Avoid Bottlenecks Using PCI Express-Based Embedded Systems

Jim Henderson

Innovative Integration

... real time solutions!

Avoid Bottlenecks Using PCI Express-Based Embedded Systems

Implementing efficient data movement is a critical element in high-performance embedded systems, and the advent of PCI Express has presented us with some very effective approaches to optimize data flow.

BY JIM HENDERSON, INNOVATIVE INTEGRATION

“A good friend will help you move, but a trusted friend will help you move the bodies.” Fortunately, I haven’t any direct experience to substantiate this old quip, but I chuckle at the premise. Embedded system architects occasionally need help moving as well, and for this we turn to our trusted friends DMA, bus-mastering, shared memory and coprocessing.

Embedded systems are ubiquitous, used within communications installations, automobiles, cell phones and even table saws. Most systems interact with their environment through transducers connected to analog-to-digital converters (ADCs/DACs), communications ports such as USB and Ethernet, and myriad other specialized devices. It is commonplace for such devices to generate or consume gigabytes per second, well beyond the capacity of a typical embedded processor to manipulate directly. Usually, this problem is addressed by including some form of coprocessing or direct memory access (DMA) within the embedded system.

For example, consider the narrowband receiver inside a software radio depicted in Figure 1. A wideband IF signal is digitized at 250 MHz, with 16-bit resolution, down-converting to produce 12.5 MHz of narrowband data. This operation might be implemented using a COTS device such as the TI GC6016 DDC or using custom VHDL firmware within an FPGA, such as a Xilinx Virtex 6. Regardless, the output data rate is substantially reduced—the complex, 16-bit output data samples are produced at 12.5 MHz, equivalent to just 50 Mbyte/s. However, even this mitigated rate may represent a substantial burden to an embedded processor. It might be necessary to implement a decoder to bring the signal to baseband, reducing the data rate by another order of magnitude before that rate is suitable for the embedded controller. In this scenario, the efficiency of data flow is improved by delegating the per-sample manipulations to the coprocessor, reducing the bandwidth from 500 to just 5 Mbyte/s. The bandwidth of the embedded computer’s bus is preserved by reducing its load.

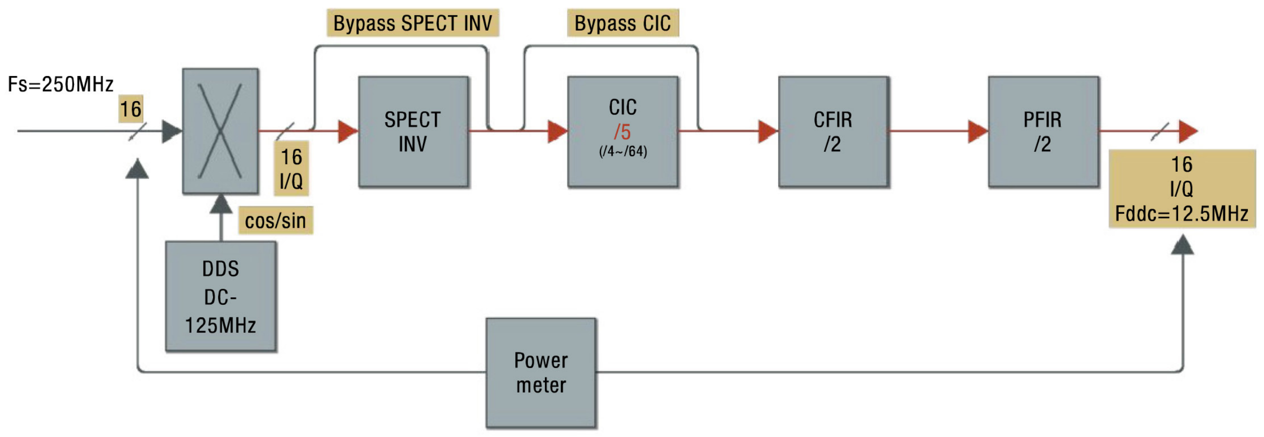

Figure 1
Digital Down Converter implementation.

But in some situations, coprocessing is not feasible or desirable. Implementing the necessary coprocessing functionality might exceed available FPGA resources. Or, such computations may be deferred or implemented on another system in non-real-time. For example, it is commonplace to capture wideband recordings of signals to disk, subsequently analyzing the waveforms in environments such as Matlab. Multipath, Doppler fading and other impairments on a cellular transmission might be captured by a portable 250 MHz IF recorder mounted in the trunk of a car driving throughout a city. These real-world signals could subsequently be used to stimulate prototype receiver equipment in a lab. What architecture is required to implement such a system at these rates?

Direct Memory Access (DMA) is a feature of many embedded systems that allows I/O subsystems to access memory-mapped peripherals independently from the embedded CPU, microcontroller or digital signal processor. Once initialized, a DMA controller autonomously reads from source memory addresses and writes to destination addresses. Typically, DMA controllers initiate data movement when signaled by an external stimulus signal, such as an FIFO level signal indicating that data can be read or written to a device. Many modern DMA controllers provide substantial addressing flexibility, allowing transactions to/from fixed addresses (memory mapped FIFOs), address ranges (shared memory) and even non-contiguous ranges (image buffers). Since DMA responds to the stimulus signal independently from the CPU and is granted higher priority than the CPU in accessing the bus, DMA transactions are a mainstay for reliable, deterministic data flow in real-time applications.

Nearly all desktop PCs and many embedded systems now incorporate PCI Express (PCIe) as the communications bus between I/O cards and host system memory. PCIe is a standard, point-to-point serial implementation that has displaced the earlier (parallel) PCI standard that was a dominant expansion card interface used until circa 2005. PCIe gained market acceptance very rapidly, primarily due to the signal routing density intrinsic to its serial implementation and its correspondingly scalable performance. PCIe traffic flows in lanes, each implemented as a matched pair of differential signals implementing a bidirectional data path.

Data flow is 8/10B encoded at a bit rate of 2.5 Gbit/s in the PCIe v1 standard, providing a rated 200 Mbyte/s for each lane. PCIe v2-compliant hardware doubles that bit rate to 5.0 Gbit/s, doubling throughput. To support high bandwidth devices, lanes may be bonded to improve throughput. For example, graphics adapters in desktop PCs often bond sixteen v2-compliant lanes to establish an effective 6400 Mbyte/s bidirectional link between system memory and the GPU (graphics processor unit).

Use of PCIe is not restricted to the desktop form factor. VITA Standard 42 combines the popular PCI Mezzanine Card (PMC) format with the PCIe serial fabric technology to support standardized PCIe peripheral cards in embedded applications, known as XMC modules (Figure 2). Likewise, VITA Standard 46 supports use of PCIe within VPX platforms, the popular rugged successor to VME (Figure 3).

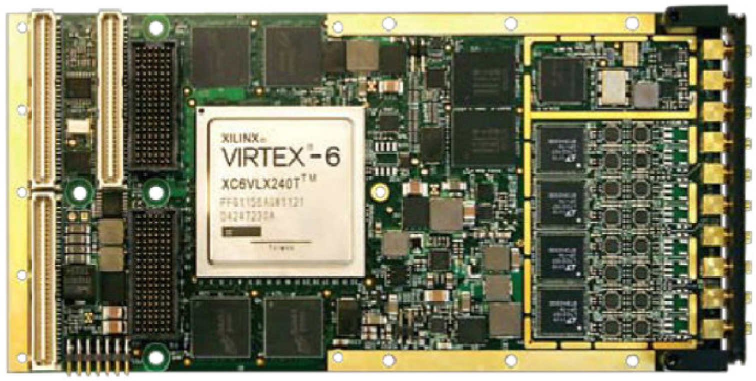

Figure 2
XMC Module with FPGA and Analog I/O.

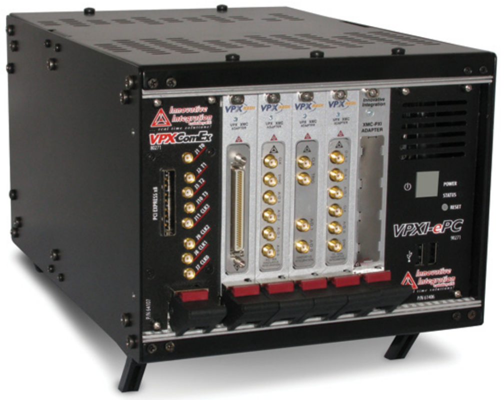

Figure 3
Embedded VPX System.

To effect a transfer, a PCIe device first arbitrates with the host processor(s) for memory bus ownership. Once granted, the master writes or reads small packets of 32- or 64-bit words to another memory-mapped peripheral, the slave. Packet sizes are governed by cache memories within the PCIe chipset. These are typically less than 2K words in size, although they are frequently much smaller on the chipsets found in embedded systems. The slave device accepts or rejects the packet by emitting an acknowledgement or negative acknowledgement, respectively. This protocol allows for flow control if data is being sourced too rapidly for the slave by the master.

PCIe cards typically implement embedded data movement engines that implement bus mastering—a specialized form of DMA transfer. Bus mastering offers several advantages compared to using a legacy DMA controller. While both DMA and bus mastering arbitrate with the host processor for access to the host memory bus (exhibiting similar transaction latency), each DMA transaction involves both a bus read and write. However, a bus mastering peripheral can access onboard I/O devices via a local bus, so a transfer to system memory or another memory-mapped PCIe peripheral requires only a bus write operation, effectively doubling the efficiency of DMA.

Bus mastering transfers occur between a PCIe device and another memory-mapped peripheral. But the latter need not be memory, per se. For instance, one PCIe device may transfer directly to another, bypassing host system memory. PCIe systems can incorporate switch devices that allow multiple attached devices to communicate locally, mitigating traffic on the primary system bus. Traffic between such devices behaves as if interconnected directly. Switch devices are typically configured either statically from flash ROM during system initialization or dynamically by the host processor during OS initialization. Once configured, even persistent PCIe traffic between local devices will consume no system bus bandwidth.

The high-speed cellular data storage application referenced earlier seems a suitable candidate to exploit inter-device PCIe communications. One would assume that a high-speed PCIe digitizer could bus master directly to a RAID disk controller to implement wideband recording, without loading the host system bus or processor. While local, inter-board communications is exploited frequently in VPX platforms to share data between two FPGA cards, it is rarely feasible when mixing PCIe devices from multiple vendors. Most RAID controllers, for instance, are closed designs provided with drivers only for popular operating systems such as Windows or Linux. These drivers perform all low-level access to the array and are the exclusive vehicle for interacting with the array. The manufacturers do not document or support use of the RAID controller as a slave device.

The zero work solution is to accept the implicit inefficiency of a single bus mastering transfer from the source PCIe device to host system memory. If acquiring at a relatively modest rate like 250 MHz from 16-bit ADC devices, this represents a load of only 1000 Mbyte/s on the system bus (500 Mbyte/s for the BM from the acquisition device to system memory, and another 500 Mbyte/s when the RAID controller BMs from system memory to the RAID array), which is well within the available bandwidth of modern motherboards and RAID subsystems.

But if the data rate escalates, this approach becomes increasingly unattractive. A typical i7-based embedded system will provide approximately 6-8 Gbyte/s of aggregate system memory bandwidth. If the acquisition card samples at 2 Gsamples/s at 12-bit resolution with no packing, the system bus utilization becomes an unmanageable 8000 Mbyte/s for the real-time storage to the array. Interestingly, CPU utilization is nearly zero—the CPU is involved only occasionally to initiate a write of data to the array when the acquisition device completes each bus mastering transfer. But the miniscule CPU usage is irrelevant in this application; We're bus-bound.

To address this problem, it is necessary to redesign the interface on the acquisition device. Instead of using a bus mastering interface in which the module autonomously delivers data into host system memory—consuming host memory bus bandwidth—available memory on the acquisition card must be mapped as shared memory. Acquired samples from the ADC on the card are stored into consecutive memory locations in onboard memory, similar to a FIFO. But these memory locations are also mapped into the memory space of the host, and appear as a memory bank addressable by the host CPU and the RAID controller. The key advantage of this approach is that during acquisition, samples will appear in the memory space of the read-only memory, without requiring a memory bus arbitration or write cycles. Effectively, the data transfer is free. Of course, when the RAID controller reads from this memory area, the bus will be utilized normally and a fraction of the available bus bandwidth will be consumed. However, the utilization is halved, compared to the bus-master scenario described earlier.

On modern acquisition cards that employ reprogrammable FPGAs, this sort of repurposing is relatively easy to implement. What a difference compared to the cut and jumper world in which we used to live!