Transmit frequency is f_0 = 435.450 MHz and crystal frequency of first test board is 30 MHz. Data rate at radio channel is r = 9600bps, modulation is FSK (deviation Δf = 3 kHz). Packet preamble is 8 bytes (64 bits) long repeating pattern 101010…(0xAA..)

More detailed overview and description of packet structure can be found in [1] and [2]. Here is presented only as much as necessary for realization of transmission protocol. Structure of frame can be seen on figure 3.1. All octets of a frame, except the FCS field, shall be transmitted LSB first. The FCS shall be sent MSB first [1].

Flag	Address	Control	PID	Info	FCS	Flag
01111110 0x7E	112 Bits	00000011 0x03	11110000 0xF0	*N* (up to 256) x 8 Bits	16 Bits	01111110 0x7E

Figure 3.1 Radio Frame structure

1. The flag field is one octet long. Because the flag delimits frames, it occurs at both the beginning and end of each frame. Two frames may share one flag, which would denote the end of the first frame and the start of the next frame. A flag consists of a zero followed by six ones followed by another zero, or 01111110 (0x7E hex). As a result of bit stuffing, this sequence is not allowed to occur anywhere else inside a complete frame [2].

Bit stuffing is a technique to assure the flag delimiter does not appear accidentailly during frame content. The station sending shall monitor data to be sent and whenever detecting 5 consequtive 1's insert a 0 right after the 5'th 1, making the transmitted bit sequence effectively longer. The receiving station shall implement the inverse scheme – whenever detecting 5 consequtive 1's it shall remove the 0 from the bit-stream following the 5'th 1. The only data sequences not going through this bit-stuffing and unstuffing procedure are the Flags [1].

2. The address field, according to AX.25 protocol [2], is to identify the source of frame and its destination. In this space system application, the satellite is designed to talk only to one defined ground station. Therefore, the address field has always one form in frames from ground station to satellite and another fixed form when frames are sent by satellite to ground station.
If foreign amateur UHF receiver(s) are communicating with satellite, then their respective address should be used. So address field should be fixed while sending data from our ground station to satellite. But satellite should take Source address of received request and use it as Destination address while (if) responding.

This space system uses only „Nonrepeater Address-Field Encoding" as defined in [2], and consists of 14 octets as described on 3.2.

112-bit Address Field													
Destination Address							**Source Address**						
A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14

Figure 3.2 Address Field structure

The Address field (see 3.2) consists of 56-bit destination station address call sign and SSID, and 56-bit Source station address amateur radio call sign and SSID.

The octets A1-A13 are the respective bytes of amateur radio call sign and SSID shifted left by 1 bit. The A14 (sent last) is the last octet of source amateur radio call sign and SSID shifted left by 1 bit and ORed with 1 (LSB set to 1) to indicate the end of Address Field.

3. The control field identifies the type of AX.25 [2] frame. This space system uses only AX.25 „UI“ and therefore is fixed to value 0x03. The P/F bit in Control Field is not used and therefore is set to 0 according to AX.25 [2] requirements.

4. The PID field in AX.25 frame [2] identifies the protocol of Info Field. This Space System does not use any AX.25 defined protocols in Info Field and therefore this octet is set to fixed value 0xF0 (No Layer 3 implemented).

5. The info field carries information from ground station to a specific system or service on satellite and vice versa. It has maximum length of 256 bits. This length constraint applies before insertion of zero bits (bit stuffing). In our case it contains satellite bus communication protocol frame.

6. The 16-bit frame check sequence is used to ensure that the frame is received without corruption. The field is calculated according to ISO 3309 Recommendations. The polynom for FCS calculation is as follows:

$$FCS = X^{16} + X^{12} + X^5 + 1$$

The FCS field is computed over Address, Control, PID and Information fields. Initial value of calculation is 0xFFFF and result of division is XOR-ed with final value 0xFFFF. All data bytes are transmitted LSB first except 16-bit long FCS witch is transmitted MSB first.

There are few common ways how AX.25 frames are transmitted over the air [12]. One way to do it is using G3RUH FSK [13]. This is normally used for a rate of 9600 baud on the UHF bands. The HDLC bit stream is scrambled using a multiplicative scrambler with polynomial $1+x^{12}+x^{17}$ before NRZ-I encoding is done (Fig 3.3). The NRZ-I signal is then shaped (low-pass filtered) and used to drive an FM modulator directly, in order to produce an FSK signal [12]. Other sources claim that NRZ-I encoding is done before scrambler [14 p. 36]. Last approach seems more common so it will be used.

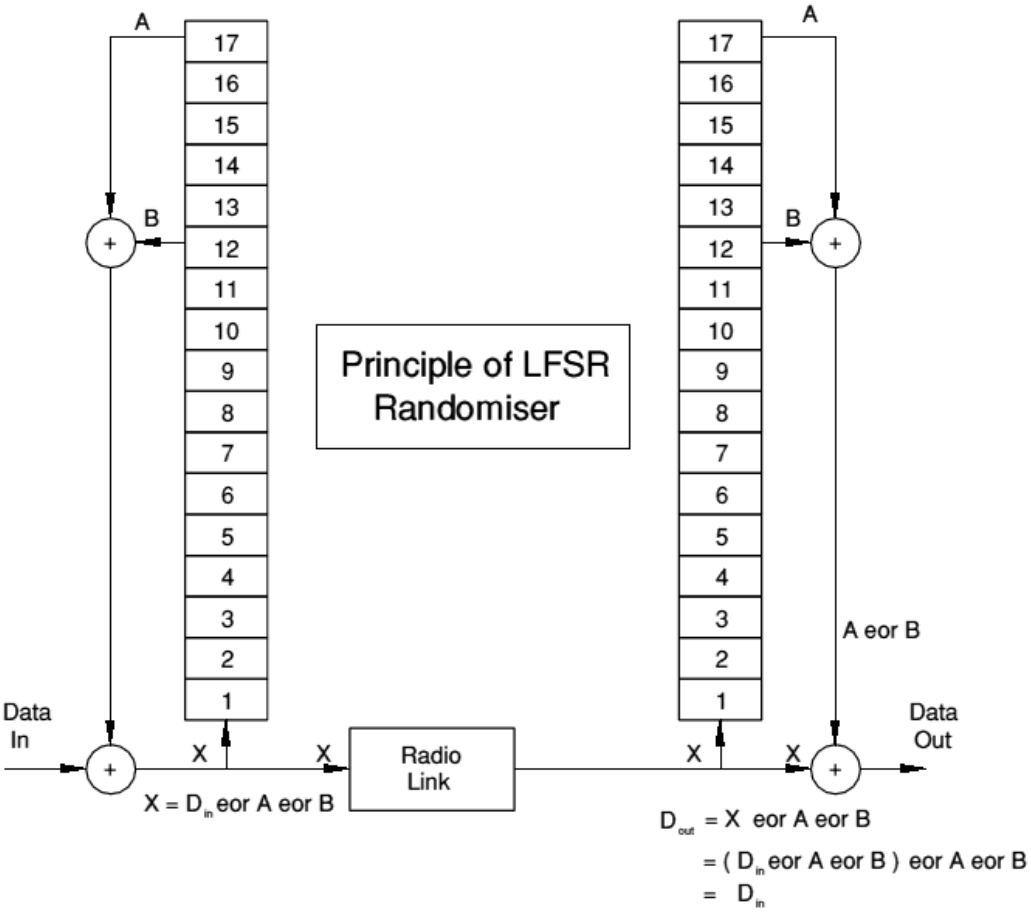

Fig 3.3 Multiplicative scrambler and descrambler [13]

Table 3.1 contains (initial) values of fourteen address Bytes (octets). It is assumed that call sign of ground station is ES1ZW and call sign of satellite is ES1W/S. Satellite is assumed to reply to command received from GS, indicated by Command/Response bit (0/1). Address extension bit, being lowest in each octet, is 1 only at last octet (A14), indicating thus the end of address field.

Table 3.1 Address field of satellite transmitted frame

Octet	ASCII	Binary	Hex
A1	E	10001010	0x8A
A2	S	10100110	0xA6
A3	1	01100010	0x62
A4	Z	10110100	0xB4
A5	W	10101110	0xAE
A6	space	00100000	0x40
A7 (SSID)	none	01100000	0x60
A8	E	10001010	0x8A
A9	S	10100110	0xA6
A10	1	01100010	0x62
A11	W	10101110	0xAE
A12	/	01011110	0x5E
A13	S	10100110	0xA6
A14 (SSID)	none	01100001	0x61

After fourteen octet long address field a control field with fixed value of 0x03 comes. Immediately after that fixed PID field with value 0xF0. Then SBCP frame with length of six to 47 data bytes and finally end flag 0x7E.

It is not enough for radio packet to contain only scrambled and bit-stuffed AX.25 packet in NRZ-I format. Additional fields (see figure 6.3) must be inserted in order to synchronize receiver clock, indicate beginning of frame to Si4468 transceiver IC and to initiate scrambler shift register.

Preamble	**Sync Word**	**Packet length**	**Scrambler sync**	**AX.25 frame(s)**
8 x 0xAA	0x7C 0x56	2 bytes	At least 3 bytes	

Figure 6.3 Additional fields of radio packet

Raadiokaadri moodustamine:

Satelliidi sisese siini kaadri formaat on kujutatud joonisel 1.1. Kaader koosneb alguse lipust (*Flag*) 0x7E, kaadri pikkuse väljast (*Len*) , kuni *N* = 256 baidist andmetest (*Data*) ja kahebaidisest kontrollsummast FCS.

Flag	Len	Data	FCS
0x7E	2 B	*N* B	2 B

Joonis 1.1 Satelliidi sisemise siini kaadri struktuur

Raadio toimib lüüsina, seega eemaldatakse enne andmete (*Data*) eetrisse saatmist kaadrist alguse lipp ning pikkuse ja kontrollsumma väljad, ehk raadio teel kantakse üle ainult *N* baiti, kus *N* väärtus on vähemalt neli ja mitte suurem kui 256 baiti ($4 \leq N \leq 256$).

Esmalt lisatakse raadio poolt edastatavatele andmetele AX.25 aadressi, CONTROL ja PID väljad, kokku 16 baiti, tulemus on kujutatud joonisel 1.2. Üle kõigi nelja välja arvutatakse vigade tuvastamiseks kahebaidine kaadri kontrollkood FCS. Viimane edastatakse baithaaval kõrgeima tähtsusega bitist alustades (*MSB-first*). Kõik ülejäänud baidid esitatakse vastupidises järjekorras (*LSB-first*).

Address	Control	PID	Info (Data)
14 B	0x03	0xF0	*N* B

Joonis 1.2 Raadiokaadri moodustamise esimene samm

Kontrollkood lisatakse joonisel 1.2 kujutatud struktuuri lõppu ja seejärel teostatakse üle kogu struktuuri farssbittide lisamine (*bit stuffing*). Kontrollkoodiga ja farssbittidega kaadri struktuur on toodud joonisel 1.3. Antud joonisel toodud väljade suurused vastavad andmetele ilma farssbittideta. Halvimal juhul võib kaadri pikkus nende tõttu olla kuni 20% suurem.

Address	Control	PID	Info (Data)	FCS
14 B	0x03	0xF0	*N* B	2 B

Joonis 1.3 Raadiokaadri moodustamise teine samm

Järgmise sammuna lisatakse kaadri alguse lipp 0x7E ja kaadri lõppu omana bitid 0111 1110. Bititäitmise tõttu ei pruugi edastatavate bittide hulk anda täisarvu baite, sellisel juhul lisatakse pärast kaadri lõppu niipalju nullibitte, kui on vaja kaheksaga jaguva tulemuse saamiseks.

Flag	Address	Control	PID	Info (Data)	FCS	Flag
0x7E	14 B	0x03	0xF0	*N* B	2 B	0x7E

Joonis 1.4 Raadiokaader enne skrämblerit

Järgnevalt teisendatakse andmed NRZL formaadist (biti väärtus „1“ – kõrge, „0“ – madal) ümber NRZI formaati, kus biti väärtusele „0“ vastab nivoo muutus ja väärtusele „1“ nivoo püsimine eelmise väärtuse juures. Eelmise väärtuse algväärtuseks on „0“. Seejärel läbib kaader multiplikatiivse skrämbleri genereeriva polünoomiga $1 + x^{12} + x^{17}$.

Nüüd on kaader valmis ja see edastatakse 3 kHz deviatsiooniga FSK modulatsiooni kasutades, kiirusega 9600 bitti/s kandesagedusel 435,45 MHz üle raadioeetri.

Enne AX.25 kaadri algust saadab raadio välja preambuli, mis koosneb järgnevatest osadest. Esmalt saadetakse 64 biti pikkune vahelduvatest „1“ ja „0“ koosnev sünkrojada (8 x 0xAA), viimane on vajalik vastuvõtja sünkroniseerimiseks. Järgmisena tuleb kahebaidine sünkrosõna (*Sync Word*) 0x7C 0x56,

mis on vajalik ainult juhul, kui kaader edastatakse Si446x tüüpi raadiokivile vastuvõtmiseks. Samas ei sega see teiste vastuvõtjate tööd, kuna paikneb enne kaari alguse lippu. Järgmine kahebaidiline väli (*Packet length*) on samuti vajalik Si446x vastuvõtja jaoks. Siiamaani kirjeldatud esimesed 12 baiti edastatakse otse, ilma bititäitmise ja skrämbleerimiseta. Järgmised kolm baiti (*Scrambler sync*) on vajalikud vastuvõtja skrämbleri algväärtustamiseks ja sellel järgneb juba joonisel 1.4 toodud struktuuriga skrämbleeritud AX.25 raadiokaader.

Preamble	**Sync Word**	**Packet length**	**Scrambler sync**	**AX.25 frame**
8 x 0xAA	0x7C 0x56	2 bytes	At least 3 bytes	

Joonis 1.5 Raadiokaadri preambuli struktuur

Kaadri vastuvõtul teostatakse kirjeldatud protseduurid vastupidises järjekorras.

Appendix A. Satellite Bus Communication Protocol frame checksum computation.

Given example shows how to compute IFCS for a two byte frame consisting of 0x03 and 0x3F. Binary representation of given bytes is: 0000 0011 0011 1111. As all bytes are transmitted LSB bit first, then also CRC is calculated over flipped version: 1100 0000 1111 1100.

Generator polynomial $P(x)$ of degree 16 is having the form $P(x) = x^{16} + x^{12} + x^5+1$. In short it can be also written down as hexadecimal number 0x1021. Initial reminder of IFCS is all ones or 0xFFFF. So polynomial $G(x)$ to be divided is initially: 1100 0000 1111 1100 1111 1111 1111 1111 and generator polynomial $P(x)$ is: 1 0001 0000 0010 0001.

Result of division is 0x25C8 (0010 0101 1100 1000). Result will be bitwise inverted (XOR-ed with 0xFFFF) resulting in 0xDA37 (**1101 1010 0011 0111**).

IFCS is sent, unlike the rest of data, MSB first, so frame to be sent will be following:

0111 1110 1100 0000 1111 1100 1101 1010 0011 0111 0111 1110

or in hexadecimal form as: 0x7E 0xC0 0xFC 0xDA 0x37 0x7E

As EUSART module hardware is built so, that sends it's contents LSB first, then within memory all bytes can (and must) be in usual MSB- first order. Only exception being IFCS as it must be transmitted MSB-first then it must be stored LSB-first, because of EUSART module reverses bit order while transmitting and receiving. So example frame to be sent, must look inside transmit buffer like this:

0111 1110 0000 0011 0011 1111 0101 1011 1110 1100 0111 1110

or in hexadecimal form as: 0x7E 0x03 0x3F 0x5B 0xEC 0x7E

Appendix B. C code of IFCS calculation.

```
void crc(unsigned char *ptr, unsigned short nBytes, unsigned char *ptcrc,
unsigned short remainder)
// Calculates 16-bit CRC-CCITT
// Initial value of reminder is: 0xFFFF
// Generator polynomial is: 0x1021 (x^16+x^12+x^5+1)
// Reminder is XOR-ed with: 0xFFFF
{
    unsigned char inverted;

    for (unsigned short byte = 0; byte < nBytes; ++byte)
    {
        inverted = reverse(*ptr++); // Input bytes are reversed as
        remainder ^= inverted << 8; // transmission is carried out LSB-first
        for (unsigned char bit_nr = 8; bit_nr > 0; --bit_nr)
        {
            if (remainder & 0x8000)
                remainder = (remainder << 1) ^ 0x1021;
            else
                remainder = (remainder << 1);
        }
    }
    remainder^=0xFFFF;
    *ptcrc++ = reverse(remainder>>8); // CRC is reversed as it must be
    *ptcrc++ = reverse(remainder);    // transmitted MSB first
}

unsigned char reverse(unsigned char b)
/* Reverses bit order within byte b */
{
    b = (b&0xF0) >> 4 | (b&0x0F) << 4;
    b = (b&0xCC) >> 2 | (b&0x33) << 2;
    b = (b&0xAA) >> 1 | (b&0x55) << 1;
    return b;
}
```

Appendix C. C code of scrambler and sescrambler.

```
void NRZM_scrambler (unsigned char *inptr, unsigned char *outptr, unsigned
char nBytes)
/* Self synchronizing scrambler with polynomial 1 + x^12 + x^17
   Output of scrambler is in NRZM (NRZ-I) format
   NRZM: 1 - togle; 0 - constant.                                    */
{
    unsigned char inbyte, outbyte, y17, y12, outbit, last_bit = 0;
    unsigned short long shreg = SCR_SH_INIT;
                              //Initial state of shift register

    for (unsigned char byte = 0; byte < nBytes; ++byte)
    {
        inbyte = *inptr++;
        for (unsigned char bit_nr = 8; bit_nr > 0; --bit_nr)
        {
            y17 = (shreg & 0x010000) ? 1 : 0; // Reads value of 17th output
                                              // of shift register
            y12 = (shreg & 0x000800) ? 1 : 0; // Reads value of 12th output
                                              // of shift register
            outbit = 0x01&(inbyte >>(bit_nr-1)); //Reads input bit (bit_nr)
            outbit ^= (y17^y12);   // Adds values of 12th and 17th bits of
                                   // shift register to input (mod 2)
            shreg  <<= 1;       // Shift register shifts one bit to left
            shreg |= outbit;    // New value is written into shift register

            /** NRZM (NRZ-I) coding */
            if (outbit == 0)
                outbit = last_bit;
            else
                outbit = 0x01&(~last_bit);
            last_bit = outbit;
            outbyte <<= 1;    // Shifts output byte one bit left
            outbyte |=outbit; // Writes result into output byte
        }
        *outptr++ = outbyte;  // Writes scrambled and NRZ-I encoded
    }                          // byte to output
}


void NRZL_descrambler (unsigned char  *inptr, unsigned char  *outptr,
unsigned char  nBytes)
{
/* Self synchronizing descrambler with polynomial 1 + x^12 + x^17
   Input of scrambler is in NRZM (NRZ-I) format
   NRZM: 1 - togle; 0 - constant.
   Output of scrambler is in NRZL format
   NRZL: 1 - high level; 0 - low level.                              */

     unsigned char  inbyte, outbyte, y17, y12, outbit, inbit, last_bit = 0;
     unsigned short long shreg= SCR_SH_INIT;
                              // Initial state of shift register
```

```
        for (unsigned char byte = 0; byte < nBytes; ++byte)
        {
              inbyte = *inptr++;
              for (unsigned char  bit_nr = 8; bit_nr > 0; --bit_nr)
              {
               y17 = (shreg & 0x00010000) ? 1 : 0; // Reads value of 17th
                                                   // output of shift register
               y12 = (shreg & 0x00000800) ? 1 : 0;  // Reads value of 12th
                                                   // output of shift register
                  shreg  <<= 1;      // Shift register shifts one bit to left
                  inbit = 0x01&(inbyte >>(bit_nr-1));  // Reads value of
                                                      // input bit (bit_nr)
                  if (inbit == last_bit)
                  {
                     outbit = 0^y17^y12;// Adds values of 12th and 17th bits
                                        // of shift register to input (mod 2)
                     shreg |= 0;  // New value is written into shift register
                  }
                  else
                  {
                      outbit = 1^y17^y12; // Adds values of 12th and 17th
                                  // bits of shift register to input (mod 2)
                      shreg |= 1; // New value is written into shift register
                  }
                  last_bit = inbit;
                  outbyte <<= 1;    // Shifts output byte one bit left
                  outbyte |=outbit; // Writes result into output byte
              }
              *outptr++ = outbyte;  // Writes scrambled and NRZ-I encoded
        }                           // byte to output
}
```