Creating Multicomputer Test Systems using PCI and PCI Express

Lee Mohrmann
National Instruments
Austin, Texas USA
lee.mohrman@ni.com

Jason Tongen
National Instruments
Austin, Texas USA
jason.tongen@ni.com

Matthew Friedman
National Instruments
Austin, Texas USA
Matthew.friedman@ni.com

Mark Wetzel
National Instruments
Austin, Texas USA
Mark.wetzel@ni.com

Abstract— As test systems grow larger with higher channel counts and faster digitization, new techniques are needed for managing and processing the large amount of data being generated. One technique to handle this is with multiple computing nodes to distribute the acquisition, processing and data storage. Current communication busses exist to create a multicomputer system but must often make tradeoffs in terms of bandwidth or latency capabilities. In this paper we will look at using the PCI and PCI Express busses to create both a high bandwidth and low latency multi-computing system.

Keywords *– PXI, PCI, PCI Express, cabled PCI Express, multicomputing, instrument contro, test systens*

I. INTRODUCTION

The approach taken to address the high bandwidth and low latency communications model is to use the concept of a PCI Non-Transparent Bridge (NTB). An NTB is a function within some PCI devices that translates PCI transactions within one PCI hierarchy into corresponding transactions in another PCI hierarchy. Mapping transactions between PCI hierarchies allows two PCI trees to be logically connected and enabling them to communicate. This communication between two PCI based buses does not break the model of a central host-bus bridge and single memory space per PCI hierarchy.

While PCI-based NTBs have been around for years, they have simply acted as a buffer between two PCI-based systems. PCI-Express NTB's have been around since the packet switches were developed, but present unique challenges in an open, modular environment. While NTB technology is not new, the industry has lacked a standardized means by which to easily communicate between PCI hierarchies. NTB solutions required custom hardware and software design prior to using them as a communications channel. In this paper we will specifically focus on the development of an open industry standard that implements this communication across independent PCI buses. Specific standardization issues we will focus on include how to initialize the systems, manage clocking, handle interrupts, define interconnects and address communication protocols.

The PXI MultiComputing (PXImc) specification addresses the hardware and software requirements to connect two or more systems together using PCI based interfaces [1]. From the hardware standpoint, several issues have to be reconciled to allow two independent systems to communicate directly over PCI or PCI-Express. From the software view, a communications scheme must be create to allow each system to discover and configure its own resources in a way to communicate with the other system.

II. PHYSICAL CONNECTIVITY

The first level of interoperability is the physical connection. The PXI Systems Alliance (PXISA) wanted to leverage existing physical infrastructures while increasing system capabilities. PXImc utilizes existing chassis and module form-factors to increase computing options while also utilizing the Cabled PCI-Express specification to provide interconnection between physical boxes [2]. In utilizing this approach, the PXISA has allowed both suppliers and end users to maintain their existing investments in equipment while providing the ability to expand system performance to tackle the ever increasing complexity of instrumentation configurations and increased computational demands. PXI vendors will be able leverage this specification by creating new processor modules that will be able fit in a peripheral slot of current PXI chassis. It also allows the cabling over cabled PCI-Express to additional PXI systems, general computing devices, and other instrumentation systems.

III. CLOCKING ARCHITECTURE

The second level of interoperability addressed is the lowest level of hardware- synchronization for data communication. For PCI based systems, a common 33MHz clock is typically used as a timing reference for all PCI devices in a system. In PCI-Express systems, it is typically a 100 or 125MHz clock used for a common timing reference. Each system will have a central source for the reference clock generation and each of these sources will have its own jitter characteristics. There will likely be a phase offset between the two sources as well. While jitter on a 33MHz clock will not likely be an interfacing issue, the phase between the two system references will likely be a problem for the synchronous nature of the PCI bus. With a limited timing window for a bus, devices in different clock domains may not be able to communicate due to the high likelihood of phases not being aligned and therefore synchronous timing budgets will be broken.

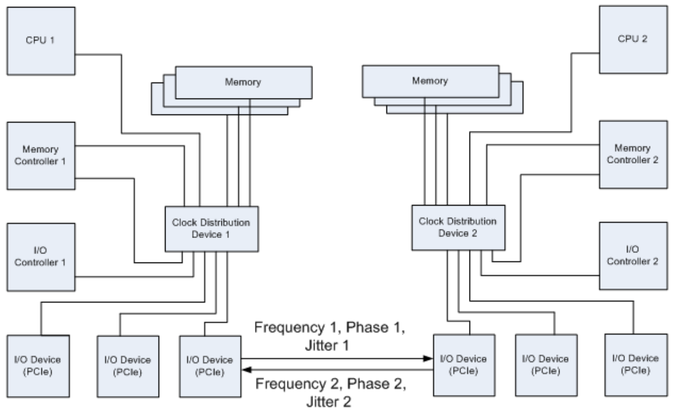
CPU 1
CPU 2
Memory
Memory
Memory Controller 1
Memory Controller 2
Clock Distribution Device 1
Clock Distribution Device 2
I/O Controller 1
I/O Controller 2
I/O Device (PCIe)
Frequency 1, Phase 1, Jitter 1
Frequency 2, Phase 2, Jitter 2

Figure 1. PCI clocking architecture

PCI-Express interfaces have an even more sensitive clocking situation. The PCI-Express specification requires that devices maintain a clocking difference no greater than 600ppm. This limits the speed difference between two devices, which in turn allows cost saving measures to be used in the design of PCI-Express silicon. While PCIe links do not require devices receive the clocks with a particular phase relationship, the devices do depend on a low differential jitter environment. This can be an issue if the two clock sources do not have identical jitter profile and therefore have slight frequency differences. While many serial standards are designed to handle frequency differences and independent jitter profiles, they do so at the expense of cost of clocking circuits and silicon memory. The low-cost nature of PCI-Express imposes a few requirements on these characteristics but still allows a high-performance interface. In addition to inherent jitter, many systems employ a modulation to the reference clocks, called spread spectrum clocking (SSC), to reduce peak radiated emissions. This modulation will appear as differential jitter between two devices that reference independent clock sources.

The PXImc specification addresses these clocking architectural requirements and creates a scheme that must be adhered to for interoperability. This is done by leveraging both the existing PCI and PCI-Express specifications for clock characteristics and defining where the clock domain crossing must be accomplished [3]. In doing this, off the shelf silicon may used by suppliers, while system integrators will have a known operation methodology to work from when building a system.

IV. NON-TRANSPARANT BRIDGING TO CONNECT PCI SYSTEMS

The second challenge after the devices can communicate is bus ownership. The general topology for a PCI bus is a root structure. A single host bus interface, or root complex, is the interface for the computing system to communicate with PCI devices. The main system will interface with this root complex to assign resources before communication begins. With some minor exceptions, PCI devices are expected to be controlled from a single point. By connecting two or more systems together via a PCI interface, several issues can occur during the assigning of resources for each system. Key questions that must be addressed include:

1. Which system would own the bus?
2. How would System A assign or control resources that are contained in System B?
3. Once the systems are running, where does the data go into memory, System A or B?
4. How does the device deal with simultaneous access or data integrity?

The PCI bus was not originally designed to interface between two systems, but to be a local bus. To deal with these logical issues, the concept of the non-transparent bridge is used. A NTB, is a device that has two endpoints meant to interface to two different logical systems.

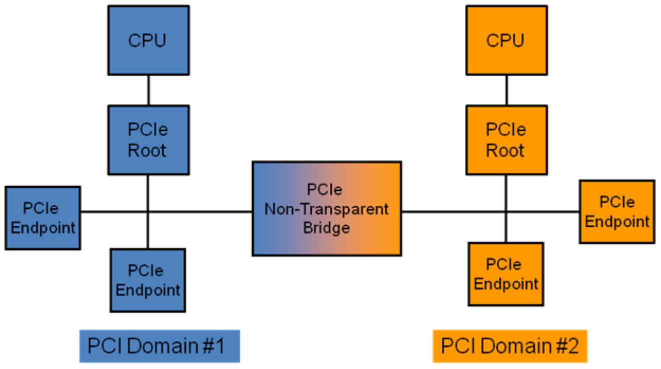

Figure 2. Non-Transparant Bridging between PCI domains

The use of a non-transparent bridge to separate the two PCI domains solves the issue of resource allocation. The non-transparent bridge isolates the physical resources of PCI Domain #1 from the physical resources of PCI Domain #2. Both System A and System B have complete control of resource allocation within their own domains, and the presence of the non-transparent bridge does not affect either's allocation algorithm. The non-transparent bridge responds to resource requests as all other PCI endpoints in the system, requesting some amount of physical address space. The system BIOS or system software will assign a specific range of physical address space to the non-transparent bridge. As this resource allocation occurs both on System A and System B, the non-transparent bridge will acquire resources in both PCI Domains. The address space acquired by the non-transparent bridge within PCI Domain #1 will act as a window into physical address space within PCI Domain #2, and the address space acquired within PCI Domain #2 will act as a window into physical address space within PCI Domain #1.

After resource allocation executes on both System A and System B, the non-transparent bridge will contain memory mechanisms to transfer data between the two systems. These include items such as scratch pad registers to send data, doorbell registers to assert interrupts, and large blocks of address space to translate across the NTB into the opposing address space.

V. Software Connectivity

Device drivers for the non-transparent bridge on both System A and System B can use scratchpad and doorbell registers to perform some limited communication across the non-transparent bridge. Device drivers can allocate some region of physical memory, and use the scratchpads and doorbell registers to inform the corresponding device driver on the remote system of the amount and location of the physical memory allocated. The pair of device drivers can work together to initialize the non-transparent bridge so that accesses to the address space of the non-transparent bridge in one domain can be translated into a transaction to the device-driver owned memory in the opposing domain.

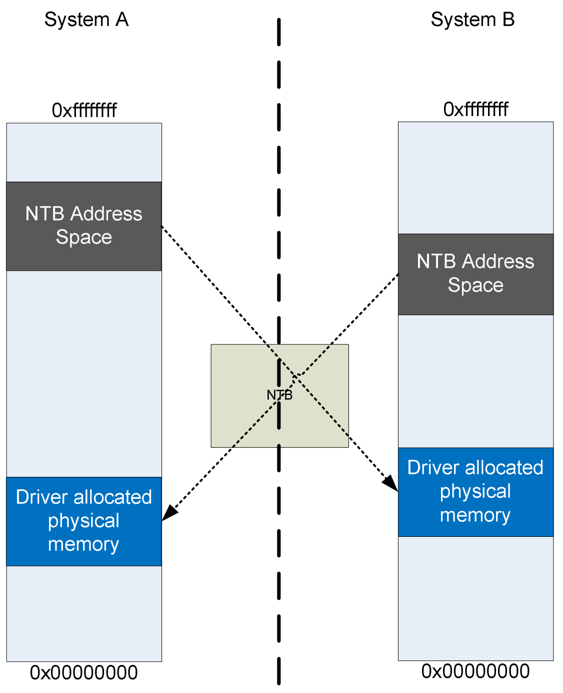

Figure 3. NTB address translation

In cases where the non-transparent bridge has multiple mechanisms for mapping between address spaces, the additional mapping regions can be used for either targeting additional regions of physical memory, or for targeting other entities that reside within physical address space, such as other PCI endpoints. The use of multiple mapping regions can allow the processor in one domain to read and write directly to any PCI endpoint present in the other PCI domain, or allow a PCI endpoint in one PCI domain to communicate directly with a PCI endpoint in the other PCI domain without involving either domain's processor.

After this fundamental initialization has completed to enable reads, writes, and interrupt generation across PCI domains, some mechanism must be established for the non-transparent bridge device drivers to publish its provided resources to potential consumers, such as application software or other PCI endpoint driver software. The PXImc specification provides a standardized API that can be used by clients that desire to communicate across the non-transparent bridge with some client within the other PCI domain. The PXImc specification API also allows application software to query the PXImc compliant device drivers present in the system to determine what potential connections exist and to initiate a connection across the non-transparent bridge. When compatible connection requests have been made on both sides of the PXImc compliant interface, the two connection requests will be paired together and given ownership of memory resources that will allow data transmission and interrupt generation across the non-transparent bridge. Once connection is established, the client applications can communicate using these resources, or pass ownership of the resources to some other entity, such as a hardware device.

Finally, to provide vendor interoperability, a shared library was introduced to guarantee client applications a uniform behavior independent of the vendor of the PXImc compliant interface. This shared library allows client applications to link against a single software library that will abstract the client application from vendor-specific libraries on the system.

VI. Conclusion

In conclusion, the PXImc specification provides a vendor interoperable solution for connecting multiple PCI domains together. This was done by standardizing NTB components and defining a common software interface to abstract vendor specific implementations. With PXImc, test system developers will be able to create distributed test systems over a high throughput, low latency link and leverage increased computing power,

References

[1] PXI Systems Alliance "PXImc Specification", unpublished.

[2] Peripheral Component Interconnect Special Interest Group, "PCI Express® External Cabling 1.0 Specification", http://www.pcisig.com/specifications/pciexpress/pcie_cabling1.0/

[3] Peripheral Component Interconnect Special Interest Group, "PCI Express Base Specification 1.1", http://www.pcisig.com/specifications/pciexpress/base