

# Here Today — Multiprocessing on PCI and CompactPCI

Chris Williams, Vice President of Marketing Force Computers

Does PCI—and its industrial derivatives like CompactPCI—feature a multiprocessing capability, or not? There have been frequent allegations that CompactPCI is not suitable for asymmetrical multiprocessing - referring to the type of multiprocessing where each of a PCI-based system's multiple processors has its own operating system. Closer scrutiny reveals a different reality, showing how CompactPCI can, indeed, address the multiprocessing real-time environment.

Originally, the PCI (Peripheral Component Interconnect) bus was developed as a high-speed I/O bus for PC systems. In recent years, the PCI bus has also found its way into the embedded market in various form factors. The most popular representative is PCI's electrical and logical equivalent, CompactPCI, with its rugged, tried-and-true Eurocard format.

The increasing demands for asymmetrical multiprocessing capabilities have triggered new solutions that address the restrictions previously imposed on this type of multiprocessing by PCI/CompactPCI architectures. Asymmetrical multiprocessing ties together several PCI-based boards via the PCI/CompactPCI bus. Each board is "intelligent" in that it has its own proces-

sor and copy of the operating system. Figure 1 shows this basic configuration, with the user interface and the network part of the system being implemented on the host CPU, and with additional real-time processing power provided by I/O modules.

## Central System Controller Functionality in PCI/CompactPCI Systems

A brief discussion of the fundamental aspects of PCI is now necessary. It is important to understand that a PCI/CompactPCI system requires a central system controller (host) with the following multiprocessing properties:

- Central clock signal generation for the PCI bus
- Configuration management
- Arbitration (multimaster system)
- Central interrupt handling

Let's take a closer look in order to form the basis for further multiprocessing considerations.

**Central clock signal generation**— The PCI bus is a synchronous bus, which requires the system controller to generate a central clock signal for the entire system.

**Configuration management**— Both PCI and CompactPCI busses are based on

a hardware and software specification that defines the auto-configuration cycles which in turn enable Plug 'n Play capabilities. During the auto-configuration cycle that occurs after power-up, the central system controller uses special bus transfers to detect what PCI devices are connected to its local PCI bus, as well as what PCI devices may be connected on other PCI buses behind a PCI-to-PCI bridge (PPB). The central system controller first configures and assigns address ranges for its local PCI devices, then repeats the process for the PPB and for any PCI devices found on PCI backplanes behind the PPB.

**Arbitration**— The PCI bus is also a multimaster bus, which means that several masters can be connected on one PCI bus and can compete for possession of the bus via a request/grant mechanism. Possession of the PCI bus is allocated by a central arbitrator which is usually located in the processor-PCI bridge. One master on the PCI bus may, for instance, be a SCSI controller that collects data, then requests for possession of the bus, and finally transfers its data to main memory by means of DMA transfer.

**Central interrupt controller**— The PCI bus has 4 interrupt lines that can be used by several devices. The central interrupt controller is located directly on the local PCI bus of the system controller board. This is adequate for a PC system, but processing interrupts in a CompactPCI system become more complicated. This is because the central controller must also process interrupts generated by other CompactPCI boards on the backplane, and not just its local PCI bus interrupts. A standard CompactPCI system has 8 slots and if more than 4 interrupt-capable boards are installed, some of the 4 available interrupt lines must be shared by more than one interrupt source.

## Problems for Multiprocessing

All of these points are important for multiprocessing and apply to CompactPCI systems when the local I/O bus of the system controller board and the intelligent

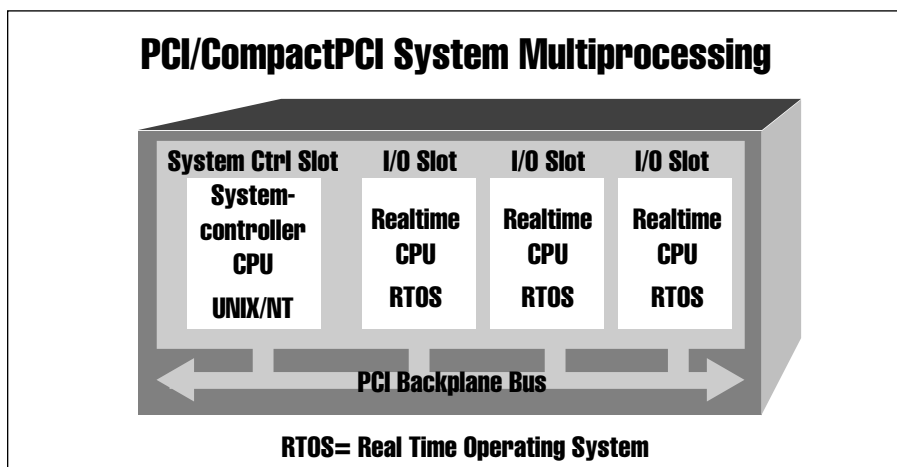


Figure 1: Example of a PCI/CompactPCI system with asymmetrical multiprocessing

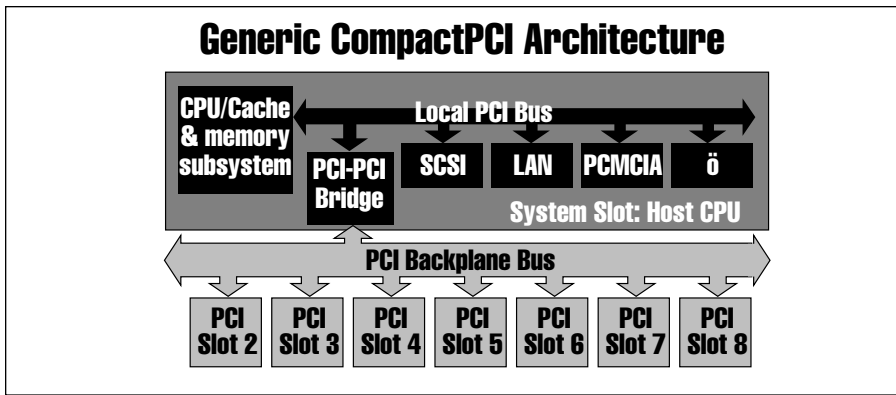


Figure 2: Generic CompactPCI architecture

I/O boards are PCI-based.

There are three different methods that can be used for connecting various PCI buses:

1. Direct connection of two PCI buses without a bridge
2. Connection via standard ('transparent') PPB
3. Embedded PCI-to-PCI bridges

### Direct connection of two PCI buses without a bridge

The first and most obvious method is to use a direct and parallel connection between the local PCI bus and the bus designated as the PCI backplane. The first disadvantage of this method is the restriction on the number of loads that can be driven by a PCI module.

In a PCI system, the signals are driven directly from the IC without any intermediate buffer and are subject to certain timing restrictions on the high-speed PCI bus. This simple hardware restriction limits the number of loads in any PCI system to 10. From the view of the PCI bus, a PCI device on the motherboard counts as one load and each additional slot counts

as 2 more loads. For a standard PC this means a maximum of 4 expansion slots.

Additional disadvantages to the direct PCI method become apparent when multiprocessing is required and the slots are populated with intelligent I/O rather than just "dumb" I/O modules. These disadvantages are identical to the problems with the 'transparent' bridge connection method discussed in the following section.

### Connection via standard ('transparent') PPB

In 1994, the 'PCI-to-PCI bridge Architecture Specification' was defined by the PCI Special Interest Group (PCISIG). This specification defines a 'transparent' or 'standard' PCI-to-PCI bridge (PPB) that avoids the load restrictions of the direct connect method.

After normal identification during the auto-configuration cycle, the PPB becomes transparent to the host processor. From the multiprocessing point of view, this transparent bridge behaves in exactly the same way as the direct connection of two PCI buses described above.

This type of PPB does not contain any resources such as DMA or mailboxes that would require a separate device driver. Nor does it convert any addresses from one PCI bus to another (flat addressing). Interrupt lines are simply routed around this 'transparent' PPB.

Transparent PPBs solved the load restriction problem, but we are now faced with the following new challenges:

**No Plug'n Play capability-** The auto-configuration cycles from the two different processors will collide. After power-up, each processor will first configure its local PCI I/O subsystem, then configure its 'transparent' PPB, and finally attempt to identify and set the registers and address ranges of the attached devices found behind the PPBs on the PCI backplane. The first problem is that 'transparent' PPBs can only be configured from the primary (local) PCI side via a special configuration cycle coming from their local processor. PCI to PCI bridges cannot be selected by configuration cycles coming from their secondary or backplane interface. This means that it is not possible for the host board to identify the intelligent I/O board and vice versa.

**There is no communication feature (e.g. mailboxes) between the two processors-** A 'transparent' bridge does not have a device driver that could manage such a resource.

### New solution involving embedded PCI-to-PCI bridges

Fortunately, a new type of PCI-to-PCI bridge has been developed in order to avoid the above problems. These new PPBs are called embedded, or 'non-transparent', bridges. They have been designed specifically for intelligent I/O boards and thus for asynchronous multiprocessing. Here is how they specifically solve previous problems and other important issues:

Plug 'n Play capabilities are now possible without collision of the configuration cycles. The embedded bridge effectively separates the areas allowed to be scanned and configured by the local and host processors. After power-up, the intelligent I/O processor will scan (auto-configure) its own I/O subsystem found on the primary or local side of the embedded bridge. It will then configure the embedded bridge, install address windows, define address translations and map the address registers for its local PCI devices into the embedded bridges' primary

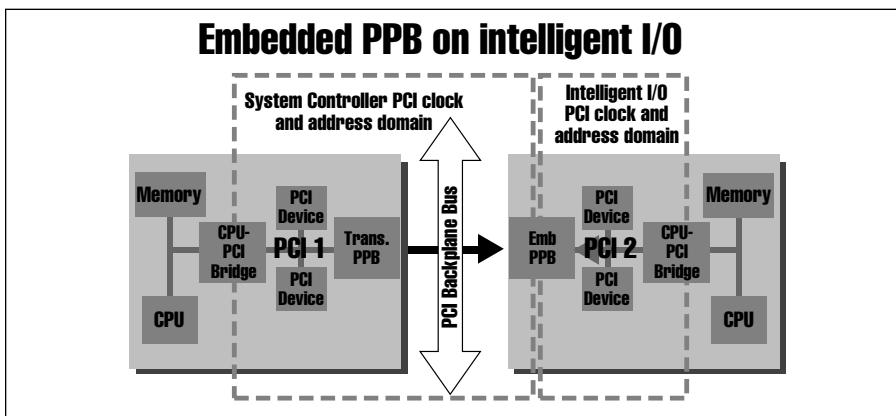


Figure 3: Asymmetrical multiprocessing with embedded ('non-transparent') PPB on the intelligent I/O board

(local) and secondary (PCI backplane bus) sides. The local processors' configuration range ends at the embedded bridge and it will not scan past the bridge to find devices on the PCI backplane bus (figure 3).

During the same power-up sequence, the processor on the host board is configuring its PCI I/O subsystem (PCI1 in figure 3) and its 'transparent' PPB. It is also scanning for PCI devices found on the PCI backplane bus. When the host processor finds the embedded bridge on the intelligent I/O module, it only sees the secondary side of the bridge (which looks like a normal PCI target device) and configures it accordingly. The underlying PCI bus of the intelligent I/O module (PCI2 in figure 3) is not visible to the host processor.

Addressing conflicts are avoided by means of address conversion. For instance, the processor of the I/O board is able to 'hide' the resources it requires from the host processor (hidden resources). In addition, address conversion enables a device on the local PCI bus of the intelligent I/O board to be allocated a PCI address that already exists on the backplane without causing any address conflicts.

The host processor drives the clock signal via its 'transparent' PPB to the PCI backplane bus and to the attached PCI devices. The embedded bridge on the intelligent I/O board receives the host clock signal on its secondary (PCI backplane bus) side, but does not forward the host clock signal to its primary side (PCI2). Instead, the primary side of the 'non-transparent' bridge runs with the clock signal from its own local processor. Both clock signals are still synchronous for their PCI bus, but asynchronous with respect to each other.

The 'non-transparent' PPB has functions (e.g. mailboxes and I2O FIFOs) that can be used for communication between the host processor and the intelligent I/O processor.

Figure 3 illustrates how a hardware solution for asymmetrical multiprocessing in PCI and CompactPCI systems is possible by using 'transparent' PPBs on the host board and 'non-transparent' PPBs on intelligent I/O boards. For the sake of completeness, it should also be stated that 'transparent' PPBs could also be used on 'dumb' I/O boards.

### **Summary - Embedded bridges bring asymmetrical multiprocessing to CompactPCI**

This article has described the hardware requirements for asymmetrical multiprocessing on a CompactPCI system. Asymmetrical multiprocessing systems have been implemented on CompactPCI architectures, by combining transparent and embedded bridge technologies.

The key multiprocessing requirements - clocking, plug'n play configuration management, arbitration and interrupt handling - have all been addressed by the embedded bridge solution. Embedded bridges also offer additional features like mailboxes, I2O support and hidden resource capabilities.

Before deciding to implement a specific asymmetrical multiprocessing system, the application of such a system should be scrutinized with respect to the real-time conditions and also subsequent expandability.

Force Computers is on the web at: [www.forcecomputers.com](http://www.forcecomputers.com)

B&W Ad for  
Harting