Chapter 8
Protocol Layer

This chapter presents a bottom-up view of the protocol starting with field and packet definitions. This is followed by a description of packet transaction formats for different transaction types. Link layer flow control and transaction level fault recovery are then covered. The chapter finishes with a discussion of retry synchronization, babble, and loss of bus activity recovery.

8.1 Bit Ordering

Bits are sent out onto the bus LSB first, followed by next LSB, through to MSB last. In the following diagrams, packets are displayed such that both individual bits and fields are represented (in a left to right reading order) as they would move across the bus.

8.2 SYNC Field

All packets begin with a synchronization (SYNC) field, which is a coded sequence that generates a maximum edge transition density. The SYNC field appears on the bus as IDLE followed by the binary string 'KJKJKJKK', in its NRZI encoding. It is used by the input circuitry to align incoming data with the local clock and is defined to be eight bits in length. SYNC serves only as a synchronization mechanism and is not shown in the following packet diagrams (refer to Section 7.1.7). The last two bits in the SYNC field are a marker that is used to identify the first bit of the PID. All subsequent bits in the packet must be indexed from this point.

8.3 Packet Field Formats

Field formats for the token, data, and handshake packets are described in the following section. Packet bit definitions are displayed in unencoded data format. The effects of NRZI coding and bit stuffing have been removed for the sake of clarity. All packets have distinct start and end of packet delimiters. The start of packet (SOP) is part of the SYNC field, and the end of packet (EOP) delimiter is described in Chapter 7.

8.3.1 Packet Identifier Field

A packet identifier (PID) immediately follows the SYNC field of every USB packet. A PID consists of a four bit packet type field followed by a four-bit check field as shown in Figure 8-1. The PID indicates the type of packet and, by inference, the format of the packet and the type of error detection applied to the packet. The four-bit check field of the PID insures reliable decoding of the PID so that the remainder of the packet is interpreted correctly. The PID check field is generated by performing a ones complement of the packet type field.

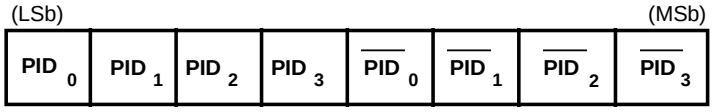

Figure 8-1. PID Format

The host and all functions must perform a complete decoding of all received PID fields. Any PID received with a failed check field or which decodes to a non-defined value is assumed to be corrupted and it, as well as the remainder of the packet, is ignored by the packet receiver. If a function receives an otherwise valid PID for a transaction type or direction that it does not support, the function must not respond. For example, an IN only endpoint must ignore an OUT token. PID types, codings, and descriptions are listed in Table 8-1.

Table 8-1. PID Types

PID Type	PID Name	PID[3:0]	Description
Token	OUT	b0001	Address + endpoint number in host -> function transaction
	IN	b1001	Address + endpoint number in function -> host transaction
	SOF	b0101	Start of frame marker and frame number
	SETUP	b1101	Address + endpoint number in host -> function transaction for setup to a control endpoint
Data	DATA0	b0011	Data packet PID even
	DATA1	b1011	Data packet PID odd
Handshake	ACK	b0010	Receiver accepts error free data packet
	NAK	b1010	Rx device cannot accept data or Tx device cannot send data
	STALL	b1110	Endpoint is stalled
Special	PRE	b1100	Host-issued preamble. Enables downstream bus traffic to LS devices.

PIDs are divided into four coding groups: token, data, handshake, or special, with the first two transmitted PID bits (PID<1:0>) indicating which group. This accounts for the distribution of PID codes.

8.3.2 Address Fields

Function endpoints are addressed using two fields: the function address field and the endpoint field. A function needs to fully decode both Address and Endpoint fields. Address or endpoint aliasing is not permitted, and a mismatch on either field must cause the token to be ignored. Accesses to non-initialized endpoints will also cause the token to be ignored.

8.3.2.1 Address Field

The function address (ADDR) field specifies the function, via its address, that is either the source or destination of a data packet, depending on the value of the token PID. As shown in Figure 8-2, a total of 128 addresses are specified as ADDR<6:0>. The ADDR field is specified for IN, SETUP, and OUT tokens. By definition, each ADDR value defines a single function. Upon reset and power-up, a function's address defaults to a value of 0 and must be programmed by the host during the enumeration process. The 0 default address is reserved for default and cannot be assigned for normal operation.

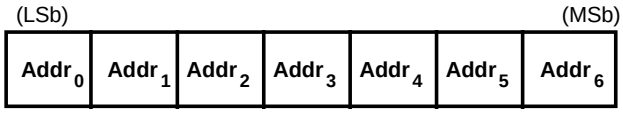

Figure 8-2. ADDR Field

8.3.2.2 Endpoint Field

An additional four-bit endpoint (ENDP) field, shown in Figure 8-3, permits more flexible addressing of functions in which more than one sub-channel is required. Endpoint numbers are function specific. The endpoint field is defined for IN, SETUP, and OUT token PIDs only. All functions must support one control endpoint at 0. Low speed devices support a maximum of two endpoint addresses per function: 0 plus one additional endpoint. Full speed functions may support up to the maximum of 16 endpoints.

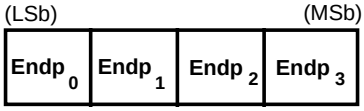

Figure 8-3. Endpoint Field

8.3.3 Frame Number Field

The frame number field is an 11-bit field that is incremented by the host on a per frame basis. The frame number field rolls over upon reaching its maximum value of x7FF, and is sent only for SOF tokens at the start of each frame.

8.3.4 Data Field

The data field may range from 0 to 1023 bytes and must be an integral numbers of bytes. Figure 8-4 shows the format for multiple bytes. Data bits within each byte are shifted out LSB first.

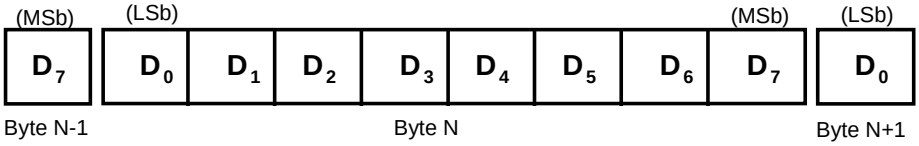

Figure 8-4. Data Field Format

Data packet size varies with the transfer type as described in Chapter 5.

8.3.5 Cyclic Redundancy Checks

Cyclic redundancy checks (CRCs) are used to protect the all non-PID fields in token and data packets. In this context, these fields are considered to be protected fields. The PID is not included in the CRC check of a packet containing a CRC. All CRCs are generated over their respective fields in the transmitter before bit stuffing is performed. Similarly, CRCs are decoded in the receiver after stuffed bits have been removed. Token and data packet CRCs provide 100% coverage for all single and double bit errors. A failed CRC is considered to indicate that one or more of the protected fields is corrupted and causes the receiver to ignore those fields, and, in most cases, the entire packet.

For CRC generation and checking, the shift registers in the generator and checker are seeded with an all ones pattern. For each data bit sent or received, the high order bit of the current remainder is XORed with the data bit and then the remainder is shifted left one bit and the low order bit set to ‘0’. If the result of that XOR is ‘1’, then the remainder is XORed with the generator polynomial.

When the last bit of the checked field is sent, the CRC in the generator is inverted and sent to the checker

MSB first. When the last bit of the CRC is received by the checker and no errors have occurred, the remainder will be equal to the polynomial residual.

Bit stuffing requirements must be met for the CRC, and this includes the need to insert a zero at the end of a CRC if the preceding six bits were all ones.

8.3.5.1 Token CRCs

A five-bit CRC field is provided for tokens and covers the ADDR and ENDP fields of IN, SETUP, and OUT tokens or the time stamp field of an SOF token. The generator polynomial is:

$$G(X) = X^5 + X^2 + 1$$

The binary bit pattern that represents this polynomial is '00101'. If all token bits are received without error, the five-bit residual at the receiver will be '01100'.

8.3.5.2 Data CRCs

The data CRC is a 16-bit polynomial applied over the data field of a data packet. The generating polynomial is:

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

The binary bit pattern that represents this polynomial is '1000000000000101'. If all data and CRC bits are received without error, the 16-bit residual will be '1000000000001101'.

8.4 Packet Formats

This section shows packet formats for token, data, and handshake packets. Fields within a packet are displayed in the order in which bits are shifted out onto the bus in the order shown in the figures.

8.4.1 Token Packets

Figure 8-5 shows the field formats for a token packet. A token consists of a PID, specifying either IN, OUT, or SETUP packet type, and ADDR and ENDP fields. For OUT and SETUP transactions, the address and endpoint fields uniquely identify the endpoint that will receive the subsequent data packet. For IN transactions, these fields uniquely identify which endpoint should transmit a data packet. Only the host can issue token packets. IN PIDs define a data transaction from a function to the host. OUT and SETUP PIDs define data transactions from the host to a function.

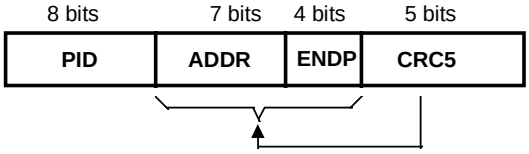

Figure 8-5. Token Format

Token packets have a five-bit CRC which covers the address and endpoint fields as shown above. The CRC does not cover the PID, which has its own check field. Token and SOF packets are delimited by an EOP after three bytes of packet field data. If a packet decodes as an otherwise valid token or SOF but does not terminate with an EOP after three bytes, it must be considered invalid and ignored by the receiver.

8.4.2 Start of Frame Packets

Start of Frame (SOF) packets are issued by the host at a nominal rate of once every 1.00 ms ±0.05. SOF packets consist of a PID indicating packet type followed by an 11-bit frame number field as illustrated in Figure 8-6.

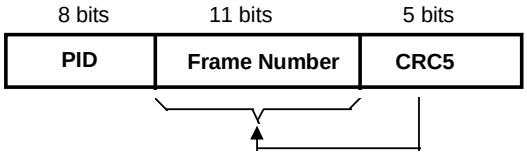

Figure 8-6. SOF Packet

The SOF token comprises the token-only transaction that distributes a start of frame marker and accompanying frame number at precisely timed intervals corresponding to the start of each frame. All full speed functions, including hubs, must receive and decode the SOF packet. The SOF token does not cause any receiving function to generate a return packet; therefore, SOF delivery to any given function cannot be guaranteed. The SOF packet delivers two pieces of timing information. A function is informed that a start of frame has occurred when it detects the SOF PID. Frame timing sensitive functions, which do not need to keep track of frame number, need only decode the SOF PID; they can ignore the frame number and its CRC. If a function needs to track frame number, then it must comprehend both the PID and the time stamp.

8.4.3 Data Packets

A data packet consists of a PID, a data field, and a CRC as shown in Figure 8-7. There are two types of data packets, identified by differing PIDs: DATA0 and DATA1. Two data packet PIDs are defined to support data toggle synchronization (refer to Section 8.6).

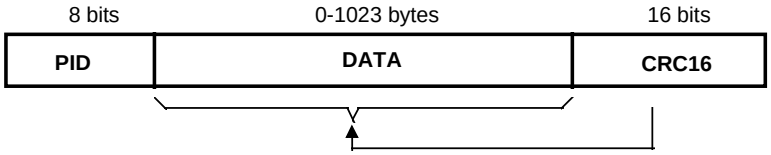

Figure 8-7. Data Packet Format

Data must always be sent in integral numbers of bytes. The data CRC is computed over only the data field in the packet and does not include the PID, which has its own check field.

8.4.4 Handshake Packets

Handshake packets, as shown in Figure 8-8, consist of only a PID. Handshake packets are used to report the status of a data transaction and can return values indicating successful reception of data, flow control, and stall conditions. Only transaction types that support flow control can return handshakes. Handshakes are always returned in the handshake phase of a transaction and may be returned, instead of data, in the data phase. Handshake packets are delimited by an EOP after one byte of packet field. If a packet decodes as an otherwise valid handshake but does not terminate with an EOP after one byte, it must be considered invalid and ignored by the receiver.

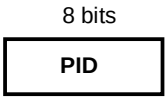

Figure 8-8. Handshake Packet

There are three types of handshake packets:

- **ACK** indicates that the data packet was received without bit stuff or CRC errors over the data field and that the data PID was received correctly. ACK may be issued either when sequence bits match and the receiver can accept data or when sequence bits mismatch and the sender and receiver must resynchronize to each other (refer to Section 8.6 for details). An ACK handshake is applicable only in transactions which data has been transmitted and where a handshake is expected. ACK can be returned by the host for IN transactions and by a function for OUT transactions.
- **NAK** indicates that a function was unable to accept data from the host (OUT) or that a function has no data to transmit to the host (IN). NAK can only be returned by functions in the data phase of IN transactions or the handshake phase of OUT transactions, and the host can never issue a NAK. NAK is used for flow control purposes to indicate that a function is temporarily unable to transmit or receive data, but will eventually be able to do so without need of host intervention. NAK is also used by interrupt endpoints to indicate that no interrupt is pending.
- **STALL** is returned by a function in response to an IN token or after the data phase of an OUT (see Figure 8-9 and Figure 8-13). STALL indicates that a function is unable to transmit or receive data, and that the condition requires host intervention to remove the stall. Once a function's endpoint is stalled, the function must continue returning STALL until the condition causing the stall has been cleared through host intervention. The host is not permitted to return a STALL under any condition.

8.4.5 Handshake Responses

Transmitting and receiving functions must return handshakes based upon an order of precedence detailed in Table 8-2 through Table 8-4. Not all handshakes are allowed, depending on the transaction type and whether the handshake is being issued by a function or the host.

8.4.5.1 Function Response to IN Transactions

Table 8-2 shows the possible responses a function may make in response to an IN token. If the function is unable to send data, due to a stall or a flow control condition, it issues a STALL or NAK handshake, respectively. If the function is able to issue data, it does so. If the received token is corrupted, the function returns no response.

Table 8-2. Function Responses to IN Transactions

Token Received Corrupted	Function Tx Endpoint Stalled	Function Can Transmit Data	Action Taken
Yes	Don't care	Don't care	Return no response
No	Yes	Don't care	Issue STALL handshake
No	No	No	Issue NAK handshake
No	No	Yes	Issue data packet

8.4.5.2 Host Response to IN Transactions

Table 8-3 shows the host response to an IN transaction. The host is able to return only one type of handshake, an ACK. If the host receives a corrupted data packet, it discards the data and issues no response. If the host cannot accept data from a function, (due to problems such as internal buffer overrun) this condition is considered to be an error and the host returns no response. If the host is able to accept data and the data packet is received error free, the host accepts the data and issues an ACK handshake.

Table 8-3. Host Responses to IN Transactions

Data Packet Corrupted	Host Can Accept Data	Handshake Returned by Host
Yes	N/A	Discard data, return no response
No	No	Discard data, return no response
No	Yes	Accept data, issue ACK

8.4.5.3 Function Response to an OUT Transaction

Handshake responses for an OUT transaction are shown in Table 8-4. A function, upon receiving a data packet, may return any one of the three handshake types. If the data packet was corrupted, the function returns no handshake. If the data packet was received error free and the function's receiving endpoint is stalled, the function returns a STALL handshake. If the transaction is maintaining sequence bit synchronization and a mismatch is detected (refer to Section 8.6 for details), then the function returns ACK and discards the data. If the function can accept the data and has received the data error free, it returns an ACK handshake. If the function cannot accept the data packet due to flow control reasons, it returns a NAK.

Table 8-4. Function Responses to OUT Transactions in Order of Precedence

Data Packet Corrupted	Receiver Stalled	Sequence Bits Mismatch	Function Can Accept Data	Handshake Returned by Function
Yes	N/A	N/A	N/A	None
No	Yes	N/A	N/A	STALL
No	No	Yes	N/A	ACK
No	No	No	Yes	ACK
No	No	No	No	NAK

8.4.5.4 Function Response to a SETUP Transaction

Setup defines a special type of host to function data transaction which permits the host to initialize an endpoint's synchronization bits to those of the host. Upon receiving a Setup transaction, a function must accept the data. Setup transactions cannot be stalled or NAKed and the receiving function must accept the Setup transfer's data. If a non-control endpoint receives a SETUP PID, it must ignore the transaction and return no response.

8.5 Transaction Formats

Packet transaction format varies depending on the endpoint type. There are four endpoint types: bulk, control, interrupt, and isochronous.

8.5.1 Bulk Transactions

Bulk transaction types are characterized by the ability to guarantee error free delivery of data between the host and a function by means of error detection and retry. Bulk transactions use a three phase transaction consisting of token, data, and handshake packets as shown in Figure 8-9. Under certain flow control and stall conditions, the data phase may be replaced with a handshake resulting in a two phase transaction in which no data is transmitted.

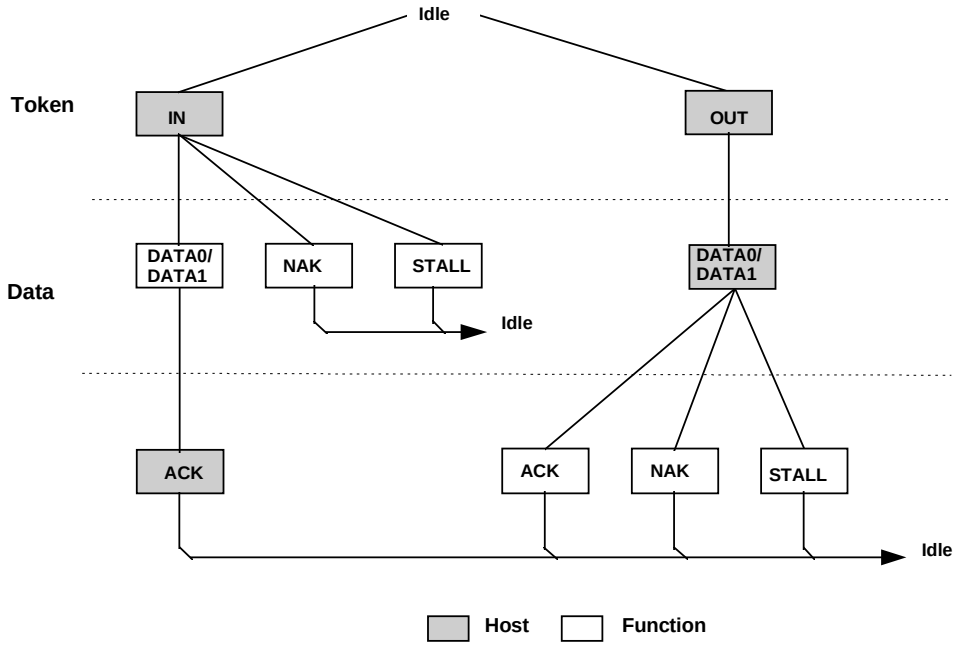

Figure 8-9. Bulk Transaction Format

When the host wishes to receive bulk data, it issues an IN token. The function endpoint responds by returning either a DATA packet or, should it be unable to return data, a NAK or STALL handshake. A NAK indicates that the function is temporarily unable to return data, while a STALL indicates that the endpoint is permanently stalled and requires host software intervention. If the host receives a valid data packet, it responds with an ACK handshake. If the host detects an error while receiving data, it returns no handshake packet to the function.

When the host wishes to transmit bulk data, it first issues an OUT token packet followed by a data packet. The function then returns one of three handshakes. ACK indicates that the data packet was received without errors and informs the host that that it may send the next packet in the sequence. NAK indicates that the data was received without error but that the host should resend the data because the function was in a temporary condition preventing it from accepting the data at this time (e.g., buffer full). If the endpoint was stalled, STALL is returned to indicate that the host should not retry the transmission because there is an error condition on the function. If the data packet was received with a CRC or bit stuff error, no handshake is returned.

Figure 8-10 shows the sequence bit and data PID usage for bulk reads and writes. Data packet synchronization is achieved via use of the data sequence toggle bits and the DATA0/DATA1 PIDs. Bulk endpoints must have their toggle sequence bits initialized via a separate control endpoint.

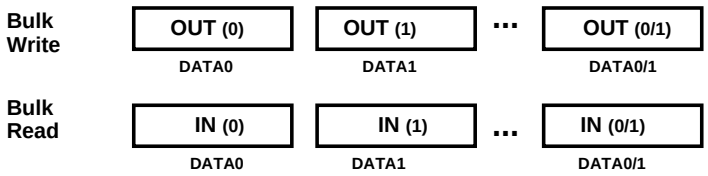

Figure 8-10. Bulk Reads and Writes

The host always initializes the first transaction of a bus transfer to the DATA0 PID. The second transaction uses a DATA1 PID, and successive data transfers alternate for the remainder of the bulk transfer. The data packet transmitter toggles upon receipt of ACK, and the receiver toggles upon receipt and acceptance of a valid data packet (refer to Section 8.6).

8.5.2 Control Transfers

Control transfers minimally have two transaction stages: Setup and Status. A control transfer may optionally contain a data stage between the setup and status stages. During the Setup stage, a Setup transaction is used to transmit information to the control endpoint of a function. Setup transactions are similar in format to an OUT, but use a SETUP rather than on OUT PID. Figure 8-11 shows the Setup transaction format. A Setup always uses a DATA0 PID for the data field of the Setup transaction. The function receiving a Setup must accept the Setup data and respond with an ACK handshake or, if the data is corrupted, discard the data and return no handshake.

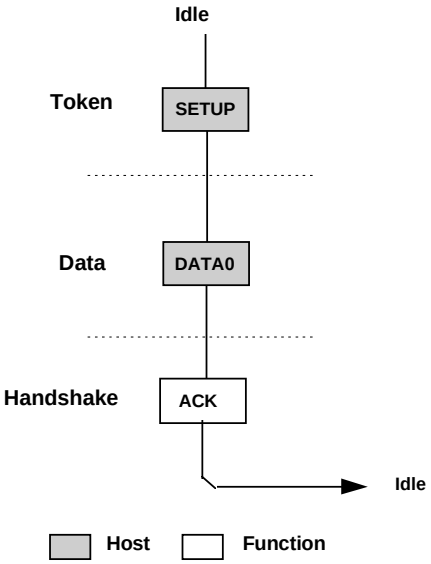

Figure 8-11. Control Setup Transaction

The data stage, if present, of a control transfer consists of one or more IN or OUT transactions and follows the same protocol rules as bulk transfers. All the transactions in the data stage must be in the same direction, i.e., all INs or all OUTs. The amount of data to be sent during the data phase and its direction are specified during the Setup stage. If the amount of data exceeds the prenegotiated data packet size, the data is sent in multiple transactions (INs or OUTs) which carry the maximum packet size. Any remaining data is sent as a residual in the last transaction.

The status stage of a control transfer is the last operation in the sequence. A status stage is delineated by a change in direction of data flow from the previous stage and always uses a DATA1 PID. If, for example, the data stage consists of OUTs, the status is a single IN transaction. If the control sequence has no data stage, then it consists of a Setup stage followed by a Status stage consisting of an IN transaction. Figure 8-

12 shows the transaction order, the data sequence bit value and the data PID types for control read and write sequences. The sequence bits are displayed in parentheses.

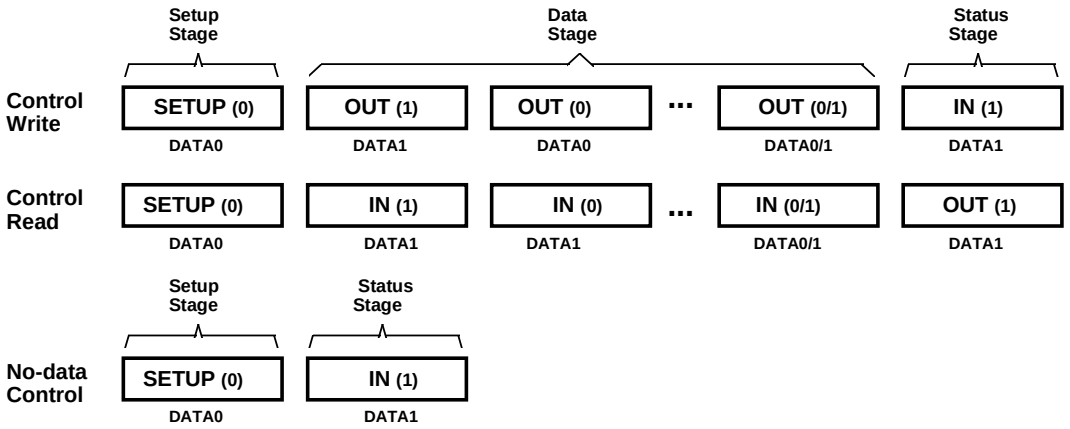

Figure 8-12. Control Read and Write Sequences

8.5.2.1 Reporting Status Results

The status stage reports to the host the outcome of the previous setup and data stages of the transfer. Three possible results may be returned:

- The command sequence completed successfully
- The command sequence failed to complete
- The function is still busy completing command

Status reporting is always in the function to host direction. The following table summarizes the type of responses required for each. Control write transfers return status information on the data phase of the transfer. Control read transfers return status information on the handshake phase after the host has issued a zero length data packet during the previous data phase.

Table 8-5. Status Phase Responses

Status Response	Control Write Transfer (sent during data phase)	Control Read Transfer (send during handshake phase)
Function completes	0 length data packet	ACK handshake
Function has an error	STALL handshake	STALL handshake
Function is busy	NAK handshake	NAK handshake

For control reads, the host sends a zero length data packet to the control endpoint. The endpoint's handshake response indicates the completion status. NAK indicates that the function is still processing the command and that the host should continue the status phase. ACK indicates that the function has completed the command and is ready to accept a new command and STALL indicates that the function has an error that prevents it from completing the command.

For control writes, the function responds with either a handshake or a zero length data packet to indicate its status. A NAK indicates that the function is still processing the command and that the host should continue the status phase, return of a zero length packet indicates normal completion of the command, and STALL

indicates that the function has an error that prevents it from completing the command. Control write transfers which return a zero length data packet during the data phase always cause the host to return an ACK handshake to the function.

If, during a data or status stage, a command endpoint is sent more data or is requested to return more data than was indicated in the setup stage, it should return a STALL. If a control endpoint returns STALL during the data stage, there will be no status stage for that control transfer.

8.5.2.2 Error Handling on the Last Data Transaction

If the ACK handshake on an IN transaction gets corrupted, the function and the host will temporarily disagree on whether the transaction was successful. If the transaction is followed by another IN, the toggle retry mechanism will detect the mismatch and recover from the error. If the ACK was on the last IN of a control transfer, then the toggle retry mechanism cannot be used and an alternative scheme must be used.

The host which successfully received the data of the last IN, issues an OUT setup transfer, and the function, upon seeing that the token direction has toggled, interprets this action as proof that the host successfully received the data. In other words, the function interprets the toggling of the token direction as implicit proof of the host's successful receipt of the last ACK handshake. Therefore, when the function sees the OUT setup transaction, it advances to the status phase.

Control writes do not have this ambiguity. The host, by virtue of receiving the handshake, knows for sure if the last transaction was successful. If an ACK handshake on an OUT gets corrupted, the host does not advance to the status phase and retries the last data instead. A detailed analysis of retry policy appears in Section 8.6.4.

8.5.3 Interrupt Transactions

Interrupt transactions consist solely of IN. Upon receipt of an IN token, a function may return data, NAK, or STALL. If the endpoint has no new interrupt information to return, i.e., no interrupt is pending, the function returns a NAK handshake during the data phase. A stalled interrupt endpoint causes the function to return a STALL handshake if it is permanently stalled and requires software intervention by the host. If an interrupt is pending, the function returns the interrupt information as a data packet. The host, in response to receipt of the data packet, issues either an ACK handshake if data was received error free or returns no handshake if the data packet was received corrupted.

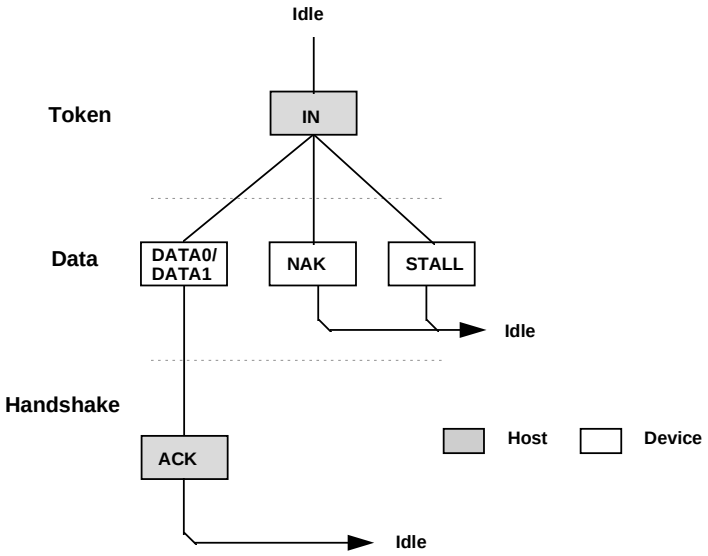

Figure 8-13. Interrupt Transaction Format

When an endpoint is using the Interrupt transfer mechanism for actual interrupt data, the data toggle protocol must be followed. This allows the function to know that the data has been received by the host and the event condition may be cleared. This 'guaranteed' delivery of events allows the function to only send the interrupt information until it has been received by the host rather than having to send the interrupt data every time the function is polled and until host software clears the interrupt condition. When used in the toggle mode, an interrupt endpoint is initialized to the DATA0 PID and behaves the same as the bulk IN transaction shown in Figure 8-10.

An Interrupt endpoint may also be used to communicate rate feedback information for certain types of isochronous functions. When used in this mode, the data toggle bits should be changed after each data packet is sent to the host without regard to the presence or type of handshake packet.

8.5.4 Isochronous Transactions

ISO transactions have a token and data phase, but no handshake phase, as shown in Figure 8-14. The host issues either an IN or an OUT token followed by the data phase in which the endpoint (for INs) or the host (for OUTs) transmits data. ISO transactions do not support a handshake phase or retry capability.

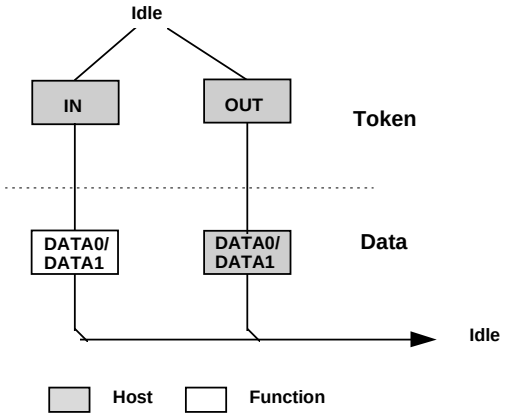

Figure 8-14. Isochronous Transaction Format

ISO transactions do not support toggle sequencing, and the data PID is always DATA0. The packet receiver does not examine the data PID.

8.6 Data Toggle Synchronization and Retry

USB provides a mechanism to guarantee data sequence synchronization between data transmitter and receiver across multiple transactions. This mechanism provides a means of guaranteeing that the handshake phase of a transaction was interpreted correctly by both the transmitter and receiver. Synchronization is achieved via use of the DATA0 and DATA1 PIDs and separate data toggle sequence bits for the data transmitter and receiver. Receiver sequence bits toggle only when the receiver is able to accept data and receives an error free data packet with the correct data PID. Transmitter sequence bits toggle only when the data transmitter receives a valid ACK handshake. The data transmitter and receiver must have their sequence bits synchronized at the start of a transaction, and the mechanism for doing this varies with the transaction type. Data toggle synchronization is not supported for ISO transfers.

8.6.1 Initialization via SETUP Token

Control transfers use the SETUP token for initializing host and function sequence bits. Figure 8-15 shows the host issuing a SETUP packet to a function followed by an OUT. The numbers in the circles represent the transmitter and receiver sequence bits. The function must accept the data and ACK the transaction. When the function accepts the transaction, it must reset its sequence bit so that both the host's and function's sequence bits are equal to '1' at the end of the SETUP transaction.

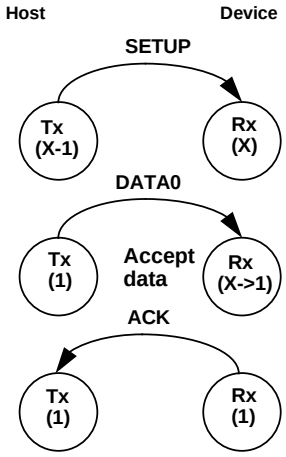

Figure 8-15. SETUP Initialization

8.6.2 Successful Data Transactions

Figure 8-16 shows the case where two successful transactions have occurred. For the data transmitter, this means that it toggles its sequence bit upon receipt of an ACK. The receiver toggles its sequence bit only if it receives a valid data packet and the packet's data PID matches the receiver's sequence bit.

During each transaction, the receiver compares the transmitter sequence bit (encoded in the data packet PID as either DATA0 or DATA1) with its receiver sequence bit. If data cannot be accepted, then the receiver must issue a NAK. If data can be accepted and the receiver's sequence bit matches the PID sequence bit, then data is accepted. Sequence bits may only change if a data packet is transmitted. Two-phase transactions in which there is no data packet leave the transmitter and receiver sequence bits unchanged.

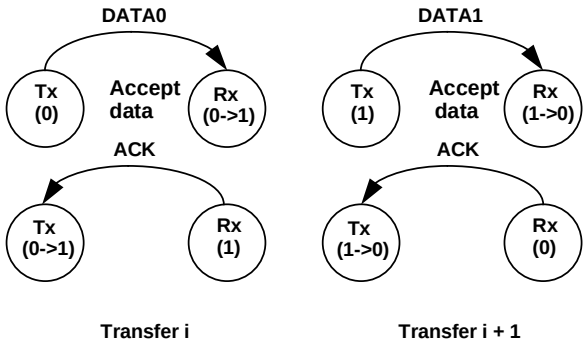

Figure 8-16. Consecutive Transactions

8.6.3 Data Corrupted or Not Accepted

If data cannot be accepted or the received data packet is corrupted, the receiver will issue a NAK or STALL handshake, or will time out, depending on the circumstances, and the receiver will not toggle its sequence bit. Figure 8-17 shows the case where a transaction is NAKed and then retried. Any non-ACK handshake or time out will generate similar retry behavior. The transmitter, having not received an ACK handshake, will not toggle its sequence bit. As a result, a failed data packet transaction leaves the transmitter's and receiver's sequence bits synchronized and untoggled. The transaction will then be retried and, if successful, will cause both transmitter and receiver sequence bits to toggle.

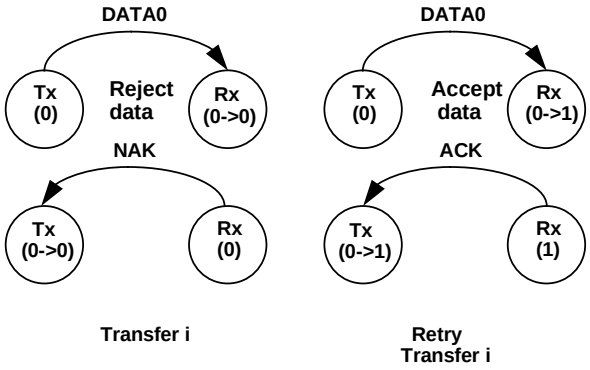

Figure 8-17. NAKed Transaction with Retry

8.6.4 Corrupted ACK Handshake

The transmitter is the last and only agent to know for sure whether a transaction has been successful, due to its receiving an ACK handshake. A lost or corrupted ACK handshake can lead to a temporary loss of synchronization between transmitter and receiver as shown in Figure 8-18. Here the transmitter issues a valid data packet, which is successfully acquired by the receiver; however, the ACK handshake is corrupted.

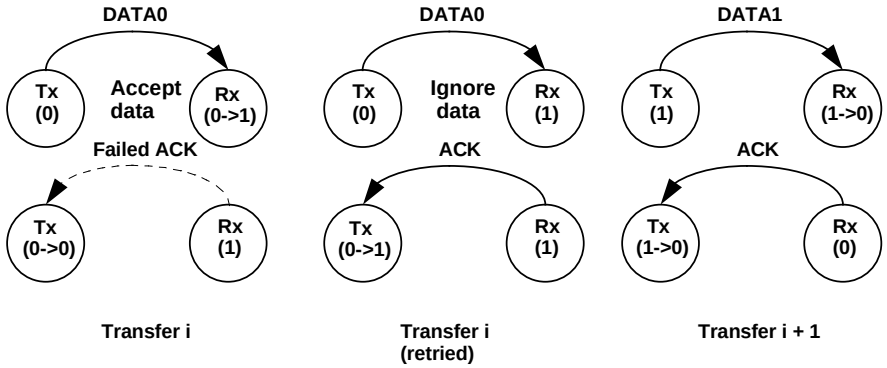

Figure 8-18. Corrupted ACK Handshake with Retry

At the end of transaction <i>, there is a temporary loss of coherency between transmitter and receiver, as evidenced by the mismatch between their respective sequence bits. The receiver has received good data, but the transmitter does not know whether it has successfully sent data. On the next transaction, the transmitter will resend the previous data using the previous DATA0 PID. The receiver's sequence bit and the data PID will not match, so the receiver knows that it has previously accepted this data. Consequently, it discards the incoming data packet and does not toggle its sequence bit. The receiver then issues an ACK, which causes the transmitter to regard the retried transaction as successful. Receipt of ACK causes the transmitter to toggle its sequence bit. At the beginning of transaction <i+1>, the sequence bits have toggled and are again synchronized.

The data transmitter must guarantee that any retried data packet be identical in length to that sent in the original transaction. If the data transmitter is unable, because of problems such as a buffer underrun condition, to transmit the identical amount of data as was in the original data packet, it must abort the transaction by generating a bit stuffing violation. This causes a detectable error at the receiver and guarantees that a partial packet will not be interpreted as a good packet. The transmitter should not try to force an error at the receiver by sending a known bad CRC. A combination of a bad packet with a "bad" CRC may be interpreted by the receiver as a good packet.

8.6.5 Low Speed Transactions

USB supports signaling at two speeds: full speed (FS) signaling at 12.0 Mbs and low speed (LS) signaling at 1.5 Mbs. Hubs disable downstream bus traffic to all ports to which LS devices are attached during full speed downstream signaling. This is required both for EMI reasons and to prevent any possibility that an LS device might misinterpret downstream a FS packet as being addressed to it. Figure 8-19 shows an IN LS transaction in which the host issues a token and handshake and receives a data packet.

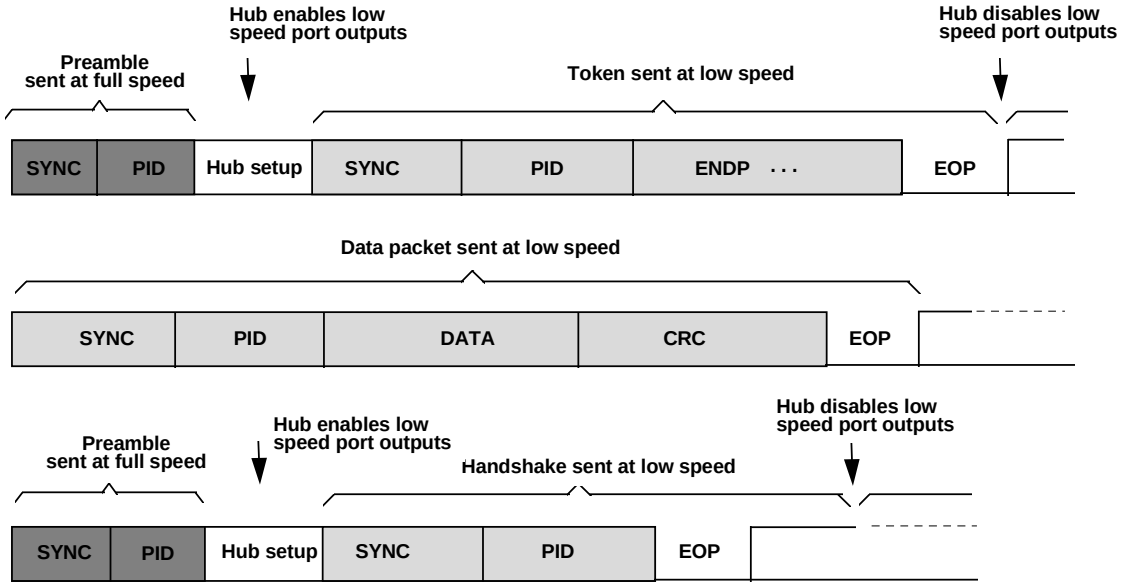

Figure 8-19. Low Speed Transaction

All downstream packets transmitted to LS devices require a preamble. The preamble consists of a SYNC followed by a PID, both sent at full speed. Hubs must comprehend the PRE PID; all other USB devices must ignore it and treat it as undefined. After the end of the preamble PID, the host must wait at least 4 full speed bit times during which hubs must complete the process of configuring their repeater sections to accept LS signaling. During this hub setup interval, hubs must drive their FS and LS ports to their respective idle states. Hubs must be ready to accept low speed signaling from the host before the end of the hub setup interval. Low speed connectivity rules are summarized below:

1. Low speed devices are identified during the connection process and the hub ports to which they are connected are identified as low speed.
2. All downstream low speed packets must be prefaced with a preamble (sent at full speed) which turns on the output buffers on low speed hub ports.
3. Low speed hub port output buffers are turned off upon receipt of EOP and are not turned on again until a preamble PID is detected.
4. Upstream connectivity is not affected by whether a hub port is full or low speed.

The start of LS signaling commences with the host issuing SYNC at low speed, followed by the remainder of the packet. The end of packet is identified by End of Packet (EOP), at which time all hubs tear down connectivity and disable any ports to which LS devices are connected. Hubs do not switch ports for upstream signaling; LS ports remain enabled in the upstream direction for both LS and FS signaling.

LS and FS transactions maintain a high degree of protocol commonality. However, LS signaling does have certain limitations which include:

- Data payload limited to eight bytes, maximum
- LS only supports Interrupt and Control types of transfers
- The SOF packet is not received by LS devices

8.7 Error Detection and Recovery

USB is designed to permit reliable end to end communication in the presence of errors on the physical signaling layer. This includes the ability to reliably detect the vast majority of possible errors and to recover from errors on a transaction type basis. Control transactions, for example, require a high degree of data reliability; they support end to end data integrity using error detection and retry. ISO transactions, by virtue of their bandwidth and latency requirements, do not permit retries and must tolerate a higher incidence of uncorrected errors.

8.7.1 Packet Error Categories

USB employs three error detection mechanisms: bit stuff violations, PID check bits, and CRCs. A bit stuff violation exists if a packet receiver detects seven or more consecutive bit times without a differential (J -> K or K -> J) transition, as detected on the physical D+ and D- lines, between the start and end of a packet. A PID error exists if the four PID check bits are not complements of their respective packet identifier bits. A CRC error exists if the computed checksum remainder at the end of a packet reception is not zero.

With the exception of the SOF token, any packet that is received corrupted causes the receiver to ignore it and discard any data or other field information that came with the packet. Table 8-6 lists error detection mechanisms, the types of packets to which they apply, and the appropriate packet receiver response.

Table 8-6. Packet Error Types

Field	Error	Action
PID	PID Check, Bit Stuff	Ignore packet
Address	Bit Stuff, Address CRC	Ignore token
Frame Number	Bit Stuff, Frame Number CRC	Ignore Frame Number Field
Data	Bit stuff, Data CRC	Discard data

8.7.2 Bus Turnaround Timing

The host and USB function need to keep track of how much time has elapsed from when the transmitter completes sending a packet until it begins to receive a packet back. This time is referred to as the bus turnaround time and is tracked by the packet transmitter's bus turnaround timer. The timer starts counting on the SE0 to IDLE transition of the EOP strobe and stops counting when the IDLE to 'K' SOP transition is detected. Both devices and the host require turnaround timers. The device bus turnaround time is defined by the worst case round trip delay plus the maximum device response delay (refer to Section 7.1.14). USB devices cannot time out earlier than 16 bit times after the end of the previous EOP and they must time out by 18 bit times. If the host wishes to indicate an error condition via a timeout, it must wait at least 18 bit times before issuing the next token to insure that all downstream devices have timed out.

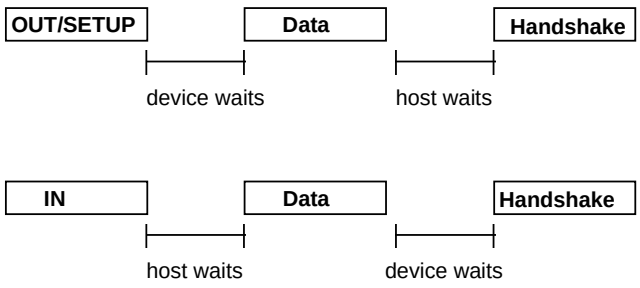

Figure 8-20. Bus Turnaround Timer Usage

As shown above, the device uses its bus turnaround timer between token and data or data and handshake phases. The host uses its timer between data and handshake or token and data phases.

If the host receives a corrupted data packet, it must wait before sending out the next token. This wait interval guarantees that the host does not attempt to issue a token immediately after a false EOP.

8.7.3 False EOPs

False EOPs must be handled in a manner which guarantees that the packet currently in progress completes before the host or any other device attempts to transmit a new packet. If such an event were to occur, it would constitute a bus collision and have the ability to corrupt up to two consecutive transactions. Detection of false EOP relies upon the fact that a packet into which a false EOP has been inserted will appear as a truncated packet with a CRC failure. (The last 16 bits of the packet will have a very low probability of appearing to be a correct CRC.)

The host and devices handle false EOP situations differently. When a device sees a corrupted data packet, it issues no response and waits for the host to send the next token. This scheme guarantees that the device will not attempt to return a handshake while the host may still be transmitting a data packet. If a false EOP has occurred, the host data packet will eventually end, and the device will be able to detect the next token. If a device issues a data packet that gets corrupted with a false EOP the host will ignore the packet and not issue the handshake. The device, expecting to see a handshake from the host, will time out.

If the host receives a corrupted data packet, it assumes that a false EOP may have occurred and waits for 16 bit times to see if there is any subsequent upstream traffic. If no bus transitions are detected within the 16 bit interval and the bus remains in the IDLE state, the host may issue the next token. Otherwise, the host waits for the device to finish sending the remainder of its packet. The 16 bit times guarantees two conditions. The first condition is to must make sure that the device has finished sending its packet. This is guaranteed by a time-out interval (with no bus transitions) greater than the worst case 6-bit time bit stuff interval. The second condition is that the transmitting device’s bus turnaround timer must be guaranteed to expire. Note that the time-out interval is transaction speed sensitive. For full speed transactions, the host must wait 16 FS bit times; for LS transactions, it must wait 16 LS bit times.

If the host receives a data packet with a valid CRC, it assumes that the packet is complete and need not delay in issuing the next token.

8.7.4 Babble and Loss of Activity Recovery

USB must be able to detect and recover from conditions which leave it waiting indefinitely for an end of packet or which leave the bus in something other than the idle state at the end of a frame. Loss of activity (LOA) is characterized by start of packet (SOP) followed by lack of bus activity (bus remains in 'J' or 'K' state) and no end of packet (EOP) at the end of a frame. Babble is characterized by an SOP followed by the presence of bus activity past the end of a frame. LOA and babble have the potential to either deadlock the bus or force out the beginning of the next frame. Neither condition is acceptable, and both must be prevented from occurring. As the USB component responsible for controlling connectivity, hubs are responsible for babble/LOA detection and recovery. All USB devices that fail to complete their transmission at the end of a frame are prevented from transmitting past a frame's end by having the nearest hub disable the port to which the offending device is attached. Details of the hub babble/LOA recovery mechanism appear in Section 11.4.3.