SCOPE

This document describes the waveform used in satellite telemetries. A Telemetry frame is generated in the moments prior to its transmission, with the latest available data.

DESCRIPTION

PERIODICITY: If the battery voltage is higher than 3550mV, telemetries are transmitted each 30second, with a period of 14 minutes.

CARRIER: The downlink frequency of the transmitter is 436.888 MHz or 436.666 MHz. The Doppler effect shifts +10kHz the carrier at a rate of up to 40Hz per second.

MODULATION: A 2FSK modulation system is used, with 1125Hz tone separation. The lowest frequency corresponds to the MARK signal or logic 1, the highest frequency corresponds to the SPACE signal or logic 0. The bit rate is 200 bits/second in all new satellites. GENESIS-U is using 50bps.

FRAME: The table shows the fields of a typical telemetry frame and its length in bits.

ID	LEN	DESCRIPTION
TRAINING	32	AAAAAAAAAAAAAA
SYNC	16	BF35
TYPE	4	PACKET TYPE
ADDRESS	4	SATELLITE IDENTIFICATION
DATA	8*N	DATA PAYLOAD
CRC	16	PACKET CHECKSUM

ORDER:
MSB Most Significant Bit is the first bit to be transmitted of each byte.
BE Big-Endian order is used for transmission of multi-byte words.

PREAMBLE: Each data packet is preceded by the sequence 0xAAAA (10101010 10101010 10101010 10101010) that allows the demodulator to identify the frequency/shift of the mark/space tone pair.

SYNC: Word 0xBF35 marks the start of the data frame.

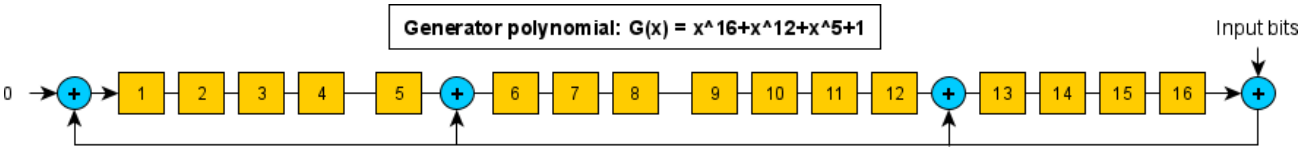
ADDRESS: A 4-bit word identifies the originating satellite of the transmission. Up to 16 satellites are possible.

ADRR	DESCRIPTION
0	Ground Station
1	GENESIS-I
7	GENESIS-U
8	GENESIS-V
9	GENESIS-M
11	GENESIS-E
12	GENESIS-F
13	GENESIS-R

TYPE: A 4-bit word identifies the type of packet and its length in bytes. Up to 16 packet types are possible.

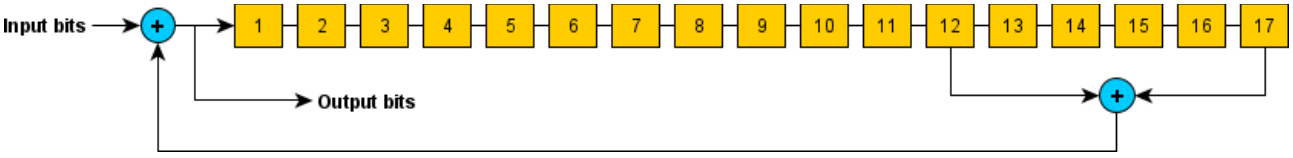
ID	LEN	DESCRIPTION
1	31	POWER
2	17	TEMP
3	29	COUNTERS
4	35	POWER MAX/MIN
5	27	TEMP MAX/MIN
6	135	SUN SENSOR
8	31	ANTENNA MECHANISM
9	123	POWER RAW DATA
14	38	SENSOR TIME SERIES
12	64	EPHEMERIDES
7	101	EXPERIMENT DATA
10	17	EXPERIMENT DATA
11	9	EXPERIMENT DATA
13	47	EXPERIMENT DATA
15	41	EXPERIMENT DATA

CRC: A 16-bit word used to check the integrity of DATA field. The figure shows the CRC calculation algorithm. The polynomial used is called CRC-CCITT-FALSE.

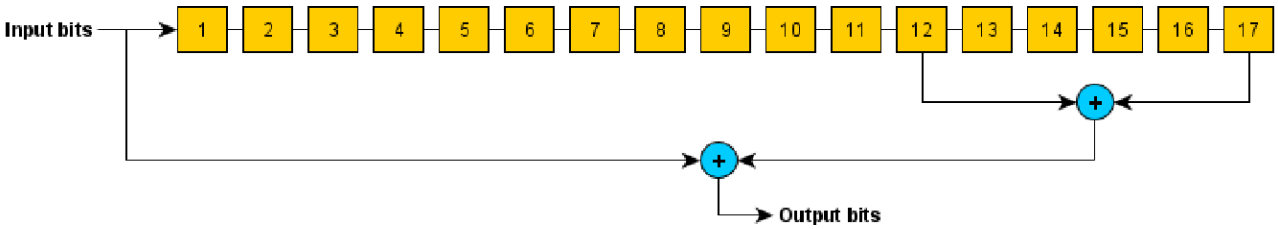


SCRAMBLER: To do mark/space equiprobable, DATA field is scrambled. The encoding and decoding algorithms are based on a multiplicative scrambler. Its implementation is defined by the following polynomial: $G(x) = x17 + x12 + 1$. For each packet received, the shift register is initialized with 0x2C350000. The figures show the multiplicative encoder and decoder respectively.

encoder



decoder



OPERATIONS: To create a valid frame, operations must be applied following this order

OPERATION	FIELD
	data
SCRAMBLE	DATA
CHECKSUM	DATA CRC
HEADER	TYPE ADR DATA CRC
BIT/FRAME SYNC	TRAIN SYNC TYPE ADR DATA CRC
MODULATION	

IMPLEMENTATION

```
#define U8 unsigned char
#define U16 unsigned short
#define U32 unsigned int

const u8 lengths[16]={0,31,17,29,35,27,135,101,31,123,17, 9,64,47,38,41};
                    //0  1  2  3  4  5   6   7  8   9 10 11 12 13 14 15

u16 checksum(u8* d, int len) {
    u8 x;
    u16 crc = 0xFFFF;
    while (len--) {
        x = crc >> 8 ^ *d++;
        x ^= x >> 4;
        crc = (crc << 8) ^ ((u16)(x << 12)) ^ ((u16)(x << 5)) ^ ((u16)x);
    }
    return crc;
}

void scrambler(u8 *d, int len) {
    u32 reg = 0x2C350000;
    u8 in, out;
    while (len--) {
        for (int b = 7; b > 0; b--) {
            in = ( *d >> b) & 1;
            out = (in ^ (reg >> 16) ^ (reg >> 11)) & 1;
            if (out) *d |=  (0x01 << b);
            else *d &= ~(0x01 << b);
            reg = ((reg << 1) | (out & 1)) & 0x1FFFF;
        }
        d++;
    }
}

void descrambler(u8* data, int len){
    u8 inBit, outBit;
    u32 descrambler_LFSR = 0x2C350000;
    while (len--) {
        for (int b = 7; b > 0; b--) {
            inBit = (*data >> b) & 1;
            outBit = (inBit ^ (descrambler_LFSR >> 16) ^ (descrambler_LFSR >> 11)) & 1;
            if (outBit) *data |= (0x01 << b);
            else *data &= ~(0x01 << b);
            descrambler_LFSR = ((descrambler_LFSR << 1) | (inBit & 1)) & 0x1FFFF;
        }
        data++;
    }
}
```

end