TUBiX10

Telemetry Frames

Dok.-Nr.: TUBiX10_3800_TN03
Version: 1.0
Datum: 08.10.2019

Gefördert durch:

Bundesministerium für Wirtschaft und Technologie

Deutsches Zentrum für Luft- und Raumfahrt e.V.

aufgrund eines Beschlusses des Deutschen Bundestages

DLR Förderkennzeichen: 50 YB 1225

Contact:
Dr.-Ing. Zizung Yoon (project manager S-NET)
Tel.: +49 (30) 314-24438
Fax: +49 (30) 314-21306
E-Mail: zizung.yoon@tu-berlin.de
Dipl.-Ing. Walter Frese (systems engineer TUBiX10)
Tel.: +49 (30) 314-25611
Fax: +49 (30) 314-21306
E-Mail: walter.frese@tu-berlin.de
M.Sc. Jens Grosshans (project manager SALSAT)
Tel.: +49 (30) 314-29144
Fax: +49 (30) 314-21306
E-Mail: jens.grosshans@tu-berlin.de

Revision History

Version	Date	Change	Author
1.0	08.10.2019	First version	Z. Yoon
1.1			

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

ACRONYM AND SYMBOLS

Acronym	Description
ADCS	Attitude Determination and Control System (Lageregelungssystem)
BMWi	Bundesministerium für Wirtschaft und Technologie
EPS	Electric Power System
GS	Ground Station
ISL	Inter-Satelliten-Link
TC	Telecommand
TM	Telemetry
TNC	Terminal Node Controller
UHF	Ultra High Frequency (70 cm-Band der Funkamateure)

Decoding of TUBiX10 Telemetry Frames

This document describes the telemetry frame format of standard telemetry used for the TUBiX10 satellite bus. Currently, the four nanosatellites of mission S-NET have adopted this format. Mission SALSAT (Spectrum Analysis Satellite), which is scheduled for launch on mid of 2020, will use the format as well with some adaptions. As an example, the frame definition of the EPS (Electric Power System) and ADCS (Attitude Determination and Control System) standard telemetry is provided.

1 General Information

Table 1: General information of TUBiX10 satellites

Sat Name	NORAD	Cospar	Call sign	Frequency	Modulation
S-NET A	43188	2018-014J	DP0TBB	435.950 MHz for Up and Downlink	FFSK
S-NET B	43187	2018-014H	DP0TBC		
S-NET C	43189	2018-014K	DP0TBD		
S-NET D	43186	2018-014G	DP0TBE		
SALSAT	tbd	tbd	tbd		

2 AIR Frame

2.1 TMTC Communication

TUBiX10 uses the CMX469 FFSK modem chip to transmit on 70cm. This chip can generate 1k2, 2k4 or 4k8 AFSK, ready to be sent to an FM modulator. S-NET uses the 1k2 configuration, which uses tones at 1200 and 1800Hz, with the lower tone representing the bit 1 and the higher tone representing the bit 0. Note that the tone frequencies are different from the tones at 1200 and 2400Hz of the Bell 202 modem, used in 1k2 AFSK packet radio[1].

[1] https://destevez.net/2018/03/decoding-s-net/

2.2 Frame Structure

An air frame (a packet) always consists of a LTU (Link Transmission Unit) frame header and of immediately following PDU (Payload User Data) with min. length of 0 byte. Both the LTU and PDU are identically modulated (FFSK) and have the same symbol rate. The LTU frame header is always coded with block code (r=0.5) and has the max. possible SNR for the active symbol rate. The PDU data have a flexible length on 0 to 2047 bytes. If block coding is used, the data is interleaved, following in this case the data is organized in short user data blocks with constant length.

2.3 ARQ

For the ARQ (Automatic Repeat Request), a Go-Back-N method is implemented. The responder answers only one time and the caller wait a minimal time for an answer and repeats if no FrSync found. The packets can collide, but only for one time. The calls and answers are not synchronized in time on the symbol level. There is NO constant TDD (Time Division Duplex) frame interval, consisting of two packets: forward and return; each participant answers immediately. Caller and responder send always one frame/packet at one time, while the packets of caller and responder can have different symbol rates/coding schemas

2.4 LTU Frame Header

Table 2: Air Frame Header

LTU frame header	bits	Description	Value
BitSync	24	Preamble data pattern for bit synchronization	010101…
CallSign	48	Amateur radio call sign LSB first	S-NET A= DP0TBB S-NET B= DP0TBC S-NET C= DP0TBD S-NET D =DP0TBE
FrSync	32	Frame Syncronization Marker LSB first	0x20F3FA13
SrcId	7	Source Identifier	S-NET A TRx1=0 S-NET A TRx2=1 S-NET B TRx1=2 S-NET B TRx2=3 S-NET C TRx1=4 S-NET C TRx2=5 S-NET D TRx1=6 S-NET D TRx2=7
DstId	7	Destination Identifier	
FrCntTx	4	Frame Counter Transmitt	

FrCntRx	4	Frame Counter Receive	
SNR	4	Signal-to-Noise Ratio	
AiTypeSrc	4	Air Interface Type Source	0. Uncoded. All the 15 bits represent data. 1. BCH(15,11). The last 11 bits represent data. 2. BCH(15, 7). The last 7 bits represent data. 3. BCH(15, 5). The last 5 bits represent data.
AiTypeDst	4	Air Interface Type Destination	
DfcId	2	Data field contruction ID	
Caller	1	Caller/Responder	
Arq	1	Acknowledging flag	
PduTypeId	1	Payload data unit type ID	
BchRq	1	BCH_TM request flag	
Hailing	1	Hailing flag	
UdFl1	1	User defined flag #1	
PduLength	10	PDU length [Byte]	
CRC13	13	CRC for LTU Hdr (BBC) polynomial 0x1CF5	
CRC5	5	CRC for PDU (ITU) polynomial 0x15	

	Bits	
Packet length [Bit]	**70**	
Uncoded words BCH[15,5,7]	**14**	**Integer!**
Packet length [Byte]	**8.750**	
FEC length BCH[15,5,7] [Bit]	**140**	
Coded words BCH[15,5,7]	**14**	**Integer!**
Block length [Bits]	**210**	
Frame header length [Bit]	**314**	

After the Frame Sync Marker, the packet header (LTU frame header) is sent. This header consists of 70 bits, but 210 bits are sent since it uses FEC (Forward Error Correction) and interleaving. First, the header (70 bits) are encoded by using 14 BCH(15,5,7) codewords. When storing the 5 bits of each 14 chunks, the first bit of the header is stored in the last bit of the first BCH codeword, the second bit of the header is stored in the second to last bit of the first BCH codeword, and so on.

Then, the codewords are transmitted interleaved. Thus, the order of transmission is:

- 1st bit of 1st codeword, 1st bit of 2nd codeword, …, 1st bit of last (=14th codeword)
- 2nd bit of 1st codeword, 2nd bit of 1st codeword, …, 2nd bit of last (=14th codeword), etc.

The code for BCH [15,5,7] (Bode-Chaudhuri-Hocquenghem) is given in following table:

```
enum BCH_CODE_PARAMS_15_5_7
{
    BCH_15_5_7_N        = 15,                   ///< n for BCH (15,5,7).
    BCH_15_5_7_M        = 4,                    ///< m for BCH (15,5,7).
    BCH_15_5_7_K        = 5,                    ///< k for BCH (15,5,7).
    BCH_15_5_7_D        = 7,                    ///< d for BCH (15,5,7).
     BCH_15_5_7_LENGTH    =  15,                    ///<  length  for  BCH
(15,5,7).
    BCH_15_5_7_T        = 3,                    ///< error correcting ca-
pability for BCH (15,5,7).
    BCH_15_5_7_GENPOL   = 0b0000010100110111,   ///< Generator polinomial
for BCH (15,5,7).
    BCH_BITMASK_15_5_7  = 0b1111111111100000,  ///< Bitmask for clearing
FEC bits.
};
```

The decoding is done with Berlekamp/Chien's search. The 210 bits (14 codewords á 15 bits) are mapped into a 14 x uint16_t array and decoded codeword by codeword.

The CRC13 (polynomial 0x1CF5) is computed over the 65 bits comprising the header without the CRC5 field, followed by the sequence 1011011, which is used to pad the data to a multiple of 8 bits. The bytes are processed in reverse (from the last byte to the first byte), and within each byte the most significant bit is processed first. The CRC13 computation code is as following:

Table 3: CRC13 code of LTU frame

```
/**
 * Calculates LTU PDU CRC13 BBC format.<p>
 * Input: Complete LTU PDU data.<p>
 * Output: Modified CRC13 in the header.
 */
void calcLtuPduCrc(LtuStruct * frame)
{
    uint16_t    i;                ///< Byte counter.
    uint8_t        j;                ///< Bit counter.
    uint8_t        dataByte;        ///< Byte buffer.
    uint8_t        a;                ///< Variable for checking MS bit in dataByte.
    uint16_t    c;                ///< Variable for checking MS bit in crc.
    uint16_t     crc=0x1FFF;        ///< CRC13 shift register. Start value.
    i=frame->hdr.PduLength;
    while (i>0){
        //i--;
        switch (--i){
            /**
             * From here: PDU (first S-Net header, than S-Net data).
             */
            case 0:    dataByte = frame->pdu.hdr.FSYNC>>16;        break;
            case 1:    dataByte = frame->pdu.hdr.FSYNC>>8;            break;
            case 2:dataByte = (frame->pdu.hdr.FSYNC | (frame->pdu.hdr.CRC14>>8));break;
            case 3:dataByte = frame->pdu.hdr.CRC14;     break;
            case 4:dataByte = ((frame->pdu.hdr.FCID_MAJOR<<2) | (frame->pdu.hdr.FCID_SUB>>8));break;
            case 5:dataByte = frame->pdu.hdr.FCID_SUB;     break;
            case 6:
                /*dataByte = (frame->pdu.hdr.UrgentFlag<<7)
                |(frame->pdu.hdr.FutureUse<<6)
                |(frame->pdu.hdr.CrcFlag<<5)
                |(frame->pdu.hdr.Multiframe<<4)
                |(frame->pdu.hdr.TimeTaggedSetting<<3)
```

```
                |(frame->pdu.hdr.TimeTagged<<2)
                |(frame->pdu.hdr.FrameLength>>8);*/
                dataByte = (frame->pdu.hdr.FlagReg<<2) |(frame->pdu.hdr.FrameLength>>8);
                break;
            case 7:dataByte = frame->pdu.hdr.FrameLength;         break;
            default:
                //if (frame->pdu.hdr.TimeTagged == OBSSNETFRAME_TIMETAGGED_TT){
                if (CHECKBIT8(frame->pdu.hdr.FlagReg,OBSSNETFRAME_TIMETAGGED_FLAG)!=0){
                    if (i == 8){dataByte = frame->pdu.hdr.TT;        break;}
                    if (i == 9){dataByte = frame->pdu.hdr.TT>>8;    break;}
                    if (i == 10){dataByte = frame->pdu.hdr.TT>>16;    break;}
                    if (i == 11){dataByte = frame->pdu.hdr.TT>>24;    break;}
                    dataByte = frame->pdu.data[i - OBSSNETFRAME_MAX_HEADER_LENGTH];
                }
                else{
                    dataByte = frame->pdu.data[i - OBSSNETFRAME_MIN_HEADER_LENGTH];
                    break;
                }
                break;
            }
        for (j=0;j<8;j++){            ///< Count bits in g data block byte
            a = dataByte & 0x80;     ///< Check most significant bit in the byte buffer and safe in a
variable.
            c = crc & 0x1000;         ///< Check most significant bit in the CRC buffer and safe in a
variable.
            c >>= 8;               ///< Shift variable to make the compare op. possible (see beneath).
            crc <<= 1;               ///< Shift CRC to the left and write 0 into the least significant
bit.
            if (c != a){crc ^= CRC_13_BBC_POLY;}        ///< CRC polynomial.
            crc &= 0x1FFF;            ///< erase three most significant bits.
            dataByte <<= 1;            ///< Shift to the left.
        }
    }
    frame->hdr.CRC13=crc;            ///< Safe calculated value.
    return;
}
```

The CRC5 (polynomial 0x15) is computed over the entire PDU frame. The bytes are processed in reverse (from the last byte to the first byte), and within each byte the most significant bit is processed first. The CRC5 computation code is as following:

Table 4: CRC5 code of LTU frame

```
/**
 * Calculates LTU header CRC5 ITU format.<p>
 * Input: Complete LTU header (w/o CRC5 bits).<p>
 * ATTENTION! CRC13 has to be already calculated.<p>
 * Output: Modified CRC5 in the header.
 */
void calcLtuHdrCrc(LtuStruct * frame)
{
    uint8_t       i;               ///< Byte counter.
    uint8_t       j;               ///< Bit counter.
    uint8_t       dataByte;        ///< Byte buffer.
    uint8_t       a;               ///< Variable for checking MS bit in dataByte.
    uint8_t       c;               ///< Variable for checking MS bit in crc.
    uint8_t     crc=0x1F;        ///< CRC shift register. Start value.
    i=FEC_LTUHDR_CRC_LENGTH;
    while (i>0){
        //i--;
        switch (--i){
            /**
             * From here: LTU header without CRC bytes.
             *      SrcId            DstId            FrCntTx FrCntRx   SNR       AiTypeSrc  AiTypeDst
DfcId  Caller   QoS PduTypeId  BchRq    UdFl0  UdFl1   FrLength              CRC13 <p>
             *    | 6 5 4 3 2 1 0 | 6   5 4 3 2 1 0 | 3 2   1 0 | 3 2 1 0 | 3 2   1 0 | 3 2 1 0 | 3 2   1
0 | 1 0   | 0     | 0   | 0     | 0     | 0   | 0    | 9 8 7 6 5 4   3 2 1 0 | 12 11 10 9   8 7 6
5 4 3 2 1   0<p>
             *    | 7 6 5 4 3 2 1   0 | 7 6 5 4 3 2   1 0 | 7 6 5 4 3 2   1 0 | 7 6   5 4 3 2   1 0 | 7
6   5 4    3      2 1       0      | 7     6      5 4 3 2 1 0 | 7 6 5 4   3  2 1  0 | 7 6 5 4 3
2 1 0 | 7 6 5 4 3 2 1 0<p>
```

```
            *  | 0                 | 1                 | 2                  | 3                  | 4
| 5                    | 6                 | 7                | 8<p>
            */
           case 0: dataByte    = frame->hdr.SrcId<<1;                 ///< 7 MS bits (from 7 bit Id).
                   dataByte    |= frame->hdr.DstId>>6;        break;    ///< 1 MS bit (from 7 bit Id).
           case 1:    dataByte    = frame->hdr.DstId<<2;                ///< 6 LS bits (from 7 bit Id).
                   dataByte    |= frame->hdr.FrCntTx>>2;        break;    ///< 2 MS bits (from 4 bit
FrCnt).
           case 2:    dataByte     = frame->hdr.FrCntTx<<6;                 ///< 2 LS bits (from 4 bit
FrCnt).
                   dataByte    |= frame->hdr.FrCntRx<<2;              ///< 4 bits (from 4 bit FrCnt).
                   dataByte    |= frame->hdr.SNR>>2;            break;    ///< 2 MS bits (from 4 bit
SNR).
           case 3:    dataByte     = frame->hdr.SNR<<6;                     ///< 2 LS bits (from 4 bit
SNR).
                   dataByte    |= frame->hdr.AiTypeSrc<<2;          ///< 4 bits (from 4 bit AiType).
                   dataByte     |= frame->hdr.AiTypeDst>>2;     break;     ///< 2 MS bits (from 4 bit
AiType).
           case 4:     dataByte     = frame->hdr.AiTypeDst<<6;              ///< 2 LS bits (from 4 bit
AiType).
                   dataByte    |= frame->hdr.DfcId<<4;              ///< 2 bits (from 2 bit DfcId).
                   dataByte    |= frame->hdr.Caller<<3;              ///< 7th bit reserved for future
use (always zero).
                   dataByte    |=    frame->hdr.Arq<<2;
                   dataByte    |= frame->hdr.PduTypeId<<1;
                   dataByte    |=    frame->hdr.BchRq;
           case 5:    dataByte    = frame->hdr.Hailing<<7;
                   dataByte    |= frame->hdr.UdFl1<<6;
                   dataByte    |= frame->hdr.PduLength>>4;    break;
           case 6:    dataByte    = frame->hdr.PduLength<<4;
                   dataByte     |= frame->hdr.CRC13>>9;        break;
           case 7:    dataByte    = frame->hdr.CRC13>>1;        break;
           case 8:    dataByte    =    COM_DUMMY_7BIT_FOR_CRC;
                   dataByte    |= frame->hdr.CRC13<<7;        break;
           default:    break;
           }
       for (j=0;j<8;j++){            ///< Count bits in g data block byte
           a = dataByte & 0x80;     ///< Check most significant bit in the byte buffer and safe in a
variable.
           c = crc & 0x10;            ///< Check most significant bit in the CRC buffer and safe in a
variable.
           c <<= 3;                ///< Shift variable to make the compare op. possible (see beneath).
           crc <<= 1;                ///< Shift CRC to the left and write 0 into the least significant
bit.
           if (c != a){crc ^= CRC_5_ITU_POLY;}        ///< CRC polynomial.
           crc &= 0x1F;            ///< erase three most significant bits.
           dataByte <<= 1;            ///< Shift to the left.
       }
    }
    frame->hdr.CRC5=crc;            ///< Safe calculated value.
    return;
}
```

Please note, that the S-NET satellites have a bug in the CRC5 code as following. This will be corrected for the SALSAT mission.

Table 5: Bug in CRC5 code for S-NET

```
dataByte    = frame->hdr.Caller<<3;             ///< 7th bit reserved for future use
```

2.5 PDU (PAYLOAD DATA UNIT) FRAME

After the LTU header, the PDU (Payload Data Unit) header is sent block by block. Each block also uses FEC and interleaving and consists of 16 codewords of 15 bits. The interleaving is done in the same way as for the header. In contrast to the header, the data is written in the usual way (from left to right) in the last bits of each codeword.

3 PDU TELEMETRY DEFINITION

3.1 DEFINITION OF SINGLE FRAME (VERSION S-NET)

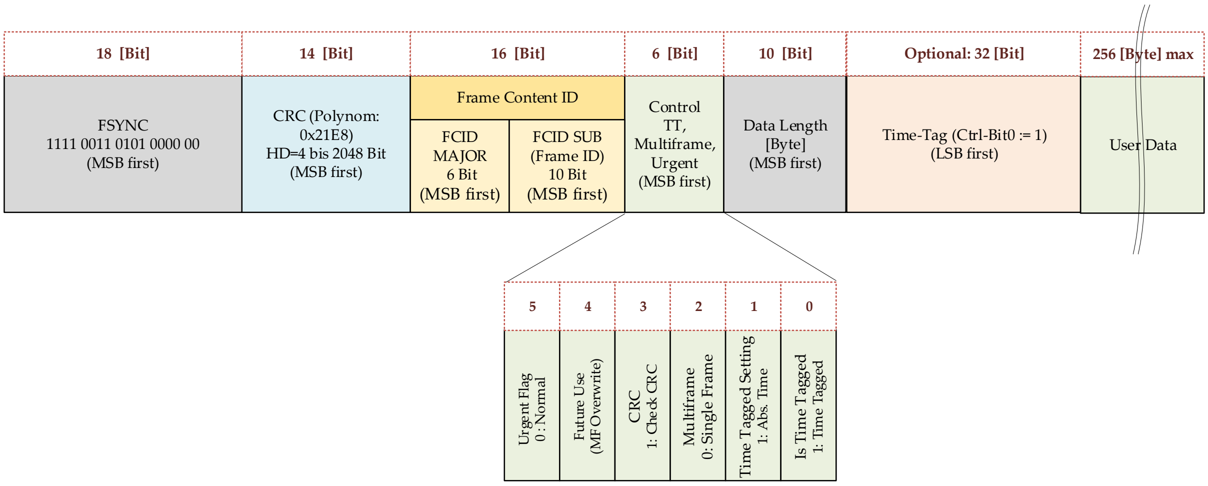

Figure 1: S-NET Single Frame

The single frame definition is illustrated in Figure 1. The frame consists of a 12 byte message header, a 4 byte time stamp and up to 256 byte of user data.

3.1.1 Frame-Header for Standard-Telemetry

The onboard software generates standard telemetry for EPS and ADCS each 500 ms. The time stamp corresponds to the moment of generation. For all standard TM, the FCID-Sub-ID=0 (Frame Content ID Sub).

Table 6: S-Net Frame Header (EPS und ADCS STD)

FSYNC	1111 0011 0101 0000 00	
CRC		
	ADCS Standard-Telemetry	EPS Standard-Telemetry
FCID-Major	0 – ADCS Housekeeping	9 – EPS Housekeeping

FCID-Sub	0 – Standard-Telemetry	
Control-Bits	0_1011	
	0 – not urgent 0/1 – not relevant 1 – CRC used 0 – single frame 1 – time stamp available 1 – absolute time stamp	
Data Length	57 (Bytes)	50 (Bytes)
Time Tag	UTC in 0.5 seconds since 2000.01.01, 00:00	

3.1.2 Cyclic Redundancy Check (CRC)

The CRC-14 check sum is obtained by starting from byte 4 (FCID) until end of data. 0x21E8 is used for the generator polynomial. The initial value is initialized with 1, 0x3FFF respectively.

3.1.3 User Data

The user data apart of the frame is subdivided into the following parameters. The number of bytes for a parameter is determined by its data type.

3.1.3.1 Parameter Types

Table 7: Size of Parameter

Data type	Bytes
Float	4
Double	8
Int8_t, uint8_t	1
Int16_t, uint16_t	2
Int32_t, uint32_t	4
Int64_t, uint64_t	8
Bool	0.125 - 1

The transmission of data words is done by LSB-First. As an example, if 2807309080(0xA7542318) is transmitted as an 32-bit unsigned integer, the bytes are arranged as following:

Byte 0	Byte 1	Byte 2	Byte 3
0x18	0x23	0x54	0xA7

Floatings-points are transmitted with the same principle. The bytes must be written into correct order and position and must be interpreted as floating-points.

If type Boolean is used, a byte is added to the frame and Bit0 is assigned with the Boolean value. All other Bits are set to zero. If additional Boolean parameters (up to 7) are used in the frame definition, these are written incrementally into higher bit positions. If another parameter type is added, a new Byte is assigned.

3.1.3.2 Parameter Conversion

Each parameter is modified in order to store, process, transmit and display its value. Incoming data is type casted, spread and scaled before being stored or displayed. The variables for the conversion process is listed in Table 8. The method to convert incoming parameters into the display and storage format is given by the following equation:

$$\hat{x} = \frac{Double\langle x_{Ch}\rangle}{S}$$

$$x_{View} = c_0 + c_1\hat{x} + c_2\hat{x}^{-1}\ while\ c_2\hat{x}^{-1} = \{0, \hat{x} = 0$$

$$\rightarrow x_{View} = c_0 + c_1\frac{Double\langle x_{Ch}\rangle}{S} + c_2\left(\frac{Double\langle x_{Ch}\rangle}{S}\right)^{-1}\ while\ c_2\left(\frac{Double\langle x_{Ch}\rangle}{S}\right)^{-1} = \{0, x_{Ch} = 0$$

$$S \neq 0$$

Table 8: Variables for parameter conversion

Symbol	Description
x_{View}	Displayed (real) value in defined unit
$\hat{x}$	Intermediate value
x_{Ch}	„Channel" data – tranmitted data
$c_{0,..,3}$	Coefficient of scaling function
S	Spread factor to scale up/down according tot he data type

3.2 Definition of Single Frames (Version SALSAT)

The SALSAT frame definition is identical to the S-NET version except the changes described herein.

3.3 Definition of Multi Frames (Version SALSAT)

The SALSAT frame definition is identical to the S-NET version except the changes described herein.

4 TELEMETRY FRAMES

Here the standard TM of S-NET satellites are described.

4.1 EPS STANDARD-TELEMETRY

For all parameters: c_0= 0, c_1=1, c_2 = 0

Table 9: S-Net Frame EPS_STD_TM

Parameter Name	Description	Unit	s	Data type
EPS_PGET_S00_CUR_SOLX_POS	current X+ solar panel	mA	50	int16_t
EPS_PGET_S01_CUR_SOLX_NEG	current X- solar panel	mA	50	int16_t
EPS_PGET_S02_CUR_SOLY_POS	current Y+ solar panel	mA	50	int16_t
EPS_PGET_S03_CUR_SOLY_NEG	current Y- solar panel	mA	50	int16_t
EPS_PGET_S04_CUR_SOLZ_POS	current Z+ solar panel	mA	50	int16_t
EPS_PGET_S05_CUR_SOLZ_NEG	current Z- solar panel	mA	50	int16_t
EPS_PGET_S06_V_SOL	main solar voltage (2,42V = 25V)	mV	1	int16_t
EPS_PGET_S24_V_BAT0	voltage battery 0 (2.37V@12.6V)	mV	2	int16_t
EPS_PGET_S26_A_IN_CHARGER0	input current charger0 (2.5V@2500mA)	mA	12	int16_t
EPS_PGET_S25_A_OUT_CHARGER0	output current0 (2.5V@5000mA)	mA	6	int16_t
EPS_PGET_S13_V_BAT1	voltage battery 1 (2.37V@12.6V)	mV	2	int16_t
EPS_PGET_S23_A_IN_CHARGER1	input current charger1 (2.5V@2500mA)	mA	12	int16_t
EPS_PGET_S14_A_OUT_CHARGER1	output current charger1 (2.5V@5000mA)	mA	6	int16_t
EPS_PGET_S22_V_SUM	sum voltage unregulated bus (2.39V = 14V)	mV	2	int16_t
EPS_PGET_S44_V_3V3	voltage 3V3 bus (2.4V@3.6V)	mV	8	int16_t
EPS_PGET_S45_V_5V	voltage 5V bus (2.39V@5.5V)	mV	5	int16_t
THM_PGET_S31_TH_BAT0	temperature of battery 0	°C	256	int16_t
THM_PGET_S15_TH_BAT1	temperature of battery 1	°C	256	int16_t
THM_PGET_TH_OBC	temperature OBC external sensor (LMT85)	°C	1	int16_t
EPS_PGET_A_OBC	current of OBC (2V@400mA)	mA	1	uint16_t
EPS_PGET_V_OBC	voltage of OBC (2.048V@4.096V)	mV	1	uint16_t
EPS_PGET_S30_A_IN_BAT0	charge current battery 0 (2.5V@2500mA)	mA	12	int16_t
EPS_PGET_S29_A_OUT_BAT0	discharge current battery 0 (2.5V@2500mA)	mA	12	int16_t
EPS_PGET_S12_A_IN_BAT1	charge current battery 1 (2.5V@5A)	mA	12	int16_t
EPS_PGET_S20_A_OUT_BAT1	discharge current battery 1 (2.5V@2500mA)	mA	12	int16_t

4.2 ADCS STANDARD-TELEMETRIE

For all parameters: c_0= c_2 = 0

Table 10: S-Net Frame ADCS STD TM

Parameter Name	**Description**	**Unit**	S	**Data type**	c_1
ADCS_PGET_iModeChkListThisStepActive	Active ADCS mode [this step]	1	1	int8_t	1
ADCS_PGET_iAttDetFinalState	Current state of attitude determination	1	1	uint8_t	1
ADCS_PGET_iSensorArrayAvailStatusGA	Sensor availability of this GA	1	1	uint8_t	1
ADCS_PGET_iSensorArrayAvailStatusMFSA	Sensor availability of this MFSA	1	1	uint8_t	1
ADCS_PGET_iSensorArrayAvailStatusSUSEA	Sensor availability of this SUSEA	1	1	uint8_t	1
ADCS_PGET_iActArrayAvailStatusRWA	Actuator availability of this RWA	1	1	uint8_t	1
ADCS_PGET_iActArrayAvailStatusMATA	Actuator availability of this MATA	1	1	uint8_t	1
ADCS_PGET_AttDetMfsDistCorrMode	Auto Correction Mode MFS Distorsion	1	1	uint8_t	1
ADCS_PGET_AttDetSuseDistCorrMode	Auto Correction Mode SUSE Distorsion	1	1	uint8_t	1
ADCS_PGET_AttDetTrackIGRFDeltaB	Change of external magnetic field by virtual magnetic field vectors		1	bool	1
ADCS_PGET_AttDetSuseAlbedoTracking	Albedo analysis of sun sensor		1	bool	1
ADCS_PGET_SUSE1AlbedoFlag	Albedo flag of sun sensor 1		1	bool	1
ADCS_PGET_SUSE2AlbedoFlag	Albedo flag of sun sensor 2		1	bool	1
ADCS_PGET_SUSE3AlbedoFlag	Albedo flag of sun sensor 3		1	bool	1
ADCS_PGET_SUSE4AlbedoFlag	Albedo flag of sun sensor 4		1	bool	1
ADCS_PGET_SUSE5AlbedoFlag	Albedo flag of sun sensor 5		1	bool	1
ADCS_PGET_SUSE6AlbedoFlag	Albedo flag of sun sensor 6		1	bool	1
ADCS_PGET_AttDetAutoVirtualizeMFSA	MFSA not valid and not used if narrow vector detected		1	bool	1
ADCS_PGET_AttDetAutoVirtualizeSUSEA	SUSEA not valid and not used if narrow vector detected		1	bool	1
ADCS_PGET_AttDetNarrowVectors	Detect narrow vector between sun and magnetic field		1	bool	1
ADCS_PGET_AttDetMismatchingVectors	Mismatch between vector measurement and model		1	bool	1
ADCS_PGET_omegaXOptimal_SAT	Angular rate X for control loop	°/s	260	int16_t	1
ADCS_PGET_omegaYOptimal_SAT	Angular rate Y for control loop	°/s	260	int16_t	1
ADCS_PGET_omegaZOptimal_SAT	Angular rate Z for control loop	°/s	260	int16_t	1
ADCS_PGET_magXOptimal_SAT	Magnetic field X for control loop	nT	0.1	int16_t	1
ADCS_PGET_magYOptimal_SAT	Magnetic field Y for control loop	nT	0.1	int16_t	1
ADCS_PGET_magZOptimal_SAT	Magnetic field Z for control loop	nT	0.1	int16_t	1
ADCS_PGET_sunXOptimal_SAT	Sun vector x-axis for control loop	mm	32000	int16_t	1
ADCS_PGET_sunYOptimal_SAT	Sun vector y-axis for control loop	mm	32000	int16_t	1
ADCS_PGET_sunZOptimal_SAT	Sun vector z-axis for control loop	mm	32000	int16_t	1
ADCS_PGET_dCtrlTorqueRWAx_SAT_lr	Control torque on RW X	µNm	38484	int8_t	1000000
ADCS_PGET_dCtrlTorqueRWAy_SAT_lr	Control torque on RW Y	µNm	38484	int8_t	1000000
ADCS_PGET_dCtrlTorqueRWAz_SAT_lr	Control torque on RW Z	µNm	38484	int8_t	1000000
ADCS_PGET_dCtrlMagMomentMATAx_SAT_lr	Control torque mag. torquer X	Am²	127	int8_t	1
ADCS_PGET_dCtrlMagMomentMATAy_SAT_lr	Control torque mag. torquer Y	Am²	127	int8_t	1
ADCS_PGET_dCtrlMagMomentMATAz_SAT_lr	Control torque mag. torquer Z	Am²	127	int8_t	1
ADCS_PGET_iReadTorqueRWx_MFR	Measured control torque of RW X	µNm	9696969	int16_t	1000000

Parameter Name	**Description**	**Unit**	S	**Data type**	c_1
ADCS_PGET_iReadTorqueRWy_MFR	Measured control torque of RW Y	µNm	9696969	int16_t	1000000
ADCS_PGET_iReadTorqueRWz_MFR	Measured control torque of RW Z	µNm	9696969	int16_t	1000000
ADCS_PGET_iReadRotSpeedRWx_MFR	Measured speed of RW X	rpm	1	int16_t	1
ADCS_PGET_iReadRotSpeedRWy_MFR	Measured speed of RW Y	rpm	1	int16_t	1
ADCS_PGET_iReadRotSpeedRWz_MFR	Measured speed of RW Z	rpm	1	int16_t	1
ADCS_PGET_SGP4LatXPEF	Latitude of Satellite	°	355	int16_t	1
ADCS_PGET_SGP4LongYPEF	Longitude of Satellite	°	177	int16_t	1
ADCS_PGET_SGP4AltPEF	Altitude of Satellite	km	0.25	uint8_t	1
ADCS_PGET_AttitudeErrorAngle	angle in deg to unify SAT and TGT	°	177	uint16_t	1
ADCS_PGET_TargetData_Distance	Distance to target	km	1	uint16_t	1
ADCS_PGET_TargetData_ControlIsActive	Control active		1	bool	1