

# Extending SPI4.2 capabilities for Ethernet services

WITH THE PROLIFERATION OF INTERNET PROTOCOL-BASED SYSTEMS IN THE TELECOMMUNICATIONS MARKET, DESIGNERS ARE TURNING TO FPGAs TO CREATE INTELLIGENT ETHERNET BRIDGES AND TRAFFIC MANAGERS.

As carriers and cable providers start delivering integrated video, voice, and data over a common broadband infrastructure to their customers, OEMs are increasing their efforts to roll out IP (Internet Protocol)-based systems, including passive-optical networks, cable-modem-termination systems, DSLAMs (digital-subscriber-line-access multiplexers), multiservice switches, and other access and back-haul equipment. The underlying physical layer for this equipment is the ubiquitous Ethernet technology. The system-level components in next-generation Ethernet-service cards are framers, NPUs (network-processing units), and off-the-shelf Ethernet switches—leaving design engineers with the challenge of developing a programmable bridge between the parts, fitting the solution, and cost-effectively implementing it.

Line cards with network processors and framers often use SPI4.2 (system-packet interface, Level 4, Phase 2). Although the specification addresses the challenge of achieving fast, low-latency, point-to-point, 10-Gbit physical connectivity, it leaves the user to implement efficient buffer-management schemes as they relate to the system design. The consequences of the user's decisions affect system-bandwidth efficiency and are concerns to system vendors, which must demonstrate the suitability of an all-IP network to service providers seeking the determinism, reliability, and SLA (service-level-agreement) guarantees of SONET (synchronous-optical-networking) and SDH (synchronous-digital-hierarchy) multiplexing protocols and ATM (asynchronous-transfer-mode)-based systems. These service providers also desire the ubiquity and reduced development, capital-equipment, and operational costs of Ethernet.

## SPI4.2-BASED BRIDGES

An FPGA can play a host of roles, but, in its most basic form, it is a programmable gasket or bridge between the framer and the NPU or between the NPU and the carrier-class-Ethernet switch (Figure 1). There are a number of must-have components in this context.

The first component is an

XAUI (10-Gbit attachment-unit interface) that supports class-of-service and port-switching-information overlays. These interfaces need to run above and beyond the IEEE 802.3-mandated rates of 3.125 Gbps to maintain a 10-Gbps line rate. One example of this interface is the proprietary Broadcom HiGig+ interface, which runs at 3.75 Gbps. Conversely, if the system is aggregating multiple lanes of 1-GbE (gigabit-Ethernet) or 10/100-GbE streams, the best choice is usually multiple 1000BaseX-compliant interfaces in the form of IEEE 802.3z, a GbE interface, or SGMII (serial-gigabit media-independent interface).

The second must-have component is a fully compliant 10-GbE interface or multiple triple-speed MACs (media-access controllers) that can add, strip, and process-overlay the headers.

The third necessary component is the bridge, which is responsible for accepting packets in the ingress direction, ensuring that transfers occur to the Ethernet domain under the right conditions—for example, throttling during flow control. In the egress direction, the system presents traffic depending on the condition of the SPI4.2 status channel. The user requests Ethernet flow control based on the system criteria. The bridge also performs bus translation in both directions.

Finally, for control, a system bus monitors status, provides program registers for user-controlled options, and provides interrupt support.

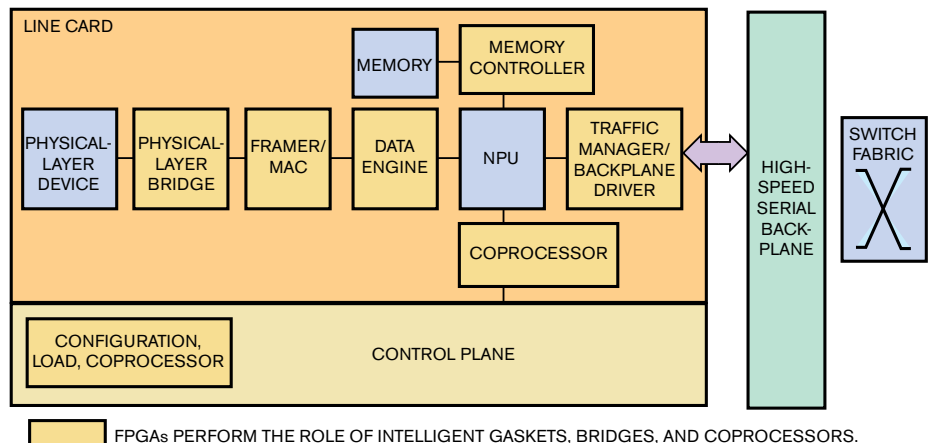


Figure 1 An FPGA can play a host of roles, but, in its most basic form, it is a programmable gasket or bridge between the framer and the NPU or between the NPU and the carrier-class-Ethernet switch.

You can find these attributes in either off-the-shelf ASSPs (application-specific standard products) or FPGAs. However, FPGAs provide several additional benefits. First, they allow the user to instantiate multiple bridges into a single monolithic device. Because carrier-Ethernet systems require more determinism and quality-of-service guarantees than the traditional IEEE 802.3 specification provides, switch vendors are taking proprietary approaches to provide flow control, channelization, interleaving, and OAM (operation/administration/maintenance). With each generation of products, switch vendors are moving away from traditional Ethernet and adding their secret sauces, making FPGAs ideal for keeping pace with burgeoning proprietary interfaces.

One of the biggest advantages of programmability is that the customer can define the bandwidth-management and -provisioning capabilities that best suit the nuance of the end application. An ASSP or ASIC would have to support multiple schemes to be a truly catchall product. However, a low-cost FPGA can implement the scheme the user needs, eliminating the unnecessary power, real estate, and cost of ASSPs and ASICs.

### SPI4.2 BANDWIDTH AND BUFFER MANAGEMENT

Many bridging applications have packets arriving at full 10-Gbps rates over XAUI, for example, but may need to play out at a much slower rate for a given channel on the SPI4.2 side—155 Mbps, for example. The system must also support dynamic channel re-provisioning so that you can add or subtract individual subscriber lines or bandwidth to and from a channel while a system is live without disrupting service on the other channels. It is best to implement a suitable buffer manager that resides outside the basic SPI4.2 datapath and

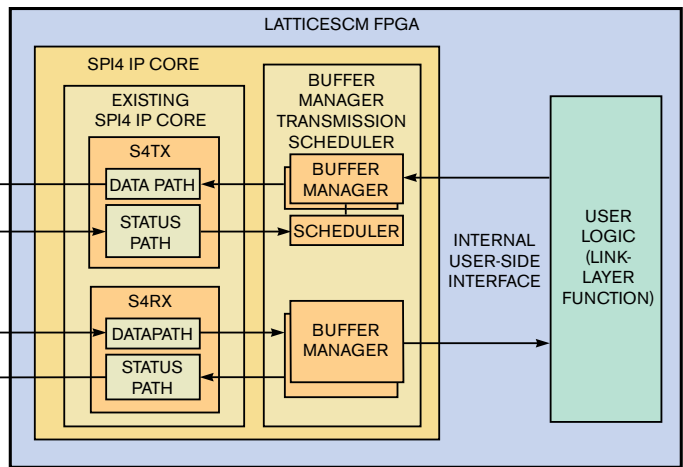


Figure 2 It is best to implement a suitable buffer manager that resides outside the basic SPI4.2 datapath and dictates transmit bandwidth commensurate with the end application.

dictates transmit bandwidth commensurate with the end application (Figure 2).

### PHYSICAL PER-CHANNEL BUFFER

A physical-buffer approach is ideal when the application requires a few FIFO buffers and serves multiple asynchronous physical interfaces that are independent of each other—for example, when bringing multiple 2.5- or 1-Gbps interfaces into one 10-Gbps SPI4.2. Individual FIFOs work well because the number of FIFOs is small—10 in the 1-Gbps scenario—and each provides an independent interface, including independent clocks for the external interfaces. This approach is the simplest and most straightforward for solving the per-channel-buffer-design question. The architecture is  $N$  channels= $N$  physical buffers of equal depth= $N$  physical interfaces= $N$  FIFO controllers, where  $N$  does not exceed 16 channels.

Additionally, this architecture would handle error and overflow conditions—usually, a packet-drop capability with FIFO-pointer readjustments—that you base on user-defined criteria. However, this architecture has several potential drawbacks that the virtual buffer addresses. With Ethernet, you must con-

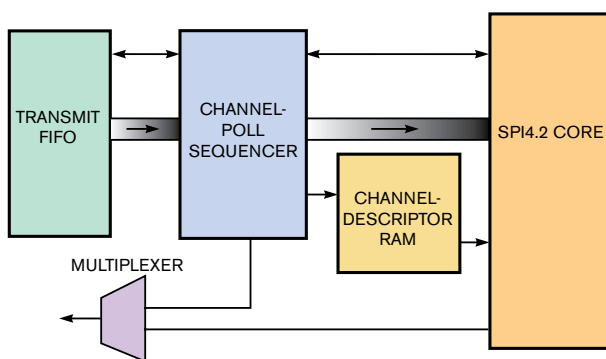


Figure 3 The scheduler could comprise a poll sequencer and a channel descriptor.

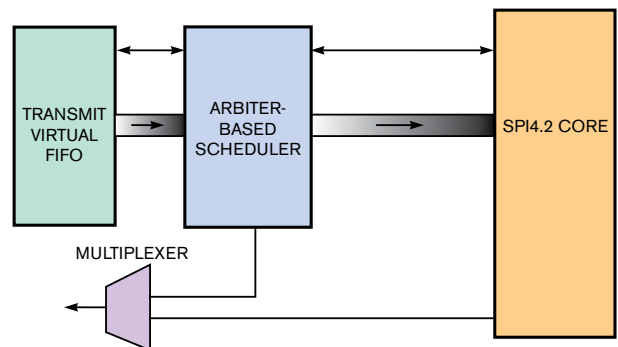


Figure 4 The objective of an arbitrer-based architecture is to skip channels that have nothing to send and prioritize those channels that do.

sider packet sizes of 9 kbytes or greater for worst-case scenarios. If your application requires more than one or two instances of jumbo-packet storage in both the transmitting and the receiving directions for one of the physical channels, you cannot use any of the other memory from other underused channels to reduce overall memory demands. This architecture does not work well for applications in which you must add or subtract bandwidth on the fly without affecting other channels. Therefore, you should budget the worst-case amount of per-channel memory beforehand.

Because this architecture assumes multiple independent asynchronous clock domains—one for each physical channel—the system cannot share FIFO-controller intelligence across channels to optimize logic resources.

### VIRTUAL PER-CHANNEL BUFFER

The virtual buffer uses a large memory structure in which one FIFO controller can manage many logical FIFO buffers. The system treats the large memory as a global resource that it can allocate dynamically for a channel through linked lists. Because the virtual buffer uses dynamic-memory allocation, it allows the engineering of peak memory-resource requirements at the system level rather than at the physical channel level, lowering overall memory needs. This approach is viable when a large number of synchronous channels are in operation—

+ Go to [www.edn.com/ms4293](http://www.edn.com/ms4293) and click on Feedback Loop to post a comment on this article.

+ For more feature articles, go to [www.edn.com/features](http://www.edn.com/features).

for example, when aggregating multiple SONET/SDH tributaries through a virtual-concatenation framer to an XAUI bridge.

You can implement the link list by storing the current state information, including descriptor locations, fill level, and other information, in a high-density RAM that loads into the FIFO controller when you select a channel. The system separately stores data memory and dynamically allocates it to the channels without regard to any per-channel constraints. When an application needs to control the amount of memory for a channel, this process can occur in the user domain of the buffer manager simply by using the per-channel status information, which indicates the number of memory segments each channel uses.

This architecture achieves the objective of a buffer design that supports a large number of synchronous channels, ensuring dynamic-memory allocation and maintaining full and constant traffic flows. The obvious drawback is that this scheme assumes fully synchronous physical interfaces. You can design a retrofit with small clock-domain-conversion FIFOs in front of the large virtual FIFO, allowing it to operate more asynchronously.

### SEQUENCER-BASED SCHEDULER

You can implement transmitter scheduling through the use of the SPI4.2 calendar sequence, which allows the user to re-

ceive status updates regarding each channel's bandwidth allocation. You can then design a scheduler to schedule channels for transmission, basing its action on the direct-channel-sequence information that the user enters in the SPI4 status calendar's RAM.

From an implementation perspective, the designer needs to ensure that the SPI4.2 circuit supports random access to the status of any channel in a single clock cycle for the scheduler. The scheduler itself could comprise a poll sequencer and a channel descriptor (Figure 3). The poll sequencer would use channel ID to poll channel status. Transmission for the channel would begin as soon there were enough data available to satisfy far-end-receiver requirements. If the poll sequencer did not meet the criterion, the scheduler would poll the next channel for transmission. The channel descriptor, stored in RAM, contains the channel ID and burst parameters for that channel.

One potential drawback of such an architecture is the time it spends polling channels with no assurance that any of the channels have anything to send. In such a case, you could consider an arbiter that would service only relevant channels.

### ARBITER-BASED SCHEDULER

The objective of an arbiter-based architecture is to skip channels that have nothing to send and prioritize those channels that do (Figure 4). The scheduled RAM sequencer polls every channel, which can add undesirable latency.

One approach is to assign each channel a location in a RAM table and then to add a weight for each channel. The weight

would specify how much of the total SPI4.2 pipe the system could allocate to each of the channels. A conventional round-robin arbiter would then flag channels that have a larger weight and have data to send first. Traffic-management and shaping circuits commonly use this scheme, weighted round robin, to handle and police priority requests for traffic. You can effectively use the same scheme in SPI4.2 buffer management.

### SUMMARY

With both the power consumption and the prices of programmable-logic devices plummeting, as well as the advent of embedded high-speed Ethernet-related interfaces, it becomes difficult to ignore the case for FPGAs in carrier-Ethernet systems. Most important, for buffer-management schemes, it is improbable that an ASSP can achieve the type of cost- and power-effective flexibility that FPGAs can offer in tailor-made, intelligent bridges, gaskets, and traffic managers. **EDN**

### AUTHOR'S BIOGRAPHY



Shakeel Peera is director of marketing for high-performance solutions at Lattice Semiconductor (Bethlehem, PA). He previously held various technical-marketing positions in FPGAs, ASICs, and networking intellectual property at AT&T Microelectronics, Lucent Technologies, and Agere Systems. Peera holds a bachelor's degree in electrical engineering from the University of Rochester (New York). You can reach him at [shakeel.peera@latticesemi.com](mailto:shakeel.peera@latticesemi.com).