TECHNOLOGY WHITEPAPER

Minimizing Latency in Peer-to-Peer Communications

with Dolphin PCIe® Fabric Communications Library

CURTISS-WRIGHT
DEFENSE SOLUTIONS

Read About

Dolphin eXpressWare™

SuperSockets

SISCI

Considerations for picking the right API

PCIe fabric implementation on Dolphin eXpressWare and performance considerations

Introduction

Today's embedded systems are made up of powerful processing subsystems, each of which is uniquely designed to serve a particular function. In high performance embedded systems, each of these subsystems are often fully functional processing nodes themselves, and the limiting factor when optimizing for the highest system performance often lies in the processor-to-processor data paths. Passing data from processing node to processing node in the most efficient way possible, with the lowest latency and highest throughput, can often have the greatest impact on system performance.

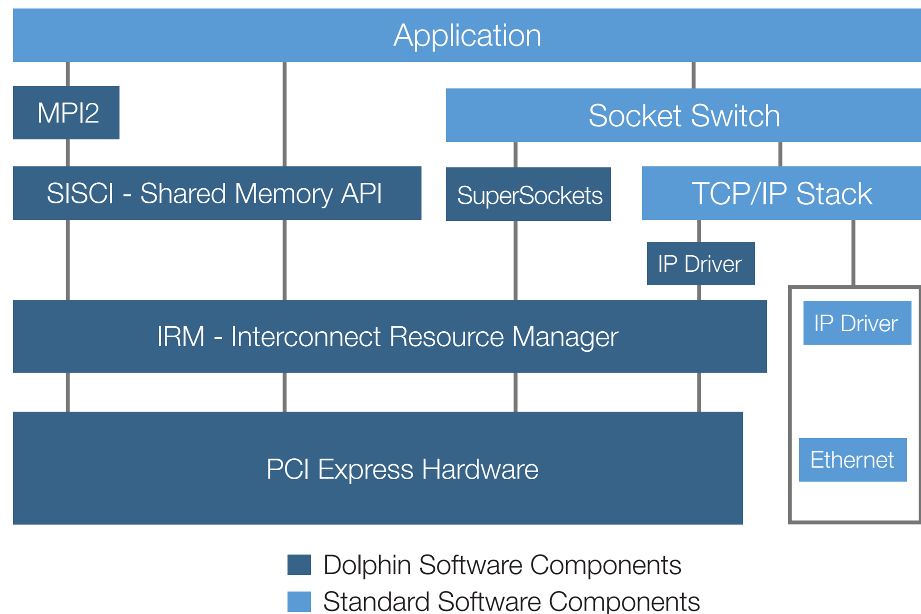


Figure 1: Dolphin eXpressWare Software Model

Curtiss-Wright has partnered with Dolphin Interconnect Solutions to bring their eXpressWare PCI Express® (PCIe) fabric software to the embedded VPX world. Uniquely optimized to take advantage of hardware features such as DMA and multi-core processing, eXpressWare can be used to exploit the highest levels of data fabric performance for the rugged defense industry.

In our first part of this white paper series, we introduced the use of fabrics for high-performance embedded systems, and focused on the hardware and architectural options available to the systems designer when using PCIe as a high performance fabric. In this second part, we present several flexible software interfaces provided for applications development, comparing their advantages and tradeoffs. Finally, we present performance benchmarks using a variety of Curtiss-Wright 3U VPX modules.

Dolphin eXpressWare – A Solution for PCIe Fabric Ease-of-Use

Dolphin Interconnect Solutions has been developing processor-to-processor communications solutions for many years. Hugely successful with their StarFabric technology, they have created eXpressWare, a system for processor-to-processor communications using PCIe connections to create an extremely fast, flexible, and feature rich message and data transfer mechanism.

Multiple Operating Systems, Multiple Processor Architectures

To support the needs of today's most demanding real-time systems, Curtiss-Wright has worked with Dolphin to extend eXpressWare, to include Intel® and Power Architecture® processors for both Linux® and Wind River® VxWorks® operating system.

With pre-tested and optimized support for Curtiss-Wright COTS Single Board Computers (SBCs) and Digital Signal Processing (DSP) engines, eXpressWare offers the most powerful and flexible high-performance fabric interface for today's real-time embedded processing systems.

Hiding Hardware and Complex Configuration of PCIe Interfaces and Switches

As we have learned in Part 1, PCI technology does not directly support host-to-host communications, and requires complex setup and configuration of PCIe devices and switches to enable this functionality. eXpressWare software has been designed to hide the complexities of PCIe setup, greatly simplifying the setup and configuration of these host-to-host architectures. Curtiss-Wright combines eXpressWare with all the predefined PCIe switch configurations which allows eXpressWare to automatically detected and configure PCIe endpoints and transparent or non-transparent ports. In addition, eXpressWare will setup message queues and data transfer windows, as well as configure and manage data transfer resources such as DMA engines. By supporting standard software APIs, eXpressWare aims to accelerate and simplify software development.

Software Models

With root nodes, endpoints devices, transparent and non-transparent ports, programming PCIe fabric communications can be difficult at best. Dolphin eXpressWare software masks the complex details of directly programming PCIe devices while supporting high-speed, low-latency, peer-to-peer communications.

By abstracting the hardware interface, the software developer no longer faces the time consuming task of managing hardware configurations or mastering different networking protocols. As seen in Figure 2, both Dolphin specific and open standard software components build the Dolphin software stack. By supporting a number of common APIs, Dolphin eXpressWare provides the user the flexibility to select the solution that best matches their specific application and performance needs.

At the simplest level, an IP stack driver is supported, offering developers a simple TCP/IP based software interface no different than using regular Ethernet communications. Although the least efficient of the available interfaces, this model permits customers with existing applications using Ethernet communications to convert to PCIe based communications will little to no application software changes.

eXpressWare also supports a unique implementation of the Berkeley Sockets API (BSD) that capitalizes on the PCI Express transport to transparently achieve performance gains for existing socket-based network applications. This interface is called SuperSockets.

Finally, for the highest possible performance, eXpressWare offers an interface called Shared-Memory Cluster Interconnect or SISCI.

SuperSockets

SuperSockets first appeared in 2004 to address time- and data-critical applications. For standard socket-based inter-process communications, SuperSockets offers a safe reliable alternative to the traditional TCP/UDP/IP protocol stack as well as supporting UDP multicast. SuperSockets can accelerate any application that uses generic BSD sockets with no configuration changes since the host names will remain the same. Simply by moving data directly using PCIe instead of Ethernet will typically reduce the minimum latency by a factor of 10 or more. For example, the average latency across the a number of Curtiss-Wright SBC and DSP modules was 1.60 microseconds, with some transfers latencies as low as 1.18 microseconds. In addition to offering lower effective latency, SuperSockets also yield high message

processing rates. Because of the low protocol overhead of SuperSockets on both the CPU and host adapter, SuperSockets ensures low CPU utilization and good scalability as the number of processing cores increases. If your application requires real-time performance, SuperSockets has a mode that does not trigger interrupts during communications, which promotes the greatest timing predictability and the lowest achievable overhead.

Why are SuperSockets so fast? The SuperSockets protocol streamlines the transfer with no need to lock down or register memory. Also, consider that small messages use basic CPU instructions, and that a single store instruction can send 8 bytes of data with a raw worst-case latency of approximately 70 nanoseconds.

For increased fault tolerance and speed, SuperSockets support multiple host adapters per host. This commitment to speed included local loopback acceleration of up to 10 times faster than the standard Linux loopback device.

SuperSockets supports all these valuable features in both user and kernel space applications. SuperSockets can also accelerate kernel services that employ sockets; however, these services will need to be modified to take advantage of SuperSockets. Typically, implemented by either re-configuring the service or patching and recompiling the code, this trivial modification will specify a different address range when opening sockets.

Delivered as a Linux independent binary object, the SuperSockets code does not interface with, or use, any Linux functionality directly. Drivers that interface directly with Linux are compiled from source during installation to match the running kernel, and the SuperSockets package consists of both kernel modules and a user-space library.

The implementation of SuperSockets at the kernel level ensures full compatibility with the TCP/UDP/IP and RDS datagram sockets that already exist in the operating systems. Operating between the unmodified binary of the user's applications and the operating system, the explicitly, preloaded, user space SuperSockets library intercepts any socket-related function calls.

Automatic Fail-Over

Depending on the system or the user's configuration, if set to override the system configuration, the library will pass the function call to either the SuperSockets or the standard socket implementation. With the selection of SuperSockets, the kernel module then performs the transfer using the PCIe interconnect. If a network problem is detected, SuperSockets will automatically and transparently switch to the standard socket implementation even while the socket is passing data. When the PCIe connection is restored, it will switch back to using SuperSockets. In addition, SuperSockets will revert to the Ethernet port when connecting to nodes outside of the cluster.

SISCI

The EU-funded Esprit Project 23714, "Standard Software Infrastructures for SCI-based Parallel Systems" created the Software Infrastructure for Shared-Memory Cluster Interconnect (SISCI) API. The purpose of this project was to encourage the development of software for parallel processing on clusters of workstations connected by a fast "memory mapped" interconnect initially called the Scalable Coherent Interface (SCI). The SISCI API supports data transfers between CPU memories and IO devices using either distributed remote memory access or Direct Memory Access (DMA). In addition, users can trigger remote interrupts as well as catch and handle events from the underlying interconnect. The SISCI API protects system security by preventing software from acting badly and accessing remote memory outside of exported SISCI segments. Without adding overhead or performance penalties, the user can write portable applications to communicate across both little and big endian systems. Like SuperSockets, the SISCI library and tools are available in both user and kernel space.

Understanding of the "resource" concept is essential to successfully using the SISCI API. For example, a virtual device is a resource, a memory segment is a resource, a DMA queue is a resource, etc., and a resource can depend on another resource. A descriptor collects the properties that described a resource. While the user does not have direct access to the descriptor, the user controls the descriptor handler that are used in conjunction with the API functions. Fortunately, the naming conventions for the descriptors and handles relate to the name of the underlying resource. For example, the resource of a local memory segment has a descriptor named "sci_local_segment" with a matching "sci_local_segment_t" handle to as local segment resource.

Memory

Safely accessing memory physically resident on another machine is the fundamental characteristic and strength of the SISCI software. When remote memory is mapped into addressable space of a local process, the remote memory appears to be local and any data transfer becomes as simple as a normal "memcpy()". This transfer method is called Programmed I/O (PIO) and the "memcpy()"

equivalent function is implemented as a sequence of CPU load and store instructions that sends and retrieves data from remote memory. PIO has the lowest overhead and lowest latency of any of the SISCI methods for accessing remote memory, but the drawback is that the CPU is consumed reading and writing data to and from remote memory. For example, on one set of Curtiss-Wright test boards connected via PCIe, the one-way latency was measured in the range from 0.54 microseconds to 0.7 microseconds.

Instead of allocating memory using standard functions such as “malloc()”, a memory segment on the local host is allocated with a custom function because the driver must be aware of the segment and its associated parameters. By using the custom functions, implementation requirements such as the memory being non-swappable and physically contiguous are hidden from the user, which helps ensures portability. Upon successful memory allocation of the local segment, the function returns the local segment resource, which includes a segment id, to the user. Let us assume that one local segment memory has been allocated on a node designated at the receiver node. Using the local segment resource, the receiver makes that block of memory available to the rest of the system. Now, another node, the sender node, wants to access that memory segment on the receiver node. The sender node’s first step is to “connect” to the remote memory segment on the receiver node. A connect request is sent containing the receiver’s node id and the id of the memory segment as well as other parameters. The request returns a handle to remote segment resource. At this point, the size of the remote segment can be determined with another function call. Once a valid resource handle is available, the memory segment can be mapped into the address space of your process and accessed like regular memory (i.e. pointer operations).

DMA

As an alternative method to the PIO data transfers, the SISCI API also provides for Direct Memory Access (DMA) transfers when it is available on your hardware. DMA functionality is only implemented on certain hardware platforms, such as the Curtiss-Wright CHAMP-XD1 (VPX3-482) and Power Architecture SBCs (VPX3-131/133). The user’s application defines the desired data transfers and the CPU in turn passes the information to the DMA engine. This frees the CPU to continue processing in parallel with the transfer, or the CPU can just wait for the transfer to complete. If the CPU continues processing, the application can specify a callback function to be invoked when the transfer is complete. DMA operations have a high startup cost compared to PIO and should only be invoked for larger data movements. Another option would be to join several DMA transfers together to amortize the overhead. Sometimes, PIO and DMA operations can work together for maximum application benefits and performance. DMA programming requires the same code sequence to setup local and remote memory segments as described for PIO transfers.

So how do you know when the DMA transfer is complete? For synchronous behavior, the CPU can just spin and wait. Other than wasting precious CPU time, the transfer could error and the CPU could be waiting forever. To avoid this lockup condition, simply set a timeout value as part of the call. Instead of waiting for the completion of the DMA transfer, the CPU can poll at selected intervals until the transfers completes or returns an error. Using the DMA queue, the application can also start multiple transfers from one API call. In addition, the SISCI API provides functionality for Direct Remote DMA (RDMA).

Interrupts

Interrupts notify a remote application when a predefined condition occurs. Like memory segments, a SISCI interrupt is a resource and is allocated on one node and connected to, and used from, one another. These interrupts can also pass data as part of the interrupt; however, interrupts with data normally consume more resources and incur more latency. To start, the local node allocates, initializes and makes available an interrupt resource. By default, the create interrupt function returns an identifier to the interrupt that the remote application must use to trigger the interrupt. The local node must share the identifier with the remote node by some method, usually by shared memory. To avoid the extra step of passing the identifier, the local node declares a constant interrupt number that is passed to the interrupt creation routine. If using this method, the user must ensure the same interrupt number is defined on both the send and receive nodes. Similar to memory segments, the remote node has to connect to the remote interrupt and receive the handle to the interrupt resource. Once the remote node has the handle, it can trigger the interrupt as needed. Given that the application on the local is waiting or has set a callback, the interrupt will be handled; otherwise, the interrupt will be lost. An error occurs when the timeout expires, or if some other thread removes the interrupt.

Events and Callbacks

Hardware, drivers, and even the application can generate events. Some examples of an event include a cable disconnection, a node failure resulting in the disappearance of a remote segment, the completion of a DMA transfer or an interrupt. The SISCI driver handles some events directly while forwarding other events to the application, which makes the choice to ignore or handle them. As mentioned above, the application can block an event or setup a callback. To setup a callback, the mechanics are the same whether the function is creating a local memory segment, connecting a remote section, starting DMA transfer or creating an interrupt. In all instances, the setup function has parameters that include a flag indicating the intention to use the callback mechanism, a pointer to the callback function and the arguments for the callback function. Defining the argument parameter as "void*" allows anything from nothing, to a single value to a pointer to a larger data structure to be passed to it.

Multicast

Multicast, sometimes referred to as reflective memory, transfers the same data to multiple remote nodes. Since multicast is implemented in hardware, no software overhead is incurred. Multicasting of the data buffers can use PIO, DMA, or direct data moves from PCIe devices, such as GPUs and FPGAs. The PCIe multicast uses main system memory, which is considerably faster than specialized device memory. Because main memory is cached, data updates from remote nodes will automatically invalidate the CPU's cache, which helpfully guarantees data consistency. The ability to use up to four independent reflective memory segments, and the selection of which nodes that will receive the multicast data, are a few of the strengths of Dolphin's implementation.

Another major differentiator is the utilization of two different addresses, one for reading and another for writing. The PCIe multicast process distributes the entire PCIe bandwidth simultaneously to all remote nodes. Ping-pong testing performed on a 2-8 nodes of different Curtiss-Wright boards resulted in an average 1.98 microseconds latency, along with very low jitter. One way latency for three nodes measured at just 0.99 microseconds, with eight nodes resulting in 1.27 microsecond latency. The timing was recorded when all the remote nodes replied with an ACK after receiving the data.

Note: multicast is only supported on systems configured with a central switch.

Remote Peer-to-Peer

PCIe peer-to-peer (P2P) communications enables regular PCIe devices to perform direct data transfers without using main memory as temporary storage and without using the CPU to move the data. P2P reduces latency and communication overhead and typically benefits GPUs, FPGA, and high-speed data input devices. The SISCI API simplifies the setup and management of P2P transfers, and the P2P functionality can be combined with the reflective memory functionality to multicast data to multiple devices transparently.

Each hardware resource on the PCIe fabric must be mapped to the controlling application with the appropriate SISCI API functions. First, the user should specify the physical address and the number of bytes inside the PCIe device that will form the SISCI segment. After the segment is prepared, a remote host can connect and map the physical memory. To enable a local PCIe device to access a remote SISCI memory or remote device segment, the code must retrieve the corresponding I/O address in the local address space. This information will be available via a function call after the remote segment has been connected and mapped. To ensure the master access is passing through the required NT mapping function, the PCIe device must register as an approved PCIe master. For example, an FPGA device that is setup as a PCIe master can direct memory using the address provided.

Picking the right API: Tradeoffs and Performance Considerations

For applications that are already written using socket communications, the use of TCP/IP sockets represents the lowest risk and shortest software development effort. This API incurs the highest software overhead, and results in the lowest performance of the available eXpressWare software APIs. Systems moving from Ethernet to PCIe communications will still benefit from an increase in overall performance.

SuperSockets increases performance dramatically. Although some software rework will be required, most well designed software can be adapted to use SuperSockets with relative ease, and the performance benefits will be seen quickly.

The highest performance API is the SISCI interface. New applications using PCIe fabric communications should be written with the SISCI API, which will achieve the highest possible performance of all eXpressWare APIs.

Curtiss-Wright PCIe Fabric Implementation with Dolphin eXpressWare

Performance Considerations

PCIe speeds, Lane Widths, and Throughput

Although there are many combinations of PCIe connections, the most common and practical connections for a 3U VPX module are Gen2 x4 and x8 lane, and Gen3 x4 and x8 lane.In this section, these are the PCIe transfer speeds and lane widths used for benchmark examples.

Transferring Data: Simple R/W vs. DMA

Endpoints connected with PCIe are configured with memory addressable windows, permitting simple communications using simple processor read and write operations. A processor can write data directly to the PCIe mapped memory address, and endpoint device receives the data via PCIe transactions.

This form of communications is sometimes called Processor I/O or PIO mode. It is extremely simple to write software using this data passing mechanism within an application, however it will clearly use processor CPU cycles. If the processor has DMA capabilities, the DMA engine can be used to transfer data across the PCIe bus with block transfer read/write operations, freeing up precious CPU cycles.

Intel processors have a unique feature called PCIe write combining, where sequential write operations are combined into a single PCIe burst write operation. This results in extremely high bus utilization and data throughput.

Optimization

The eXpressWare software has been optimized to operate in PIO or in DMA mode, depending on the type and size of data being transferred. For small size messages, PIO mode is extremely efficient, with the lowest possible latency and fast data transfers. For large block data transfers, the DMA mode of operation can be used to free up CPU resources.

eXpressWare is smart enough to select PIO or DMA modes of operation dynamically, making the best use of hardware resources for every type of data transaction. Small data transactions would be inefficient if extra cycles are spent setting up DMA operations. In many case, especially for short messages, the entire message can be transferred with simple PIO operations before a single DMA cycle would have been executed.

Supported Curtiss-Wright Modules

The following Curtiss-Wright modules are presently supported by the eXpressWare PCIe Fabric Software:

- VPX3-1220 = Intel Xeon® 7th Gen “Kaby Lake” SBC
- VPX3-1259 = Intel Core i7 5th Gen “Broadwell” SBC
- VPX3-1260 = Intel Xeon 8th Gen “Coffee Lake” SBC
- VPX3-131 = NXP P4080 SBC
- VPX3-133 = NXP T2080 SBC
- VPX3-482 = CHAMP-XD1 Intel Xeon D DSP Engine
- XMC-121 = Intel Xeon 7th Gen “Kaby Lake” XMC Mezzanine

The eXpressWare PCIe Fabric Software has been optimized to make use of available hardware resources on each of these modules. The VPX3-1220, VPX3-1259, and VPX3-1260 with Intel Core i7 and Xeon processors do not have high performance DMA capabilities, and thus operate in PIO mode exclusively. On these modules, eXpressWare has been optimized to leverage Intel’s PCIe write combining to achieves the highest possible PCIe bus utilization. The VPX3-131/133 and CHAMP-XD1 modules do support DMA operations, and automatically select PIO or DMA operations for each transaction to maximize throughput and minimize processor overhead.

Performance - Data Throughput and Latency

Curtiss-Wright has performed extensive testing and benchmarks with the eXpressWare PCIe Fabric software under various combinations of modules and operating systems using both PIO and DMA operations. This section summarizes some of these benchmarks.

CHAMP-XD1 to CHAMP-XD1 under Linux

Figure 2 shows the performance of two CHAMP-XD1 Intel Xeon-D DSPs via a 4-lane PCIe Gen3 interface which sports a maximum theoretical bandwidth of 3.94 GBps. For this test, both XD1s were running Linux and the data transfers tested use the PIO mode of operation for comparison to the SBCs such as the VXP3-1259. As expected, the shape of the graphs are very similar with the increase of performance due to Gen3 interface noted.

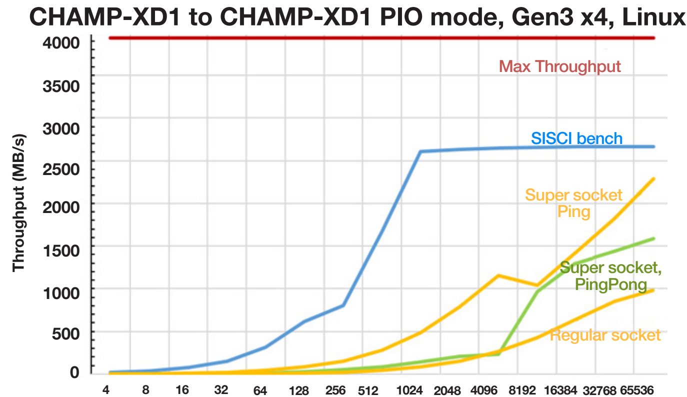
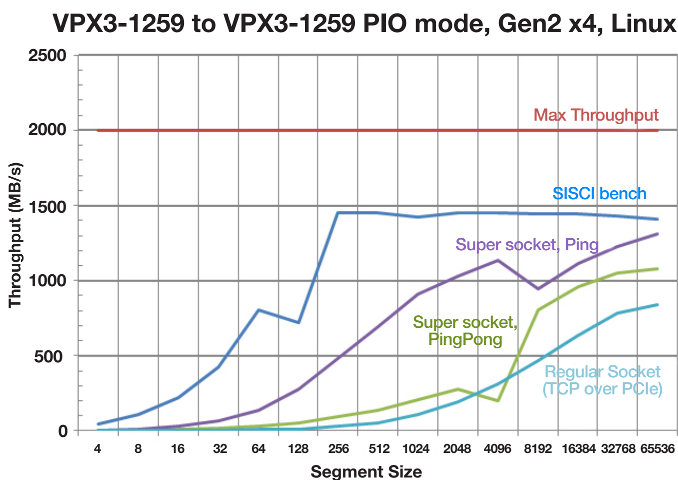


Figure 2: CHAMP-XD1 to CHAMP-XD1 under Linux, PIO Mode Performance

VPX3-1259 to VPX3-1259 under Linux

Figure 3 shows the performance of two VPX3-1259 Intel Core i7 Broadwell SBCs connected with a 4-lane PCIe Gen2 interface, which has a maximum theoretical bandwidth of 2.0 GBps. In this benchmark, both SBCs are running Linux. All data transfers are via PIO mode of operation, as a high performance DMA engine is not available in the Core i7 chipset.



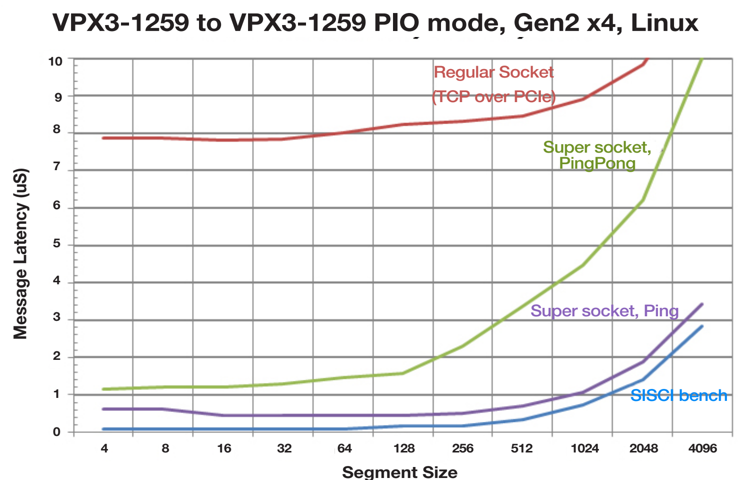
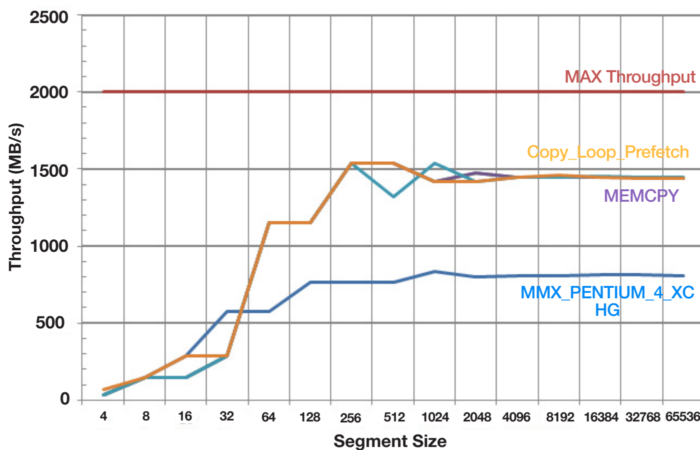


Figure 3: VPX3-1259 to VPX3-1259 under Linux, PIO Mode Performance

Data throughput and latency are shown, and as expected, the SISCI API exhibits the highest performance and lowest latency of all software APIs. Throughput quickly reaches a peak of 1.45 GBps at a data transfer size of 256 bytes, representing 73% of theoretical PCIe bus performance. Larger data transfer sizes do not achieve higher throughput, as the PCIe bus itself limits transactions to 128 or 256 bytes (chipset dependent), so even larger transfers are limited by the PCIe bus transaction process. SuperSocket protocol throughput is not as efficient as the SISCI protocol, and standard TCP socket throughput is even lower, reaching a maximum of only 0.84 GBps even for large 64K size packets. This is due to the Linux software stack that handles regular sockets, which is not as efficient as the SISCI or SuperSocket drivers. Message latency is also shown. Using SISCI, a messages of 1 KB has a latency of approximately 0.72 microseconds. SuperSocket latency is slightly higher at 1.07 microseconds and the SuperSocket PingPong latency, transferring messages in both directions, makes the round trip in just 4.46 microseconds. Regular TCP socket interface latency is much higher for all size packets, primarily due to socket software stack execution, requiring almost 8 microseconds for even the shortest of messages.

VPX3-1259 to VPX3-1259 under VxWorks

Figure 4 shows the same two VPX3-1259 SBCs connected with the same 4-lane PCIe Gen2 interface, but this time running the VxWorks operating system. Data throughput peaks at 1.54 GBps, or 77% of theoretical PCIe bus performance, slightly higher than the same configuration under Linux. This is due to the better real-time performance of VxWorks over Linux. Latency is also shown, with higher performance (lower latency) than Linux. A 256 byte messages transfers in as low as 0.18 microseconds.



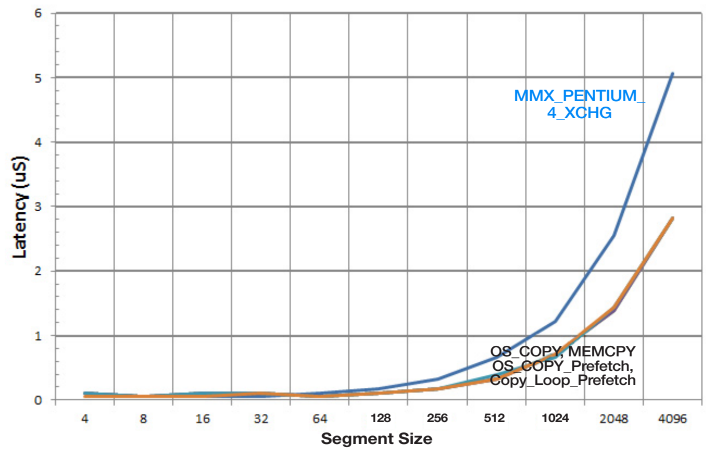


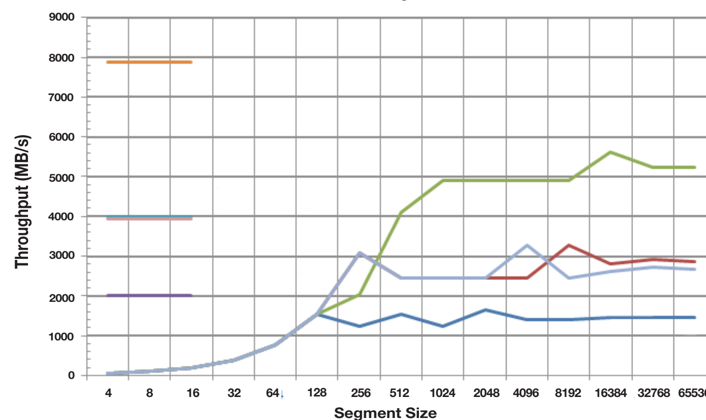
Figure 4: VPX3-1259 to VPX3-1259 under VxWorks, PIO Mode Performance

CURTISS-WRIGHT

VPX3-1259 to VPX3-1259 under VxWorks, PCIe Lane Widths and Speeds

The effects of wider PCIe lane widths (ie: 8-lane .vs. 4-lane) and higher PCIe transfer speeds (ie: Gen3 .vs. Gen2) are shown in Figure 5. All these benchmarks are using the SISCI API under VxWorks.

As expected, wider lane widths and higher PCIe bus speeds produce better results. Using an 8-lane PCIe Gen3 interface, the two SBCs achieve data transfer rates of up to 5.62 GBps, or 71% of the theoretical 7.88 GBps bandwidth of this PCIe connection. Interestingly, for data transfers under 128 bytes, all three configurations have the same data throughput. This is due to the way the Intel PCIe controller stores up PCIe data transactions and then bursts data onto the PCIe bus. Data throughput of a 4-lane Gen3 and an 8-lane Gen2 interface are almost identical, as is their theoretical maximums. Similarly, latency is almost identical for transfer lengths up to 128 bytes, and then the effect of faster data transfers begins to show with larger data transfers.



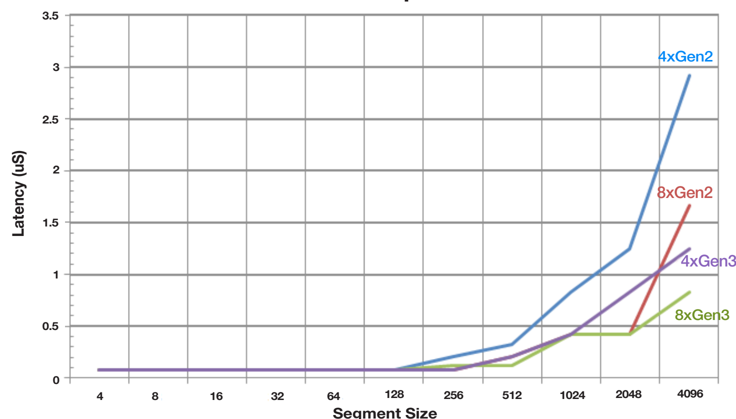
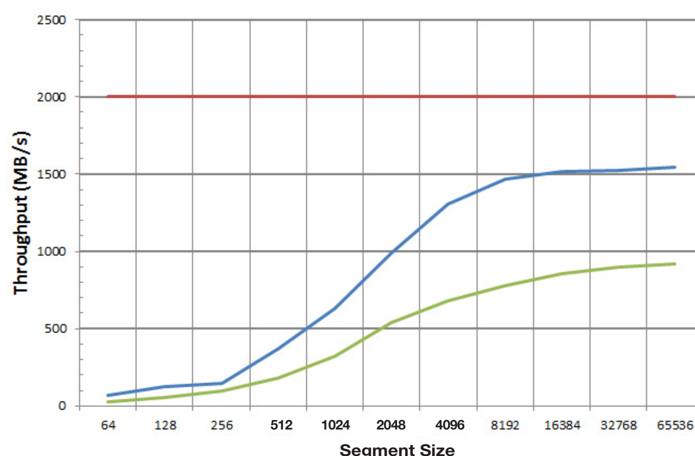


Figure 5: VPX3-1259 to VPX3-1259 under VxWorks, Effect of PCIe Lane Widths and Speeds

VPX3-133 to VPX3-133 with DMA under Linux

Figure 6 shows two VPX3-133 Power Architecture T2080 SBCs communicating, again using 4-lane PCIe Gen2 interfaces with maximum theoretical performance of 2.0 GB/s. DMA performance is slightly higher than the Intel Core i7 (1258) PIO performance, hitting a peak of 1.54 GBps (77%). This demonstrates that PIO and DMA modes both achieve similar throughput, thus we can conclude the maximum throughput is not limited not by the processor transfer mechanism. Not shown is the CPU overhead for these two modes of data transfer, where we would see the PIO mode of the Intel Core i7 consuming CPU cycles, and the DMA mode of the Power Architecture T2080 leaves the CPU sitting idle, free to perform other tasks or waiting for DMA operations to be complete.

Latency with these Power Architecture SBCs is slightly higher than the Intel Core i7 SBCs, taking 1.75 microseconds to push a 256 byte message to the other host, most likely due to the lower overall CPU performance of these Power Architecture processors.



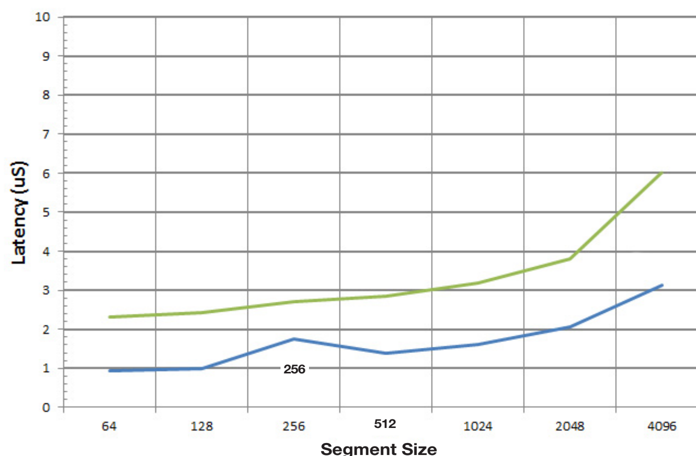


Figure 6: VPX3-133 to VPX3-133 under Linux, DMA Mode Performance

CURTISS-WRIGHT

Author(s)

Aaron Frank, BaSC
Senior Product Manager
Curtiss-Wright Defense Solutions

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

Summary

Rugged embedded systems depend on high performance fabrics to reduce latency in data transfer times. In this second part of our Dolphin white paper series, we discussed several different software interfaces provided for applications development, compared their advantages and tradeoffs. Dolphin's support of a number of common software APIs offers high-speed, low-latency, peer-to-peer communications while masking the complex details of programming PCIe devices. Curtiss-Wright's partnership with Dolphin provides our customers with access to all these benefits on our embedded hardware, enabling them the flexibility to select a solution tailored to their own unique application and performance needs. In part three of this series, we will take an in-depth look at device sharing and multicast applications.

Learn More

Product Sheets

- Dolphin PCIe Fabric Communications Library
- OpenHPEC Accelerator Suite
- VPX3-133 3U VPX SBC
- VPX3-131 3U VPX SBC with NXP Power Architecture P4080 Processor
- VPX3-1220 3U VPX SBC with Intel Xeon 7th Gen Processor
- VPX3-1259 3U VPX SBC with Intel Core i7 Broadwell Processor
- VPX3-1260 3U VPX SBC with Intel Xeon 8th Gen Processor
- CHAMP-XD1/VPX3-482 3U VPX DSP with Intel Xeon D Processor
- XMC-121 3U XMC Mezzanine with Intel Xeon 7th Gen Processor

White Paper

- Enhancing PCIe® Communications to Eliminate Bottlenecks with Dolphin PCIe Fabric Communications Library

TRUSTED PROVEN LEADER
CURTISSWRIGHTDS.COM

© 2020 Curtiss-Wright. All rights reserved. Specifications are subject to change without notice. All trademarks are property of their respective owners. I W098.0220