

# EETIMES<sup>ONLINE</sup>

EE Times: [Latest News](#)

## Hitting the 10-Gbit Mark with SPI-4.2

SPI-4.2 has become a pop star in the comm chip space. But to keep its stardom alive, designers must prove that this I/O can effectively handle 10-Gbit streams. And that's going to require proper configuration and usage.

Kelly Marquardt, Silicon Logic Engineering

(09/10/2002 4:43 AM EDT)

URL: <http://www.eetimes.com/showArticle.jhtml?articleID=16505719>

The system packet interface (SPI)-4.2 specification is quickly becoming the pop star of the networking sector. Like Mark Walberg's "Chris" character in the movie *Rock Star*, SPI-4.2 has risen from virtual obscurity in the networking space and become a main ingredient in almost all system interfaces for serdes, network processors, traffic managers, and more.

But for this Optical Internetworking Forum (OIF) to hold its stardom, designers must quickly prove that this I/O can effectively handle 10-Gbit/s data streams. Thus, it is imperative that the interface be configured and used correctly in a system in order to achieve an effective 10-Gbit data rate.

Factors such as number of ports, calendar length, and maxburst values impact the bandwidth of the SPI-4.2. In this article will look at all of these components and examine their impact on achieving 10-Gbit/s performance on a SPI-4.2 interface.

### Bio on the Star

The SPI-4.2 spec developed by the OIF defines a 16-bit interface operating at a minimum data rate of 622-Mbits/line, which yields a raw data rate of 9.952 Gbit/s. A typical SPI-4.2 interface might run at 800-Mbits/line, yielding a raw data rate of 12.8 Gbits/s.

In addition to the data lines, the SPI-4.2 FIFO status channel sends flow control information from the receiver to the transmitter at one eighth the rate of the data channel, based on a predefined FIFO status calendar. This calendar defines, in order, which port each status update corresponds to. These status updates grant credits for the transmitter to send data for a given port. The number of credits granted is determined by the maxburst values that the interface uses for its "hungry" and "starving" indications.

SPI-4.2 bandwidth analysis is a complex problem with many factors to be considered including the number of ports, the calendar sequence used, and the maxburst values. The most significant challenges in achieving high SPI-4.2 bandwidth involve using the FIFO status channel flow control mechanism effectively so that the data channel is never idle when there is data to be sent. It is also important to ensure that the bandwidth of the SPI-4.2 is shared appropriately across the ports in order to achieve the goals of the system. Let's look at these requirements in more detail.

### Staying Busy

Keeping the SPI-4.2 channel busy, while clearly not the only goal of bandwidth allocation, is a minimal requirement when developing a chip design. If the channel is unnecessarily idle, bandwidth goals cannot be met. To prevent the channel from being idle, designers must consider several factors. Let's look at these below.

Port count is clearly one factor that will impact the usage of the SPI-4.2 channel. The number of ports dictates the minimum length of the FIFO status calendar since each port must appear in the calendar at least once. If a simple, linear calendar is used, a range of suitable values for maxburst can be determined.

The minimum suitable value for maxburst is the smallest value which, when multiplied by the number of ports, grants enough credits to keep the data channel busy for the duration of one calendar sequence. In a 10-port application, where each port is 1 Gbit/s, for example, a simple, linear calendar can be used which grants maxburst credits for a given port every 12 status clocks — one clock for each port and the two clock overhead of the DIP-2 parity codeword and the framing pattern.

Since the status clock runs at one-eighth of the data rate, the latency of one calendar is equal to  $8 \times 12 = 96$  data cycles where each data cycle transfers two bytes. Thus, 192 bytes of data can be sent during one full calendar cycle.

If the bandwidth is divided among 10 ports, the maxburst value for each port must allow  $96 \times 2 / 10 = 19.2$  bytes of data to be sent during each calendar sequence, or at least a value of 2, since maxburst is defined in units of 16 bytes. However, this minimum value for maxburst will only keep the data channel full if traffic is spread evenly across the ports.

In the case where traffic is distributed unevenly, the maxburst value must be set larger to allow any active ports to use all of the available bandwidth. In the example system discussed above, only one of the ten ports might have data available to send. To keep the channel busy, this one active port must be able to use all of the bandwidth. To do so, the maxburst for that port must grant enough credits to that one port to keep the data channel busy for the duration of the calendar sequence. Thus, in our 10-port system example, the maxburst for each port must be set to 96 data cycles (during a calendar sequence) \* 2 bytes per cycle = 192 bytes of data, or a value of 12. This can be considered the maximum suitable value for maxburst in this instance since a larger value does not allow for additional data traffic.

**Figure 1** shows the bandwidth available per port in the above example.

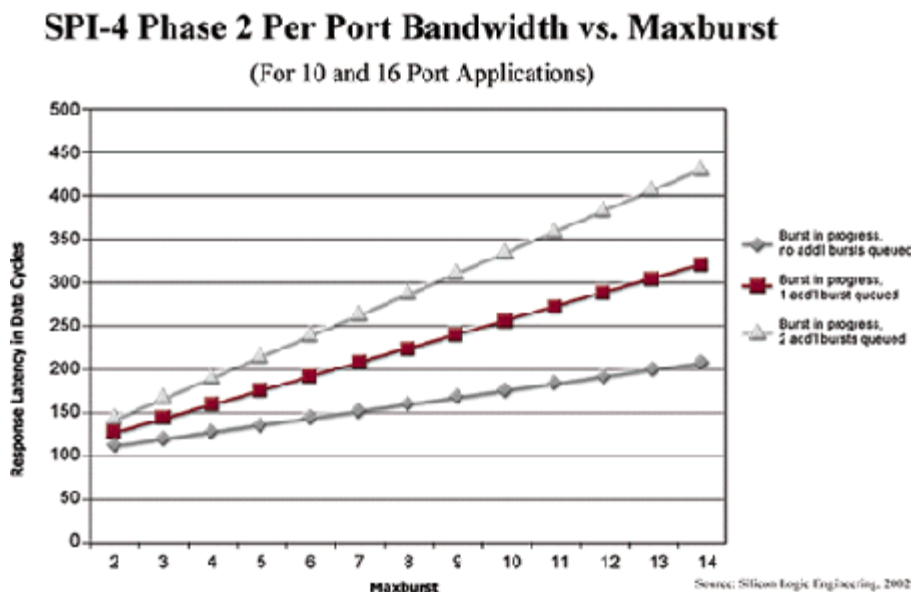


Figure 1: Chart depicting SPI-4.2 per port bandwidth versus maxburst.

While it is relatively straightforward to calculate these minimum and maximum suitable values for maxburst, determining which values to use within this range depends on various aspects of the design. There are benefits and drawbacks for both small and large values of maxburst. However, if the only goal that is considered is keeping the channel busy, then large values of maxburst are preferred.

Small values of maxburst create more challenges in keeping the channel busy. If smaller values of maxburst are used, it is important that the data traffic available to send is spread evenly across the ports. If not spread evenly, there is the risk that the only ports with data to send will use all of their credits before the calendar sequence completes and the channel will become idle. Smaller values of maxburst make it more difficult for the transmitter design to supply data fast enough to keep the channel busy, due to the challenges of arbitration and the retrieval of data to send.

### The Arbitration Challenge

In general, SPI-4.2 transmit arbiters perform better when the maxburst values are larger. In a typical design there is a great deal of overhead that happens in the transmitter between bursts. The transmitter has to first decide which port it will send data to next, taking into consideration which ports are "starving", which ports are "hungry", and which ports it has data available for. Additionally, the transmitter must try to achieve a fair arbitration algorithm on the selected port.

Arbitration itself might take a few cycles. Once the port is chosen, the transmitter needs to fetch the data for that port. This might typically involve retrieving the data from a RAM, checking error correction coding on the data, and then formatting it to send on the SPI-4.2 by adding control words and calculating DIP-4 codewords. A typical transmitter might have an interburst latency of 5-10 application clock cycles.

When there are multiple active ports, the arbitration overhead can be pipelined, minimizing or even eliminating the impact. By arbitrating before the current burst is complete, the transmitter can have the data for the next port ready to go and not have to pay the interburst penalty.

The downside here is that ideally, arbitration should occur as late as possible in order to use the most up-to-date information. If a transmitter arbitrates early in order to keep its pipeline full, responsiveness to status changes is reduced. If, for example, a port changes from a "hungry" status to a "starving" status an arbiter might want to take this change into consideration when deciding which port to send data on next. It can't respond to this change very quickly if it has

arbitrated in advance and has a pipeline full of data queued to send on the SPI-4.2 channel.

**The Single Port Case**

Single-port applications are challenging for designers looking to achieve 10-Gbit operation over a SPI-4.2 interface. In the case of SPI-4.2, single-port can be defined as a true single-port application or a multi-port application where only one port is actively sending data.

In the single-port case, the arbitration and data fetch cycles cannot be pipelined. To arbitrate and fetch data for the same port without waiting for the status update would cause multiple maxburst amounts of data to be queued up in the pipeline for a single port. This violates the SPI-4.2 spec, which only allows one maxburst unit of data to be sent to a port in response to the granting of credits for that port. Thus, the application described here needs to send one burst, wait for the next status update for that port, and then fetch and send data for the port if it has been granted credits.

The case of a multi-port application where only one port is active is more challenging than the true single-port case. In this scenario, calendar updates for the active port come less frequently and there is often more overhead associated with fetching data in a multi-port application. When small maxburst values are used, the interburst latency quickly impacts the achievable bandwidth.

**Benefits of Small Maxburst Values**

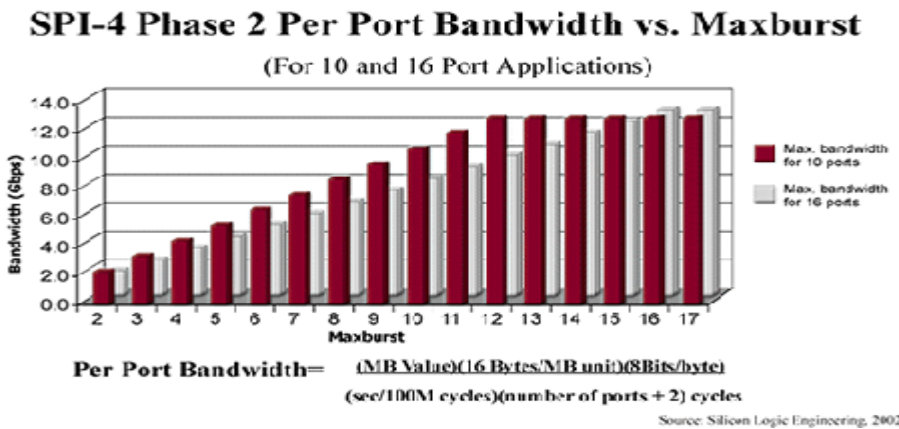
While it is clear that small values of maxburst pose challenges in the design of the transmitter, there are also benefits to using small values of maxburst. In many applications, the goals for bandwidth allocation go beyond simply keeping the channel full. It is often important to make sure that none of the receiver ports runs out of data. Many applications also require fairness in the allocation of bandwidth among the ports, which is a related goal. In these cases, a smaller maxburst value can be advantageous. Let's see why.

When an individual port in the receiver is receiving data, the port has a buffer that is being filled by the SPI-4.2 channel and consumed by the application logic. As the data is consumed by the application logic, and when the amount of data in the buffer falls to a certain watermark, the receiver logic sets the FIFO status for that port to "hungry" or "starving" to request that the transmitter send data for that port. The appropriate levels for these watermarks depend a great deal on the latencies associated with the calendar as well as the transmitter design. In the worst case, the receiver logic might realize that it has reached a watermark just after the calendar slot for that port has passed. This means it has to wait for one full calendar sequence to send its status update to the transmitter.

Once the transmitter has received the status, any data that is sent out to the requesting port must follow a burst that is currently in progress, as well as any other data that is already queued to be sent. If a large value of maxburst is being used, such as one that is large enough to cover the entire calendar latency, the response latency for the requesting port is increased. In this case, the receiving port must wait for another calendar latency for the burst that is currently in process, assuming that burst just started. The delay could be even longer if the arbiter does not immediately choose the requesting port.

Clearly, large values of maxburst make the response latency longer than it would otherwise be. However, even in the case of small maxburst, the transmitter will likely need to arbitrate early and have data for multiple ports queued to send in order to pipeline the data retrieval. The depth of the pipeline of data depends on the application and determines the resulting response latency.

**Figure 2** shows the response latency for different values of maxburst for the 10-port example discussed above. The response latency is expressed in terms of data cycles, which are two bytes. The three curves represent the different cases where there is a burst in progress, but no additional bursts queued, a burst in progress with one additional burst queued, and a burst in progress with two additional bursts queued.



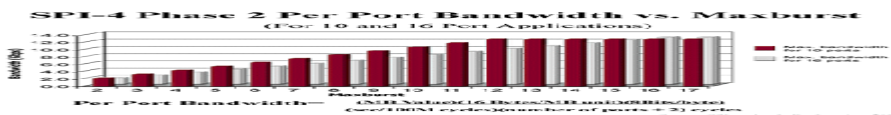


Figure 2: Response latency versus maxburst.

### Uneven Bandwidth Allocation

Some comm system designs require uneven bandwidth allocation among the ports that are used. Designers can achieve this requirement with the SPI-4.2 using several methods.

One way to achieve unequal bandwidth allocation with SPI-4.2 is by using a linear calendar with different maxburst values. Under this approach, the high bandwidth ports are assigned a proportionally higher value for maxburst than the low bandwidth ports. This works well when response time and fairness for the low bandwidth ports is not of great concern.

Another way to achieve unequal bandwidth allocation is the use of a non-linear calendar sequence that gives extra entries to higher bandwidth ports and similar maxburst values. This approach allows arbitration to be performed more often, reducing the response latency for an individual port. The non-linear calendar allows smaller values of maxburst to be used. Rather than wait for the entire calendar sequence, the credits granted for maxburst only need to cover the amount of data that can be sent until the next time that port appears in the calendar.

### Putting it All Together

The interoperability that SPI-4.2 provides in the 10-Gbit/s application space is an important advance for the networking community, allowing developers of networking systems to focus on their own products, assured that the components of the system will interoperate. With interoperability assured, designers can focus on the configuration factors that impact bandwidth of the SPI-4.2 channel.

### About the Author

**Kelly Marquardt** is the director of IP design and applications at Silicon Logic Engineering. She holds a MSEE from the University of Illinois at Chicago and can be reached at [kellym@siliconlogic.com](mailto:kellym@siliconlogic.com).

All material on this site [Copyright © 2005 CMP Media LLC](#). All rights reserved.  
[Privacy Statement](#) | [Your California Privacy Rights](#) | [Terms of Service](#)