

**PHASE 2 OF THE SYSTEM PACKET INTERFACE-4 IS EMERGING AS THE NEW TELECOMMUNICATIONS AND NETWORKING STANDARD, BUT CLAIMING COMPLIANCE DOESN'T NECESSARILY MEAN *BEING* COMPLIANT.**

# All SPI-4s are not created equal

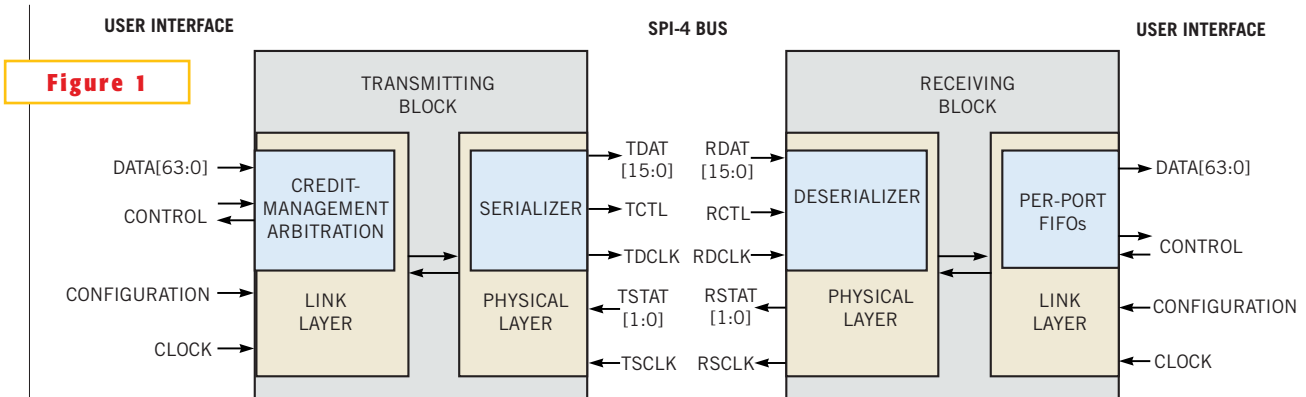
**T**HE OIF (OPTICAL INTERNETWORKING FORUM) developed Phase 2 of the SPI (System Packet Interface)-4 to ensure interoperability among essential networking-system components. It is emerging as an important integration standard in telecommunications and data networking (Reference 1). SPI-4 Phase 2 accelerates data movement and reduces bottlenecks in 10-Gbps Ethernet and optical-networking systems. A growing number of network-processor vendors and system designers are adopting SPI-4 Phase 2 for interoperability among 10-Gbps applications.

IP (intellectual-property) vendors that claim compliance with the OIF specification may not provide all the components required to design chips to communicate with the SPI-4 Phase 2 protocol. Some vendors offer only the physical layer as a full-custom digital or analog approach, and others offer fully digital, standard-cell-based implementations. Some offer only the link layer, and others provide both the physical and the link layers but without dynamic deskew. Still others offer both the

physical and the link layers, including the dynamic deskew, but not the credit-management and arbitration functions. Finally, vendors may or may not provide an asynchronous interface within the SPI-4 block to allow the user logic's clock to run at a frequency independent of the SPI-4 bus. The following guidelines ensure that the SPI-4 Phase 2 block you acquire will meet your system-level requirements.

A SPI-4 Phase 2 implementation comprises an independent transmitter and receiver (Figure 1). At the system level, the transmitter and the receiver exchange data, control, status, and clocks with each other over the SPI-4 bus. OIF defines this interface, and you cannot customize it within the context of an OIF-compliant SPI-4 Phase 2 implementation. OIF does not define the other side of the SPI-4 block that communicates with the user logic; what occurs between these two interfaces, though, makes things interesting.

An ASIC needs to implement both the physical and the link layers to process traffic coming across



**A SPI-4 Phase 2 implementation comprises an independent transmitter and receiver that exchange data, control, status, and clocks over the SPI-4 bus.**

a SPI-4 Phase 2 bus. The physical layer, which includes a high-speed SERDES, deals with physically converting the wide, slow data from the user logic into the narrow, fast data going across the SPI-4 bus and then piecing it back together at the other end. The link layer deals with the SPI-4 traffic's logical protocol.

### THE PHYSICAL LAYER

The transmitting side of the SPI-4 bus serializes data on the bus—typically at a 4-to-1 or an 8-to-1 serialization with the associated frequency multiplication—and sends it as 16 bits of data, 1 bit for control, and 1 bit for clock. OIF specifies the minimum data rate for this interface at 622 Mbps, and vendors often achieve that rate by implementing a 311-MHz DDR interface.

The clock an implementation requires can take one of two forms: An implementation can use only the rising edges from a full-rate, 622-MHz clock, or it can generate the specified data rate with both edges of a half-rate clock. Use of a full-rate clock, particularly for applications that run the SPI-4 interface above 800 Mbps, imposes significant and sometimes difficult-to-achieve requirements on both the clock source and the on-chip clock distribution. Using both edges of a half-rate clock, such as in a DDR implementation, puts a strict requirement on the duty cycle of the incoming clock. Either method is acceptable. It's a matter of balancing the trade-off between a higher speed clock source and distribution and a lower speed clock with strict duty-cycle requirements.

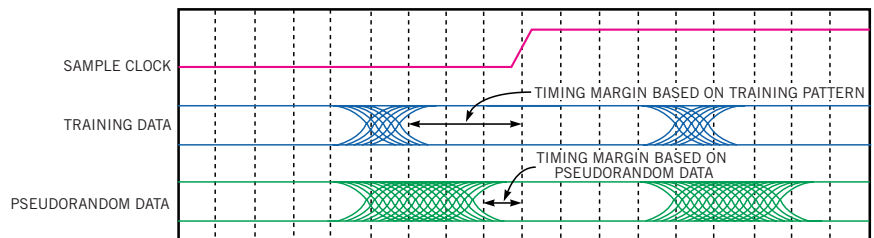
The receiver must reliably capture the serialized data coming over the SPI-4

bus, deserialize it back to its original 64- or 128-bit-wide form, and finally re-assemble it back into words that match those that were the input to the serializer at the transmitter. OIF specifies that the channel clock transmits along with the data across the SPI-4 bus to eliminate the need for the SPI-4 receiver to do CDR (clock-data recovery). Similarly, as with the transmitter, a full- or half-rate clock can support the deserialization of the incoming data stream. Because the clock on the SPI-4 bus is a half-rate clock, a deserializer that uses a full-rate clock must generate it, typically with a PLL, from the incoming channel clock.

The deserialization algorithm can evaluate either multiple phases of the clock against the incoming data or multiple phases of the incoming data against a single clock. Evaluating multiple phases of the clock against the incoming data typically requires an on-chip, multi-phase PLL or a DLL to generate the clock phases, and it often can operate over only a limited frequency range. Evaluating multiple phases of the incoming data against a single clock requires neither a PLL nor a DLL, assuming that you use a half-rate clock, but requires a potentially long delay chain to create the multiple data phases. Either method works; it's a matter of trading off the added requirement and possible frequency limitations of a PLL or DLL with the potentially larger gate area and physical restrictions of the delay-chain approach.

### THE LINK LAYER

The link layer deals with the SPI-4 protocol. Data from the user logic on the SPI-4's transmitting side can come from multiple sources, each of which has a port into the SPI-4 link layer. Arbitration



**Figure 2** Intersymbol interference can occur when you are transferring pseudorandom data and can cause a significant change in the data-to-clock relationship at the receiver, which can, in turn, lead to data-integrity problems.

and credit-management functions manage the data flow into these ports. The receiver side's user logic allocates credits for each port and sends them to the SPI-4 transmitter via the status channel. Each port has a satisfied, hungry, or starving status. The arbiter determines which port's data next travels across the SPI-4 bus based on the status value and information from the transmitter side's user logic. Although OIF specifies no arbitration algorithm and does not require that you implement this function as part of the SPI-4, a fundamental need exists for this function in the

user logic or as part of a SPI-4 IP implementation. Once the arbiter decides to accept data for a given port, the transmitter link layer assembles the SPI-4 burst that includes information about which port it came from, DIP4 parity,

and some additional control information, and ships it to the physical layer for serialization. On the receiver side, the physical layer presents the SPI-4 data to the link layer for protocol processing. The link layer uses the in-band control information to decide which port is the target for the data. It checks the DIP4 parity and forces the SPI-4 interface to retrain if it detects a programmable number of DIP4 parity errors. A number of options now exist regarding what happens with the data, all of which help define how the user interface works.

The simplest case to visualize is when data from the SPI-4 channel transfers directly, via the link layer, to the user logic without applying additional intelligence. Unfortunately, such a simple interface is problematic because of how OIF specified the protocol. The data coming across the SPI-4 bus may not be in even 8-byte chunks, because the user interface is 8 bytes wide in a 4-to-1 SERDES function. As such, the receiver could receive data destined for two ports in the same user clock cycle.

Thus, the logic must include in the implementation of the user interface a mechanism for simultaneously dealing

with two data transfers. One approach is to simply have an alternative data bus on the user side of the SPI-4 that is active only during these end cases when two transfers need to occur simultaneously. Another approach is to put the data into a FIFO that can receive either one or two updates per clock and then stream it one word at a time to the user logic. This approach involves another complication because no low-latency back-pressure mechanism exists to stall the SPI-4 bus when the FIFO fills. Both of these implementations are reasonable, and which

is better depends largely on a user's needs.

A more sophisticated approach involves performing intelligent packet reassembly and then sending contiguous packets to the user. This approach may be undesirable because it involves a poten-

tially significant amount of buffering within the SPI-4 block. In cases in which the user logic is responsible for managing the per-port data from SPI-4, the existence of the packet-reassembly logic may duplicate or even conflict with the functions of the user logic. Designers must have a clear understanding of the user interface to evaluate a SPI-4 IP block for use in an application.

The frequency at which the SPI-4 bus operates is often independent of the user-logic frequency. In such cases, the SPI-4 block needs an asynchronous interface to support the data transfer between clock domains. This asynchronous boundary often exists between the physical layer and the link layer. Although it may exist elsewhere, such as within the link layer, it is usually desirable to allow for arbitrary clock relationships between the channel and the user logic, because other system-level requirements often dictate the user-logic frequency. In cases where there is an asynchronous relationship between the user logic and the channel clock, the application must maintain appropriate clock-frequency ratios to guarantee that the SPI-4 channel receives enough data to prevent un-

**THE LOGIC MUST INCLUDE  
IN THE IMPLEMENTATION  
OF THE USER INTERFACE  
A MECHANISM FOR SIMULTA-  
NEOUSLY DEALING WITH  
TWO DATA TRANSFERS.**

derflow on the transmitting side and overflow or back pressure on the receiving side. It is difficult to efficiently manage any back pressure in the receiver because of the high latency of the SPI-4's status-channel flow control.

#### TRAINING

The physical layer, the link layer, or both handle the training sequence. SPI-4 needs training when it operates in dynamic mode so that the transmitter and receiver can synchronize, compensating for any board-level routing or on-chip latency mismatches that exist. The receiver uses an OIF-defined data pattern sent across the SPI-4 bus as a reference signal to be able to reliably capture data on the bus. The deskew function encompasses bit recovery, in which the receiver attempts to center the capture point for data within a valid data "eye," and byte or word alignment, in which each of the 16 bits of data and control align with each other to compensate for any latency differences in their respective paths.

If you perform training for bit recovery based solely on the OIF-defined training pattern, the additional intersymbol interference that could occur when you are transferring pseudorandom data could cause a significant change in the data-to-clock relationship at the receiver and, in turn, lead to data-integrity problems (Figure 2). This problem occurs because the OIF-defined training pattern is of a periodic nature, and no encoding of the data occurs to guarantee a minimum transition density on the SPI-4 bus. Some IP vendors solve this problem by maintaining bit alignment based on the pseudorandom data.

In dynamic mode, the SPI-4 design can tolerate as much as  $\pm$  one bit time of skew between the 17 lines—16 data plus one control—relative to the clock. The static mode of operation does not need dynamic deskew, but you need to carefully match the board line lengths and respective system latencies so that you can reliably recover the individual bits. The link layer includes hooks to initiate training either directly from the transmitter or indirectly via the status channel from the receiver. The receiver must be able to detect when training is in progress so

that it can respond appropriately.

The status channel is another small, yet important piece of the SPI-4. OIF defines a low-speed LVTTTL-based or a high-speed LVDS-based protocol. The low-speed option runs at one-eighth the rate of the data channel, which is adequate for many applications because the volume of information flowing across the status channel is low. However, for systems in applications requiring a large number of ports or with a significant amount of bandwidth mismatches between ports, a higher bandwidth flow control is desirable. The choice between high-speed and low-speed status is often a configurable option within a SPI-4 IP block.

#### MEETING THE 10-GBPS CHALLENGE

A fully OIF-compliant SPI-4 Phase 2 core can help designers and architects address the speed and interoperability requirements of today's systems. Each component of a fully OIF-compliant SPI-4 Phase 2 core, however, has significant challenges. With SPI-4 Phase 2's increase in popularity, engineers must be aware of the options when selecting a SPI-4 IP core for use in their designs. □

---

#### REFERENCE

1. Cravotta, Nicholas "RapidIO versus HyperTransport", *EDN*, June 27, 2002, pg 32.

---

#### AUTHOR'S BIOGRAPHY



*Mike Berry is chief technology officer at Silicon Logic Engineering (Eau Claire, WI), a semiconductor IP and design-services company, where he has worked for six years. In his current position,*

*he oversees the electrical- and physical-design teams and provides technology-based input to the development teams for the company's IP offerings. He earned a bachelor's degree in physics from Drew University (Madison, NJ) and a master's in electrical engineering from the New Jersey Institute of Technology (Newark, NJ). In his spare time, he enjoys boating, skiing, tinkering with his 1967 Mustang, and spending time with his children. You can reach him at [mberry@siliconlogic.com](mailto:mberry@siliconlogic.com).*