entity, such as a disk controller, which corresponds to unique I/O driver software. On a multifunction node, for example, the processor and I/O interfaces could be different units on the same node. Unless the node is reset or reconfigured, the I/O driver software for each unit can operate independently. A unit may be defined by a unit architecture which describes the format and function of the unit's software-visible registers. Within a unit there may be multiple subunits, which can be accessed through independent control registers or uniquely addressed DMA-command sequences. Although unit architectures should use the subunit concept to simplify I/O driver software, the definition of subunit architectures is beyond the scope of this standard.

Cable Environment

The physical topology for the cable environment is a non-cyclic network with finite branches and extent. "Non-cyclic" means that closed loops are unsupported. The medium consists of 2 conductor pairs for signals and one pair for power and ground which connect together ports on different nodes. Each port consists of terminators, transceivers and simple logic. The cable and ports act as bus repeaters between the nodes to simulate a single logical bus. Since each node must continuously repeat bus signals, the separate power and ground wires within the cable enable the physical layer of each node to remain operational even if node local power is off. The pair can even power an entire node if its requirements are modest. The Serial Bus cable provides 8 to 40 VDC at up to 1.5 A. The actual current available is system-dependent.

Developments are underway to provide IEEE-1394 connectivity over standard network CAT-5 cabling with distances as long as 100 meters.

Addressing

The Serial Bus follows the CSR Architecture for 64-bit fixed addressing, where the upper 16 bits of each address represent the node_ID. This allows address space for up to 64k nodes. The Serial Bus divides the

node_ID into two smaller fields: the higher order 10 bits specify a bus_ID and lower order 6 bits specify a physical_ID. Each of the fields reserves the value of all "1"s for special purposes, so this addressing scheme provides for 1023 buses each with 63 independently addressable nodes. This standardization is continued within the node, with 2^48 bytes (256 terabytes) divided between initial register space (2048 bytes reserved for core CSR Architecture resources, registers specific to the Serial Bus, and the first 1024 bytes of a ROM ID area), initial units space (area reserved for node-specific resources), private space (area reserved for node-local uses) and initial memory space. Clause 4 of the CSR Architecture defines these terms in more detail. Figure 3-3 illustrates the address structure of the Serial Bus.

When practical, the node should use only the first 2048 bytes of the initial unit's space. This simplifies the design of heterogeneous bus systems, since the 2048 bytes of standard CSRs and ROM and first 2048 bytes of the initial units space use up the 4096 bytes allocated to CSR space on buses using the CSR-Architecture-defined 32-bit extended addressing (Futurebus+ has several profiles that use 32-bit extended addressing).

Subaction Queue Independence

For the split-response Serial Bus design model, separate and (effectively) independent queues exist for incoming/outgoing transaction requests and responses. For a simple responder which never explicitly generates read, write, or lock transactions, two queues are sufficient: one for incoming transaction requests and one for outgoing transaction responses, as illustrated in figure 3-7. These queues are independent, in the sense that the sending of response-queue subactions isn't affected by the retry state of transaction request-queue packets (see clause 3.6.2.4 for a discussion of retries). Of

- An asynchronous subaction - a variable amount of data and several bytes of transaction layer information are transferred to an explicit address and an acknowledgement is returned.
- An isochronous subaction or "channel" - a variable amount of data is transferred on regular intervals with simplified addressing and no acknowledge.

The subaction has three possible parts:

1. Arbitration sequence - a node that wishes to transmit a packet requests the physical layer to gain control of the bus. The physical layer may respond immediately if the node already controls the bus (as it would be if the subaction is the transaction response corresponding to the immediately preceeding acknowledge).
2. Data packet transmission - for asynchronous subactions, the source node sends a data prefix signal (including a speed code, if needed), addresses of the source and destination nodes, a transaction code, a transaction label, a retry code, data, one or two CRCs and a packet termination (either another data prefix or a data end signal). Isochronous subactions include a short channel identifier rather than source or destination addresses and do not have the transaction label or retry code.
3. Acknowledgment - a uniquely addressed destination returns a code indicating the action taken by the packet receiver. Isochronous packets and asynchronous broadcast packets do not have acknowledgments. Acknowledgments are also preceded by a data prefix and terminated by another data prefix or a data end.

All asynchronous subactions are normally separated by periods of idle bus called "subaction gaps" as shown in figure 3- 8. Note that a gap opens up on the bus between the packet transmission and ack reception. This "ack gap" is of varying lengths depending on where the receiver is on the bus with respect to the senders of the link request and ack. However, the maximum length of the ack gap is sufficiently shorter than a subaction gap to ensure that other nodes on the bus will not begin arbitration before the acknowledge has been received. Similarly, isochronous subactions are separated by periods of idle bus called "isoch gaps" as shown in figure 3-8.

Conclusion

Delta Tau is aggressively pursuing each of the technologies outlined in this white paper. The 3U Chassis with embedded MS Windows, OPC Server, and USB interface are undergoing extensive testing today. The OPC Broker is under development and has been publicly demonstrated. The FireWire interface is being designed today and will integrate MACRO, USB, and FireWire on a single PMAC expansion card.

The PMAC of tomorrow is being built as you read this paper. You can expect extremely flexible PMACs capable of connecting to most automation environments at the turn of the century!

APPENDIX – FIELDBUS CHARACTERISTICS

The information contained below is taken from the Synergetics web site at www.synergetic.com.

Fieldbus Background Information				
	Technology Developer	Year Introduced	Governing Standard	Openness
PROFIBUS DP/PA	PTO	DP-1994, PA-1995	DIN 19245 part 3/4	Products from over 150 vendors
INTERBUS-S	Phoenix Contact	1984	DIN 19258	Products from over 400 manufacturers
DeviceNet	Allen-Bradley	March 1994	ISO 11898 &11519	6 chip vendors, 100+ products
ARCNET	Datapoint/ SMC	1975	ANSI 878	Chips, boards, ANSI docs
AS-I	AS-I Consortium	Fall 1993	Submitted to IEC	AS-II.C. Market item
Fieldbus Foundation	Fieldbus Foundation	1995	ISA SP50/IEC TC65	Chips/software from multiple vendors
IEC/ISA SP50 Fieldbus	ISA & Fieldbus F.	1992 - 1996	IEC 1158/ANSI 850	Multiple chip vendors
Seriplex	APC, Inc.	1990	Seriplex spec	Chips available multiple interfaces
WorldFIP	WorldFIP	1988	IEC 1158-2	Multiple chip vendors
LonWorks	Echolon Corp.	March 1991	ASHRAE of BACnet	Public documentation on protocol
SDS	Honeywell	Jan., 1994	Honeywell Specification, Submitted to IEC, ISO11989	6chip vendors, 200+ products
MACRO	Delta Tau Data Systems	1997	FDDI	Chips, boards, documents

Physical Characteristics				
	Network Topology	Physical Media	Max. Devices (nodes)	Max. Distance
PROFIBUS DP/PA	Line, star & ring	Twisted-pair or fiber	127 nodes	24 Km (fiber)
INTERBUS-S	Segmented with "T" drops	Twisted-pair, fiber, and slip-ring	256 nodes	400 m/segment, 12.8 Km total
DeviceNet	Trunkline/dropline with branching	Twisted-pair for signal & power	64 nodes	500m
ARCNET	Bus, multidrop, star	Twisted-pair coax fiber	255 nodes	5 miles
AS-I	Bus, ring, tree star, of al	Two wire cable	31 slaves	100 meters, 300 with repeater
Fieldbus Foundation	Multidrop with bus powered devices	Twisted-pair	240/segment, 65,000 segments	1900m @ 31.25K 500m @ 2.5M
IEC/ISA SP50 Fieldbus	Star or bus	Twisted-pair fiber, and radio	IS 3-7 non IS 128	1700m @ 31.25K 500M @ 5Mbps
Seriplex	Tree, loop, ring, multi-drop, star	4-wire shrolded cable	500+ devices	500+ ft
WorldFIP	Bus	Twisted-pair, fiber	256 nodes	up to 40 Km
LonWorks	Bus, ring, loop, star	Twisted-pair, fiber, power line	32,000/domain	2000m @ 78 kbps
SDS	Turnkline/Dropline	Twisted-pair for signal & power	64 nodes, 126 addresses	500m
MACRO	Ring	RJ-45/and/or Fiber Pair	256 nodes	100 ft RJ45/10,000ft

Transport Mechanism						
	Communication Methods	Transmission Properties	Data Transfer Size	Arbitration Method	Error Checking	Diagnostics
PROFIBUS DP/PA	Master/slave peer to peer	DP up to 12 Mbps PA 31.25 kbps	244 bytes	Token passing	HD4 CRC	Station, module & channel diagnostics
INTERBUS-S	Master/slave with total frame transfer	500kBits/s, full duplex	512 bytes h.s., unlimited block	None	16-bit CRC	Segment location of CRC error and cable break
Device Net	Master/slave, multi-master, others	500 kbps, 250 kbps, 125 kbps	8-byte variable message	Carrier-Sonac Multiple Access	CRC check	Bus monitoring
ARCNET	Peer to peer	31.25K to 10M	508 bytes	Token	16-bit CRC	Built in
AS-I	Master/slave with cyclic polling	Data and power, EMI resistant	31 slaves with 4 in and 4 out	Master/slave with cyclic polling	Manchester Code, hamming-2, partly	Slave fault, device fault
Fieldbus Foundation	Client/server publisher/ subscriber, Event notification	31.25 kbps 1 Mbps 2.5Mbps	16.6 M objects/device	Deterministic centralized scheduler, multiple backup	16-bit CRC	Remote diagnostics, network monitors, parameter status
IEC/ISA SP50 Fieldbus	Client/server Publisher/ subscriber	31.25 kbps IS+1, 2.6, 5 Mbps	64 octets high & 256 low priority	Scheduler, tokens, or master	16-bit CRC	Configurable on network management
Seriplex	Master/slave peer to peer	200 Mbps	7680/transfer	Sonal multiplexing	End of frame & echo check	Cabling problems
WorldFIP	Peer to peer	31.25 kbps, 1 & 2.5 Mbps, 6 Mbps fiber	No limit, variables 128 bytes	Central arbitration	16-bit CRC, data "freshness" indicator	Device message time-out, redundant cabling
LonWorks	Master/slave peer to peer	1.25 Mbs full duplex	228 bytes	Carrier Sense, Multiple Access	16-bit CRC	Database of CRC errors and device errors
SDS	Master/slave, Peer to peer, Multi-cast, Multi-master	1Mbps, 500 kbps, 250 kbps, 125 kbps	8-byte variable message	Carrier-Sonac Multiple Access	CRC check	Bus montoring, Diagnostic slave
MACRO	Master/slave, Master/master, Multi-master	125Mbps	96 bit fixed token passing	Serial sequential addressing	CRC check 4B/5B check	Error data base Auto ring- Break detect

Performance			
	Cycle Time: 256 Discrete 16 nodes with 16 I/Os	Cycle Time: 128 Analog 16 nodes with 8 I/Os	Block transfer of 128 bytes 1 node
PROFIBUS DP/PA	Configuration dependent typ <2ms	Configuration dependent type <2ms	not available
INTERBUS-S	1.8 ms	7.4 ms	140 ms
DeviceNet	2.0 ms Master-slave polling	10 ms Master-slave polling	4.2 ms
ARCNET	<2 ms @ 2.5M	<2 ms @ 2.5M	<2 ms @ 2.5M
AS-I	4.7 ms	Not possible	not possible
Fieldbus Foundation	100 ms @ 31.25k <1 ms @ 2.5M	600 ms @ 31.25k <8 ms @ 2.5M	36 ms @ 31.25k 0.45 ms @ 2.5M
IEC/ISA SP50	Configuration dependent	Configuration dependent	0.2 ms @ 5 Mbps 1.0 ms @ 1 Mbps
Seriplex	1.32 ms @ 200 kbps, m/s	10.4 ms	10.4 ms
WorldFIP	2 ms @ 1 Mbps	5 ms @ 1 Mbps	5 ms @ 1 Mbps
LonWorks	20 ms	5 ms @ 1 Mbps	5 ms @ 1 Mbps
SDS	<1 ms, event driven	<1 ms per event	2 ms @ 1 Mbps
MACRO	12.31 nX (1 ring cycle)	Config dependent approx. 200 Msecs	1 Master, 1 Slave 46.6 nS

Index

S

T

U