CHAPTER 7
Packet Telemetry Downlink

List of Figures

Acronyms

EP	encapsulation packet
IP	Internet Protocol
IPv4	Internet Protocol, Version 4
IPv6	Internet Protocol, Version 6
LLEP	low-latency encapsulation packet
MAC	media access control
msb	most significant bit
PCM	pulse code modulation
SP	source packet
TmNS	Telemetry Network Standard
TP	transport packet

This page intentionally left blank.

CHAPTER 7

Packet Telemetry Downlink

This standard defines the method for transporting variable-length, well-defined data formats in a Chapter 4 pulse code modulation (PCM) stream.

7.1 Packet Telemetry

Packet telemetry defines the method for asynchronously inserting data from one or more data streams into PCM minor frames. This standard defines various source packet (SP) types that are used to encapsulate well-defined data formats, including:

- Chapter 11 packets;
- Chapter 24 TmNSMessages;
- Ethernet frames.

An SP is encapsulated into an integral number of encapsulation packets (EPs); nominally each EP contains only one SP. Different SP types may be multiplexed simultaneously into a single, logical stream of EPs. The encapsulation packet stream is then segmented into fixed-length transport packets (TPs), each of which is inserted into a single PCM minor frame. While a TP may be segmented and interspersed with PCM data within a PCM minor frame, each PCM minor frame shall contain only one TP. A low-latency encapsulation packet (LLEP) mechanism allows for the insertion of one or more EPs with low-latency requirements into the transmission of a long EP. Specific structure-critical elements are protected using a Golay code; see Appendix 7-A for details. Figure 7-1 provides an overview of the entire packet telemetry encapsulation process.

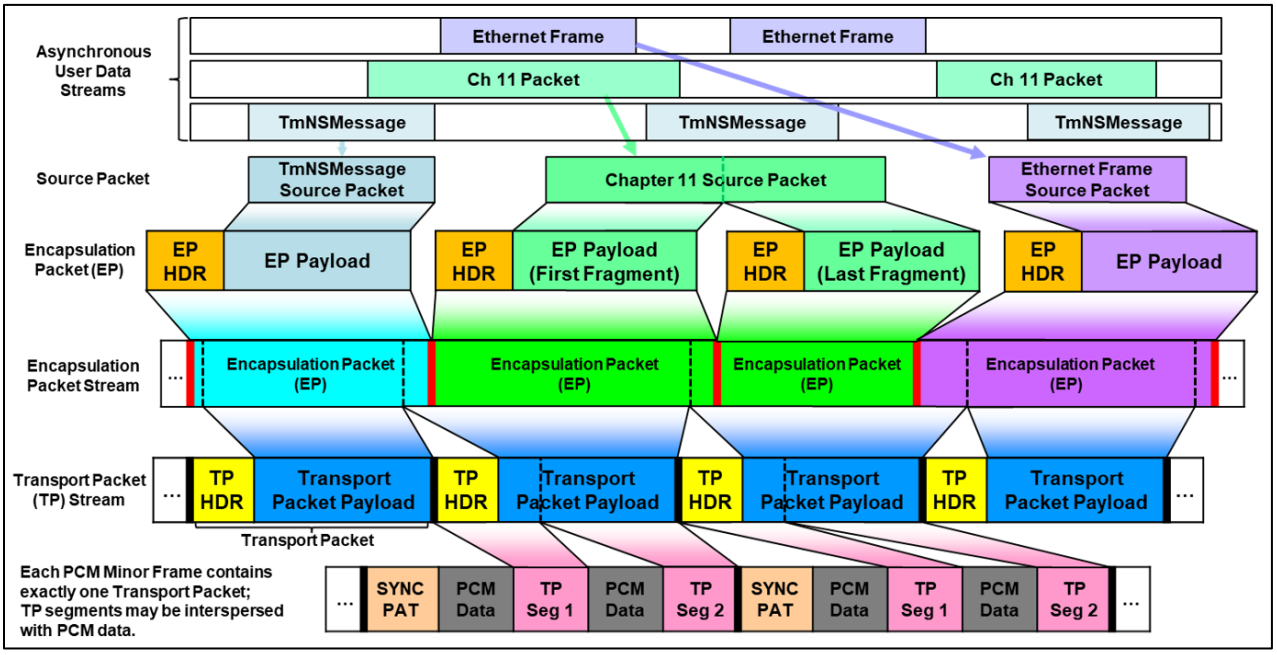


Figure 7-1. Packet Telemetry Overview

NOTE	An SP may be fragmented into multiple EPs where each EP's payload contains a fragment of the SP. Subsection 7.2.3 describes SP fragmentation. A TP may be segmented for insertion into a PCM minor frame. A single PCM minor frame may contain multiple TP segments. See Section 7.5 for more details.

NOTE	The IRIG 106-15 restricted a PCM minor frame to a single, contiguous, unsegmented TP. The current standard supports a single, contiguous or segmented TP per PCM minor frame.

NOTE	Chapter 9 supports defining TP segments contained within a PCM minor frame.

7.2 Encapsulation Packet Structure

An EP shall include a header and a payload as shown in Figure 7-2.

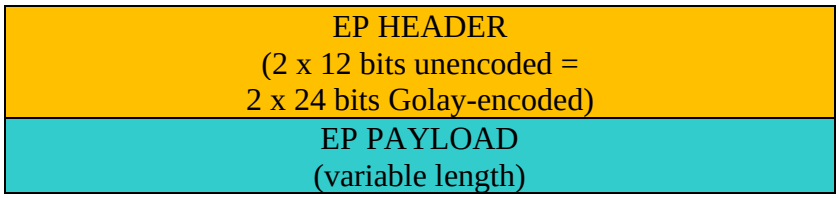


Figure 7-2. Encapsulation Packet Structure

7.2.1 Encapsulation Packet Header

The EP header shall consist of two 12-bit words and shall be encoded as two 24-bit Golay code words prior to insertion into the EP, encoded most significant bit (msb) first and in the word order as shown in Figure 7-3.

	11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Word 0	CRC	Res	Content				Fragment		Length (15 – 12)			
Word 1	Length (11 – 0)											

Figure 7-3. EP Header Protected Fields (Unencoded)

The EP header consists of the following fields.

- CRC (bit 23). A CRC trailer is present on this packet. A 16-bit CRC ins included at the end of the packet and is included in the packet length. The CRC shall be calculated as a CRC-16-ANSI CRC as specified in Chapter 4 Subsection 4.3.3.
- Res (bit 22). Reserved. This bit shall be set to zero (e.g., 2'b0) on transmission; ignored on reception.
- Content (bits 21 – 18). The ST packet type field shall specify the type of SP contained in the EP payload (see Subsection 7.2.2 for details).
    - 4'b0000: Fill SP

- 4'b0001: Application-Specific SP
- 4'b0010: Test Counter SP
- 4'b0011: Chapter 11 SP
- 4'b0100: Raw Ethernet Media Access Control (MAC) Frame SP
- 4'b0101: Internet Protocol (IP) SP
- 4'b0110: Chapter 24 TmNSMessage SP
- 4'b0111 – 4'b1111: Reserved

- Fragment (bits 17 – 16). The SP fragmentation flags shall identify whether the EP payload contains a complete SP or a fragment.
  - 2'b00: Complete SP
  - 2'b01: First Fragment of an SP
  - 2'b10: Middle Fragment of an SP
  - 2'b11: Last Fragment of an SP
- Length (bits 15 – 0). The EP length field shall provide the length (in bytes) of the EP payload (note the EP header length is not included in the EP length). If an SP exceeds the maximum EP length, the SP must be fragmented using multiple EPs as specified in Subsection 7.2.3.

7.2.2 Encapsulation Packet Payload

The EP payload shall contain either a complete SP or an SP fragment; the Content field identifies the SP type. The following subsections contain a detailed description for each SP type.

7.2.2.1 Fill Source Packet

If no SPs are available for embedding into an EP stream, fill SPs shall be generated and inserted into the EP stream to assure a constant data flow to the PCM stream. Each fill SP byte shall be set to 8'b10101010 (0xAA) as shown in Figure 7-4. A fill SP size may be an arbitrary integral number of bytes.

7 *0*	*7* *0*	*7* *0*	*7* *0*
10101010 (0xAA)	10101010 (0xAA)	…	10101010 (0xAA)

Figure 7-4. Fill Source Packet Payload

7.2.2.2 Application-Specific Source Packet

This standard does not define the format of application-specific SPs. While application-specific SPs are allowed, they shall not be used to encapsulate data that conforms to another defined SP format (e.g., Chapter 11 packets, Chapter 24 TmNSMessages, Ethernet data, etc.).

7.2.2.3 Test Counter Source Packet

The test counter SP is defined as a free-running 12-bit counter. The test counter SP shall consist of one 12-bit word and shall be encoded as one 24-bit Golay code word, encoded msb first as shown in Figure 7-5.

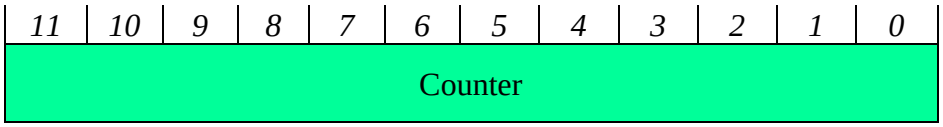


Figure 7-5. Test Counter Source Packet Payload (Unencoded)

NOTE	The test counter SP is optional and this standard does not specify the transmission rate.

7.2.2.4 Chapter 11 Source Packet

The Chapter 11 SP structure contains a slightly modified version of a Chapter 11 packet. The primary differences between the original Chapter 11 packet header and the Chapter 11 SP header are:

- The Chapter 11 SP does not contain the packet sync pattern field;
- The Chapter 11 SP does not contain the packet length field;
- The Chapter 11 SP contains a packet trailer bytes field;
- The Chapter 11 SP may contain fewer fill bytes than the original Chapter 11 packet header.

Subsection 7.2.2.4.1 describes Chapter 11 SP composition. Subsection 7.2.2.4.2 describes Chapter 11 SP reassembly.

Figure 7-6 shows the Chapter 11 general packet structure and Figure 7-7 shows the Chapter 11 SP structure.

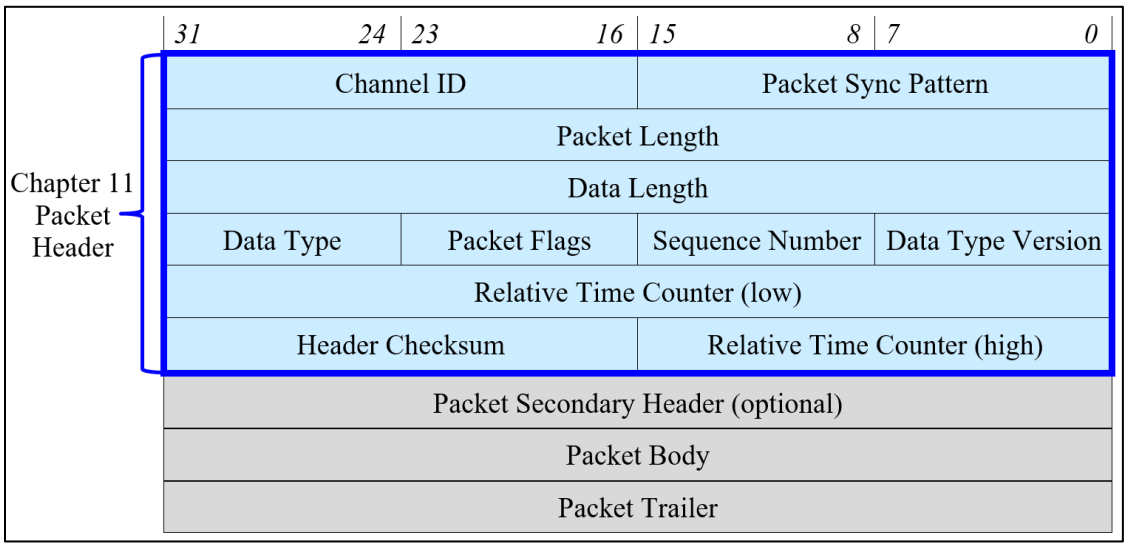


Figure 7-6. Chapter 11 General Packet Structure

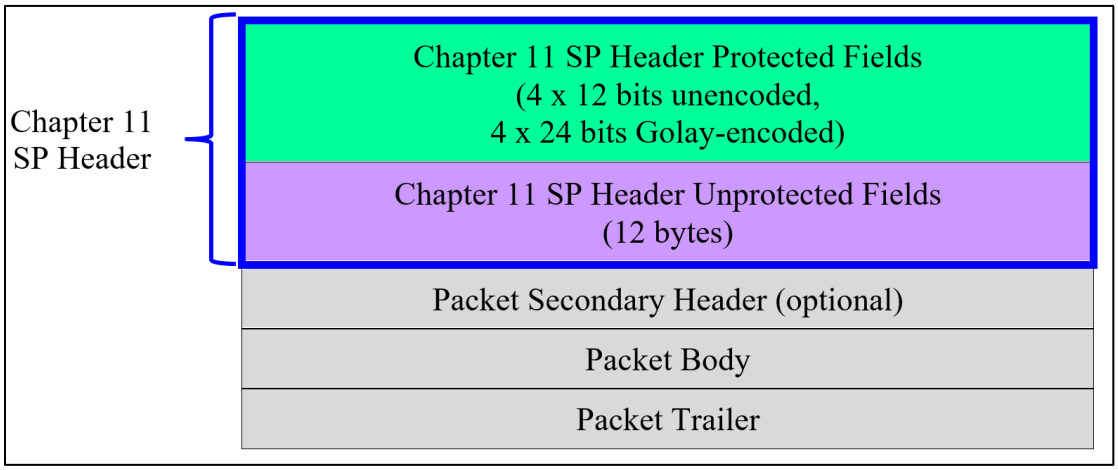


Figure 7-7. Chapter 11 Source Packet Structure

The original Chapter 11 packet header fields are partitioned into two groups for the Chapter 11 SP header:

- Golay-encoded protected fields that protect structure-critical header information;
- Unprotected fields.

The Chapter 11 SP header protected fields shall consist of four 12-bit words and shall be encoded as four 24-bit Golay code words prior to insertion into the EP payload, encoded msb first and in the word order indicated in Figure 7-8.

	11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Word 0	Reserved (0)								Channel ID (15 – 12)			
Word 1	Channel ID (11 – 0)											
Word 2	Packet Trailer Bytes[1] (4 – 0)					Data Length[2] (18 – 12)						
Word 3	Data Length[2] (11 – 0)											

[1] The packet trailer bytes shall contain the sum of the Chapter 11 SP's secondary header length, fill bytes length, and packet checksum length. See Subsection 7.2.2.4.1.

[2] The Data Length field size limit of 19 bits is sufficient for all Chapter 11 packet sizes, except Computer-Generated Data Packet, Format 1 setup record. See Subsection 7.2.2.4.1.

Figure 7-8. PT Chapter 11 Header Protected Fields (Unencoded)

The Chapter 11 SP header unprotected fields shall consist of 12 bytes as shown in Figure 7-9.

31 24	*23* 16	*15* 8	*7* 0
Data Type	Packet Flags	Sequence Number	Data Type Version
Relative Time Counter (low)			
Header Checksum		Relative Time Counter (high)	

Figure 7-9. Chapter 11 Source Packet Header Unprotected Fields

7.2.2.4.1 Chapter 11 Source Packet Composition

The following steps shall be executed prior to constructing a Chapter 11 SP.

1. Truncate the number of packet trailer fill bytes to no more than three bytes. The number of fill bytes contained in a Chapter 11 SP shall be restricted to a maximum of three bytes.
2. Update the header packet length. If the packet trailer fill bytes were truncated, the packet length shall be updated accordingly.
3. Calculate a new header checksum. If the packet trailer fill bytes were truncated, the header checksum shall be recalculated using the updated header packet length.
4. Calculate a new data checksum (if the packet trailer contains a data checksum). If the packet trailer fill bytes were truncated and non-zero packet trailer fill bytes were removed, the data checksum shall be recalculated using the truncated packet trailer fill bytes.

Once these steps are executed, the Chapter 11 packet header shall be divided into the protected and unprotected fields as described above. The packet trailer bytes field shall contain the sum of the Chapter 11 SP's secondary header length, fill bytes length (adjusted to a maximum of three bytes), and data checksum length.

If the size of the Chapter 11 packet exceeds the 19-bit limit, the data length shall be set to modulo 524,288 (2^{19}) of the Chapter 11 packet length and multiple Chapter 11 EPs shall be generated using the SP fragmentation method described in Subsection 7.2.3.

NOTE 	Compared to the original Chapter 11 packet, the resulting Chapter 11 SP is either identical, or due to fill byte truncation, shorter than the original Chapter 11 packet. If fill byte truncation occurred, the packet length and packet header checksum will be recalculated and the data checksum may be recalculated; however, the Chapter 11 SP's data content remains unchanged from the original Chapter 11 packet.

7.2.2.4.2 Chapter 11 Packet Reassembly

A Chapter 11 packet may be reassembled once an entire Chapter 11 SP has been reassembled from one or more EPs – see Subsection 7.2.3 for details. The Chapter 11 packet header shall be reassembled after Golay-decoding is performed on the Chapter 11 SP header's protected fields. The following steps shall be executed.

1. The reassembled Chapter 11 packet sync pattern shall be set to 0xEB25 (16'b1110101100100101) as specified in Chapter 11.
2. The reassembled Chapter 11 packet's packet length shall be calculated as follows:

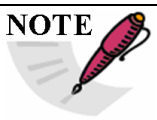
$$Packet\ Length = \sum (length\ in\ each\ EP\ Fragment\ Header)$$

3. The reassembled Chapter 11 packet's data length shall be calculated as follows:

$$Data\ Length = calculated\ Packet\ Length - Packet\ Header\ Length\ (24\ bytes) - Packet\ Trailer\ Bytes$$

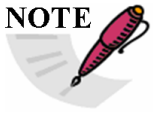
Perform this comparison to validate the reassembled Chapter 11 packet's data length:

$$Chapter\ 11\ SP's\ Data\ Length = calculated\ Data\ Length\ modulo\ 524{,}288$$

NOTE 	The modulo 524,288 (2^{19}) is required to accommodate original Chapter 11 packets that are larger than 524,288 bytes.

The following steps may be performed to verify the integrity of the reassembled Chapter 11 packet.

1. The packet header checksum should be calculated and compared to the reassembled Chapter 11 packet header checksum.
2. If present, the data checksum in the packet trailer should be calculated and compared to the reassembled Chapter 11 packet data checksum in the packet trailer.

NOTE 	If fill byte truncation occurred during Chapter 11 SP composition, the reassembled Chapter 11 packet length and packet header checksum will differ and the data checksum may differ from the original Chapter 11 packet; however, the reassembled Chapter 11 packet's data content remains unchanged from the original Chapter 11 packet.

7.2.2.5 Raw Ethernet Media Access Control Frame Source Packet

The raw Ethernet MAC frame SP contains one physical-layer MAC frame, starting with the MAC destination address and ending with the frame check sequence inclusive, as shown in Figure 7-10. The raw Ethernet MAC frame SP can contain any kind of message data, IPv4, IPv6, and jumbo messages. No extra protection is applied to the raw Ethernet MAC frame SP.

Destination MAC Address 6 bytes	Source MAC Address 6 bytes	LLC (opt) 3 – 9 bytes	Ethertype 2 bytes	Payload 46 – 1500 bytes	Frame Check Sequence 4 bytes

Figure 7-10. Raw Ethernet MAC Frame Source Packet Structure

7.2.2.6 Internet Protocol Source Packet

The IP SP contains one IPv4 as shown in Figure 7-11 or one IPv6 packet as shown in Figure 7-12. No extra protection is applied to the IP SP.

IPv4 Header (20 – 36 bytes)	IPv4 Payload (max payload length is 65,535 bytes – header size)

Figure 7-11. IPv4 Source Packet Structure

IPv6 Header (40 bytes)	IPv6 Payload (max payload length is 65,536 bytes)

Figure 7-12. IPv6 Source Packet Structure

7.2.2.7 TmNSMessage Source Packet

The TmNSMessage SP structure contains a slightly modified version of a Chapter 24 TmNSMessage. Figure 7-13 shows the Chapter 24 TmNSMessage structure and Figure 7-14 shows the TmNSMessage SP structure.

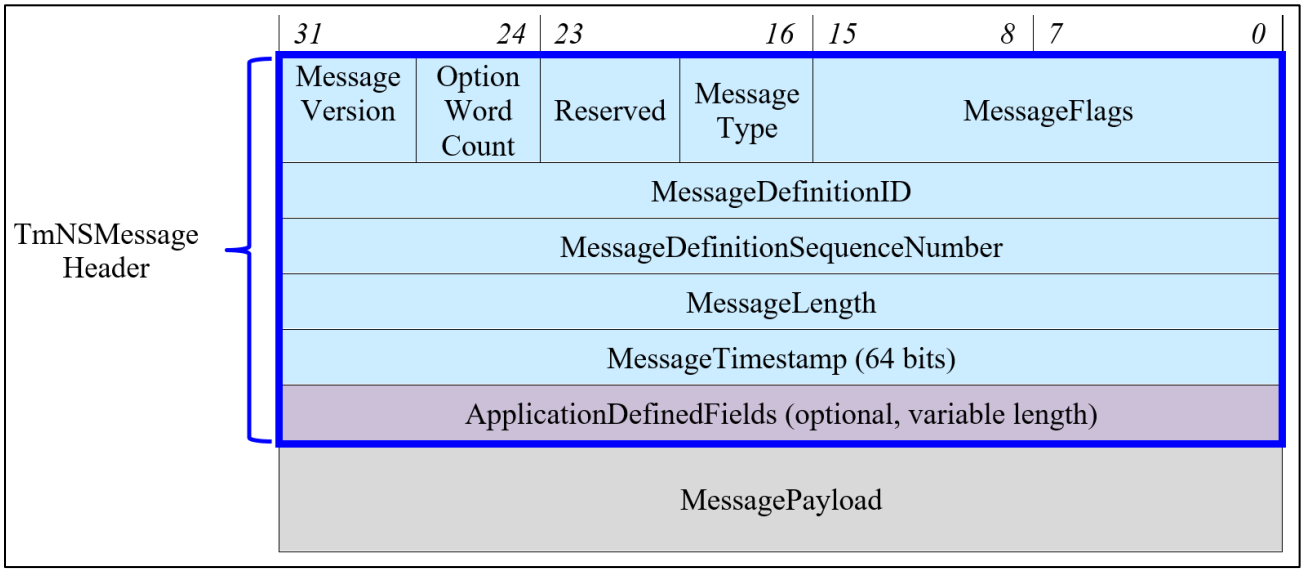


Figure 7-13. Chapter 24 TmNSMessage Structure

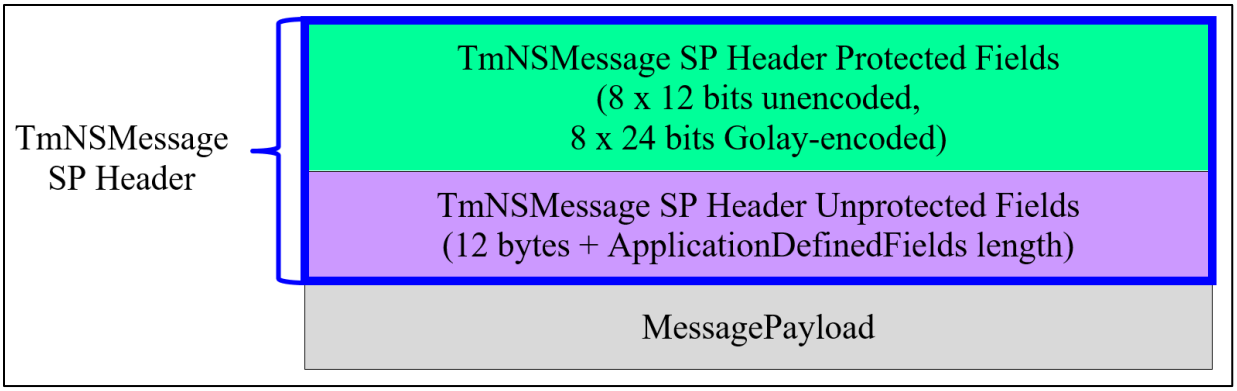


Figure 7-14. TmNSMessage Source Packet Structure

The original Chapter 24 TmNSMessage header fields are partitioned into two groups for the TmNSMessage SP header:

- Golay-encoded protected fields that protect structure-critical header information;
- Unprotected fields (those fields not part of the protected fields).

The TmNSMessage SP header protected fields shall consist of eight 12-bit words and shall be encoded as eight 24-bit Golay code words prior to insertion into the EP payload, encoded msb first and in the word order shown in Figure 7-15.

	11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Word 0	Message Version				OptionWordCount				MessageFlags (3 – 0)			
Word 1	MessageFlags (15 – 4)											
Word 2	Reserved				MessageDefinitionID (31 – 24)							
Word 3	MessageDefinitionID (23 – 12)											
Word 4	MessageDefinitionID (11 – 0)											
Word 5	MessageType				MessageLength (31 – 24)							
Word 6	MessageLength (23 – 12)											
Word 7	MessageLength (11 – 0)											

Figure 7-15. TmNSMessage SP Header Protected Fields (Unencoded)

The TmNSMessage SP header unprotected fields shall consist of 12 bytes plus the ApplicationDefinedFields (if present) as shown in Figure 7-16.

31 – *24*	*23* – *16*	*15* – *8*	*7* – *0*
MessageDefinitionSequenceNumber			
MessageTimestamp (64 bits)			
ApplicationDefinedFields (optional, variable length)			

Figure 7-16. TmNSMessage SP Header Unprotected Fields

7.2.3 Source Package Fragmentation

If an SP requires fragmentation, each EP containing an SP fragment shall be inserted sequentially into the TP stream. Only LLEPs can be inserted in between an SP's sequence of fragments by using the LLEP encapsulation mechanism as described in Subsection 7.4.2. While fragmentation is necessary if an SP's size is greater than or equal to 64 kilobytes, any SP may be fragmented. Figure 7-17 shows SP fragmentation and reassembly.

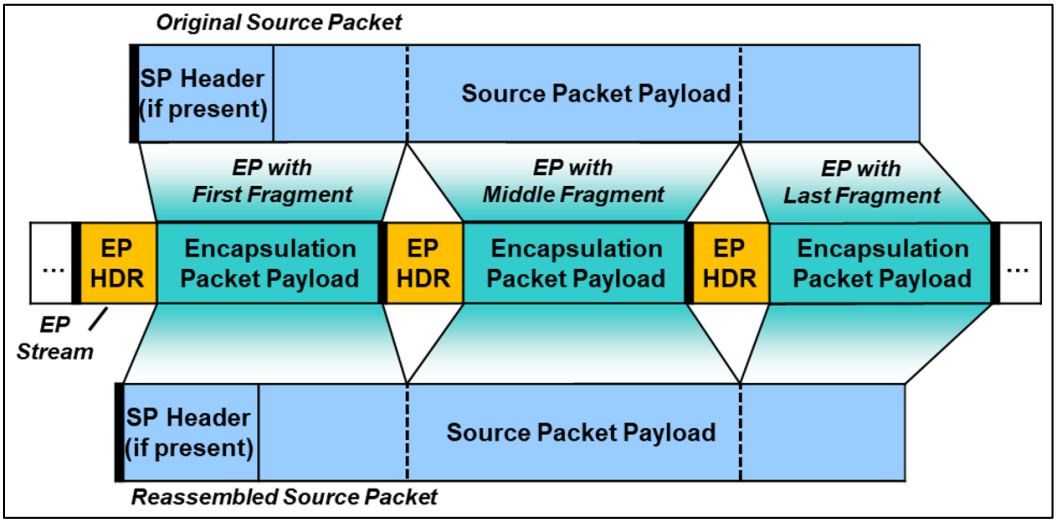


Figure 7-17. Source Packet Fragmentation and Reassembly

7.3 Transport Packet Structure

The TP is a fixed-length frame of data that contains a segment of a continuous EP stream as represented in Figure 7-18.

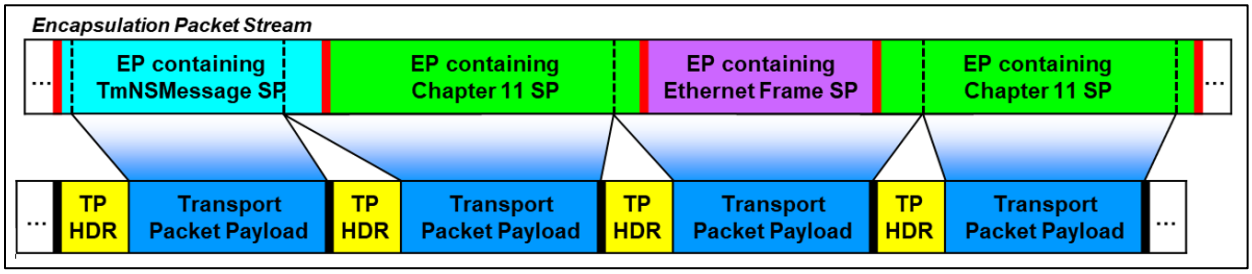


Figure 7-18. Transport Packet Overview

A TP consists of a header and a payload as shown in Figure 7-19.

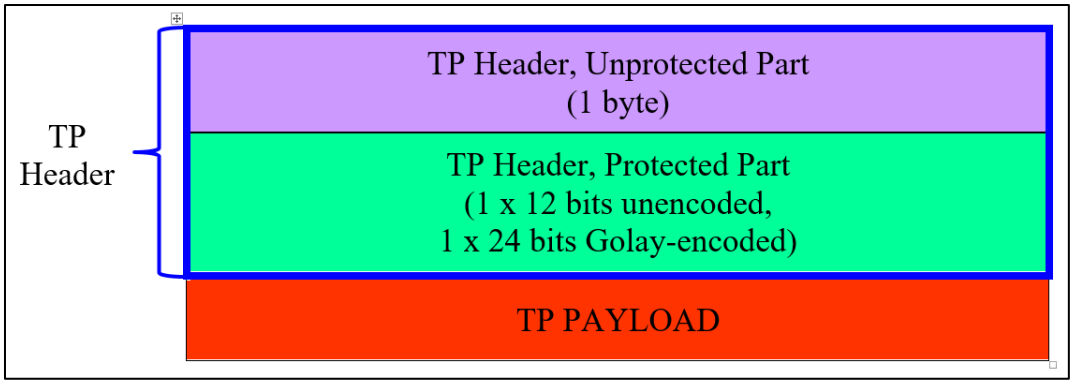


Figure 7-19. Transport Packet Structure

7.3.1 Transport Packet Header

The TP header fields are partitioned into two groups:

- Unprotected fields that contains stream and version information;
- Golay-encoded protected fields that protect structure-critical header information.

The unprotected fields shall precede the Golay-encoded protected fields.

7.3.1.1 Transport Packet Header Unprotected Fields

The TP header unprotected fields shall consist of one byte as shown in Figure 7-20.

7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Stream ID				Reserved		Version	

Figure 7-20. Transport Packet Header Unprotected Fields

The TP header unprotected fields are defined below.

- Stream ID (bits 7 – 4). The stream ID identifies an application-specific stream.
- Reserved (bits 3 – 2). All bits shall be set to zero (e.g., 2'b00) on transmission; ignored on reception.
- Version (bits 1 – 0). The TP version:
  - 2'b00: Version 1
  - 2'b01: Reserved
  - 2'b10: Reserved
  - 2'b11: Reserved

7.3.1.2 Transport Packet Header Protected Fields

The TP header protected fields shall consist of one 12-bit word and shall be encoded as one 24-bit Golay code word prior to insertion into the TP payload, encoded msb first and in the word order indicated in Figure 7-21.

11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
LL	Offset to First EP Header (bits 10 – 0)										

Figure 7-21. Transport Packet Header Protected Fields (Unencoded)

The TP header protected fields are defined below.

- LL: LLEP Exists (bit 11) – see Subsection 7.4.2 for LLEP details
  - 1'b1: indicates the TP payload contains one or more optional LLEPs and the closing LLEP end byte.
  - 1'b0: indicates no LLEP and no LLEP end byte exist in the TP payload.
- Offset to First EP Header (bits 10 – 0). If a TP contains at least one EP header, this field specifies a byte offset to the first byte of that first EP header. Otherwise, all bits shall be set to one (11'b11111111111). The value is relative to the first data byte following the TP header (e.g., the value of zero represents the first byte following the TP header). See Subsection 7.4.1 for additional information.

7.3.2 Transport Packet Payload

The TP payload contains zero or more partial or complete EPs as illustrated in Figure 7-22. Optional LLEPs may be inserted in the TP payload after the TP header (see Subsection 7.4.2 for LLEP details).

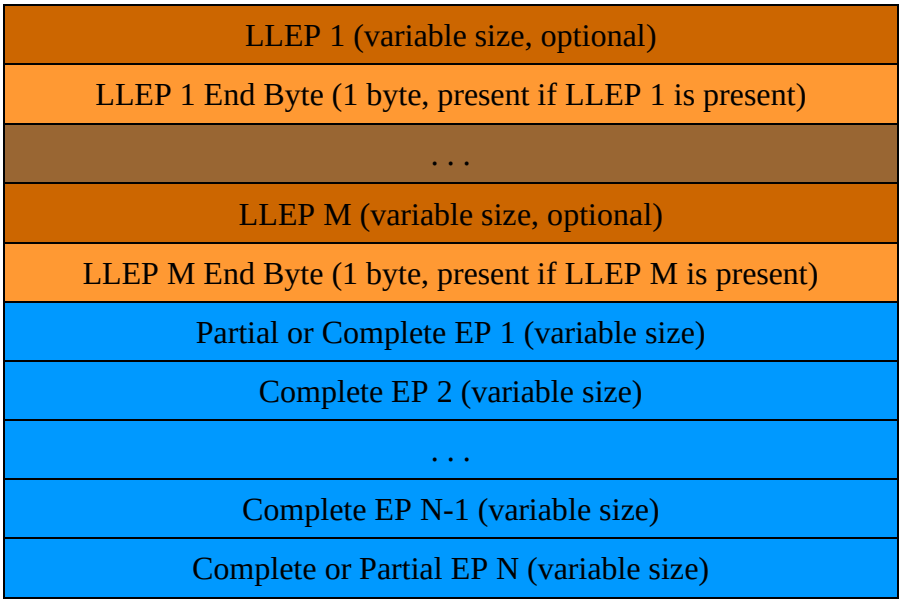


Figure 7-22. Transport Packet Payload Structure

7.3.2.1 Encapsulation Packets in Transport Packet Payload

The TP payload contains zero or more partial or complete EPs as illustrated in Figure 7-22 (blue fields). The first and last EP may be either partial or complete EPs. See Subsection 7.4.1 for details.

7.3.2.2 Low-Latency Encapsulation Packet

An LLEP is an EP having low-latency transmission requirements as described in Subsection 7.4.2. If one or more LLEPs exist in a TP, the first LLEP is placed immediately after the TP header.

7.3.2.3 Low-Latency Encapsulation Packet End Byte

For each LLEP contained within a TP, an LLEP end byte shall immediately follow the LLEP. The LLEP end byte identifies if another LLEP follows or if this is the last LLEP in the TP. The LLEP end byte encoding is as follows.

- 8b'11111111 (0xFF) indicates that another LLEP immediately follows this byte.
- 8b'00000000 (0x00) indicates there are no more LLEPs in this TP. The byte following this end byte is the first byte of the first EP, unless the LLEP end byte is the TP's last byte (i.e., there are no EPs in the TP's payload).

7.4 Asynchronous Encapsulation Packet Multiplexing

A TP contains asynchronously inserted EPs; one EP may span multiple TPs. The EP stream contains a continuous series of EPs; the start of an EP must immediately follow the last byte of a previous EP as illustrated in Figure 7-23.

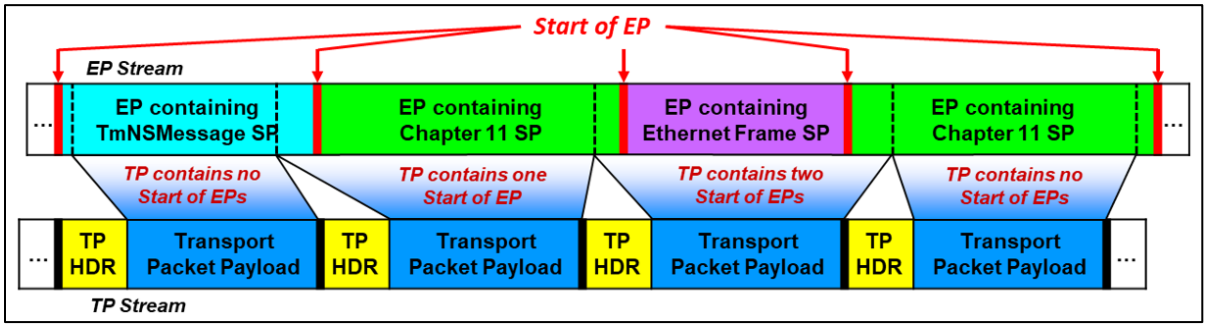


Figure 7-23. Start of Encapsulation Packet Overview

7.4.1 Standard Transport Packet Segmentation

If the start of an EP is contained within a TP, the TP header shall contain the offset to the EP's first byte. If there are multiple EPs in the TP, the TP header shall contain the offset to the start of the first EP. Since one EP may span multiple TPs, the TP header may not contain an offset to an EP. See Subsection 7.3.1.2 for details. An overview of the standard TP segmentation mechanism is shown in Figure 7-24.

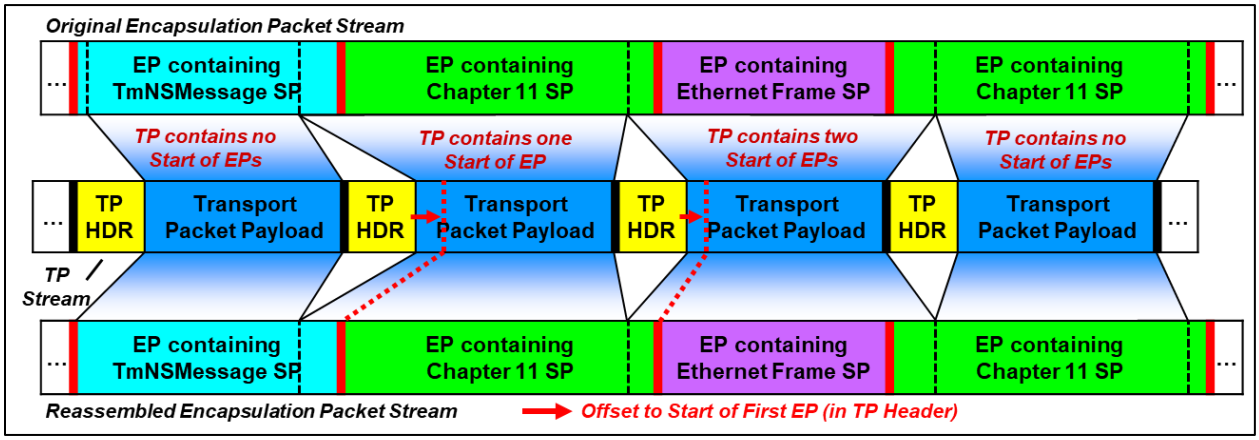


Figure 7-24. Standard Transport Packet Segmentation Overview

7.4.2 Low-Latency Encapsulation Packet Insertion

The transmission of long EPs may cause too long of latency for some critical data. Therefore, an LLEP mechanism is provided, allowing the insertion of one or more EPs with low-latency requirements within the transmission of a long EP. The interrupted long EP is resumed immediately after the LLEP part of the TP.

An LLEP shall not span multiple TPs. When constructing a TP, an entire LLEP and associated end byte shall fit into the remaining space in the TP. Figure 7-25 shows an overview of a TP with LLEPs.

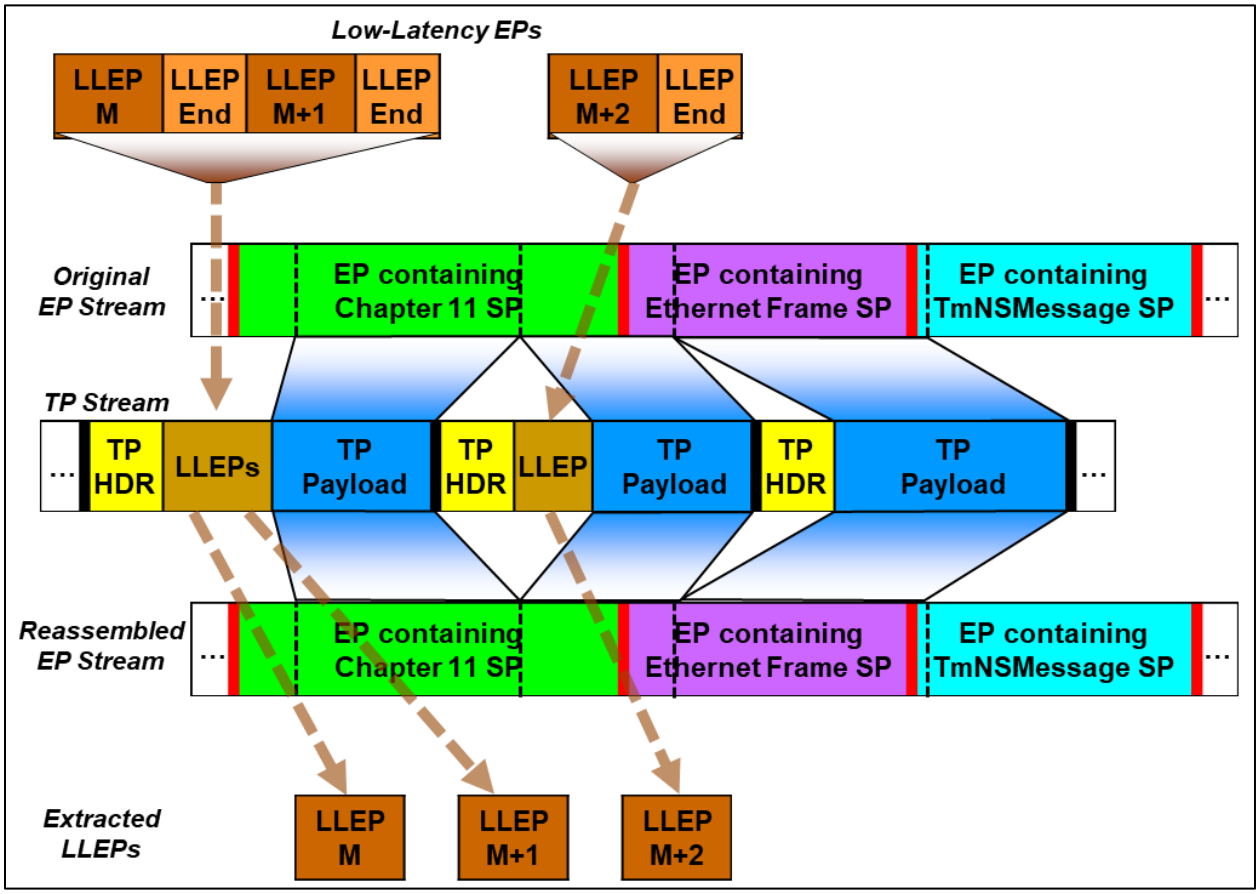


Figure 7-25. Transport Packet Segmentation with LLEPs Overview

NOTE	The offset to the first EP in the TP header is not necessarily pointing immediately after the LLEP.

7.5 Transport Packet Transport

To insert a TP into a Chapter 4 PCM minor frame, each TP segment is byte-aligned and inserted into the PCM minor frame as a byte stream with the msb first (as shown in Figure 7-26). If a TP segment's size is not an integral number of bytes, the remaining bits at the end of the TP segment are considered fill bits and should be ignored. Each PCM minor frame shall contain

exactly one complete TP structure; Figure 7-27 shows a PCM minor frame with two TP segments.

NOTE	To assist with detecting bit errors, this standard recommends adding a CRC-32 to the PCM Minor Frame as specified in Chapter 4.

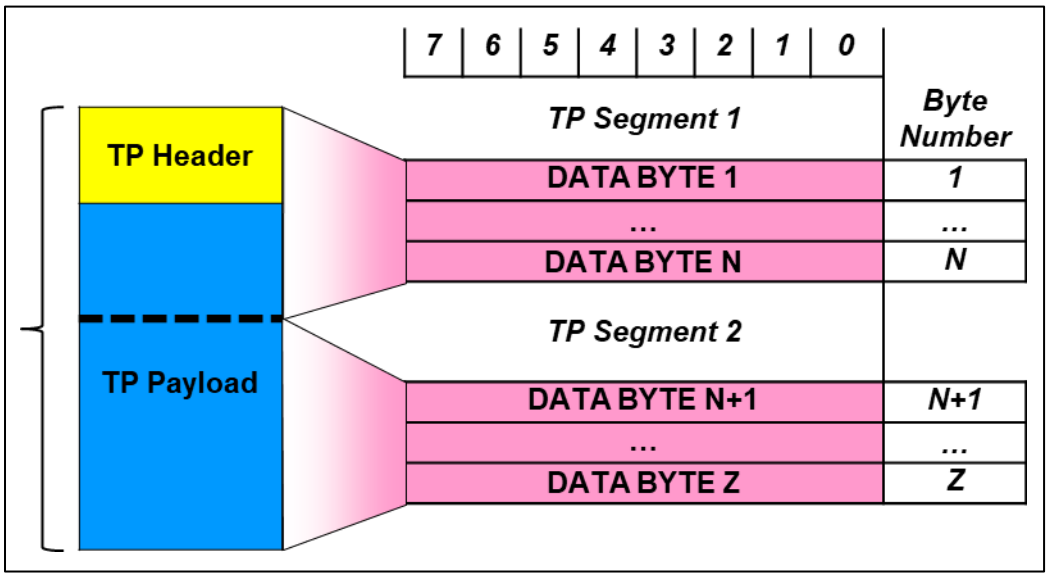


Figure 7-26. Splitting a Transport Packet into Two Segments

NOTE	Any TP segmentation does not affect how the TP header's offset to the first EP header is used to index into the TP byte array.

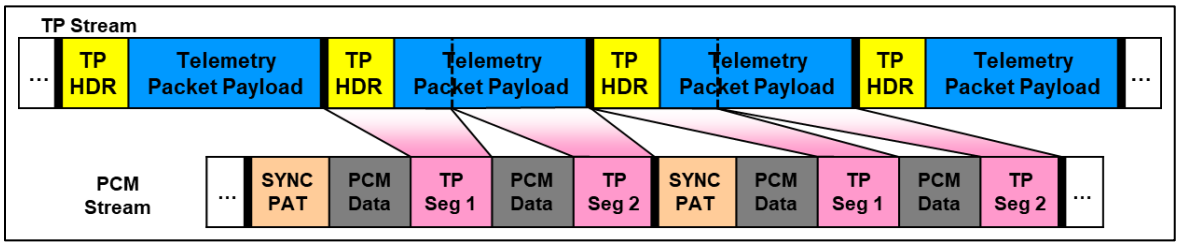


Figure 7-27. PCM Minor Frame With Two Transport Packet Segments

If no TPs are available for transmission, EPs with fill SPs shall be inserted into TPs for subsequent insertion into the PCM minor frame.

NOTE	All PCM minor frame overhead words such as the Chapter 4 sync pattern or subframe counters are not considered part of a TP.

This page intentionally left blank.

APPENDIX 7-A

Extended Binary Golay Code

A.1. Introduction

A single-bit transmission error may cause excessive data loss in packet telemetry. If the error occurs in identification or structure length fields, the error can lead to misinterpretation of a packet or a loss of a series of packets.

To help mitigate potential errors, a self-correcting coding called extended binary Golay code is applied to structure-critical elements in a packet telemetry stream. This additional coding provides protection of the packet identification and length information and supports correction of up to 3-bit transmission errors in a 24-bit sequence. This is accomplished by encoding 12-bit words into 24-bit words.

This extended binary Golay code, G_{24} (sometimes just called the "Golay code" in finite group theory) encodes 12 bits of data in a 24-bit word in such a way that any 3-bit errors can be corrected or any 7-bit errors can be detected. In standard code notation the codes have parameters (24, 12, 8) corresponding to the length of the code words, the dimension of the code, and the minimum Hamming distance between two code words, respectively.[1] The coding and decoding of the Golay code is illustrated in Figure A-1.

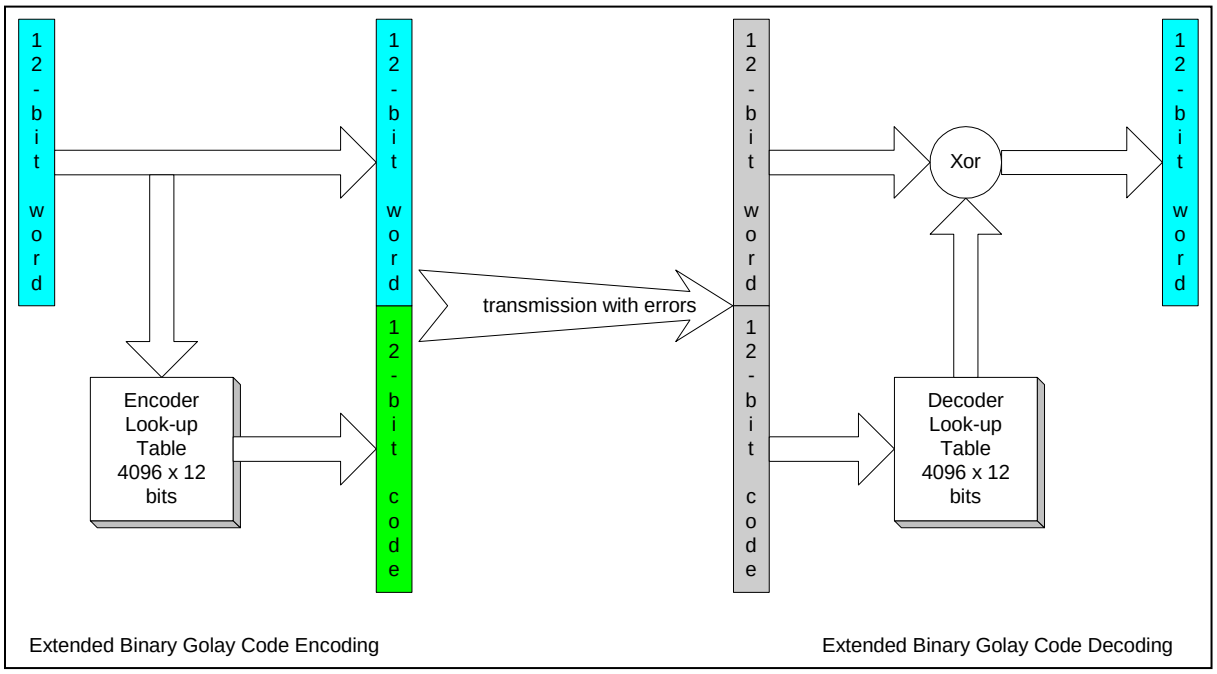


Figure A-1. Golay Code Encoding and Decoding

The following sections are C code reference implementation and define the required behavior of encoding and decoding the extended binary Golay code.

[1] Golay, Marcel J. E. *Notes on Digital Coding* in "Proceedings of the IRE," 1949, v.37, p. 657.

A.2. Encoding Golay Code

The extended binary Golay code encoding lookup table can be initialized by the InitGolayEncode() function, and the encoding can be done by the Encode(v) macro of the following C code.

```
#define GOLAY_SIZE      0x1000

// Generator matrix : parity sub-generator matrix P :

uint16_t G_P[12] = {
      0xc75, 0x63b, 0xf68, 0x7b4,
      0x3da, 0xd99, 0x6cd, 0x367,
      0xdc6, 0xa97, 0x93e, 0x8eb
};

/* Binary representation
   1 1 0 0   0 1 1 1   0 1 0 1
   0 1 1 0   0 0 1 1   1 0 1 1
   1 1 1 1   0 1 1 0   1 0 0 0
   0 1 1 1   1 0 1 1   0 1 0 0
   0 0 1 1   1 1 0 1   1 0 1 0
   1 1 0 1   1 0 0 1   1 0 0 1
   0 1 1 0   1 1 0 0   1 1 0 1
   0 0 1 1   0 1 1 0   0 1 1 1
   1 1 0 1   1 1 0 0   0 1 1 0
   1 0 1 0   1 0 0 1   0 1 1 1
   1 0 0 1   0 0 1 1   1 1 1 0
   1 0 0 0   1 1 1 0   1 0 1 1
*/
uint32_t          EncodeTable[ GOLAY_SIZE ];        // Golay encoding table

// encode a 12-bit word to a 24-bit Golay code word
#define Encode(v) EncodeTable[v&0xfff]

// initialize the Golay encoding lookup table
void InitGolayEncode( void )
{

    for( uint32_t x=0; x < GOLAY_SIZE; x++ ) {
        // generate encode LUT
      EncodeTable[x]=(x<<12);
            for( uint32_t i=0; i<12; i++ ) {
                if( (x>>(11-i)) & 1 )
            EncodeTable[x] ^= G_P[i];
            }
     }
}
```

A.3. Decoding Golay Code

The extended binary Golay code decoding lookup tables can be initialized by the InitGolayDecode() function of the following C code. The 12-bit decoded and corrected word can be calculated by the Decode(v) macro from a 24-bit code word. The number of error bits in a 24-bit code word can be gotten by the Error(v) macro from a 24-bit code word.

```
#define     GOLAY_SIZE              0x1000

uint16_t SyndromeTable[ GOLAY_SIZE ];    // Syndrome table
uint16_t CorrectTable[ GOLAY_SIZE ];     // correction table
uint8_t ErrorTable[ GOLAY_SIZE ];       // number of error bits table

#define Syndrome2(v1,v2)    (SyndromeTable[v2]^(v1))
#define Syndrome(v)         Syndrome2(((v)>>12)&0xfff,(v)&0xfff)
#define Errors2(v1,v2)      ErrorTable[Syndrome2(v1,v2)]
#define Decode2(v1,v2)      ((v1)^CorrectTable[Syndrome2(v1,v2)])

// get the number of error bits in this 24-bit code word
#define Errors(v)           Errors2(((v)>>12)&0xfff,(v)&0xfff)

// get the 12-bit corrected code from a 24-bit code word
#define Decode(v)           Decode2(((v)>>12)&0xfff,(v)&0xfff)

// Parity Check matrix
uint16_t H_P[12] = {
      0xa4f, 0xf68, 0x7b4, 0x3da,
      0x1ed, 0xab9, 0xf13, 0xdc6,
      0x6e3, 0x93e, 0x49f, 0xc75
};

/* Binary representation
   1 0 1 0   0 1 0 0   1 1 1 1
   1 1 1 1   0 1 1 0   1 0 0 0
   0 1 1 1   1 0 1 1   0 1 0 0
   0 0 1 1   1 1 0 1   1 0 1 0

   0 0 0 1   1 1 1 0   1 1 0 1
   1 0 1 0   1 0 1 1   1 0 0 1
   1 1 1 1   0 0 0 1   0 0 1 1
   1 1 0 1   1 1 0 0   0 1 1 0

   0 1 1 0   1 1 1 0   0 0 1 1
   1 0 0 1   0 0 1 1   1 1 1 0
   0 1 0 0   1 0 0 1   1 1 1 1
   1 1 0 0   0 1 1 1   0 1 0 1
*/

// calculate the number of 1s in a 24-bit word
uint8_t OnesInCode( uint32_t code, uint32_t size )
{
    uint8_t ret = 0;
    for( uint32_t i=0; i<size; i++ ) {
            if( (code>>i) & 1 )
                ret++;
    }
    return ret;
}


void InitGolayDecode( void )
{
    for( uint32_t x=0; x < GOLAY_SIZE; x++ ) {
        // generate syndrome LUT
```

```
        SyndromeTable[x]=0;         // Default value of the Syndrome LUT
        for( uint32_t i=0; i<12; i++ ) {
            if( (x>>(11-i)) & 1 ) SyndromeTable[x] ^= H_P[i];
            ErrorTable[x] = 4;
            CorrectTable[x]=0xfff;
        }
    }

    // no error case
    ErrorTable[0] = 0;
    CorrectTable[0]= 0;
    // generate all error codes up to 3 ones
    for( int i=0; i<24; i++ ) {
        for( int j=0; j<24; j++ ) {
            for( int k=0; k<24; k++ ) {
                uint32_t error = (1<<i) | (1<<j) | (1<<k);
                uint32_t syndrome = Syndrome(error);
                CorrectTable[syndrome] = (error>>12) & 0xfff;
                ErrorTable[syndrome] = OnesInCode(error,24);
            }
        }
    }
}
```

A.4. Decoding the Golay Code (8,1,3)

The one-byte 0x00 or 0xff can also be considered as a binary Golay code (8,1,3). It allows correcting the 0x00 or 0xff transmission of up to 3-bit errors, and detecting 4-bit errors. The (8,1,3) code decoding lookup tables shall be initialized by the InitGolay00FFDecode() function, and the decoding can be done by the Decode00FF(v) macro of the following C code.

```
#define     BYTE_LUT_SIZE           0x100

uint8_t Decode00FFTable[ BYTE_LUT_SIZE ]; // decode 0x00 or 0xff (8,1,3)
uint8_t Error00FFTable[ BYTE_LUT_SIZE ]; // number of error bits (8,1,3)

#define Decode00FF(v)       Decode00FFTable[v]
#define Error00FF(v)        Error00FFTable[v]

void InitGolay00FFDecode ( void )
{
    // generate (8,1,3) tables
    for( uint32_t i=0; i<BYTE_LUT_SIZE; i++ ) {
        uint32_t j = OnesInCode(i,8);
        Decode00FFTable[i] = j <= 4 ?  0 : 0xff;
        Error00FFTable[i] = j <= 4 ? j : 8-j;
    }
}
```

APPENDIX 7-B

Citations

Golay, Marcel J. E. “Notes on Digital Coding” in *Proceedings of the IRE*, 1949, v.37, p. 657.

*** * * END OF CHAPTER 7 * * ***