Telemetry Packets Definitions

Prepared by:

Benoit Cosandier

Florian George

Checked by:

Muriel Noca

Approved by:

•

EPFL
Lausanne
Switzerland

•

14 September 2009

•

Space Center EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Université Neuchâtel unine Hes·so Haute Ecole Spécialisée de Suisse occidentale EPFL ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

RECORD OF REVISIONS

ISS/REV	Date	Modifications	Created/modified by
1/0	26/02/2007	Initial release	Benoit Cosandier Florian George
1/1	02/03/2007	Update after commentaries by Serge Valera	Benoit Cosandier
1/1a	06/03/2007	Problem of page layout	Benoit Cosandier
1/2	14/03/2007	Update after the meeting after the review phase B. - Add time definition (PTC 9 and 10) - Mission constants - Time packet definition - Protocol simplification	Benoit Cosandier
1/2a	20/04/2007	Resolve problem with Packet Length.	Benoit Cosandier
1/3	20/08/2007	New major release for review phase B-C.	Benoit Cosandier
1/4	21/10/2008	Major release with updated payload service	Florian George
1/5	27/02/2009	Major update and corrections for final version	Florian George
1/6	10/09/2009	Wrong note about telecommand packet size	Florian George
1/6nu	14/09/2009	Version without uplink for publication on website	Florian George

INTRODUCTION

This document defines the application-level interface between the SwissCube ground segment and space segment. It corresponds to the tailored version of the ECSS-E-70-41A Telemetry and telecommand packet utilization as required for the SwissCube project.

This version of the document is designed for distribution to the Amateur Radio community so they can decode the telemetry coming from the SwissCube spacecraft. It does not contain the specification of the telecommand packets.

1 CONVENTIONS

The following conventions are consistently used within the SwissCube project.

1.1 Unsigned Integer values format convention

Hexadecimal values are always prefixed by the two characters "0x". Example 0x8000 is equal to the decimal value 32768.

1.2 Bit/Octet Numbering Convention

The following convention is used to identify each bit in a forward-justified N-bit field.

The first bit in the field to be transmitted (i.e. the most left-justified bit when drawing a figure) is defined to be “Bit 0”; the following bit is called “Bit 1” and so on up to “Bit N-1”.

When the field is used to express a binary value (such as an integer), the Most Significant Bit (MSB) shall be the first transmitted bit of the field (i.e. “Bit 0”).

An **octet** (i.e. a byte) is 8-bits length.

The above convention for identifying a bit is also used for identifying each octet in a forward-ordered N-octet field.

2 PACKET STRUCTURE

The TM & TC packets format used in SwissCube is compliant with the CCSDS, as defined in the PUS ([N1]). The exact structure for the SwissCube project is defined thereafter.

2.1 Telemetry packet data field structure

2.1.1 CCSDS Header

Packet Header (48 Bits)							Packet Data Field (Variable)		
Packet ID				Packet Sequence Control		Packet Length	Telemetry Data Field Header	Source Data	Packet Error Control
Version Number	Type	Data Field Header Flag	APID	Grouping Flags	Source Sequence Count				
3	1	1	11	2	14				
16				16		16	64	Variable	16

2.1.1.1 Packet Header (48 bits)

2.1.1.1.1 Packet ID (16 bits)

Version Number (3 bit): Value = 0.

Type (1 bit): Value = 0.

Data Field Header Flag (1 bit): Value = 1.

APID (Application Process ID) (11 bits): For the possible values to see [N2].

2.1.1.1.2 Packet Sequence Control (16 bits):

Grouping Flags (2 bits): value = 11

Source Sequence Count (14 bits):

A separate source sequence count shall be maintained by each APID and shall be incremented by 1 whenever it releases a packet. If the application process can send distinct packets to distinct destinations using the optional Destination ID field shown below, then a separate source sequence count is maintained for each destination. Therefore the counter corresponds to the order of release of packets by the source and enables the destination (e.g. the ground system) to detect missing packets. This counter should never re-initialize; however, under no circumstances shall it "short-cycle" (i.e. have a discontinuity other than to a value zero). The counter wraps around from 2^{14}-1 to zero.

2.1.1.1.3 Packet Length (16 bits):

The packet length field specifies the number of octets contained within the packet data field. The number shall be an unsigned integer "C" where:

C = (Number of octets in packet data field) – 1

NOTE The actual length of the entire telemetry source packet, including the source packet header, is 7 octets longer C+7.

2.1.2 Telemetry Data Field Header (64 bits)

Spare	TM Source Packet PUS Version Number	Spare	Service Type	Service Subtype	Time
Fixed BitString (1 bit)	Enumerated (3 bits)	Fixed BitString (4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Absolute Time (40 bits)

Spare (1 bit): Value = 0

TM Source Packet PUS Version Number (3 bits): Value =1

Spare (4 bits): Value = 0

Service Type (8 bits):

This indicates the service to which this telemetry source packet relates.

Service Subtype (8 bits):

Together with the service type, the subtype uniquely identifies the nature of the service report constituted by this telemetry source packet.

Remark: The definition of service type and subtype shall be unique across all application processes and the combination of these fields can be used on the ground to determine the processing priority level.

Time (40 bits):

PTC 9 and PFC 16 described in chapter 8.3.6 Absolute Time (PTC = 9)

2.1.2.1 Source Data (variable length)

The telemetry source data constitutes the data element of the service reports to the service user (e.g. the ground system).

2.1.2.2 Packet Error Control (PEC) (16 bits)

For the packets checksum, the SwissCube project uses the CRC described in annex C.

3 MISSION CONFIGURATION

The mission constants are defined in the ECSS-E-70-41A – Annex B (normative). The following table defines the value for the SwissCube project

Mission constants	SwissCube Value	Description
APPL_TIME_CODE	CUC	This mission constant identifies the presence or absence of the telemetry source packet time field as well as the time format (CUC or CDS) and the encoding of the time field. There is one such mission constant for each on-board application process.
TMPKT_MAX_LENGTH	251 octets	The maximum length of a telemetry source packet, which can be less than the maximum length defined by the CCSDS Recommendations. Remark: see TCPKT_MAX_LENGTH.
TM_CHECKSUM_TYPE	CRC	This indicates the presence and (if present) the type of checksum used for checking the integrity of telemetry source packets for the given application process. The type shall be either an ISO standard checksum or a Cyclic Redundancy Check (CRC).

4 STANDARD SERVICE FOR SWISSCUBE

4.1 Telecommand verification service (Service Type 1)

4.1.1 Service requests and reports

4.1.1.1 Telecommand acceptance

The reports of acceptance of a telecommand packet shall be as follows:

Telecommand Acceptance Report – Success (1, 1)

Telemetry source packet, source data:

Telecommand Packet ID	Packet Sequence Control
2 octets	2 octets

Telecommand Packet ID:

This is a copy of the corresponding fields from the packet header of the Telecommand to which this verification packet relates.

Packet Sequence Control:

This is a copy of the corresponding fields from the packet header of the Telecommand to which this verification packet relates.

Telecommand Acceptance Report - Failure (1, 2)

Telemetry source packet, source data:

Telecommand Packet ID	Packet Sequence Control	Code
2 octets	2 octets	Enumerated 2 octets

Code (PTC=1, PFC=16):

The code indicates the reason for the failure of the telecommand at this verification stage.

At the acceptance stage, the following standard code values are defined:

0 = illegal APID (PAC error);

1 = incomplete or invalid length packet;

2 = incorrect checksum;

3 = illegal packet type;

4 = illegal packet subtype;

5 = illegal or inconsistent application data.

Other values of the code can be application process specific or command-specific (i.e. dependent on combinations of the type, subtype and individual command function). Theses error code values are defined in the document "Flight Software Architecture" [N2].

4.1.1.2 Telecommand execution started

The reports of start of execution of a telecommand packet shall be as follows:

Telecommand Execution Started Report - Success (1, 3)

Telemetry source packet, source data: Same as for subtype 1.

Telecommand Execution Started Report - Failure (1, 4)

Telemetry source packet, source data: Same as for subtype 2.

4.1.1.3 Telecommand execution complete

The reports of completion of execution of a telecommand packet shall be as follows:

Telecommand Execution Completed Report - Success (1, 7)

Telemetry source packet, source data: Same as for subtype 1.

Telecommand Execution Completed Report - Failure (1, 8)

Telemetry source packet, source data: Same as for subtype 2.

4.2 Housekeeping & diagnostic data reporting service (Service Type 3)

4.2.1 Service reports

4.2.1.1 Reporting housekeeping or diagnostic data

The provider-initiated reports of the values of a set of housekeeping or diagnostic parameters shall be:

Housekeeping Parameter Report (3, 25)
Telemetry source packet, source data:

SID	Parameters
Enumerated 1 octet	Any

← Optionally repeated →

SID (PTC=2, PFC=8):

SID of the following reported parameters.

Parameters:

This field consists of one sequence of values of housekeeping parameters for real-time SIDs or a fixed array of sequences for archiving SIDs. In the later case, the time field of the telemetry source packet refers to the capture time of the first sequence.

For ground system processing purposes, the SID, together with the application process ID and the nature of the packet (housekeeping or diagnostic) implicitly identifies the structure of the parameter field.

Parameters and SIDs are described in [N4].

5 MISSION SPECIFIC SERVICES

This chapter defines the PUS mission specific services; in the standard PUS, the services types 127 to 255 can be used for the mission specific services.

5.1 SwissCube Image Service (Service Type 128)

5.1.1 Scope

The SwissCube Image Service is a mission specific service add-on to the standard ECSS-E-70-41A for the spacecraft SwissCube. The payload of this satellite takes images that must be transmitted to the ground segment, and the service, described hereafter, defines how it will be done.

5.1.2 Service concept

The payload of SwissCube satellite provides images of a size of 188x120 pixels with a color depth of 8 bits per pixel. For the transfer, the image is cut by lines. The size of one line is 188 octets. Lines are numbered from 0 to 119 where 0 is the number of the first line (at the top of the image).

5.1.3 Service reports

5.1.3.1 Reporting the available image

Available Image Report (128, 3)

Telemetry source packet, source data:

Image ID	Time	ADCS HK 1	ADCS HK 2
Unsigned Integer 2 octets	Unsigned Integer 4 octets	Fixed Octet-String 80 octets	Fixed Octet-String 80 octets

Image ID (PTC=3, PFC=12):

The Image ID is a unique number that identifies the images. It's a counter associated with each image. The counter may start back from zero in case of a reset of the power bus or the EPS.

Time (PTC=3, PFC=14):

The absolute time in ticks at which the image was taken.

ADCS HK 1 (PTC=7, PFC=80):

The complete housekeeping data structure of the ADCS 20 seconds before image capture.

ADCS HK 2 (PTC=7, PFC=80):

The complete housekeeping data structure of the ADCS at the time of the image capture.

5.1.3.2 Transferring an image

Image Line Report (128, 7)

Telemetry source packet, source data:

Image ID	Line Number	Line Data
Unsigned Integer 2 octets	Unsigned Integer 1 octet	Fixed Octet-String 188 octets

Image ID (PTC=3, PFC=12):

ID of the image to which the reported line belong.

Line Number (PTC=3, PFC=4):

Number of the reported line.

Line Data (PTC = 7, PFC = 188):

Data associated with the "Image ID" and "Line Number" that represents a line. The length is 188 octets (188 pixels at 8bpp).

6 REFERENCES

6.1 Normative references

[N1] ECSS-E-70-41A - Ground systems and operations – Telemetry and Telecommand packet utilization – 30 January 2003

[N2] S3-C-SE-1-1-Flight Software Architecture – 16/07/2008

[N4] S3-D-ICD-1-0-Housekeeping – 23/03/2009

7 ABBREVIATED TERMS

Ack acknowledgement

APID application process ID

Bpp Bits per pixel

CCSDS Consultative Committee for Space Data Systems

CRC cyclic redundancy code

ESA European Space Agency

OBT on-board time

PC parameter code

PFC parameter format code

PSS procedures, specifications, standards

PTC parameter type code

PUS packet utilization standard

SID structure identification

UTC universal time coordinated

VC virtual channel

8 ANNEX A: PARAMETER TYPES

8.1 Introduction

Each field in a telecommand or telemetry packet described in this document is designed to hold a parameter value. Each parameter field has a type, defining the set of values that can be assigned to the parameter. The parameter types used by the SwissCube mission are defined below.

This annex defines the physical encoding rules for each type, i.e. the permitted lengths of the parameter fields and the internal format used to encode values. This annex does not define the conversion of data parameters into physical or engineering units or user messages.

When defining telecommand and telemetry packets only parameter types defined in this section shall be allowed.

8.2 Encoding Formats of Parameter Types

The parameter type defines the range of possible parameter values. A given parameter type can vary in format and length. Each combination of parameter type and encoding format has an associated parameter code, which defines the type and its physical encoding.

The parameter code shall be used whenever a definition of a parameter field is required. The parameter codes shall be applicable to both telecommand and telemetry data.

The parameter code PC, is defined as follows:

Parameter Type Code (PTC)	Parameter Format Code (PFC)
enumerated	enumerated

The parameter code is written as (PTC, PFC) in the next subchapter.

8.3 Parameter Type Definitions

8.3.1 Boolean (PTC = 1)

PFC = 0

This is a one-bit parameter, with two distinguished values only, involved in logical operations where:

value 1 denotes "TRUE"

value 0 denotes "FALSE".

8.3.2 Enumerated Parameter (PTC = 2)

PFC = 1, 3, 4, 8, and 16: length in bits of the parameter values

This is a parameter, with discrete integer values only, involved in logical and comparative expressions (but not in numeric and relational expressions). The values that such a parameter can take are discrete and unordered. Each value has a meaning which is interpreted as a character string value. An error code is a typical example (e.g. 0 means "unchecked", 3 means "invalid").

8.3.3 Unsigned Integer (PTC = 3)

The values that this parameter can take are positive and can be involved in arithmetical, relational and comparative expressions. An unsigned integer shall be encoded with Bit-0 being the most significant bit (MSB) and Bit-N-1 the least significant bit (LSB).

The formats of an unsigned integer shall be:

Format Code	Format Definition	Lowest Value	Highest Value
0 ≤ PFC ≤ 12	PFC +4 bits, unsigned	0	$2^{PFC+4}-1$ (15 to 65 535)
PFC = 13	3 octets, unsigned	0	$2^{24}-1$ (16 777 215)
PFC = 14	4 octets, unsigned	0	$2^{32}-1$ ($\approx 4{,}3*10^{9}$)
PFC = 15	6 octets, unsigned	0	$2^{48}-1$ ($\approx 2{,}8*10^{14}$)
PFC = 16	8 octets, unsigned	0	$2^{64}-1$ ($\approx 18{,}5*10^{18}$)

8.3.4 Signed integer parameter (PTC = 4)

The values that such a parameter can take are positive or negative and can be involved in arithmetical, relational and comparative expressions.

- If Bit-0 = 0, the value shall be positive following the unsigned integer convention.
- If Bit-0 = 1, the value shall be negative and the field is the 2's complement of the positive value.

The formats of a signed integer shall be:

Format Code	Format Definition	Lowest Value	Highest Value
0 ≤ PFC ≤ 12	PFC +4 bits, signed	-2^{PFC+3} (-8 to -32 768)	$2^{PFC+3}-1$ (7 to 32 767)
PFC = 13	3 octets, signed	-2^{23} (-8 388 608)	$2^{23}-1$ (8 388 607)
PFC = 14	4 octets, signed	-2^{31} ($\approx -2{,}15*10^9$)	$2^{31}-1$ ($\approx 2{,}15*10^9$)
PFC = 15	6 octets, signed	-2^{47} ($\approx -1{,}4*10^{14}$)	$2^{47}-1$ ($\approx 1{,}4*10^{14}$)
PFC = 16	8 octets, signed	-2^{63} ($\approx -9{,}2*10^{18}$)	$2^{63}-1$ ($\approx 9{,}2*10^{18}$)

8.3.5 Octet String (PTC = 7)

PFC = 0: A variable-length octet string.

PFC > 0: A fixed-length octet string with a number of octets equal to PFC.

The values that such a parameter can take are variable-length or fixed-length sequences of octets, each octet being an ordered sequence of eight bits. The meaning and interpretation of a value shall be application process specific.

A variable-length octet-string parameter shall be of the form:

n	O_1	O_2	- - - - -	O_n
Unsigned Integer	octet	octet		octet

Where:

O1....On are octets;

n indicates the number of octets which follow.

A fixed-length octet-string parameter is of the form:

O_1	O_2	- - - -	O_n
1 octet	1 octet		1 octet

Where:

O1....On are octets;

n is the number of octets and is equal to PFC.

8.3.6 Absolute Time (PTC = 9)

PFC = 16: The CUC format with ((PFC +1) / 4, rounded down coarse time, (PFC +1) % 4 of fine time) thus 4 octets of coarse time and 1 octet of fine time.

The value of an Absolute Time parameter field shall be a number of seconds and fractions of second from a given Agency epoch. It shall be a positive time offset.

CUC format

The full CUC Format shall be as follows:

C_1	C_2	C_3	C_4	F_1
1 octet	1 octet	1 octet	1 octet	1 octet

The time in seconds from the given Agency epoch is given by:

$t = C_1*256^3 + C_2*256^2 + C_3*256 + C_4 + F_1*256^{-1}$

8.3.7 Relative Time (PTC = 10)

PFC = 10: The CUC format with ((PFC +3) / 4, rounded down) thus 3 octets of coarse time and ((PFC +1) modulo 4) thus 1 octet of fine time.

The value of a Relative Time parameter field shall be a number of seconds and fractions of a second from the occurrence time of an event whose identification can be derived from other parameters in the packet (e.g. identifying a type of on-board event) or a number of seconds and fractions of a second between two absolute times. It can be a positive or negative time offset.

The full CUC Format shall be as follows:

C_2	C_3	C_4	F_1
1 octet	1 octet	1 octet	1 octet

A positive time offset is given by:

$t = C_2*256^2 + C_3*256 + C_4 + F_1*256^{-1}$

where C_1 is in the range -128 to 127.

A negative time offset is expressed as the "2's complement" of the corresponding positive time offset

8.3.8 Deduced Parameters (PTC = 11)

PFC = 0

A type which is unspecified and can only be instantiated to one of the simple types defined in the previous subclauses. Its actual type and encoding format for an instance of the parameter field in a packet is deduced from the value(s) of other preceding parameter field(s) in the packet or from the value(s) of mission constants or a combination of both.

9 ANNEX C: SPECIFICATION OF THE CYCLIC REDUNDANCY CODE (CRC)

9.1 General specification

The packet error control field provides the capability for detecting errors which have been introduced into the telemetry source packet (or telecommand packet) by the lower layers during the transmission process or during other processing or storage activities.

The standard error detection encoding/decoding procedure, which is described in detail in the following subclauses, produces a 16-bit Packet Check Sequence (PCS) which is placed in the packet error control field. The characteristics of the PCS are those of a cyclic redundancy code, and can be expressed as follows:

a. The generator polynomial is:

$g(x) = x^{16} + x^{12} + x^5 + 1$

b. Both encoder and decoder are initialized to the "all-ones" state for each packet.

c. PCS generation is performed over the data space "D" as shown in Figure A-1 where "D" covers the entire packet including the packet header but excluding the final packet error control field.

d. The error detection properties of the CRC can be expressed as follows:

1. The proportion of all errors in the data that are not detected is approximately $1{,}53 \times 10^{-5}$.
2. An error in the data affecting an odd number of bits shall always be detected.
3. An error in the data affecting exactly two bits, no more than 65 535 bits apart, shall always be detected.
4. If an error in the data affects an even number of bits (greater than or equal to 4), the probability that the error shall not be detected is approximately 3×10^{-5} for a data length of 4 096 octets. The probability increases slightly for larger data lengths and decreases slightly for smaller data lengths.
5. A single error burst spanning 16 bits or less of the data shall always be detected. Not all intermediate bits in the error burst span need be affected.

This code is intended only for error detection purposes and no attempt should be made to utilize it for correction.

<---------- Packet ---------->	
"D" DATA SPACE	PCS
7 to 65 540 OCTETS	2 OCTETS

Figure A-1: Standard packet check sequence generation

9.2 Encoding procedure

The encoding procedure accepts an (n-16)-bit message and generates a systematic binary (n, n-16) block code by appending a 16-bit Packet Check Sequence (PCS) as the final 16 bits of the block. This PCS is inserted into the packet error control field. The equation for PCS is:

$$PCS = [x^{16} \times M(x) + x^{(n-16)} \times L(x)] \text{ MODULO } G(x)$$

where:

M(x) is the (n-16)-bit message to be encoded expressed as a polynomial with binary coefficients, n being the number of bits in the encoded message (i.e. the number of bits in the complete packet).

L(x) is the presetting polynomial given by:

$$L(x) = \sum_{i=0}^{15} x^i$$ (all "1" polynomial of order 15)

G(x) is the CCITT recommendation V.41 (Reference [10]) generating polynomial given by:

$G(x) = x^{16} + x^{12} + x^5 + 1$

+ is the modulo 2 addition operator (Exclusive OR).

NOTE The encoding procedure differs from that of a conventional cyclic block encoding operation in that the $x^{(n-16)} \times L(x)$ term has the effect of presetting the shift register to an all ones state (rather than a conventional all zeros state) prior to encoding.

9.3 Decoding procedure

The error detection syndrome, S(x) is given by:

$S(x) = [x^{16} \times C^*(x) + x^n \times L(x)]$ MODULO $G(x)$

where:

C*(x) is the received block in polynomial form.

S(x) is the syndrome polynomial which is zero if no error has been detected.

9.4 Realization of a CRC encoder - decoder

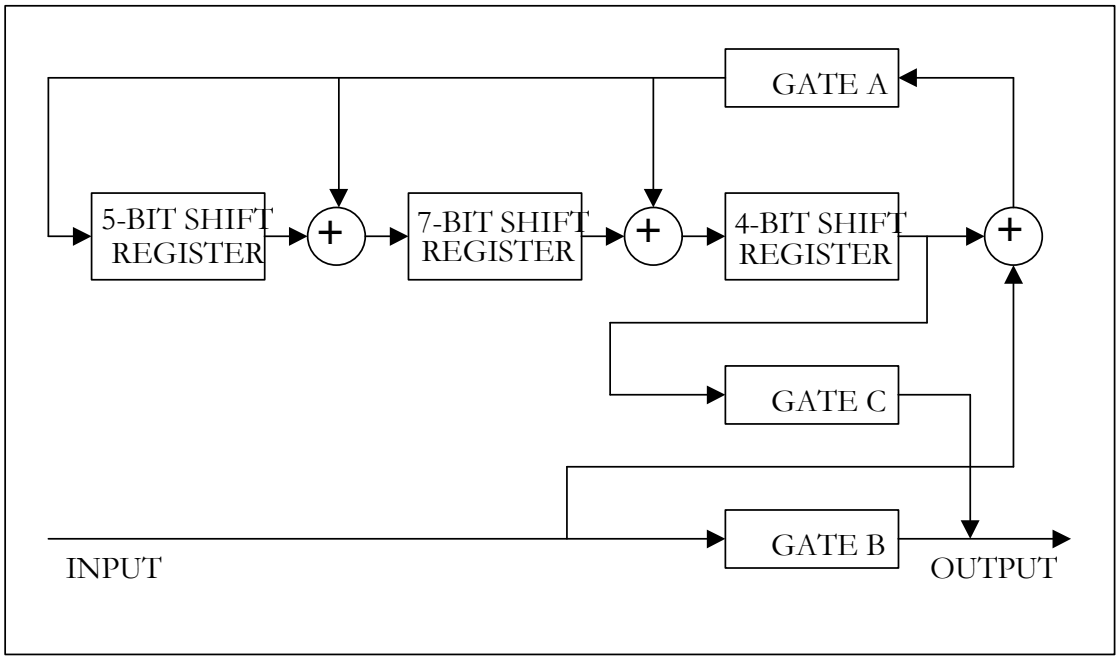
9.4.1 General

This subclause describes two arrangements, based on a shift register, for encoding and decoding a telemetry source packet (or telecommand packet) according to the packet check sequence procedures defined above.

9.4.2 Encoder

Figure A-2 shows an arrangement for encoding with the aid of a shift register. To encode, the storage stages are set to "one", gates A and B are enabled, gate C is inhibited, and (n-16) message bits are clocked into the input. They appear simultaneously at the output.

After the bits have been entered, the output of gate A is clamped to "zero", gate B is inhibited, gate C is enabled, and the register is clocked a further 16 counts. During these counts, the applicable check bits appear in succession at the output.

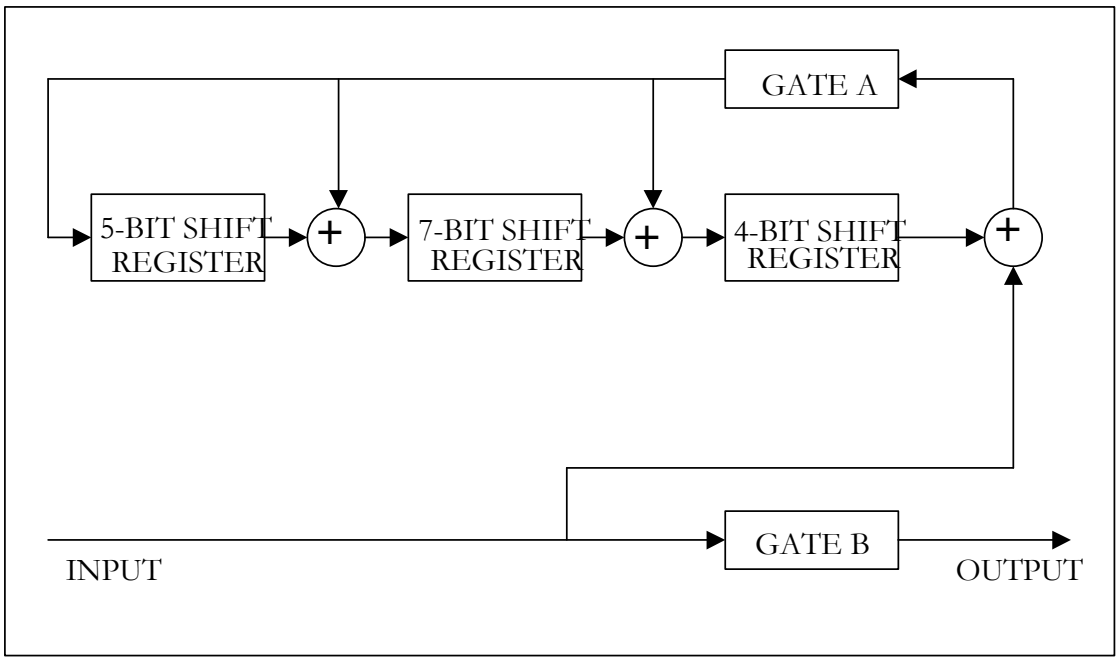


Encoder

9.4.3 Decoder

Figure A-3 shows an arrangement for decoding with the aid of a shift register. To decode, the storage stages are set to "one" and gate B is enabled.

The received n bits (i.e. the (n-16) message bits plus the 16 bits of PCS) are then clocked into the input and after (n-16) counts gate B is inhibited. The 16 check bits are then clocked into the input and the contents of the storage stages are then examined. For an error-free packet, the contents shall be "zero". Non-zero contents indicates an erroneous packet.



Decoder

9.5 Verification of compliance

The binary sequences defined in this subclause are provided to the designers of packet systems as samples for early testing, so that they may verify the correctness of their CRC error-detection implementation.

All data are given in hexadecimal notation. For a given field (data or CRC) the leftmost hexadecimal character contains the most significant bit.

Data	CRC
00 00	1D 0F
00 00 00	CC 9C
AB CD EF 01	04 A2
14 56 F8 9A 00 01	7F D5

9.6 Software implementation

9.6.1 Introduction

In addition to their interesting performance, CRC codes are particularly efficient when it comes to hardware implementation. Software implementation, on the other hand, is more complex.

The following C-language code describes the software routines to implement the CRC encoder. To implement the CRC decoder, the same routines can be used: data and the syndrome are encoded and the resulting syndrome should be equal to zero if no error is present.

9.6.2 Functions applicable to generate the CRC

-Crc FUNCTION

The Crc function calculates the CRC for one byte in serial fashion and returns the value of the calculated CRC checksum.

-Crc_opt FUNCTION

This function can be used instead of the Crc function given above. The Crc_opt function generates the CRC for one byte and returns the value of the new syndrome. This function is approximately 10 times faster than the non-optimized Crc function.

- InitLtbl FUNCTION

The InitLtbl function initiates the look-up table used by Crc_opt.

```
unsigned int Crc(Data, Syndrome)
    unsigned char Data;   /* Byte to be encoded          */
    unsigned Syndrome;   /* Original CRC syndrome        */
    {
        int i;
        for (i=0; i<8; i++) {
            if ((Data & 0x80) ^ ((Syndrome & 0x8000) >> 8)) {
                Syndrome = ((Syndrome << 1) ^ 0x1021) & 0xFFFF;
        }               else {
                Syndrome = (Syndrome << 1) & 0xFFFF;
        }
        Data = Data << 1;
    }
    return (Syndrome);
}
unsigned int Crc_opt (D, Chk, table)
    unsigned char D;                    /* Byte to be encoded          */
    unsigned int Chk;                /* Syndrome                        */
    unsigned int table [ ];       /* Look-up table                */
    {
    return (((Chk << 8) & 0xFF00)^table [(((Chk >> 8)^D) & 0x00FF)]);
    }
void InitLtbl (table)
    unsigned int table [ ];
    {
        unsigned int i, tmp;
        for (i=0; i<256; i++) {
            tmp=0;
            if ((i & 1) != 0) tmp=tmp ^ 0x1021;
            if ((i & 2) != 0) tmp=tmp ^ 0x2042;
            if ((i & 4) != 0) tmp=tmp ^ 0x4084;
            if ((i & 8) != 0) tmp=tmp ^ 0x8108;
            if ((i & 16) != 0) tmp=tmp ^ 0x1231;
            if ((i & 32) != 0) tmp=tmp ^ 0x2462;
            if ((i & 64) != 0) tmp=tmp ^ 0x48C4;
            if ((i & 128) != 0) tmp=tmp ^ 0x9188;
            table [i] = tmp;
        }
    }
```

```
/* Simple program to test both CRC generating functions */
void main ( )
    {
        unsigned int Chk; /* CRC syndrome          */
        unsigned int LTbl[256];   /* Look-up table       */
        unsigned char indata[32]; /* Data to be encoded    */
        int j;
        indata[0] = 0x31; indata[1] = 0x23; indata[2] = 0x48; indata[3] = 0x07;
        indata[4] = 0x00; indata[5] = 0xEC; indata[6] = 0xD0; indata[7] = 0x37;
        Chk = 0xFFFF;   /* Reset syndrome to all ones */
        for (j=0; j<8; j++) {
            Chk = Crc(indata[j], Chk);              /* Unoptimized CRC */
        }
        printf(" CRC = %x (should be 0)\n",Chk);
        InitLtbl(LTbl);     /* Initiate look-up table        */
        Chk = 0xFFFF;   /* Reset syndrome to all ones    */
        for(j=0;j<8;j++) {
        Chk = Crc_opt(indata[j],Chk,LTbl); /* Optimized CRC */
        }
        printf(" CRC = %x (should be 0)\n", Chk);
    }
```