Introduction to USB 3.0

By Donovan (Don) Anderson, Vice President, MindShare, Inc.

This paper is a brief review of the USB 3.0 implementation, focusing on USB 2.0 backward compatibility and on the major features associated with the SuperSpeed (SS) bus. The goal is to provide the reader with a short and concise description of USB 3.0, and enough detail to give a good feel for the technology, protocols, and techniques.

Due to the limited scope of this paper, some terminology and concepts are introduced but not fully developed. A MindShare Comprehensive USB 3.0 book is in the works that will provide all of the details. In the meantime, please check our website at www.mindshre.com to learn the availabiltity of our USB 3.0 classes and eLearning courses.

Motivation for USB 3.0

USB 3.0 enables more demanding applications compared to USB 2.0 by addressing its limitations:

- Bandwidth - 5.0 Gb/sec SuperSpeed (SS) vs. 480 Mb/sec (High Speed) rate
- Power Conservation - link power states (U0 - U3) and function power management
- Data Flow Control - poll once versus poll multiple times
- Error Handling - End-to-end and port-to-port error detection and retries versus only end-to-end retries with USB 2.0.

The additional bandwidth provided by USB SS transactions can benefit applications like real-time audio and video streaming that require higher bus bandwidth at regular intervals.

Mass storage applications can also benefit from the SS bandwidth. For example, Table 1 lists approximate download times for the different transmission rates.

Table 1: Download Speeds

	SD Movie - 6GB	USB Flash 16GB	HD Movie - 25GB
USB 1.0 (FS)	~ 2 hours	~ 6 hours	~ 9.25 hours
USB 2.0 (HS)	~ 3.25 minutes	~ 9 minutes	~ 14 minutes
USB 3.0 (SS)	~ 20 seconds	~ 54 seconds	~ 70 seconds

SD= Stanrdard Definition; HD = High Definition (Source: USB-IF)

USB 3.0 Topology

Figure 1 provides an example of a USB 3.0 topology. A major feature of this topology is its support of all wired USB speeds (LS, FS, HS & SS), and this is accomplished via the two separate buses that are integrated into USB 3.0 cables, connectors and hubs. In the illustration, the SS bus is represented in red and consists of two differential signal pairs, one to transmit packets and one to receive. The standard USB 2.0 bus consists of a single differential pair that operates in a half-duplex model. Notice also that SS devices connect to both the SS and USB 2.0 buses, and provide backward compatibility with older platforms that don't support SS.

Figure 1: Example USB 3.0 Topology

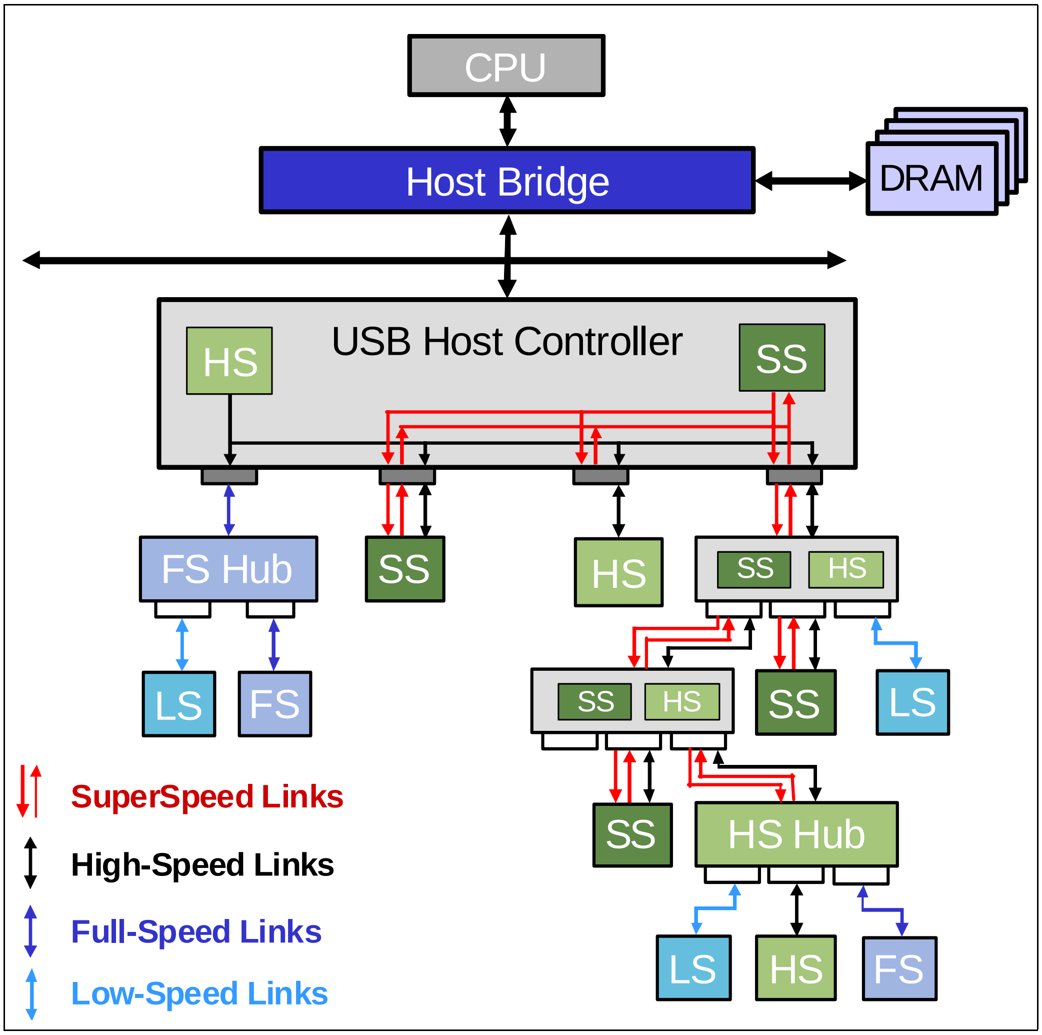
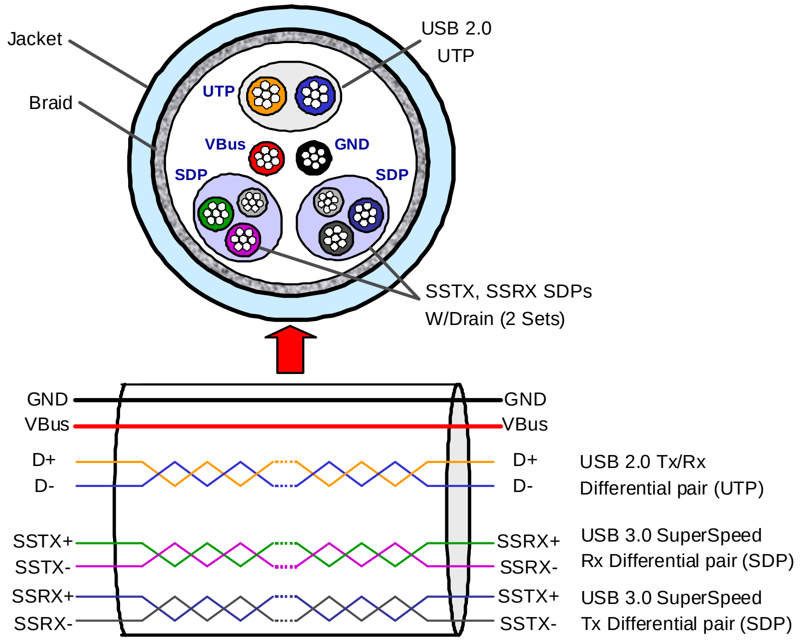


Figure 2 depicts the cross-section of a USB 3.0 cable and illustrates the SS and USB 2.0 buses along with the VBUS power pin that supplies power at 5 vdc and up to 900mA.

The SS bus employs a dual-simplex approach that allows simultaneous transmission and reception of packets. There are many cases where an SS device may be both transmitting and receiving data at the same time. For example, during burst transactions a device may be receiving data from the host and returning acknowledgements associated with data already received.

Figure 2: USB 3.0 Cable


Jacket
Braid
USB 2.0 UTP
UTP
VBus
GND
SDP
SDP
SSTX, SSRX SDPs W/Drain (2 Sets)
GND
VBus
D+
D-
SSTX+
SSTX-
SSRX+
SSRX-
GND
VBus
D+
D-
SSRX+
SSRX-
SSTX+
SSTX-
USB 2.0 Tx/Rx Differential pair (UTP)
USB 3.0 SuperSpeed Rx Differential pair (SDP)
USB 3.0 SuperSpeed Tx Differential pair (SDP)

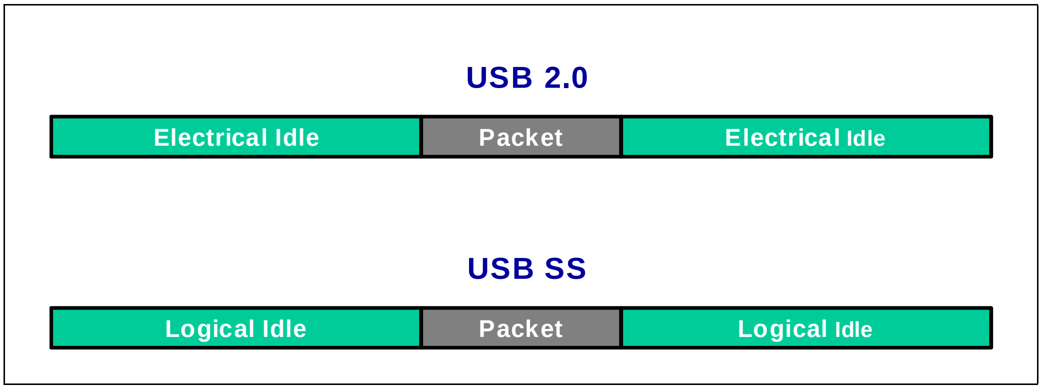
USB 2.0 Links Versus SuperSpeed Links

Unlike the USB 2.0 bus, SS links are constantly transmitting and receiving traffic to maintain synchronization in preparation for delivering the next packet. Each SS link must be trained at startup so the receivers can establish bit lock at the 5.0

Gb/s rate. When an SS link is not transmitting a packet it sends traffic to keep the link operational, known as logical idle packets. Consequently, power consumption on SS links would be very high if they didn't aggressively transition frequently into low power states (more details on link power management features are covered later). In contrast, USB 2.0 links are in the electrical idle state until it's time to send a packet (see figure 3).

To recognize packets on an SS link, a unique start-of-packet delimiter called an ordered set is required. SS USB uses a variety of ordered sets to identify the type of packet being sent.

Figure 3: Link States During Idle



SS Protocol Improvements

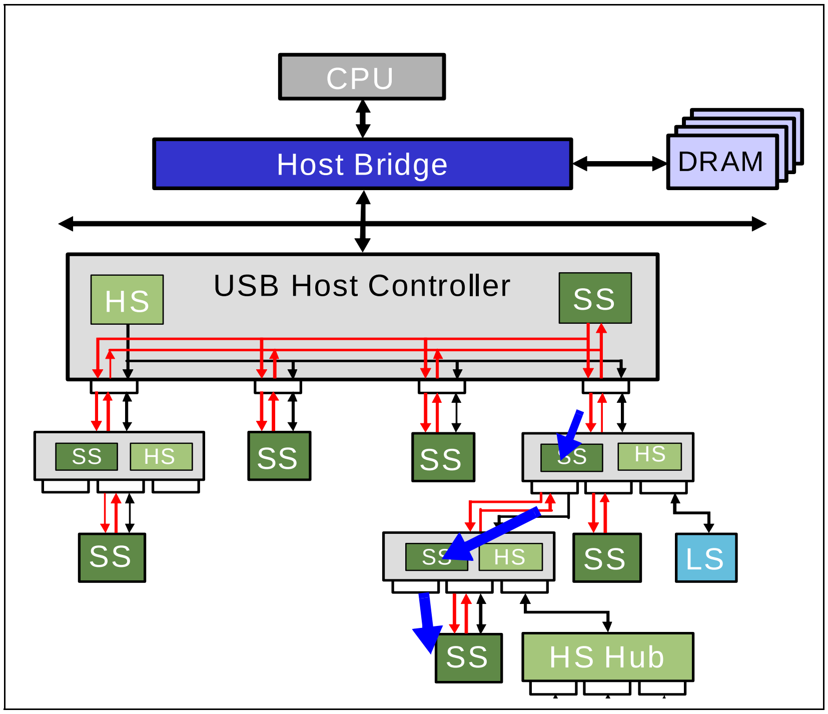
SS packet protocol is derived from the same Token/Data/Handshake model employed by USB 2.0, often referred to as the end-to-end protocol (See figure 4). Like USB 2.0 all transactions originate at the host, but SS improves the protocol and adds several new features to give better performance, efficiency, and power conservation, such as:

- **Unicast** transactions versus **broadcast**
- More efficient Token/Data/Handshake Sequence
- Data Bursting
- Improved end-to-end data Flow Control (poll once versus poll multiple)

Unicast Transactions

SS transactions are routed directly from a root port to the target device, so only links in the direct path between the root port and target device see the traffic. That lets other links in the topology to enter or remain in a low power state. Figure 4 illustrates the direct routing used when forwarding packets from the host to a target device.

Figure 4: Unicast Transactions



Token / Data / Handshake Sequences

An mentioned earlier, the SS end-to-end protocol is based on the standard USB 2.0 "Token/Data/Handshake" sequence. This section illustrates the differences between the USB 2.0 and SS implementations through an IN and OUT example.

IN Transaction Examples

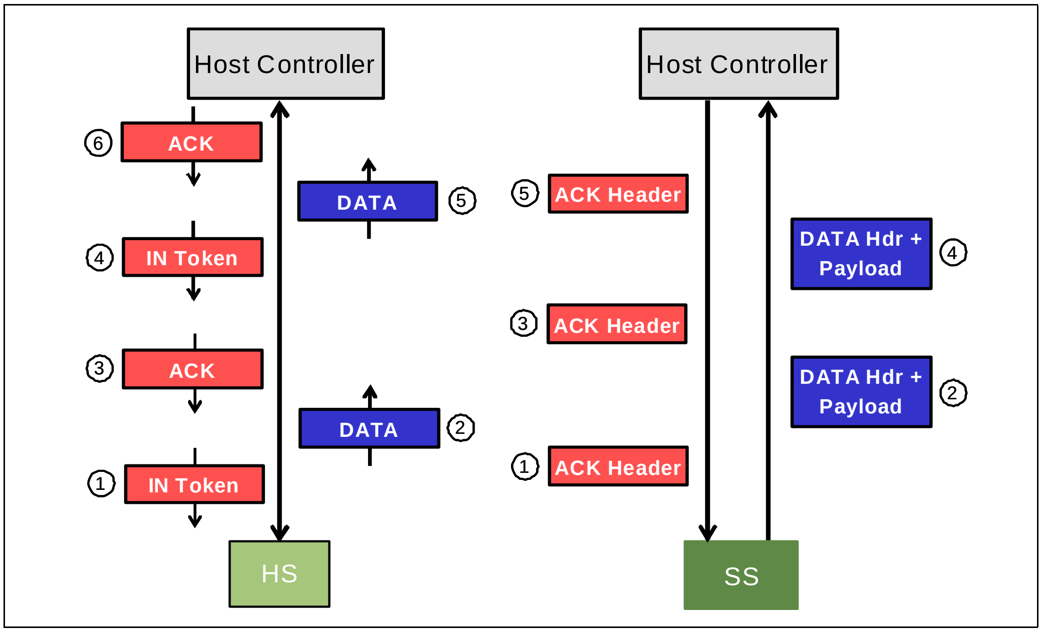
Consider the example IN transaction in figure 5. The left side indicates the sequence of packets required to perform two back-to-back "token/data/handshake" transactions, requiring 6 packets be exchanged as follows:

1. Host broadcasts an IN Token packet (1) to initiate the transaction.
2. Device returns the requested DATA packet (2).
3. Host acknowledge receipt of data with an ACK handshake packet (3).
4. Steps 1-3 are repeated.

The example on the right indicates the packet sequence needed perform two back-to-back SS IN transactions, which requires only 5 packets be exchanged.

1. SS USB uses an ACK header (packet 1) to initiate an IN transaction.
2. The SS device returns Data (packet 2).
3. The second ACK header (3) both acknowledges receipt of the data and requests a second transaction.
4. The second Data packet (4) is delivered by the device.
5. The final ACK header (5) acknowledges receipt of the data, but does not request additional data.

Figure 5: Two Back-to-Back IN Transactions -- USB 2.0 versus SS



OUT Transaction Examples

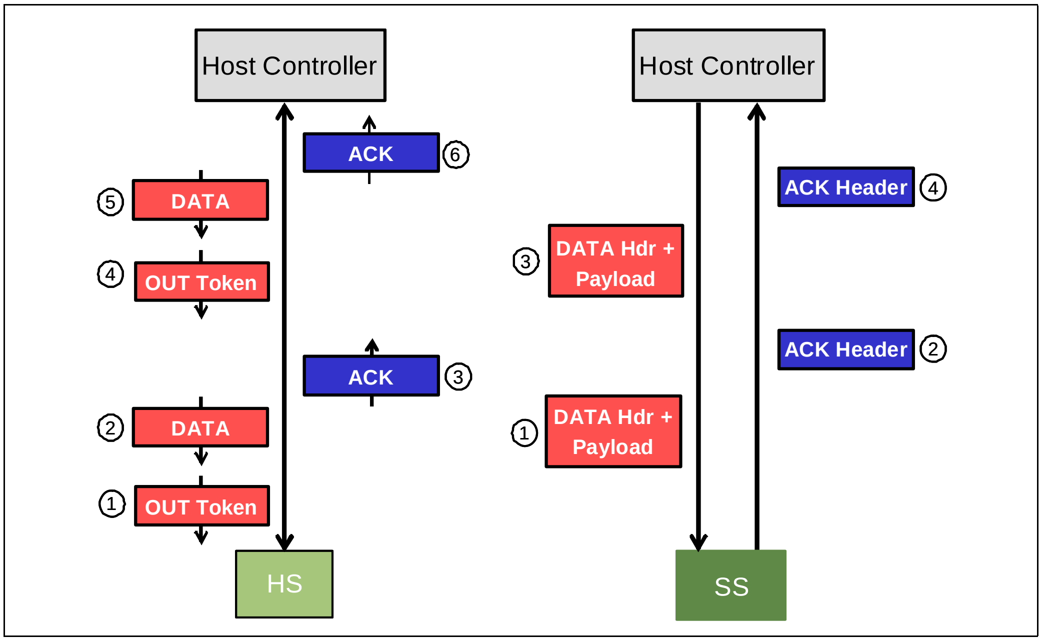
Differences between USB 2.0 and SS OUT transactions are illustrated in Figure 6. The example on the left depicts two back-to-back OUT transactions that require 6 packets:

1. Host broadcasts an OUT Token packet (1) to initiate the transaction.
2. Host sends DATA packet (2) to the Device.
3. Device acknowledges receipt of data with an ACK handshake packet (3).
4. Steps 1-3 are repeated

The right side of Figure 6 indicates the packet sequence required to perform two back-to-back SS OUT transactions, but requires only 4 packets be exchanged.

1. SS USB uses a DATA header (packet 1) to initiate an OUT transaction and to deliver data to the device.
2. Device acknowledges receipt of data via an ACK packet (2).
3. The second DATA packet (3) initiates the second transaction and delivers data to the device.
4. Device acknowledges receipt of data via an ACK packet (4), completing the sequence.

Figure 6: Two Back-to-Back OUT Transactions -- USB 2.0 versus SS

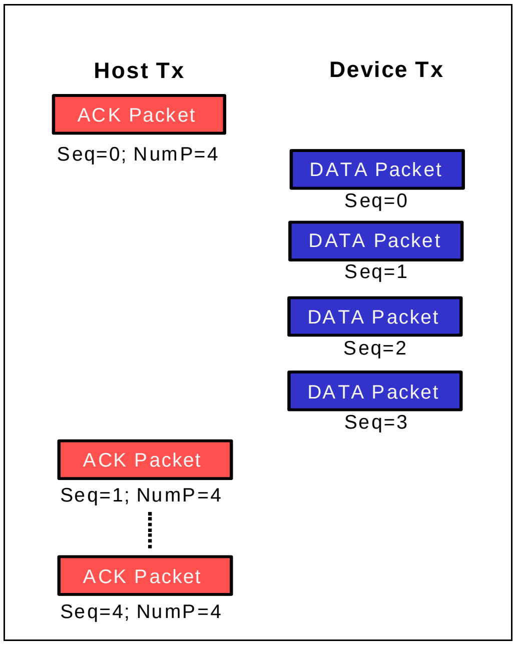


Data Bursting

SS end-to-end protocol permits data bursting to improve latency and performance. Bursting allows the host to continue sending or receiving data as long as the device can continue the transfer. Devices report their ability to support bursting in their device descriptors. The maximum burst size is 16 and the actual number to be used represents the number of DATA packets that can be sent without receiving an acknowledgement.

This bursting approach is exemplified in Figure 7 with an IN endpoint that supports a burst size of four. The host initiates the burst transfer and indicates the expected sequence number of the first DATA packet returned (Seq=0) and the number of packets it wishes to receive (NumP=4). The target device responds with a burst sequence of 4 DATA packets without receiving any handshakes. A fifth data packet cannot be returned until DATA packet zero is acknowledged and the host has indicated a request for another DATA packet (i.e., a second ACK packet with NumP=4). In this burst example, the host continues to request additional data by keeping the NumP value at 4.

Figure 7: Example Burst IN Transaction

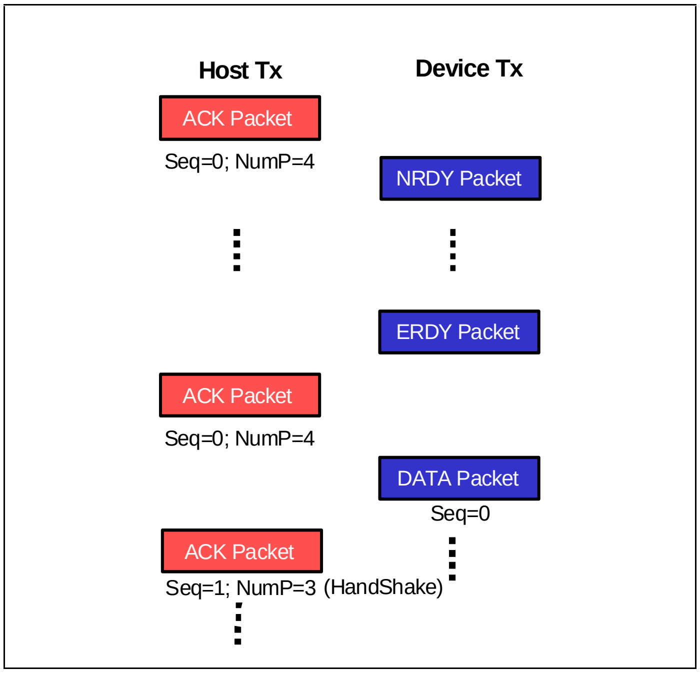


End-to-End Flow Control

USB 2.0 uses polling and the NAK handshake packet for flow control. For example, USB keyboards must be constantly polled by host software to check for activity. When an IN Token packet is delivered and no keyboard activity has occurred, the keyboard will return a NAK packet. Subsequently, host software will poll the device again and receive another NAK. This process continues until there is renewed activity.

SS flow control uses a poll-once approach coupled with an asynchronous ready notification. Consider the IN transaction illustrated in Figure 8. An ACK packet initiates the IN transaction and the keyboard has nothing to report and returns NRDY (Not Ready). This notifies the host that the device will send an ERDY (Endpoint Ready) notification when keyboard activity resumes, so the host doesn't need to continue polling. This can significantly reduce SS traffic and improve link power management.

Figure 8: Example SS Flow Control Sequence



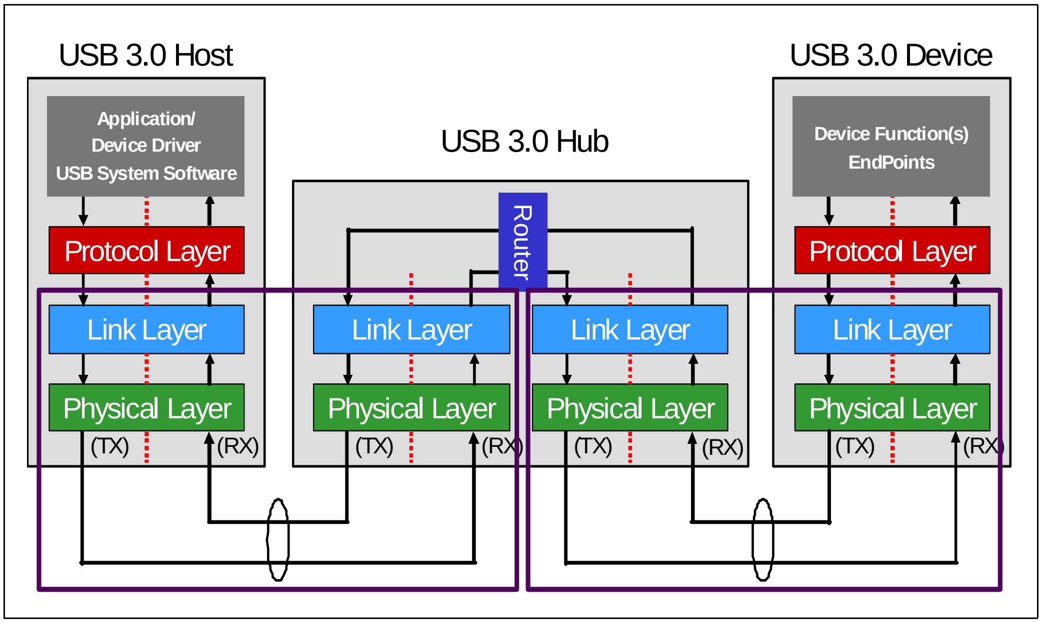
Port-to-Port Protocol

SS USB implements protocols on a link-by-link basis much like PCI Express does. Figure 9 illustrates the layers associated with the SS link interface. Note that the protocol layer focuses on the End-to-End protocol described previously. The Port-to-Port protocol involves the Link and Physical layers that manages traffic between link partners and includes:

- **Link Flow Control** — Credit-based checks that ensure receive buffer space is available at the link partner before sending a header packet
- **Link Transmission Verification** — CRC and Sequence Number checks to verify successful delivery of each packet to the link partner and perform retries if transmission fails.

Note that while Data Header Packets are subject to Link Flow Control and Link Transmission Verification, the Data Packet Payloads (DPPs) are not. Instead, DPP flow control and error handling is managed by the End-to-End protocol.

Figure 9: SuperSpeed Interface Layer and Port-to-Port Association



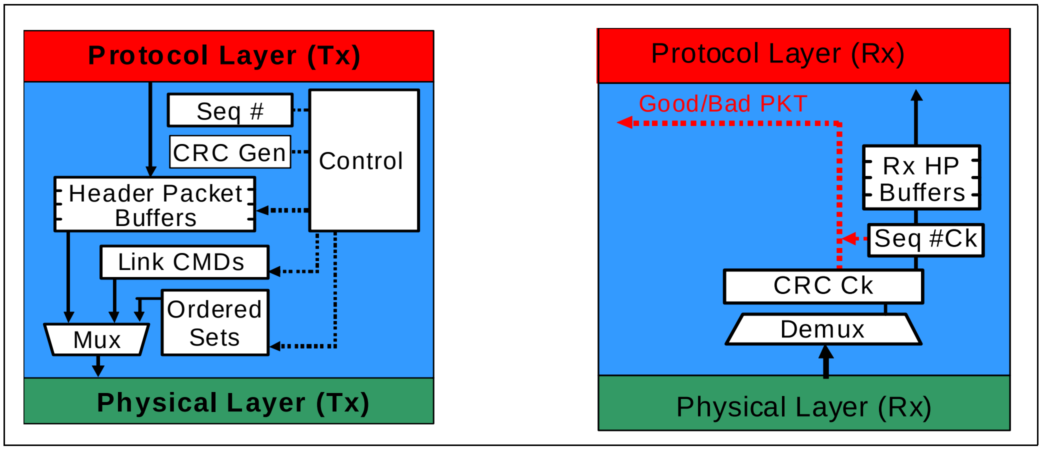
Protocol Layer packets (End-to-End packets) are prepared for delivery by the Link and Physical layers prior to being transmitted across the physical link. The following sections summarize this operation.

Link Layer Transmission and Reception

Figure 10 illustrates the primary Link layer blocks associated with the transmission and reception of Protocol packets. The link layer manages Link Flow Control and Link Transmission Verification. During transmission several different things are sent, as listed here along with the related action at the receiver:

- **Sequence Number Assignment** — A sequence number is assigned for each header packet and checked by the link partner to ensure packets are delivered in order. The value is discarded by the receiver after the check is made.
- **CRC Generation** — The link layer generates a 16-bit CRC to cover the first 12 bytes of the header. The CRC is checked by the receiver prior to checking the sequence number. If both checks pass then a Good link command packet is transmitted back to the link partner. If either check fails a Bad link command packet is returned to the link partner.
- **Header Packet Buffers** — Four buffers are used to store Header packets at the transmitter until flow control credits are verified. Once that's done, the header packet is sent to the link partner and a copy is retained until a Good link command is received. Retaining a copy allows the header to be retried if a bad link command is received.
- **Link Commands** — Link commands are used for flow control, verifying successful transmission, and link power management.
- **Ordered Sets** — Ordered sets ususally consist of bit patterns useful in packet framing, link training, clock compensation, etc., and are commonly used in other high-speed serial bus implementations. They consist of a sequence of 10-bit symbols that are uniquely identifiable by the receiver. Ordered Sets always begin with a Control (K) symbol followed by more K symbols or Data symbols.

Figure 10: Link Layer Functions



Physical Layer Transmission and Reception

The physical layer prepares packets for delivery across the differential pair. This involves several steps as illustrated in figure 11 and described below.

Transmission:

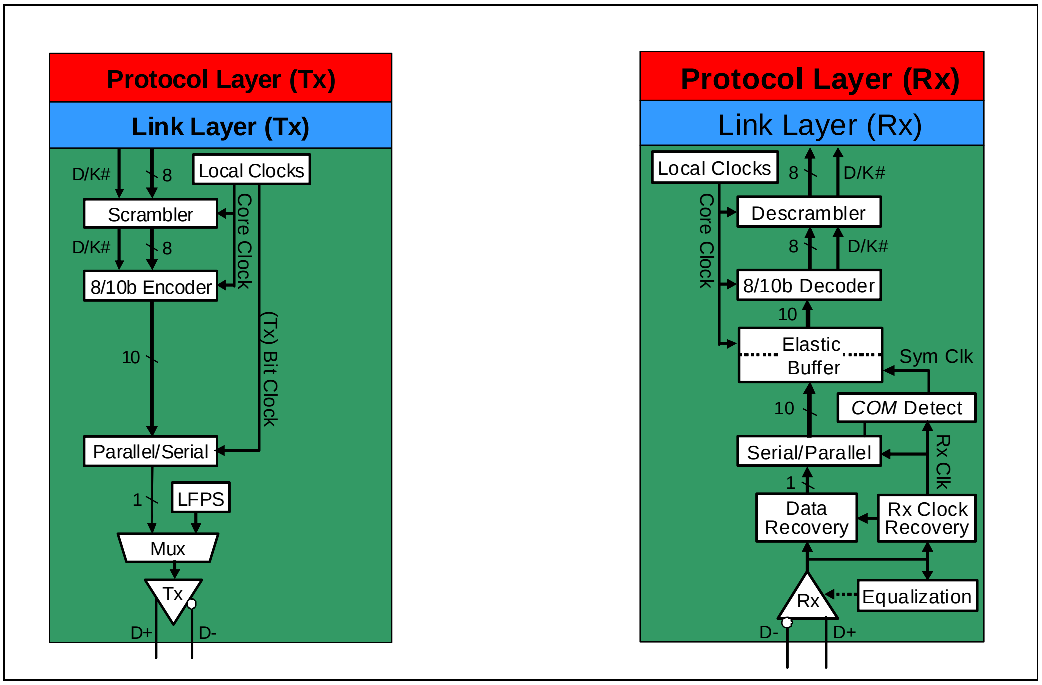
- Scrambling — Scrambling reduces EMI problems associated with repeated patterns in the data being sent across an SS link. The scrambler output is simply XORed with each byte of data to eliminate the repeated patterns.
- 8/10b Encoding — every byte that traverses the link is first converted into a 10-bit value called a symbol (this is a common encoding scheme in high-speed serial designs).
- Parallel/Serial Convertion — Bytes are converted to bit stream
- LFPS — Low Frequency Periodic Signaling is typically used in situations where the link is in an electrical idle state.
- Differential Transmission — Packets are clocked onto the link at a 5.0 Gb/s rate.

Reception:

- Differential Reception — the scrambled and encoded data is received and forwarded to the recovery blocks.
- Clock and Data Recovery — the clock is extracted from the bit stream and data is clocked into the serial/parallel converter.

- Serial/Parallel Conversion — data is clocked into the converter and 10-bit symbols are clocked into the elastic buffer.
- Elastic Buffer — The elastic buffer must absorb the worst-case clock variation between the transmitted clock frequency (recovered) and the local receive clock. The maximum variance is +300 to -300ppm. The buffer must also accommodate variations resulting from the Spread Spectrum clocking. Compensation is achieved via SKP ordered sets that are periodically inserted into the bit stream.
- 8/10b Deocoding — 10-bit symbols are converted back to bytes.
- UnScrambling — the same scrambling output is XORed with the scrambled data a second time to recover the original data.

Figure 11: Physical Layer Transmit and Receive Functions

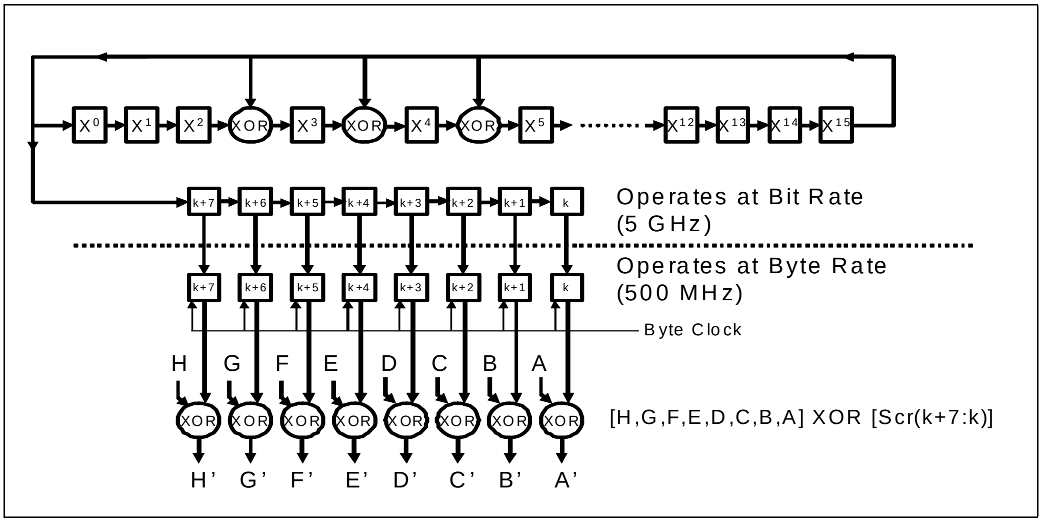


The Scrambler

Scrambling reduces repeated patterns in the bit stream and lowers EMI by preventing the concentration of emitted energy at only a few frequencies. Scrambling works by generating a pseudo-random data pattern that is XORed with the outgoing bit stream. The algorithm used for scrambling data is expressed as a polynomial implemented as a linear feedback shift register (LFSR) as shown in figure 12. The scrambler used in SS USB is identical to the scrambler used in PCIe 2.0 and uses the polynomial: G(x) = X16+X5+X4+X3+1.

An identical scrambler at the receiver applies the same output to the scrambled data a second time to recover the original value (Descrambling). The scramblers are periodically reset (loaded with FFFFh) to ensure they stay synchronized with each other. Ordered sets are never scrambled so that they will be recognized by the receiver even if the scramblers did get out of sequence for a time.

Figure 12: Scrambler



8/10b Encoding and Decoding

One of the major goals of 8b/10b encoding is to embed a clock into the serial bit stream before transmission across the link. This eliminates the need for a high frequency 5.0 GHz clock signal on the link that could generate significant EMI.

Every byte to be sent is converted to a 10-bit value, called a symbol. Figure 13 illustrates a look-up table associated with the encoder. As the two tables suggest, two types of information are encoded:

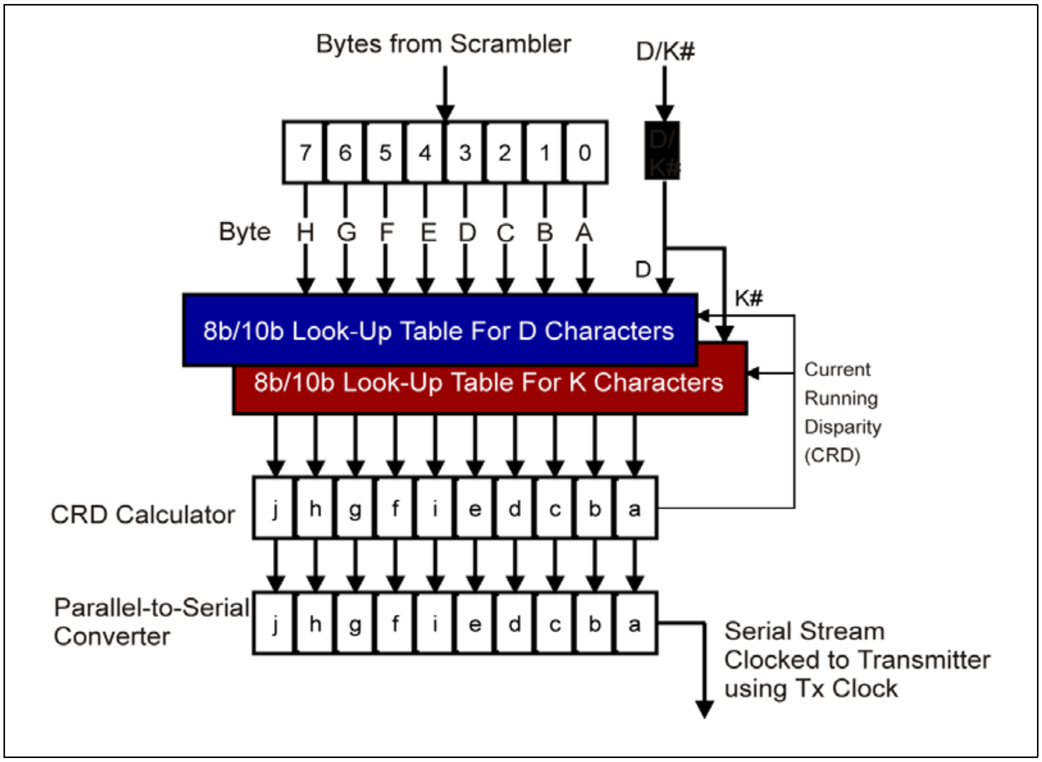
- Data bytes — consisting of every byte send across the link except ordered sets. The data lookup table must support the 256 possible input values.
- Control bytes— used in ordered sets.

The 8b/10b encoding scheme is also designed to prevent “DC wander”, meaning the possibility that too many bits of one polarity could interfere with the ability of the receiver to properly see them. This is handled by ensuring that the number of ones and the number of zeroes sent are balanced, referred to as maintaining DC balance. To do this, the transmitter observes whether the previous symbol that was sent had more ones or zeroes. This imbalance is referred to as disparity, and the possibilities for each 10-bit symbol are shown here:

- 5 zeros and 5 ones - neutral disparity
- 6 zeros and 4 ones - negative disparity
- 4 zeros and 6 ones - positive disparity

By tracking the disparity needed for the next symbol as the current running disparity (CRD), the transmitter can balance the ones and zeroes over time. Each entry within the lookup table has two possible symbol outputs depending on the disparity of the last non-neutral symbol delivered. The receiver checks to ensure that disparity is working properly, and that provides a mechanism for detecting most transmission errors. Figure 13 illustrates the 8/10b decoder.

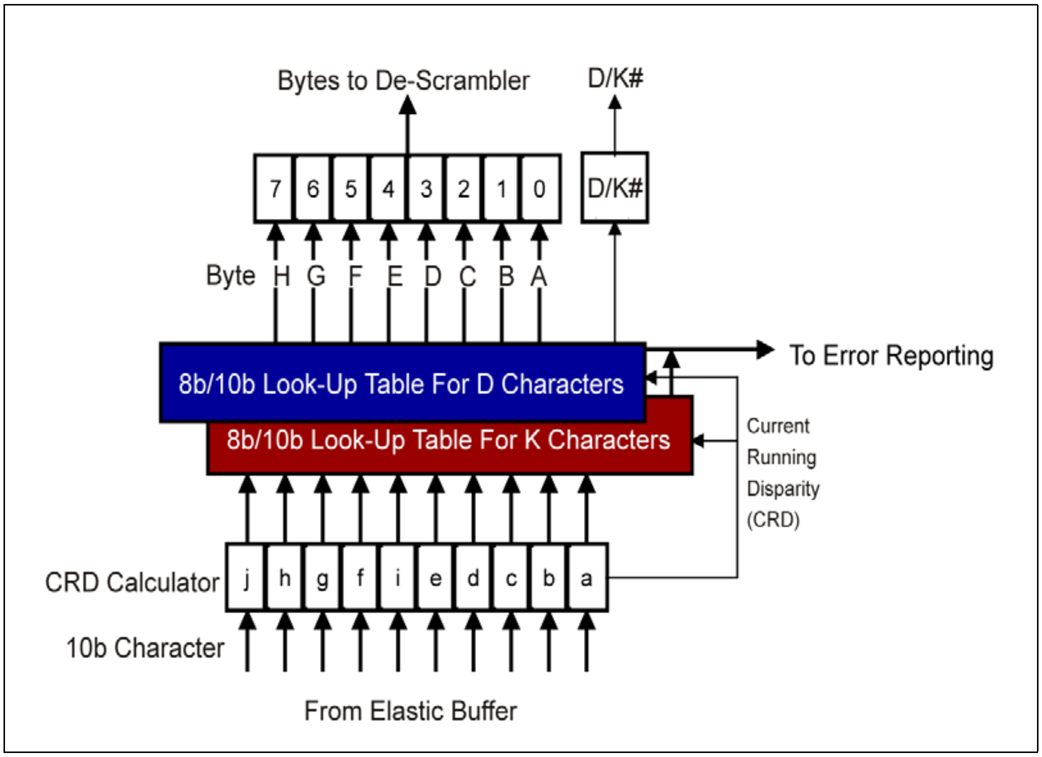
Figure 13: 8/10b Encoder


Bytes from Scrambler
D/K#
7
6
5
4
3
2
1
0
D/K#
Byte
H
G
F
E
D
C
B
A
D
K#
8b/10b Look-Up Table For D Characters
8b/10b Look-Up Table For K Characters
Current Running Disparity (CRD)
CRD Calculator
j
h
g
f
i
e
d
c
b
a
Parallel-to-Serial Converter
j
h
g
f
i
e
d
c
b
a
Serial Stream Clocked to Transmitter using Tx Clock

The 8b/10b Decoder (figure 14) uses two lookup tables (the D and K tables) to convert the 10-bit symbol stream back into bytes. Each symbol value is submitted to both lookup tables but only one of the tables will find a match for the symbol. The state of the D/K# signal indicates that the received symbol is a:

- Data (D) Symbol — a match for the received symbol is located in the D table. D/K# is driven High.
- Control (K) Symbol — a match for the received symbol is located in the K table. D/K# is driven Low.

Figure 14: 8/10b Decoder



SuperSpeed Power Management

SuperSpeed USB provides a much improved mechanism for entering and exiting low-power states. USB 2.0 implements a feature known as Suspend that forces devices to limit current to 2.5ma. Entry into the low power state requires a minimum of 3ms and exit requires more than 20ms. SuperSpeed power management provides finer granularity when entering low-power states and also reduces entry and exit times.

Link Power Management

Several architectural features aid in SS link power management:

- The much higher transmission rates mean that SS transactions complete very quickly, leaving links in the idle state for longer period of time.
- The Unicast approach involves only the links in the direct path between the originating root port and target device, leaving other links idle.
- The "poll once and notify" mechanism used in SS End-to-End flow control reduces overall link traffic.

The SS Link Power Management States are:

U0 — Fully Powered; link partners are fully powered and ready to send packets

U1 — Standby with Fast Recovery; link is in low power state and is not ready to send packets, but can transition back to U0 within microseconds.

U2 — Standby with Slow Recovery; link power saving greater than U1 and transition back to U0 within microseconds to milliseconds.

U3 — Suspend; greatest power savings and longest recovery back to U0 (milliseconds).

Transitions into these low-power states are set up by software. The downstream ports of the root hub and external hubs implement timers that software can set up to trigger entry into U1 and/or U2 based on link idle time. Entry into U3 is only possible under software control. Peripheral devices may also be enabled by software to intiate entry into U1 and/or U2.

Function Power Management

SuperSpeed power management also includes the ability to place a specific funition into a suspended state. This means that a multifunction device could have some functions suspended while others remain fully operational. Functions are placed into suspend under software control. The asynchronous Function Wake notification tells software that a suspended function or device is requesting a remote wakeup.