

# I<sub>2</sub>O Message-Passing Protocol with PCI Bus Extensions

An Intelligent I/O (I<sub>2</sub>O) message passing protocol specification with PCI bus extension is now available for embedded designs. This presentation will show how I<sub>2</sub>O building blocks, including the I<sub>2</sub>O real-time operating system (RTOS), device driver modules (DDMs), and host operating system modules (OSMs) can be used to create a variety of I/O intensive applications including ADSL central office DSLAMs, Web Servers, and real-time applications

## INTRODUCTION

This paper explores the Intelligent I/O (I<sub>2</sub>O) messaging passing protocol, with specific regard to PCI, and looks at how I<sub>2</sub>O can be valuable to the embedded system designer.

Some of the problems with adding I/O support to an embedded design are explored, and the advantages of an I<sub>2</sub>O solution are discussed. The components of an I<sub>2</sub>O system are investigated, and then it is shown how these components can be combined and organised to create a number of different advanced I/O intensive real-time applications.

## LIMITATIONS OF EXISTING DRIVER MODELS

Consider the problems with adding an Ethernet interface to your design for an embedded product. As a designer, you may not have specialised knowledge of the Ethernet interface, know how to design a LAN driver for performance or functionality (adding multicasting support, for example), or know which transceiver or front-end to use.

In the past, many embedded systems designers have had to acquire these sorts of skills to add standard I/O functionality to their products, even though these components are tangential to their core competencies.

Often the designer can spend a disproportionate amount of time developing these components rather than concentrating on the main functionality of their design, which increases the time-to-market of the product. The problem is further exacerbated as technology improves and a company can find itself facing any number of lifecycle issues, such as being locked into using a depreciated part, or having a product based on outdated technology. In these cases it may be difficult or impossible to change or re-engineer the product without a complete redesign.

With the mass market acceptance of PCI, and the proliferation of low cost CPU-PCI bus interfaces and PCI I/O chipsets, embedded system designers can now use off-the-shelf I/O adapters (for example, SCSI, Ethernet and RAID controllers). Perhaps more importantly, the ground has shifted, and whereas the designer used to be able to write a driver based on publicly available chip datasheets, he must now, typically, rely on the adapter vendor, or RTOS vendor to supply the appropriate driver.

The full advantage of the PCI revolution remain elusive to the embedded systems industry because PCI adapter vendors' primary market is the PC, and the bulk of their development expenditure and energy is focussed around Windows. I<sub>2</sub>O provides an opportuni-

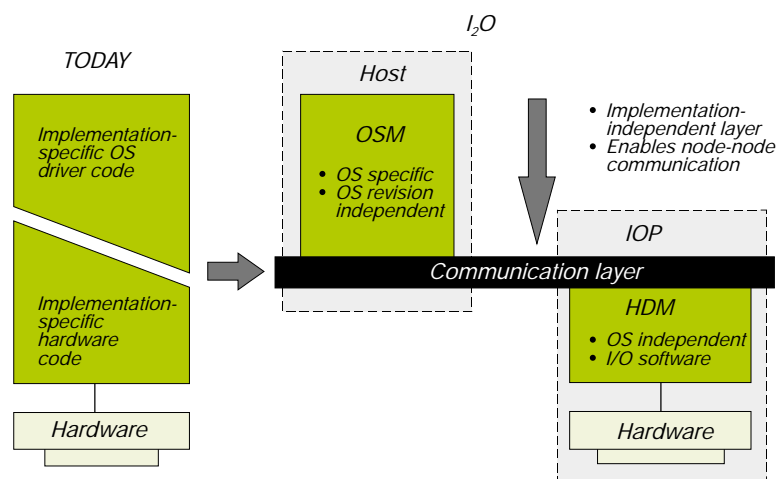


Figure 1. I<sub>2</sub>O Split Driver Model.

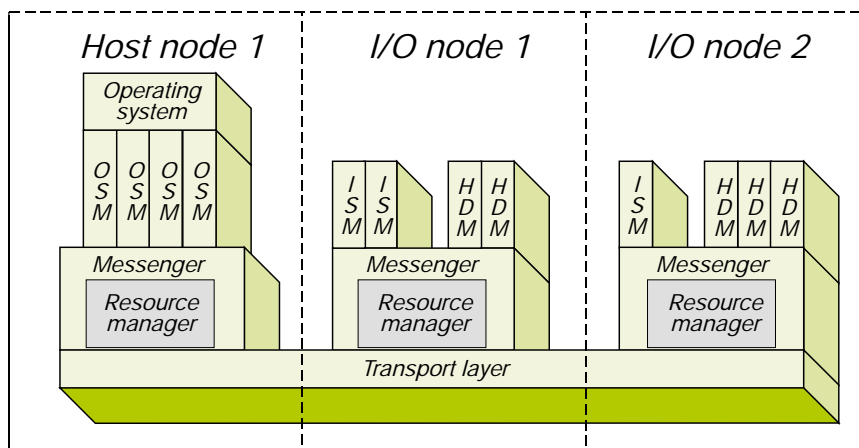


Figure 2. I<sub>2</sub>O Software Architecture.

ty for the Embedded Systems market to profit from the technological improvements and economies of scale of the mass PCI I/O adapter market.

**I<sub>2</sub>O ADVANTAGES**

I<sub>2</sub>O works by off-loading the I/O processing part of a product to a separate I/O processing subsystem, and controls this subsystem by using a standards-based set of message classes. As a result, I<sub>2</sub>O provides a number of advantages over the existing driver model:

By off-loading the host CPU, system throughput is greatly improved. The host can concentrate on non-I/O based processing, and it can reduce the effect of cache misses and pipeline flushes, etc. due to a reduced interrupt load. (A single server, for example, can support multiple I/O devices without running out of CPU cycles).

Message passing provides a natural split between the production and consumption of messages describing the I/O work that needs doing. This lets the host and the IOP work asynchronously from each other, accommodates the bursty nature of I/O transactions, as well as providing natural flow control.

The I/O system becomes more scalable, and can be designed and upgraded independently to the main CPU system. As I/O system performance is increased, additional processing can be off-loaded, such as RAID support or the TCP/IP network protocol.

Engineers can concentrate on the specifics of the system they are trying to design without having to learn how to build and design I/O systems.

Standards-based messaging and downloadable driver interfaces provide an easy way for vendors to supply value-add.

Embedded systems designers can utilise adapter drivers originally written for the PC marketplace and no longer have to rely on ports of drivers specifically for the embedded RTOS they choose to use. In addition, they are no longer locked into the I/O adapter they chose to use at the beginning of the design cycle.

Adapter vendors, Operating System vendors and driver writers no longer need to cope with the huge inter-operability matrix between multiple OSEs and adapters. OSVs write one driver per I/O class, and then forget about the problem. IHVs write one driver for each adapter they make, and then forget about the problem.

I<sub>2</sub>O messaging is independent of the bus technology. A product can be redesigned to use Future I/O rather than PCI without having to change the user's application code.

It is easy to port drivers to a new architecture.

**I<sub>2</sub>O COMPONENTS**

I<sub>2</sub>O provides a standards-based messaging passing protocol to off-load I/O processing from the primary host CPU to a specialised I/O Processor (IOP). This is achieved by defining a split-driver model, where the Operating System specific part of the driver executes

- Allows OS to batch I/O requests
- Eliminates CPU stall during I/O transactions
- Increases system TPS
- Consistent hardware interface
- Simplified software model for I<sub>2</sub>O driver
- OS support becomes pervasive
- Narrows requirements for compliance

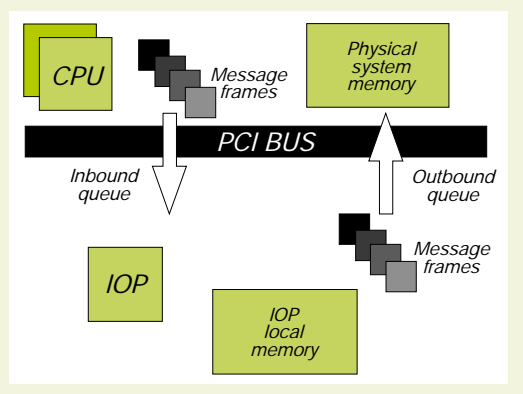


Figure 3.

on the host CPU, and the hardware specific part of the driver executes on the IOP.

In order to understand this, it is instructive to look at the various components required, and how they fit together and interact.

## A. The Shell Interface

The description and specification of the interface between the host and the IOP is called the Shell Interface. Messages are used to communicate across the Shell Interface, and the message set is subdivided into a number of classes. An Executive class supports initialisation, administration and management operations, a Utility class provides generalised per-class operations (e.g. aborting a transaction, claiming a device etc.), and a set of different Base classes support specific types of I/O, such as LAN, Block Storage and SCSI.

## B. The Host System

The host operating system needs to provide two components - an I<sub>2</sub>O messaging and transport stack, and the host part of the I/O driver.

### 1. Messaging & Transport Stack

The host's I<sub>2</sub>O Messaging and Transport stack is responsible for building I<sub>2</sub>O messages, posting them to an IOP, and receiving replies from the IOP. The bottom layer of this stack will be hardware and bus specific, but it is independent of the specific I/O subsystem

and the I/O adapter(s) being used. In addition, the transport layer is message class independent.

### 2. Operating System Service Modules (OSM)

On top of the transport layer sit a number of Operating System Service Modules (OSM), such as LAN, Block Storage or SCSI. These OSMs implement the host component of the I/O class being implemented, and are bound at their top interface to the protocols they support. For example, a TCP/IP stack would bind to a LAN OSM, which would take the TCP or UDP frames generated by the stack and build standard I<sub>2</sub>O LAN class messages to send these frames to the IOP. Similarly, frames received by an IOP are DMAed to the host, a message constructed and the LAN OSM notified, at which point it can forward it up to the TCP/IP layer.

There are a number of important things to note about the Host System:

The host OSM is hardware, bus and adapter independent. It is class specific.

Messages are buffered, and processed asynchronously from their generation.

Data transfer occurs by DMA, usually on demand.

The host is only notified at the end of an operation (for example, after a frame has been received and DMAed up to the host)

The host has a lot less work to do - it simply builds

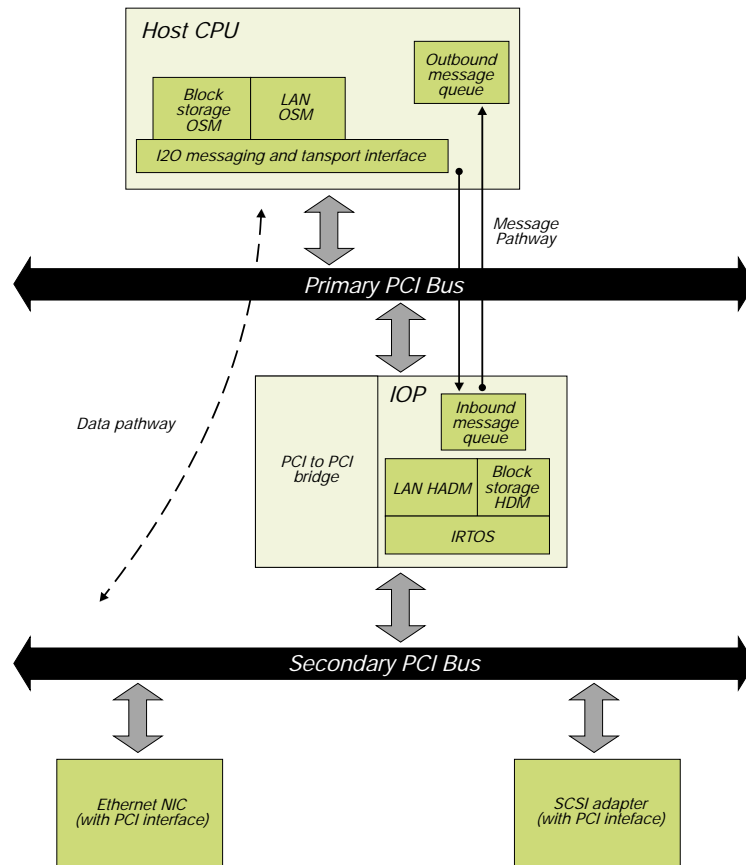


Figure 4. A typical I<sub>2</sub>O system.

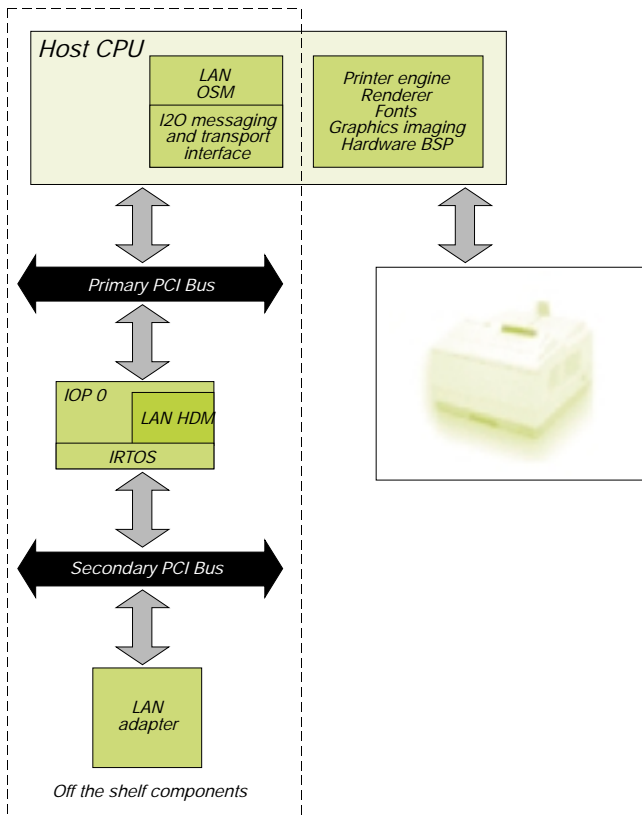


Figure 5. Printe with I2O LAN Interface.

a message describing the operating to perform, and then allows the IOP to get on and do the processing.

The top end interface between the OSM and the protocols it supports is Operating System specific, and implemented by the OS Vendor.

**C. The IOP System**

An IOP that fully implements the shell interface is I<sub>2</sub>O compliant. The I<sub>2</sub>O Specification also has support for installable device drivers.

*1. The Core Interface*

In addition to the Shell Interface, the I<sub>2</sub>O Specification defines an object module level interface, called the Core Interface, which allows users to download Device Driver Modules (DDM) to the IOP. This requires that the IOP provides a RTOS that implements the Core Interface.

An IOP system that implements the Core Interface has a number of very important advantages over a system that simply implements the Shell interface:

- Allows users to download a variety of DDMs, often by third parties
- IOP can host non-hardware driver code for additional value-add and functionality (e.g. TCP/IP offload, RAID)
- Easy upgrading of drivers
- Fast development cycle and time-to-market. No need to write your own OS.

Provides a RTOS optimised to the platform

*2. The IRTOS (I2O Real Time Operating System)*

Today, most IOPs are more than just Shell Compliant, and are based around an IRTOS. An IRTOS is important as it allows a variety of device drivers to be hosted on the IOP. The core specification as defined by the I<sub>2</sub>O SIG provides a number of advantages and services to a device driver writer, not limited to:

- Object based operating system
- Real-time response
- Threads, Semaphore, Message Queues, Pipes and Timers
- Interrupt Service routine handling and interrupt masking
- Event queue driven
- High speed memory allocator
- DMA object handling
- DDM hosting, creation and management
- Implements shell interface and provides automatic dispatch of messages to the DDM that claims them
- Inter-DDM messaging
- Abstracted access to bus and adapter memory
- Battery backup and Non-volatile memory support
- HTML based configuration management (via TCL)
- Subset of ANSI Standard C library

*3. DDMs*

Device Driver Modules (DDM) supply the off-loaded component of the I/O class being implemented. There are two types of DDM - a Hardware Device Module (HDM) and an Intermediate Service Module (ISM). An HDM is responsible for implementing the adapter specific part of the I/O class. So, typically a LAN NIC vendor would write the HDM to control their NIC. The top end of their driver understands LAN base class messages; the bottom end actually talks to their adapter.

ISMs allow developers to offload non-hardware-controlling tasks to the IOP. Typically an ISM will understand base class messages at its top end, and generate base class messages at its bottom end. A good example is a RAID ISM. From the host's point of view, it appears like a block storage device. The RAID ISM might abstract a number of SCSI adapters to appear as a single RAID volume. The RAID ISM would then send block storage class messages to the individual block storage HDMs.

Some important things about DDMs:

- A DDM is Host Operating System and bus independent.
- An HDM controls a specific adapter.
- Typically, a DDM implements one base class only.
- ISMs do not control hardware.

**D. I<sub>2</sub>O Messaging with PCI**

I<sub>2</sub>O messages are passed between the host and the IOP using hardware based message queues. Message passing is viewed from the perspective of

the IOP, so messages destined for the IOP are placed on the inbound message queue. Reply messages destined for the host are placed on the outbound message queue. These queues contain the Message Frame Addresses (MFA) of buffers to build messages in. In order to send a message to an IOP, the host gets a MFA off the IOP's inbound free queue, builds the message in the buffer pointed to by the MFA, and then places the MFA back on the inbound post queue. Thus, posting a message involves a PCI read and a PCI write.

Similarly, in order for the IOP to reply to the host, it takes an MFA off the outbound free queue, builds the message in the buffer pointed to by the MFA, and places the MFA on the outbound post queue.

The management of these queues is implemented in hardware by the IOP.

In the PCI world, the address of these inbound and outbound message queues is at a fixed offset in an IOP's memory mapped address space (offsets 0x40 and 0x44). This is the only bus specific section of the I<sub>2</sub>O Specification.

## E. Push vs. Pull

The original version of the I<sub>2</sub>O Specification only defined the Push model for posting messages. In this model, the inbound buffers described by the MFAs in the inbound queue reside in the IOP's memory space. Between reading a free MFA and posting the MFA, the host copies the message across the PCI bus to the IOP's memory space. Outbound buffers reside in host memory space and the IOP must copy the reply message into host memory space before posting an outbound MFA.

Version 2.0 of the I<sub>2</sub>O Specification defines the Pull model. In this case, the inbound message pool resides in host memory space, and it is the IOP's responsibility to DMA the message before processing it. This removes the need for the host to copy frames over the bus and increases the rate at which the host can generate messages. At the cost of slightly increased message processing latency, the IOP can also improve its message throughput by DMAing messages across the bus, possibly in bulk, improving bus utilisation.

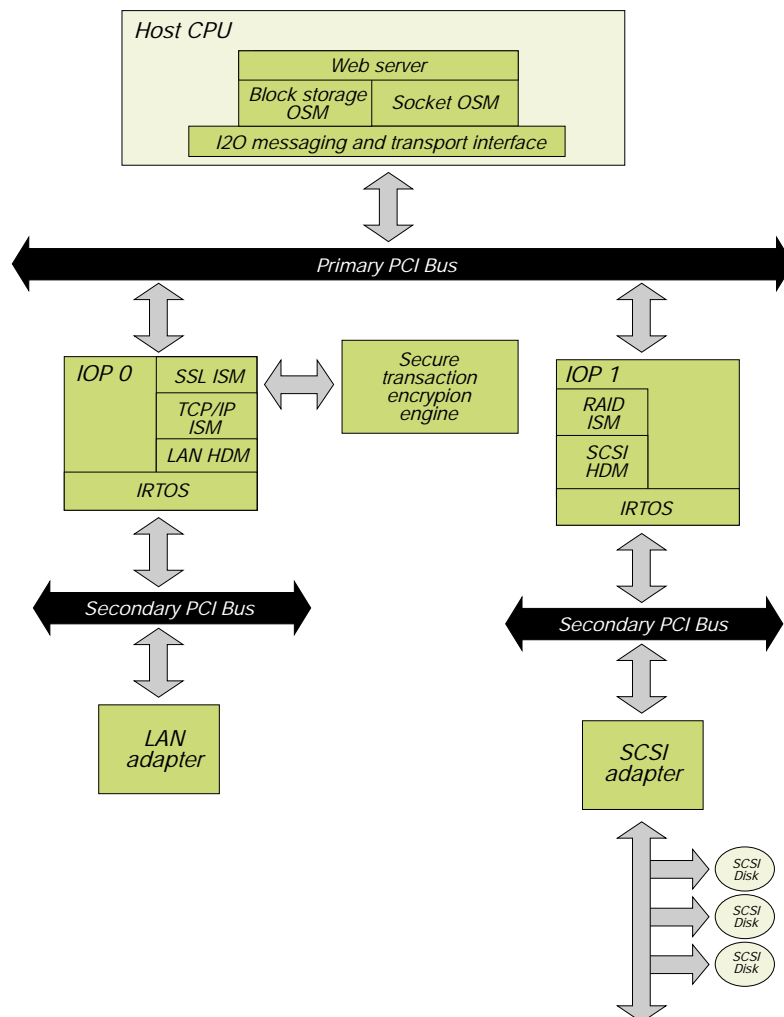


Figure 6. Secure web transaction server.

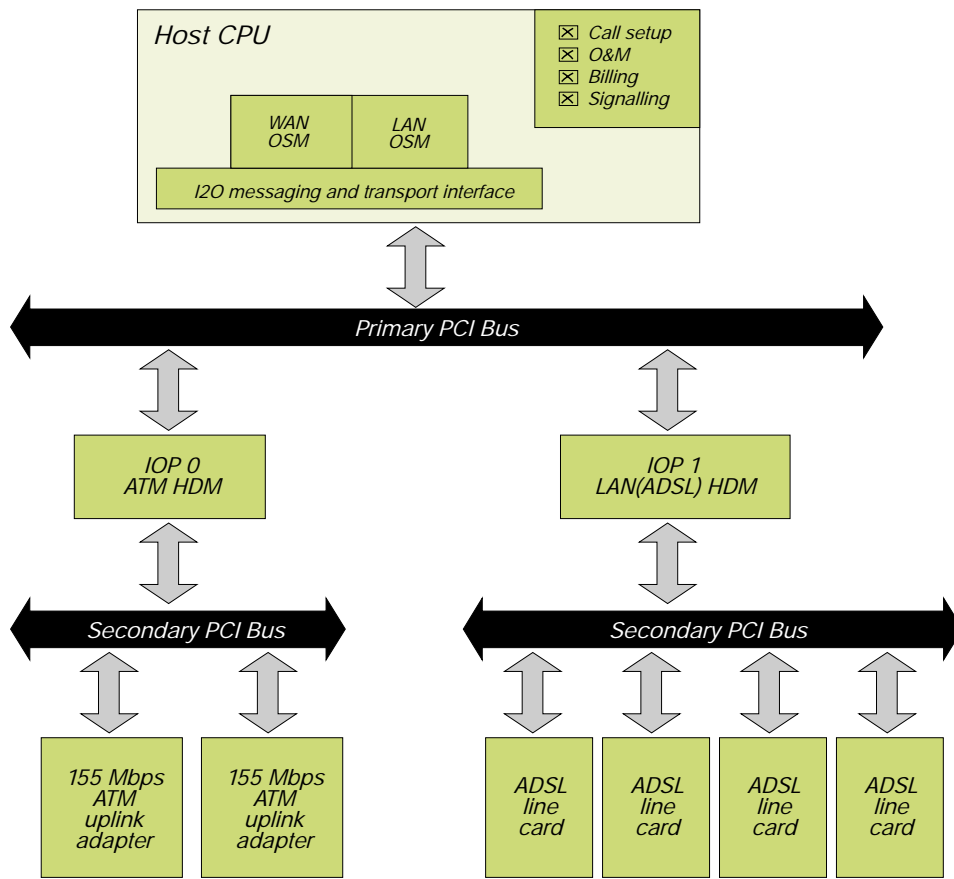


Figure 7. ADSL central office DSLAM.

**CONSTRUCTING A SYSTEM**

One of the advantages I<sub>2</sub>O provides is that pulling all the pieces together now becomes a far easier job. The use of a popular bus standard, such as PCI, is one of the key factors as this helps to reduce or remove the amount of glue-logic and custom design work needed to build a system.

A typical example might involve using an i86-based processor, memory and a PCI interface controller as the host unit. The I/O subsystem, which also connects to the PCI bus, would use an IOP. An IOP, such as the Intel i960RX range of processors, typically contains a PCI interface, a processor core, and a PCI-to-PCI bridging unit. The PCI based hardware I/O devices (such as an Adaptec 2940 SCSI interface) are then attached to the secondary PCI bus, together with any necessary physical interfacing logic.

The host processor runs the host operating system, the OSMs, and the user's application. The IOP runs an IRTOS, and the appropriate DDMs.

**USING I<sub>2</sub>O TO BUILD I/O INTENSIVE APPLICATIONS**

Using I<sub>2</sub>O to offload I/O processing to a separate I/O subsystem allows the designer to use standard off-the-shelf components when building a product. Even when specific sections of the I/O system have to be implemented by the designer, these pieces fit into a

well-developed architecture, and the designer can concentrate on the specific component required.

**A. Printer with I<sub>2</sub>O LAN interface**

In this example, a printer company wants to develop a next-generation, networked laser printer. In order to compete successfully with the competition, the company needs to achieve a particular page-per-minute rate at a particular price point. Rather than spend development time designing a network interface and writing network driver code, and then losing processor cycles to run the network driver, the designer can off-load the network driver to an I<sub>2</sub>O subsystem. One result of this is that a cheaper host processor or slower (and hence cheaper) memory may be able to deliver the required page-per-minute rate. Another result is that the company can reduce its time-to-market.

**B. Secure Web Transaction Server**

In this example, a company wants to add Secure Socket Layer technology to its Web Transaction Server product. The success of the product is defined by having a high transactions-per-second rate, but secure transactions require DES encryption, and this takes a lot of processing. In addition, the product needs to use RAID based storage to provide cheap but reliable data storage.

One solution is to use I<sub>2</sub>O to off-load both the RAID & Storage I/O component and the TCP/IP & LAN component to two IOPs. One IOP would implement RAID;

the other would implement TCP/IP. In addition, the Secure Socket Layer interface is defined to sit between TCP/IP and a socket interface, so the designer could concentrate on writing a SSL ISM to run on the IOP and interface with the company's (proprietary) secure transaction hardware engine. All of this leaves the main CPU to concentrate on managing the web interface and processing transactions.

### **C. ADSL central office DSLAM**

In this example, a company wants to develop an ADSL central office DSLAM unit. Functionally this must provide ADSL connectivity to a number of subscribers, provide some routing between ADSL connections, and provide a fat-pipe connection to the Internet via an ATM uplink.

Using I<sub>2</sub>O to off-load the I/O processing once again isolates the I/O processing component of the design, leaving the host CPU to concentrate on tasks such as call setup, signaling and routing, billing, and general operations and management tasks. In addition,

by isolating the I/O system, it is far easier to replace or upgrade specific I/O components. For example, replacing one of the 155Mbps ATM links with a 100Mbps Ethernet connection, or replacing one of the ADSL line cards with a different vendor's adapter becomes extremely easy - simply plug in the new card and run the new driver on the IOP.

### **FUTURE-PROOFING**

I<sub>2</sub>O is already extremely well future proofed. The general architecture is independent of bus type, it already supports PCI and Compact PCI, and version 2 has already started to anticipate some of the requirements of switched, fabric-based networks.

I<sub>2</sub>O is a software level message passing protocol, and as such it is an excellent fit for new switched I/O architectures such as NGIO and Future I/O. Indeed, both technologies are investigating and/or prototyping using I<sub>2</sub>O.

### **CONCLUSIONS**

By off-loading I/O processing from the host CPU, I<sub>2</sub>O provides the embedded design engineer with a powerful way to isolate and scale I/O processing, reduces time-to-market, and concentrate on the features of the product, rather than the I/O system.

In addition, the embedded community as a whole is able to benefit from the PC industry's wide scale acceptance of the PCI bus standard and maximise the application processing capabilities of the host CPUs used in their designs. This in turn improves time-to-market and drives costs down ■

---

*Jason O'Broin joined Wind River in 1998 as a Senior Development Engineer. He is currently a member of the Server Products Group in the Wind River Networks business unit, working on network offload technologies. Previously, Jason was a Senior Development Engineer at Madge Networks in the United Kingdom. He has a M.Sc. in Computing Science from the University of Newcastle Upon Tyne, UK*

### **RESOURCES**

For further information see:

<http://www.i2osig.org/>

<http://www.pcisig.com/>

<http://www.futureio.org/>

<http://www.ngioforum.org/>

<http://www.wrs.com/i2o/index.html>