Elastic Buffer Implementations in PCI Express Devices

MindShare, Inc.

Joe Winkles
joe@mindshare.com

November 2003

Printed in the United States of America.

First Printing, November 2003

Find MindShare on the World-Wide Web at:
http://www.mindshare.com/

Contents

Introduction

Elastic Buffers (also known as Elasticity Buffers, Synchronization Buffers, and Elastic Stores) are used to ensure data integrity when bridging two different clock domains. This buffer is simply a FIFO (First-In-First-Out) where data is deposited at a certain rate based on one clock and removed at a rate derived from a different clock. Because these two clocks could (and almost always do) have minor frequency differences, there is the potential for this FIFO to eventually overflow or underflow. To avoid this situation, an Elastic Buffer has the ability to insert or remove special symbols, during specified intervals that allow the buffer to compensate for the clock differences.

The Elastic Buffer, or the concept of the Elastic Buffer, has been around since at least 1960. Maurice Karnaugh was granted a patent for this technology back in 1963 for its use in transmitting Pulse-Code Modulated (PCM) signals in telephone networks (*Pat. 3,093,815*). However, this was just the first of many applications that these buffers would be used in. The robustness of Elastic Buffers is indicated by their inclusion in modern day technologies. These buffers can be found in protocols such as USB, InfiniBand, Fibre Channel, Gigabit Ethernet, and of course PCI Express.

This paper describes the necessity of Elastic Buffers in a serialized, source-synchronous timing architecture such as PCI Express. Next, a brief description of the protocol used to implement clock tolerance compensation is discussed, as well as the placement of the Elastic Buffer within the data flow of a PCI Express device. Finally, this paper shows some viable implementations (including sizes) of Elastic Buffers in PCI Express devices as well as the advantages and disadvantages of each design.

The Need for Elastic Buffers

The PCI Express technology is a high-speed, serialized, source-synchronous timing (clock-forwarding), data transfer protocol. These characteristics are significantly different than the characteristics implemented by PCI Express' predecessor buses, namely PCI and PCI-X. Both PCI and PCI-X use a multi-drop, parallel bus that utilizes a synchronous timing architecture.

A synchronous timing architecture is where a common clock source supplies a clock to all the devices on the bus, and that clock is used to enable the device's transceivers to clock data in and out. This scheme requires that the clocks arrive at each device at precisely the same time. There is a very small amount of pin-to-pin skew allowed, which means that the lengths of these clock traces have to be matched to minimize the amount of skew between devices. It now becomes apparent that as the speed of the clock increases, the allowed pin-to-pin skew must decrease, which requires the matched routing of these clock traces to become increasingly difficult.

A source-synchronous timing architecture is where both the data and clock are transmitted from the originating device's driver. This means that there is no common clock that all devices must use to clock data in and out with. Instead, a transmitting device forwards the clock it used to transmit the data and the receiver will use that forwarded clock to latch the transmitted data.

As previously mentioned, PCI Express utilizes a source-synchronous timing architecture. It employs a scheme where the forwarded clock is "embedded" into the data stream by encoding the data using IBM's 8B/10B encoding tables. This encoding mechanism ensures that the data stream will have a sufficient number of 0-to-1 and 1-to-0 transitions to allow the clock to be recovered on the receiving side with the use of a Phase-Locked Loop (PLL). With this timing architecture, we no longer have the problem of carefully routing the same clock to each device with very small amounts of skew. However, a different problem arises. Each device now has at least two clock domains: (1) the Local Clock Domain that it uses to clock all of its internal gates and to transmit data with, and (2) the Recovered (or received) Clock Domain that it uses to latch inbound data. The data being received will have to cross the clock boundary from the Recovered Clock Domain to the Local Clock Domain in order for the device to process that information. Elastic Buffers are implemented in PCI Express devices for this exact purpose, to allow received data to transition from the Recovered Clock Domain to the Local Clock Domain. The next section describes this transition in detail, including the protocol embedded in PCI Express to make this process work.

How These Buffers Work…

All of the first generation PCI Express devices must transmit data at a rate of 2.5Gbps (Giga-bits per second) with a tolerance of +/- 300 ppm (parts per million). This translates into an allowed frequency range of 2.49925GHz – 2.50075GHz at the transmitter of a device. Due to the tolerance allowed, two devices connected to each other can (and most likely will) be running at slightly different frequencies. This creates two clock domains within one device that the received data must transition across. As shown in Figure 1, the function of the Elastic Buffer is to bridge these two clock domains by compensating for their phase and frequency differences, thereby allowing the received data to maintain its integrity as it flows upstream in the target device.

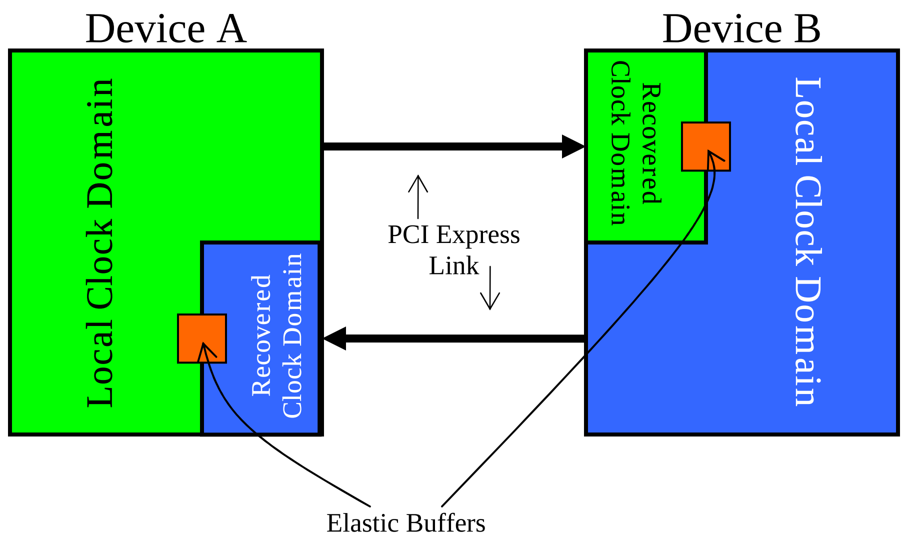

Figure 1. Elastic Buffers bridging the clock domains in each PCI Express device.

Bridging these two clock domains is accomplished by depositing the received data into the buffer using one clock domain (the recovered clock) and pulling the data out of the buffer using the other clock domain (the local clock of the device). Since these two clock domains can be running at different frequencies, the buffer has the potential to overflow or underflow. However, these error conditions are prevented by designing the buffer to monitor its own state (or fill-level) and enabling it to insert or remove special symbols at the appropriate rate (hence the name, Elastic Buffer). PCI Express defines these special symbols that can be inserted or removed as SKP symbols which are only found in SKP ordered-sets. A transmitted SKP ordered-set is comprised of a single COM symbol followed by three SKP symbols. Transmitters are required to send SKP ordered-sets periodically for the exact reason just discussed; to prevent an overflow or underflow condition in the Elastic Buffer.

The rate at which these SKP ordered-sets are transmitted is derived from the max tolerance allowed between two devices, 600ppm. This worst-case scenario would result in the local clocks of the two devices shifting by one clock cycle every 1666 cycles. Therefore, the transmitter must schedule a SKP ordered-set to be sent more frequently than every 1666 clocks. The spec defines the allowed interval between SKP ordered-sets being scheduled for transmission as being between 1180 – 1538 symbol times. (Since SKP symbols are removed and inserted an entire symbol at a time, the 1180-1538 interval is measured in symbol times, a 4ns period, and not bit times.) Upon receipt of the SKP ordered-set, the Elastic Buffer can either insert or remove up to 2 SKP symbols per ordered-set to compensate for the differences between the recovered clock and the local clock.

There will be instances where a device will not be able to transmit a SKP ordered-set within the defined interval of 1180-1538 symbol times. This occurs during the transmission of a large Transaction Layer Packet (TLP) that prevents the device sending a SKP ordered-set at the regularly scheduled time. Since the device cannot interrupt the transmission of the TLP, it must wait until the entire TLP is transmitted and then it can transmit the delayed SKP ordered-set(s). During the large TLP transmission, multiple symbol shifts can occur at the receiver between the Recovered Clock Domain and the Local Clock Domain. (The number shifts that can occur is described in a later section of this paper.) To ensure that the Elastic Buffer in the target device receives enough SKP ordered-sets to compensate for the multiple symbol shifts, the SKP ordered-sets that were scheduled for transmission during the TLP are accumulated. These accumulated SKP ordered-sets are sent consecutively, immediately following the end of the TLP.

Placement within the Data Flow

There are several components within the Physical Layer of a PCI Express device. This section discusses the logical placement of the Elastic Buffer in relation to the following three components: the Deserializer, the 8B/10B Decoder, and the Deskew circuitry. The following descriptions for the placement of the Elastic Buffer are only recommendations that are based on information given in the spec.

As described earlier in this paper, the Elastic Buffer works by inserting or removing SKP symbols contained in SKP ordered-sets. The key word in the previous sentence is “symbol.” This means that the Elastic Buffer accomplishes clock tolerance compensation (bridging clock domains) at the symbol level and not the bit level. The logic block which generates valid symbols from the received bit-stream is the Deserializer. Therefore, the received data should pass through the Deserializer before being deposited into the Elastic Buffer.

The positioning relationship of the 8B/10B Decoder to the Elastic Buffer is not as intuitive as the buffer’s relationship to the Deserializer. At first glance, it seems to make sense to place the buffer after the Decoder for various reasons, as follows:

- If the Elastic Buffer is placed after the Decoder, it does not have to be as wide, only 9-bits wide (8-bits of data plus the data or control symbol indicator) instead of 10-bits wide if it was placed before the Decoder.
- If the 8B/10B decoding was taken care of before the clock tolerance compensation, then the Elastic Buffer would not have to pay attention to the disparity of the SKP symbols in the SKP ordered-set because there is no concept of disparity after the Decoder.

Despite the reasons stated above for placing the Elastic Buffer after the Decoder, this buffer really should be placed before the 8B/10B Decoder. To understand the justification for placing the Elastic Buffer before the Decoder, we must look at the characteristics of a PCI Express device when it is acting as a Loopback Slave. A Loopback Slave is required to "retransmit the received 10-bit information exactly as received, while continuing to perform clock tolerance compensation." This implies that the 10-bit symbols being received should not be decoded and later re-encoded for transmission. The spec re-enforces this concept by also stating that "No modifications of the received 10-bit data is allowed by the Loopback Slave." For Loopback Slaves, even though the 8B/10B decoding and re-encoding is prohibited, clock tolerance compensation still has to be employed by sending this data through the Elastic Buffer. Inserting and removing SKP symbols is the only allowed modification to the received 10-bit data. (This information as well as the references above can be found on page 197 of the 1.0a version of the *PCI Express Specification*. It should also be noted that all devices are required to be able to support being a Loopback Slave. It is optional whether a device is capable of being a Loopback Master.)

Due to the Loopback Slave scenario, the Elastic Buffers should be placed before the 8B/10B decoding logical blocks. Because of this placement, the Elastic Buffer now has to pay attention to the disparity of the SKP symbols when inserting additional SKP symbols into a SKP ordered-set. (For more information on the 8B/10B decoding process and the topic of disparity, please refer to Chapter 11 of MindShare's book entitled "*PCI Express System Architecture*.")

To determine the placement of the Elastic Buffer in relation to the Deskew circuitry in the Receiver, the functionality of the Deskew circuitry must be considered.

A PCI Express link can be comprised of multiple lanes. The transmitting device delivers pieces of a packet on all of these lanes at the same time (by byte striping the packet across all the lanes). However, the receiving device on the other end of the link may not receive the data on each lane at the exact same time due to varying trace lengths for the different lanes between the two devices or possibly due to some other board characteristics. This effect is shown in Figure 2. The time difference for data arrival across the lanes is known as lane-to-lane skew. This indicates that each lane on the receiving side must recover its own clock from the data stream arriving on that lane. Each of these recovered clocks should have the same frequency (because they are all locking onto the transmit frequency), however, due to driver and trace delays, they can have different phases in relation to each other. These phases can even be off by more than 360 degrees with respect to each other, as shown in Figure 2. The purpose of the Deskew circuitry is to re-align the phases of the received data effectively. This process can be accomplished entirely in the digital realm in two simple steps. (Conversely, if this deskewing was attempted in the analog domain, it would involve having to shift received information by fractions of a clock and can become very complicated to implement.) The first step in the digital solution is to latch the symbols

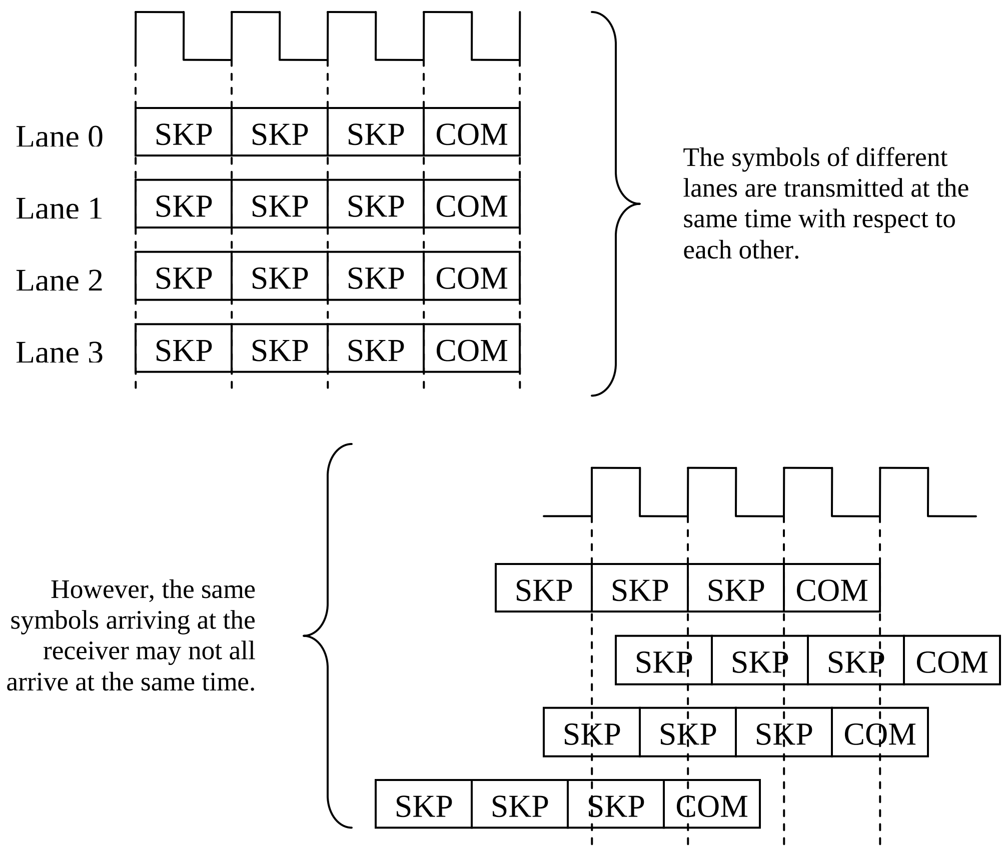
Lane 0
Lane 1
Lane 2
Lane 3
SKP
SKP
SKP
COM
The symbols of different lanes are transmitted at the same time with respect to each other.
However, the same symbols arriving at the receiver may not all arrive at the same time.

Figure 2. Illustration of Lane-to-Lane Skew.

on all of the lanes with a common clock (one of the recovered clocks could be used as this common clock). This aligns the received symbols on clock boundaries. They may still be out phase with respect to each other, but now they are out of phase in 360 degree increments (clock cycles). An illustration showing this concept can be found in Figure 3. (We will come back and revisit this first step and how it relates to the Elastic Buffer in the next paragraph). At this point, the symbols are all transitioning on clock boundaries (their phase differences are now in increments of clocks, as shown in Figure 3), so the second step of the deskewing procedure is simply to delay the "faster" lanes by the appropriate number of clocks. Now the data has been deskewed across all of the lanes for the receiver to process that information.

The first step of the deskewing process described above (and shown in Figure 3) can also be accomplished by using the skewed data as the input to the Elastic Buffer. The symbols in each lane are pulled out of the Elastic Buffers using the same clock edge accomplishing step one (because the skew is then in increments of the clock). Therefore, by placing the Elastic Buffer before the Deskew circuitry, half of the Deskew circuitry's job is already finished.

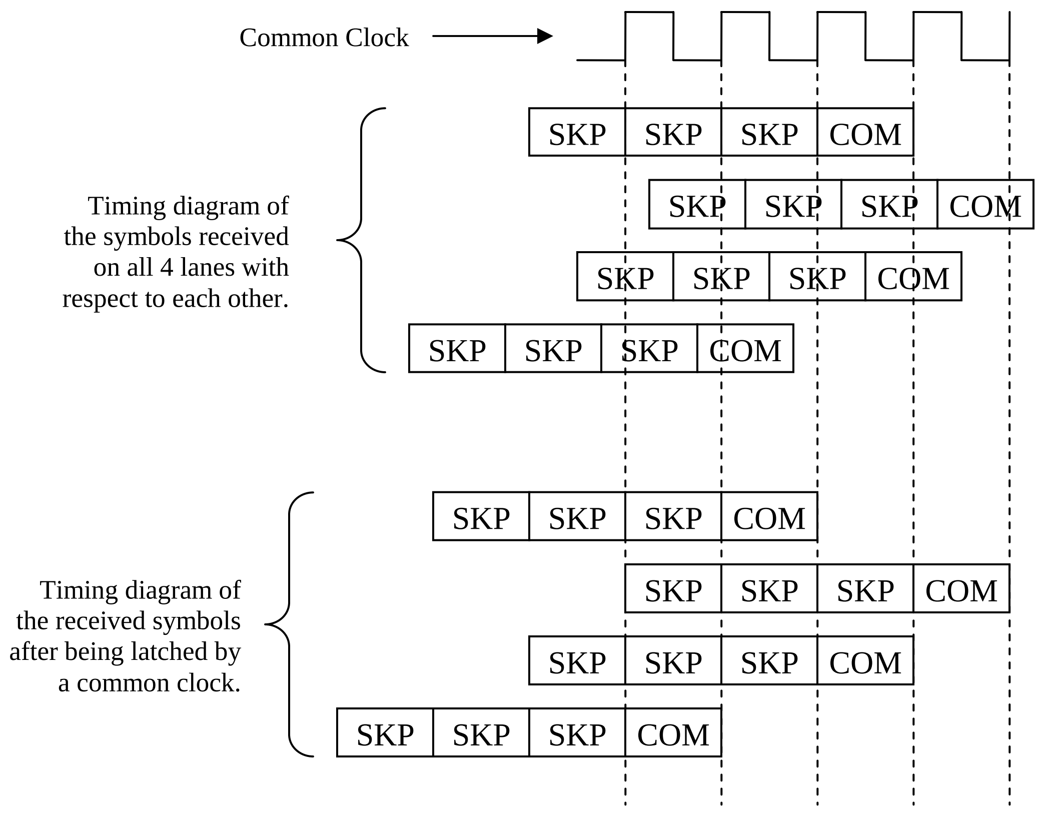
Common Clock
SKP SKP SKP COM
Timing diagram of the symbols received on all 4 lanes with respect to each other.
Timing diagram of the received symbols after being latched by a common clock.

Figure 3. Aligning Skewed data on a common clock boundary.

Implementations and Sizes

This section identifies two of the most common Elastic Buffer implementations, describe their functionality, and calculate the minimum size of each based on the device's characteristics. The sizes of these buffers are dependent on three factors: the interval at which the transmitter schedules SKP ordered-sets to be sent, the Max Payload Size supported by the device, and the Link Width. Each of these terms are used to determine the maximum amount of time between SKP ordered-sets arriving at the receiver. The maximum amount of time between SKP ordered-sets being received may be longer than the specified 1180 – 1538 symbol times scheduling interval. For example, if the transmitter is currently in the middle of sending a TLP when it determines that another SKP ordered-set needs to be sent, it must finish transmitting the current packet before sending the SKP ordered-set. This is why the Max Payload Size supported by a device and the Link Width need to be considered, because the amount of time it takes to finish transmitting that packet affects the amount of time between SKP ordered-sets being transmitted on the link.

The worst-case scenario would be when two PCI Express devices are connected together with a x1 link, the transmitter has sent the first symbol of a maximum sized packet when it determines that it needs to send a SKP ordered-set. To properly calculate the sizes of the Elastic Buffer implementations (described later) we first must determine the number of symbols that can shift during this worst-case scenario. As mentioned earlier, based on the clock tolerance allowed by the spec, two connected devices could have a symbol shift every 1666 symbol times. Therefore, the number of symbols that could shift during the worst-case scenario is described in the equation below.

$$Max\_Symbols\_Shifted = \frac{\left(\frac{Max\_Payload\_Size + TLP\_Overhead}{Link\_Width}\right) + 1538}{1666}$$

The *Max_Payload_Size* term in the equation above is the only variable and is based on the Max Payload Size parameter of the device. The *TLP_Overhead* term is a constant of 28 symbol times and is comprised of:

- start framing symbol (1)
- sequence number (2)
- header (16)
- ECRC (4)
- LCRC (4)
- end framing symbol (1)

The *Link_Width* term should be a constant of 1 and **NOT** the max link width that the device supports. This is because the smaller the link width, the longer the interval is between SKP ordered-sets, which is a component of the worst-case scenario. Since a device may not always know what the link width capability will be of the device it is connected to, the required link width of x1 should always be used in this equation. The 1538 value in the equation represents the max interval between SKP ordered-sets being scheduled by the transmitter. A sample calculation is shown below indicating the maximum number of symbols that can shift given a device's characteristics. Table 1 shows the results of the equation based on different Max Payload Sizes.

$$\frac{\left(\frac{4096 + 28}{1}\right) + 1538}{1666} = 3.4 Max\_Symbols\_Shifted$$

Max_Payload_Size	128	256	512	1024	2048	4096
Max_Symbols_Shifted	**1.02**	**1.09**	**1.25**	**1.55**	**2.17**	**3.40**

Table 1. Max number of shifted symbols based on payload size.

Primed Method

The first of two Elastic Buffer implementations described in this paper is what we will refer to as the Primed Method. This implementation involves a buffer whose normal state is half-full (primed) with enough entries before and after the middle entry to ensure that the max clock differences can be compensated for by inserting or removing SKP symbols from SKP ordered-sets. This implementation is illustrated in Figure 4.

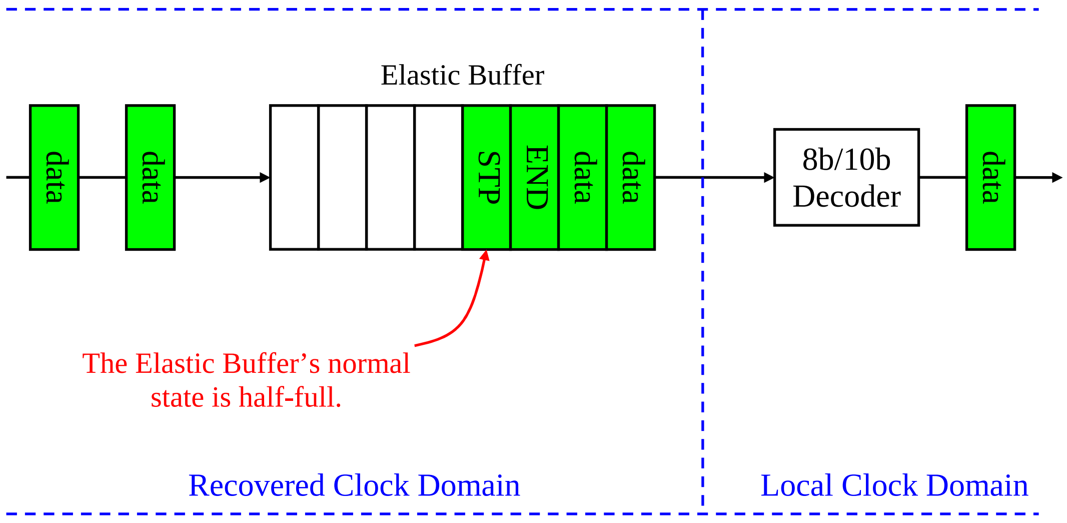

Figure 4. Implementation of Elastic Buffer using Primed Method

The number of entries before and after the middle entry is determined by the maximum number of symbols that could shift between received SKP ordered-sets. The number of symbol shifts is given in Table 1, based on the *Max_Payload_Size* parameter of the device. For example, a device which is capable of receiving a packet with a payload size of 4096 bytes should have an Elastic Buffer that is 8 entries deep. This is determined by taking the 3.40 result from Table 1 for a *Max_Payload_Size* of 4096 and rounding that value up to 4, then doubling that to allow for shifting in either direction. If the *Max_Payload_Size* parameter for a device is 2048, then using the result from Table 1, its Elastic Buffer should be 6 entries deep.

The goal of the Primed Method is to keep the Elastic Buffer half-full. Since the Recovered Clock Domain and the Local Clock Domain will have slightly different frequencies, the Elastic Buffer's fill level will vary depending on the frequency difference of the two clock domains and the interval between received SKP ordered-sets. However, each time the device does receive one (or more) SKP ordered-sets, it returns the Elastic Buffer to the half-full state by inserting or removing SKP symbols from these SKP ordered-sets. (As mentioned earlier, it should be noted that not more than 2 SKP symbols can be inserted or removed from a single SKP ordered-set.) Examples of this process are shown in Figure 5, and also in Figure 6a and 6b.

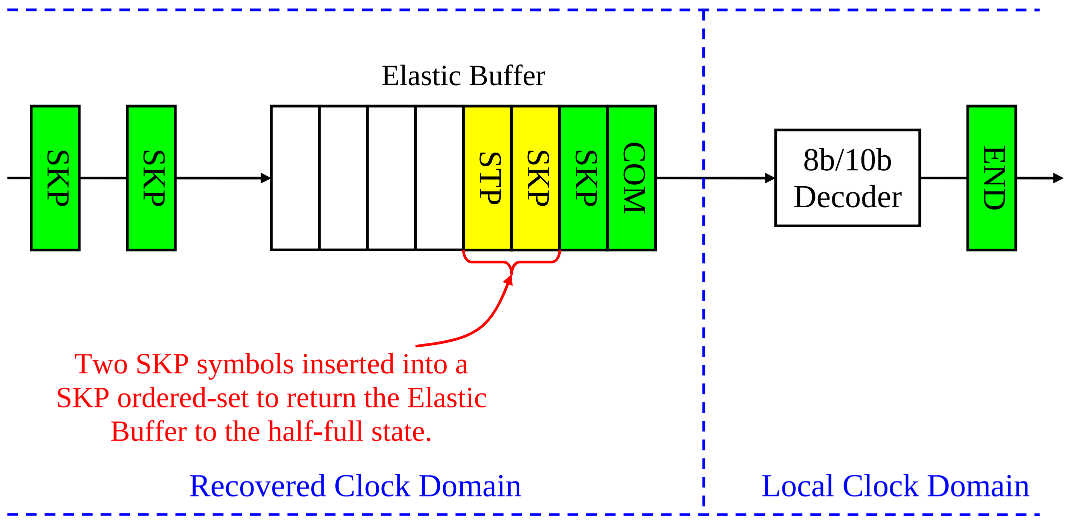
Elastic Buffer
SKP
SKP
STP
SKP
SKP
COM
8b/10b Decoder
END
Two SKP symbols inserted into a SKP ordered-set to return the Elastic Buffer to the half-full state.
Recovered Clock Domain
Local Clock Domain

Figure 5. Local Clock Faster than Recovered Clock

Figure 5 illustrates the scenario where the Local Clock is faster than the Recovered Clock, so two more symbols have been pulled out of the Elastic Buffer than have been deposited into it. This is compensated for by inserting two additional SKP symbols into the received SKP ordered-set.

In Figure 6a, the Elastic Buffer is getting close to overflowing because the Local Clock is slower than the Recovered Clock and a SKP ordered-set has not been received in a long time. This situation occurs when a very large packet (possibly of max size) is being transmitted and thus SKP ordered-sets cannot be sent until the device finishes transmitting the TLP. In this case, a single SKP ordered-set is not sufficient to return the Elastic Buffer to its half-full state. However, since the SKP ordered-sets that were scheduled to be sent during the transmission of the large TLP are accumulated, the Elastic Buffer will receive multiple SKP ordered-sets in a row following the end of the large TLP. The receiving device uses these subsequent SKP ordered-sets to return the Elastic Buffer to its normal state.

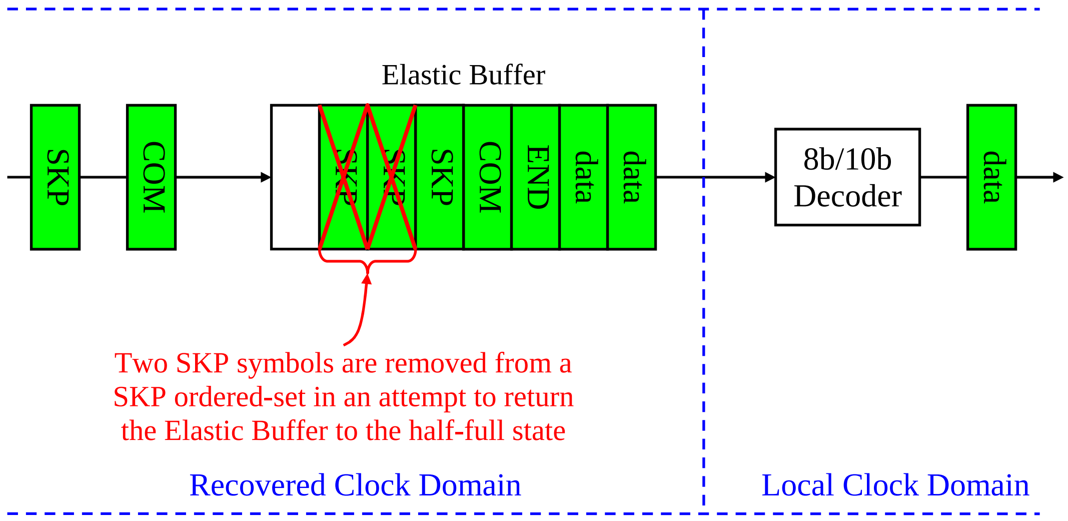
Elastic Buffer
SKP
COM
SKP
SKP
SKP
COM
END
data
data
8b/10b Decoder
data
Two SKP symbols are removed from a SKP ordered-set in an attempt to return the Elastic Buffer to the half-full state
Recovered Clock Domain
Local Clock Domain

Figure 6a. Local Clock is Slower than Recovered Clock

In this paper the examples show two SKP symbols being inserted and removed from SKP ordered-sets, which is compliant with the spec. However, it should be noted that some documents attempting to standardize portions of the PCI Express PHY, such as Intel's PIPE (*Physical Interface for PCI Express*) document, only allow one SKP symbol to be inserted or removed from a single SKP ordered-set.

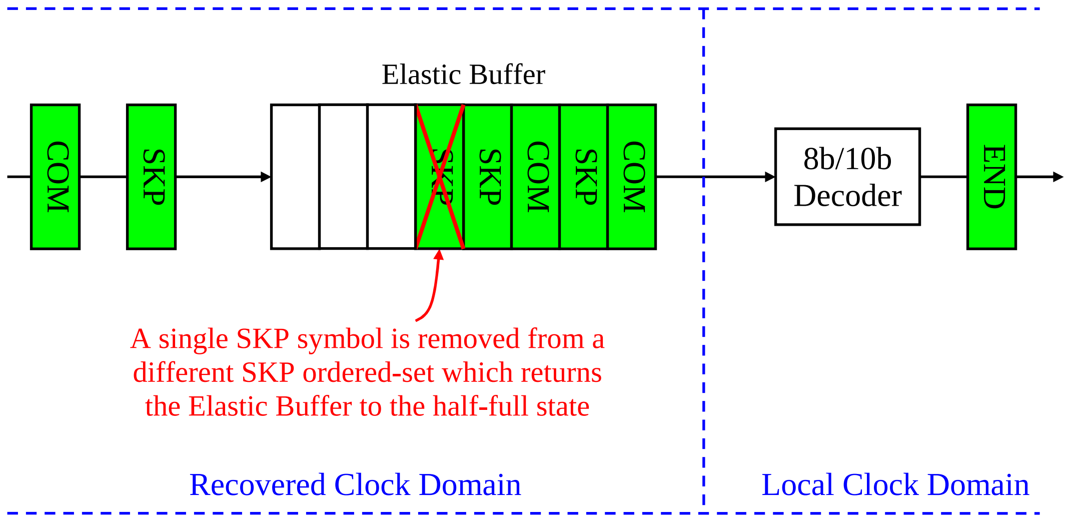
Elastic Buffer
COM
SKP
SKP
SKP
COM
SKP
COM
8b/10b Decoder
END
A single SKP symbol is removed from a different SKP ordered-set which returns the Elastic Buffer to the half-full state
Recovered Clock Domain
Local Clock Domain

Figure 6b. An additional SKP ordered-set must be used for clock tolerance compensation.

Flow Control Method

Implementing an Elastic Buffer using the Flow Control Method differs from an implementation based on the Primed Method because it does not use the same mechanism to compensate for the case where the Local Clock Domain is faster than the Recovered Clock Domain. In other words, the Flow Control Method does not use SKP ordered-sets when compensating for the case described above. However, SKP ordered-sets are still needed when compensating for the case where the Recovered Clock Domain is faster than the Local Clock Domain.

When the Local Clock Domain is faster than the Recovered Clock Domain, the Flow Control Method simply gates off the clocks supplied to the logical blocks upstream of the Elastic Buffer when the bottom entry of the buffer is not valid. Using this method, the device allows the Elastic Buffer to underflow and simply notifies the logical blocks upstream of the event. So instead of inserting an additional meaningless symbol (SKP) into the data flow to prevent an underflow condition as the Primed Method does, the Flow Control Method effectively stalls the upstream logical blocks for a single cycle when an Elastic Buffer underflow condition occurs. A diagram illustrating this method is shown in Figure 7.

When the Recovered Clock Domain is faster than the Local Clock Domain, the Flow Control Method works exactly the same as the Primed Method. SKP symbols are removed from SKP ordered-sets to return the Elastic Buffer to its normal state. The only difference is with the Flow Control Method, the normal state of the buffer is having only a single entry valid instead of having half of the entries valid as with the Primed Method.

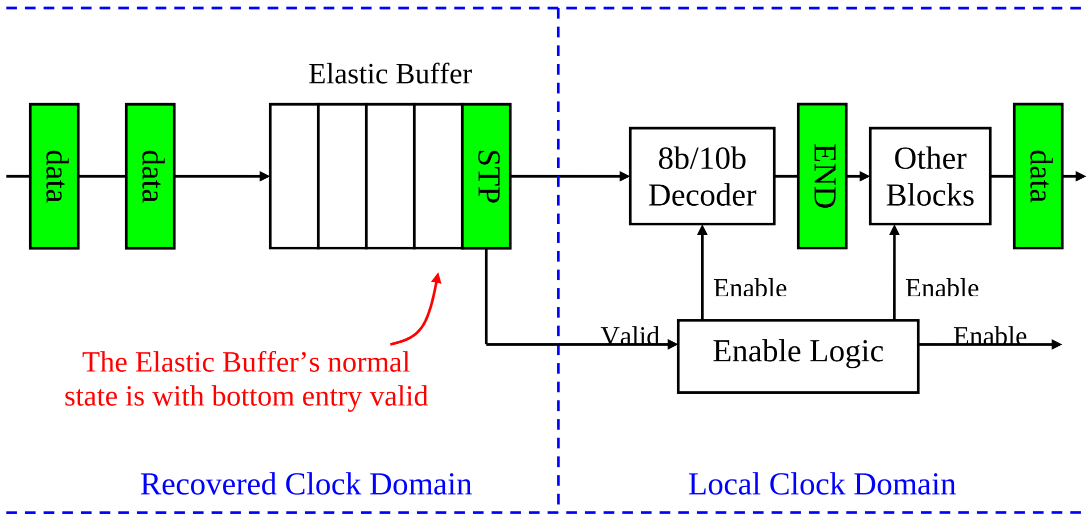

Figure 7. Implementation of Elastic Buffer using the Flow Control Method

The size of an Elastic Buffer that uses the Flow Control Method is also determined by the maximum number of symbols that could shift between received SKP ordered-sets. The number of symbol shifts is based on the *Max_Payload_Size* parameter of the device, and is given in Table 1. For example, a device which is capable of receiving a packet with a payload size of 4096 bytes should have an Elastic Buffer that is 5 entries deep. This is determined by taking the 3.40 result from Table 1 for a *Max_Payload_Size* of 4096 and rounding that value up to 4, then adding one to that for the valid bottom entry. If the *Max_Payload_Size* parameter for a device is 1024, then using the result from Table 1, its Elastic Buffer should be 3 entries deep.

The Flow Control Method has a small advantage over the Primed Method in terms of latency. Since the Prime Method tries to keep the Elastic Buffer half-full, received symbols will spend more time in a Primed Method Elastic Buffer than in a Flow Control Method Elastic Buffer. However, this latency difference between the two implementations will be very minor (a max difference of 3 symbol times). Something else that should be taken into consideration when choosing one of the described methods is that the Flow Control Method could be more complex to implement, depending on how the other logical blocks in the system are implemented and how easy it is to enable or disable them.