Optimizing PCIe® Communications through Device Lending and Multicast

with Dolphin PCIe Fabric Communications Library

Read About

- Multicast and reflective memory
- Device lending
- Non-Volatile Memory express (NVMe) sharing

Introduction

Today, ISR system integrators have three main goals: minimizing latency, maximizing system bandwidth, and optimizing configuration flexibility within their given size, weight, and power (SWaP) constraints. To address these issues, leading vendors of commercial off-the-shelf (COTS) OpenVPX™ modules are seeking ways to provide closer integration between a system's compute elements. Partnered with Curtiss-Wright, Dolphin Interconnect Solutions offers its eXpressWare to address the critical software need to squeeze every ounce of performance from the PCI Express (PCIe) interface.

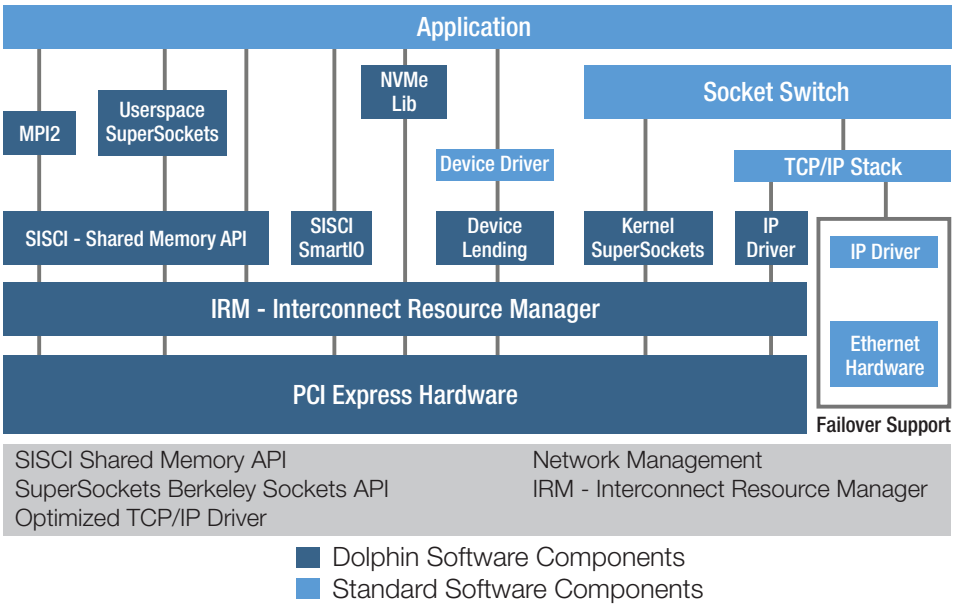

Figure 1: Software Application Layers Block Diagram

As discussed in the second white paper of our ongoing Dolphin PCIe library series, Dolphin supports several common software APIs that offer high-speed, low-latency, peer-to-peer communications. These flexible APIs also simplify the complex programming requirements of PCIe devices. In this third paper, we cover additional features of the Dolphin PCIe Fabric Communications Library that help minimize latency, maximize system bandwidth, and optimize configuration flexibility. The features we will be discussing are multicasting, device lending, and the Non-Volatile Memory express (NVMe) library.

The Benefits of Multicast

Multicast (similar to reflective memory, mirror memory, or replicated memory systems in computer literature) eliminates most communication latencies and reduces resource utilization. In embedded systems, reflective memory usually maps into the applications on multiple nodes, allowing them to share updated data without any network protocol overhead. Traditionally, systems implement reflective memory on additional adapter cards connected in a ring network. An application on one node would write to the memory of its attached card, which would then forward the data to the other memory cards. Applications around the ring would then read the received data from their attached memory card.

Removing the need for the additional memory card, Dolphin's solution utilizes each node's main memory and PCIe interface for their multicast design. The first advantage of this design is the reduced latency of data transfers, since access to a processor's main memory is faster than a copy to external memory. Due to the caching of main memory, data updates from other nodes will automatically invalidate the processor's cache, guaranteeing data consistency. Another difference from traditional reflective memory solutions is Dolphin's utilization of different addresses for reading and writing. The writing triggers the PCIe address translation and data transfer to other nodes. Depending on the hardware, eXpressWare can support up to a maximum of four independent memory segments (also known as multicast groups). The default size of each segment is 64 MB. By changing a BIOS setting, some boards will support up to 2 GB per segment. The user can add and remove segments without needing to resynchronize with the other nodes.

Multicast functionality enables a single write transaction to reach multiple targets. When mapped into a multicast address, a CPU store, direct memory access (DMA) write, or a posted write from another PCIe device can initiate a multicast transfer. For example, the CPU can issue a standard "memcopy()" or a simple pointer assignment to transmit data to multiple devices. The initialization of a multicast segment and a regular remote segment is very similar. When allocating, preparing, or mapping a multicast segment, the "SCI_FLAG_BROADCAST" flag must be true and the 'nodeId' should be set to "DIS_BROADCAST_MODEID_GROUP_ALL". Using the SISCI API, the programmer can configure other PCIe devices, such as GPUs or FPGAs, to send data directly to shared memory. During the configuration of the shared memory, the application can ensure data protection by allowing only certain nodes to write information, and other nodes to have read only permission of selected memory sections. The full PCIe bandwidth is distributed simultaneously to all remote nodes.

The multicast functionally is normally only available when using a network switch. A centralized switch will send data out on all connected ports simultaneously. If multiple switches are in the system design, each hop will add less than 200 nanoseconds delay to the data transmission. With a centralized switch, a dead node or a bad cable only results in the loss of that node while the rest of the system continues to run.

Envision the scenario where your 3U OpenVPX system is SWaP constrained, leaving no room for a centralized switch card, but the required data flow is from an FPGA to multiple CPUs and GPUs. The data generated by the FPGA could be multicast into two 3U processing modules, as well as into all connected GPGPUs connected to the processors and the processor's main memory. The key to this architecture is setting up non-transparent bridges (NTBs) back to back between the two processors. Figure 2 shows an example of one possible configuration with two 3U GPGPU modules and three CHAMP-XD1 digital signal processor (DSP modules. The data flows into the system through two data capture cards in a ping-pong manner. The data is then multicast throughout the system, including the two 3U GPU cards. At the same time, the middle XD1 could be receiving GPS and platform data (such as speed, direction, etc.), and distributing it to the rest

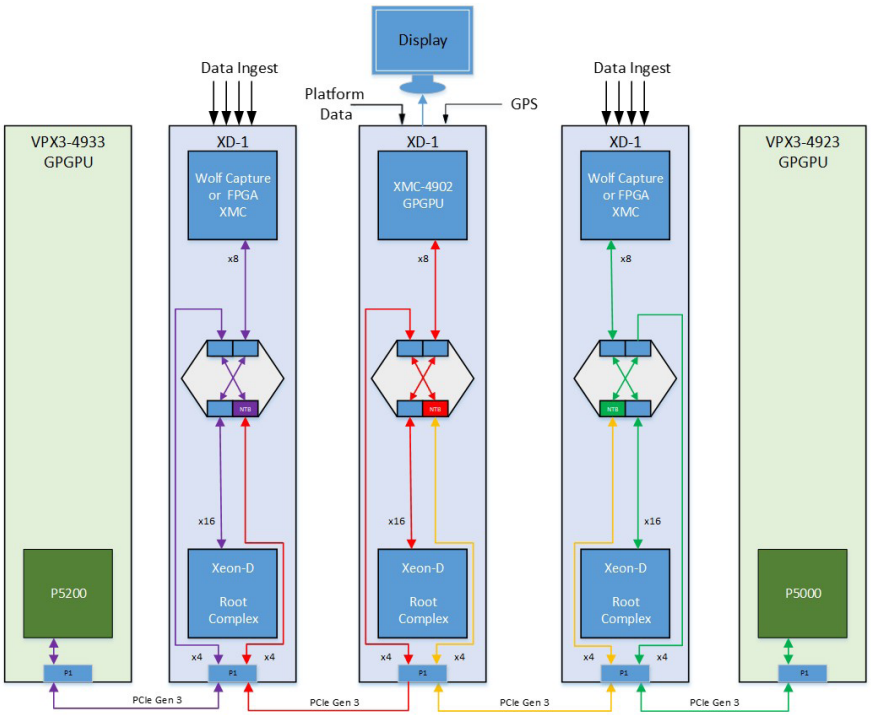

Figure 2: SWaP-Constrained 3U OpenVPX System

of the system. After the system finishes processing a dwell/data frame, the middle XD1 gathers the results, performs the final post-processing, and outputs the results. Another variation of this architecture would replace one of the 3U GPU cards with a 3U FPGA for data ingest, and the 2 data ingest XMCs with XMC GPUs.

Device Lending

ISR systems can run multiple modes, but now system integrators also want the ability to combine systems, such as radar, FLIR video, and EW, into a single processing box. The data flows and the computational requirements will vary greatly between each system's functions. Since the GPUs and FPGAs connect via the backplane or through the connectors, flexibility can be limited. Reconfiguring PCIe bridge ports might achieve some of the desired flexibility of the data flow or processing chain. But, what if a CPU could borrow a GPU, FPGA, or even NVMe SSD from another node?

Based on its SmartIO software, Dolphin offers a device lending method to access PCIe I/O devices such as FPGAs, GPUs, and NVMe drives within a PCIe network. Device lending becomes a simple way to reconfigure systems and reallocate resources without adding software overhead to the communication path.

First, How Does It Work?

As you would expect, this method uses two types of functions: lending and borrowing. The lending function makes devices available on the network for temporary access. Each processor binds itself as the driver for its local PCIe devices that are available for sharing in its tree. The processor then notifies potential borrowers of all available devices. When a borrowing node requests a device, the lending node shares the device configuration space and sets up the device in the borrower's I/O Memory Management Unit (IOMMU) domain. The borrower applies the necessary Memory Mapped I/O (MMIO) mapping using the NTB and then directs the lender to set up the reverse mapping for DMA and Message-Signaled Interrupts (MSI).

Next, the borrower injects the device into its Linux® PCIe subsystem and signals a hot-add event. After probing the device, Linux will set up and load the device driver. At this point, the device driver is able to communicate with the device using the MMIO address. Whenever the device driver sets up new DMA mapping by using the Linux DMA-API, the borrowing node will intercept the call and dynamically setup and replace the necessary IOMMU mapping. All address translations between the borrower's and lender's address domains are performed in the hardware (NTB and IOMMU) resulting in native PCIe performance along the data path.

After selecting a device to borrow from a list, the borrowing node can keep the device for long as necessary. When finished with the device, the borrower returns the device so that the local node can use it or other nodes in the system can borrow it. To the applications on the borrowing node, the device appears local and no changes are required to the standard device drivers on the Linux kernel.

Successfully Implementing Device Lending

For device lending to be successful, several issues need to be addressed. First, the mapping should be transparent between the lender and borrower. As part of enumeration, the system reads the configuration of a device to determine the size and number of address spaces to be mapped into the processor address space. MMIO allows the processor to read and write the device registers and memory as normal system memory. Instead of physical interrupts, PCIe employs MSI. When an endpoint posts an MSI, it is a memory write to a special address determined during enumeration. The processor will then post an interrupt defined by the content of the write. MSI-X, an extension to MSI, supports writes to multiple addresses to support targeting a specific CPU on a multi-core system. An NTB appears as a regular PCIe endpoint with MMIO sections, and the local processor can read and write its memory. However, memory operations in these areas are forwarded from one fabric to another with address translation performed. Since this address translation is similar to a single-level page table, the NTB is creating a shared memory architecture across the hosts. The NTB allows the local processor to read and write remote memory, including registers on the remote device. Conversely, the mapping of local resources for the remote device enables DMA and MSI interrupts across the NTB.

Without modifying the Linux kernel, the system must be able to dynamically handle the addition and removal of PCIe devices within its address space. One of the challenges is that a hot-plugged device must fit within pre-existing address windows residing inside the physical address ranges reserved by the operating system or BIOS at boot time. However, an NTB already allocates an address large enough to contain all the MMIO areas. To emulate a PCIe hot-plug event while the system is running, a virtual device can be inserted into the borrower's local device tree that appears to both the system and the device driver that a device was hot-added to the system. With the mapping of the device's Base Address Register (BAR) through the NTB, the local driver of the borrower's system can read and write the device registers and, to that driver, the device appears local.

The lender must set up the reverse mappings for DMA and MSI/MSI-X, but it does not know the address space that the local driver will use for DMA transfers in advance. Riding to the rescue, the IOMMU provides virtualization of addresses between the processor, (including its Memory Management Unit (MMU)) and the PCIe fabric. The IOMMU also features a DMA remapper to translate I/O virtual address to physical addresses.

NVMe Sharing

As part of the SmartIO, Dolphin has extended the SISCI API with device-oriented Device Lending semantics. This enables a software developer to implement a user-space device driver that supports distributed operation. Building on these extensions, Dolphin creates a user-space API implemented in C for writing custom NVMe drivers and high-performance storage operations.

Mapping user-space memory directly eliminates the cost of context switching into kernel space, and enables software-defined zero-copy. In turn, this reduces the latency of the I/O operations compared to using the Linux file system to access the storage devices. In the simplest case, direct I/O between the NVMe storage device and other PCIe devices (peer-to-peer) can be setup by arbitrary memory mapping to device memory.

By combining the parallel architecture of NVMe, the ability to use arbitrary memory addresses for buffers and queues, as well as a lock-less interface, the library provides the tools for multiple users to share a disk concurrently. With memory maps created through NTBs, multiple PCIe root complexes can also share a disk at the same time. The library also links with CUDA programs to provide storage access directly from the CUDA kernels. Creating the I/O queues and data buffers in the GPU memory excludes the CPU from the I/O path, reducing latency.

Conclusion

Today, the challenge of rugged embedded systems and their high-performance fabrics is how to minimize latency and maximize system bandwidth – all while optimizing configuration flexibility within given SWaP constraints. In this third part of our Dolphin white paper series, we covered the new exciting features of the Dolphin PCIe Fabric Communications Library enabled by its creative use of NTBs. These features include the ability to multicast without a central switch and the sharing of PCIe devices (such as GPGPUs and FPGAs) between multiple root complexes. In addition, the NVMe library enables concurrent low-latency access to NVMes from multiple users spread across the system. Curtiss-Wright, along with our partner Dolphin, is committed to increasing performance while adding flexibility to enable our customers to develop creative solutions for their application and performance needs.

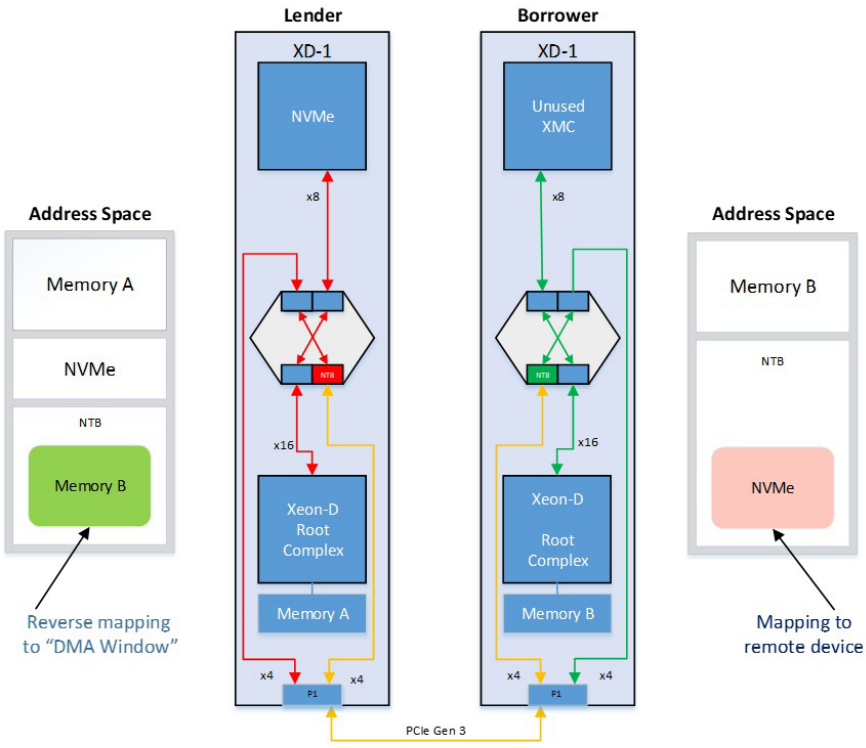

Figure 3: NVMe sharing

Author(s)

Tammy Carter, MSCS
Senior Product Manager
Curtiss-Wright Defense Solutions

Learn More

Product Sheets

- Dolphin PCIe Fabric Communications Library

White Paper

- Enhancing PCIe® Communications to Eliminate Bottlenecks with Dolphin PCIe Fabric Communications Library
- Minimizing Latency in Peer-to-Peer Communications with Dolphin PCIe® Fabric Communications Library

© 2020 Curtiss-Wright. All rights reserved. Specifications are subject to change without notice. All trademarks are property of their respective owners. I W182.0220