# Communications Infrastructure for the MOST Microsatellite Project (excerpt)

Laura Halliday

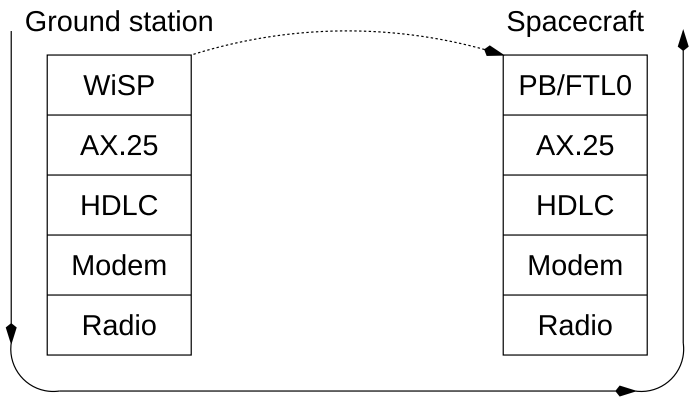

Figure 2.1: The layers of a satellite communication system

## 2.2 AX.25

Amateur digital communications, including amateur satellite communications, use the AX.25 protocol. This is a derivative of the standard X.25 protocol, with enhancements and adaptations for amateur radio use. As documented in its reference manual (1), AX.25 defines a vast protocol with sophisticated flow control and elaborate facilities for creating and managing connections. Amateur satellite communications use very little of this: with one exception they use one type of AX.25 packet, the UI (**U**nnumbered **I**nformation) packet, with all connection management and flow control handled by higher-level protocols. This results in exceptionally simple implementations: as part of this research the necessary subset of AX.25 was implemented in a few hours in Microsoft Visual Basic.

The exception to the use of UI frames is the FTL0 protocol for uploading files, which uses the connected-mode facilities of AX.25. FTL0 is discussed in Section 3.4.

AX.25 transmissions are organized in *frames*, where each frame is an individual datagram. The layout of an AX.25 information frame (there are other kinds, but they are not used in PACSAT communications) is shown in Figure 2.2.

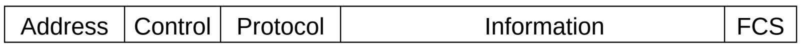

Figure 2.2: AX.25 information frame layout

The encoding of each field will be discussed in turn.

Table 2.1: Example of AX.25 address encoding

| **Character** | **Hex value** | **Shifted hex value** |
|---|---|---|
| V | `0x56` | `0xac` |
| A | `0x41` | `0x82` |
| 3 | `0x33` | `0x66` |
| S | `0x53` | `0xa6` |
| F | `0x46` | `0x8c` |
| L | `0x4c` | `0x98` |
| 0 | `0x00` | `0x60` |
| U | `0x55` | `0xaa` |
| O | `0x41` | `0x82` |
| S | `0x53` | `0xa6` |
| A | `0x41` | `0x82` |
| T | `0x54` | `0xa8` |
| 5 | `0x35` | `0x6a` |
| 11 | `0x0b` | `0x6c` |

The *Address* field specifies the origin and destination of the message. Addresses are encoded in ASCII, shifted one bit over to allow the least-significant bit to provide control information—it is set to one on the last byte of the last callsign in the field.

An example of address encoding for a packet sent from `UOSAT5-11` to `VA3SFL-0` is presented in Table 2.1. The encoding of the SSID (**S**econdary **S**tation **ID**entifier) is illustrated in Table 2.2. The address extension bit is adapted from HDLC (Section 2.3): it is `0` if there are more addresses in the frame, and `1` if the address is the last in the frame.

The AX.25 SSID performs the same function as ports under TCP, permitting a single station to have multiple presences on the network, possibly with different functions.

The *Control* field, one byte, is always `0x03` in the UI frames that constitute the vast majority of satellite traffic.

The *Protocol* field, more correctly, the Protocol ID field, is one byte in length and specifies if a higher-level protocol is in use. In PACSAT communications the value is almost invariably `0xf0`, meaning that no higher-level protocol is in use.

Table 2.2: AX.25 SSID encoding

| **Bit number** | **Item** |
|---|---|
| 7 | Command, always `0` |
| 6–5 | Reserved, always `11` |
| 4–1 | Numeric SSID |
| 0 | HDLC address extension bit |

The *Information* field contains whatever data the application wishes to send.

The FCS (**F**rame **C**heck **S**equence), also called the CRC (**C**yclic **R**edundancy **C**heck), provides a 16 bit checksum to maintain data integrity. The checksum calculations are based on the CCITT standard polynomial $g(x) = x^{16} + x^{12} + x^5 + 1$. The fading nature of the channel between the spacecraft and the ground station produces errors in bursts, and the performance of this and similar codes has been analyzed extensively (2) in such an environment. The code used by AX.25 can detect all error bursts of length 16 or less, 99.997% of all error bursts of length 17, and 99.9985% of all error bursts of length greater than 17.

## 2.3 HDLC

Like AX.25, HDLC (**H**igh-**L**evel **D**atalink **C**ontrol) defines an elaborate protocol. For satellite AX.25 purposes HDLC merely defines a *line code* (3). A line code alters the data stream in order to better adapt it to the transmission medium. In the present case HDLC provides three functions:

- NRZI differential encoding
- Bit stuffing
- Frame delimiting

HDLC is a stream-oriented protocol, and there is no notion of grouping individual bits to form bytes. In the discussion of HDLC the term “octet” will be used only as a grouping of 8 bits, with no implication that the grouping is significant to a higher-level protocol.

### 2.3.1 NRZI differential encoding

NRZI (**N**on-**R**eturn to **Z**ero **I**nverted) encoding provides differential encoding of the bit stream that is to be transmitted. Differential encoding encodes data in the *difference* between successive bits, rather than in the bits themselves. This is a convenience in some of the modulation formats used in satellite operations, like MSK (Section 2.5.4), but is mandatory in other formats like BPSK (Section 2.5.3).

In NRZI encoding a zero bit is transmitted by a change in the output, while a one bit is transmitted by no change in the output. This is illustrated in Figure 2.3. Note that the absolute value of the output is irrelevant: it is only *changes* that convey information. The two possible output waveforms are thus equivalent.

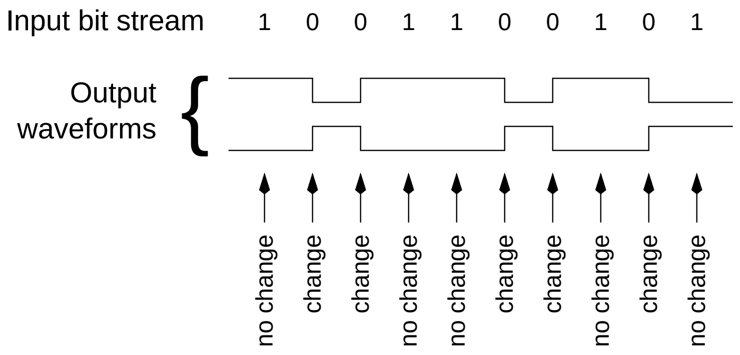

Figure 2.3: NRZI encoding

### 2.3.2 Bit stuffing

A weakness of NRZI over radio channels is that if there are too many one bits in a row, a DC component will be necessary for transmission—in systems whose bandwidth does not (and often cannot) extend to DC. Additionally, recovering the timing of individual bits—easy for zero bits under NRZI—can become unreliable when an extended series of one bits is transmitted. The solution adopted by HDLC and other stream-oriented data transmission schemes like USB (4) is *bit stuffing*: if too many bits of one kind appear in a row, insert a bit of the other kind on transmission, and remove it on reception. This is illustrated in Figure 2.4.

Under HDLC only five consecutive one bits may occur in outgoing data. Should a sixth one bit occur, a zero is inserted on transmission and removed on reception.

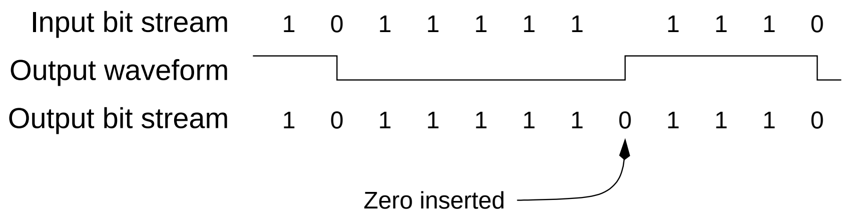

Figure 2.4: HDLC bit stuffing

### 2.3.3 Frame delimiting

HDLC indicates the beginning and end of a frame with a bit pattern that is not permitted in user data: the octet `0x7e`, with the otherwise-forbidden bit pattern `01111110`, which is not subject to bit-stuffing. By convention an idle HDLC channel sends consecutive frame characters, as illustrated in Figure 2.5.

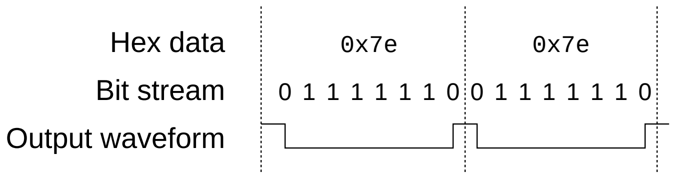

Figure 2.5: Bit patterns and waveforms on an idle HDLC channel

### 2.3.4 Polynomial scrambling

While not part of HDLC—it is, typically, a modem function—the application of polynomial scrambling to transmitted data is closely related to the functions provided by HDLC.

The process of scrambling the data enhances its transmission in several ways:

- An increased density of transitions further eases timing recovery.
- An increased density of transitions further reduces the low frequency bandwidth requirements of the system.
- The pseudo-random nature of the scrambled data renders the transmitted spectrum noise-like, with no spectral lines that could interfere with other services in shared spectrum allocations.

The standard 9600 baud analogue MSK modem was designed by Miller in the 1980s (5) and uses polynomial scrambling. Miller's design, often referred to by his amateur radio callsign G3RUH, uses the scrambling polynomial $1 + x^{12} + x^{17}$, where each output bit is computed from the current bit and the bits transmitted 12 and 17 bits ago, as illustrated in Figures 2.6 and 2.7.

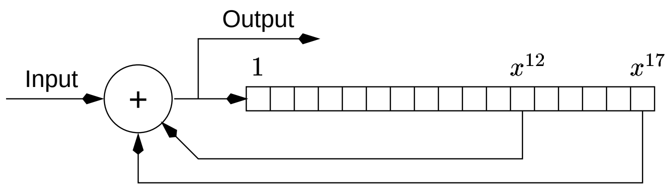

Figure 2.6: Polynomial scrambler in the G3RUH modem

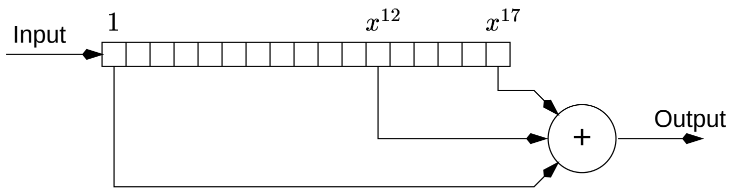

Figure 2.7: Polynomial descrambler in the G3RUH modem

The original implementation was built with discrete logic shift registers and exclusive-OR gates. Modern implementations use programmable logic (6), or perform these operations in software (7).

The scrambler and descrambler are closely related: the scrambler divides the bit sequence by the polynomial, while the descrambler multiplies by the same polynomial. In theory either circuit can serve as the scrambler with the other as the descrambler, but in practice the feedback division circuit is only used in the scrambler, because if it was used in the descrambler an erroneous received bit could circulate indefinitely. The circuit in Figure 2.7 will thus only output three corrupt bits on reception of a single erronoeus bit. This is acceptable in the AX.25 environment where packets are, in any case, rejected for a single detected bit error.

## 2.4 KISS

KISS (**K**eep **I**t **S**imple, **S**tupid) is a means of encapsulating AX.25 datagrams for transmission over serial lines. It was devised as an interim means for implementing new protocols in amateur digital radio systems (8).

Table 2.3: KISS command functions

| **Hex value** | **Function** |
|---|---|
| `0x00` | **Data frame** |
| `0x01` | **Transmitter keying delay** |
| `0x02` | Persistence |
| `0x03` | Slot interval |
| `0x05` | **Full duplex** |
| `0xff` | **Exit KISS mode** |

When radio amateurs first started experimenting with packet radio the available computers had limited capability and were not able to implement the AX.25 protocols in their own software. The solution was a separate, dedicated external device called the TNC (**T**erminal **N**ode **C**ontroller) that included protocol and modem functionality.

This approach worked, but it was inflexible: since the protocol was fixed in the TNC firmware, it was impossible to experiment with different protocols. The solution was to bypass the protocol functions of the TNC and only use the modem. The protocol devised to communicate between more-powerful host computers and the TNC, with much of its functionality now disabled, was KISS. Many modern implementations carry this progression further and perform *all* their processing in software, requiring little more than a sufficiently powerful host computer with a sound card.

KISS was modelled on the TCP/IP standard SLIP (9), with enhancements for amateur radio use.

A frame transmitted by KISS is a complete AX.25 frame (Figure 2.8) without the checksum or HDLC encoding. On transmission the TNC will compute the CRC and perform HDLC encoding. On reception it is the task of the TNC to remove the HDLC encoding and validate the checksum before making the frame available to the host.

| | | | |
|---|---|---|---|
| Frame | CMD | AX.25 datagram | Frame |

Figure 2.8: The layout of a KISS frame

The KISS command function byte is provided so that the host computer may give instructions to the TNC. The standards on this are loose, but the minimal requirements in the standard are listed in Table 2.3. Only the **bold face** commands are relevant to satellite operations; the others configure the system for terrestrial operation.

Table 2.4: KISS character usage

| **Function** | **Hex value** | **Mnemonic** |
|---|---|---|
| Frame end | `0xc0` | `FEND` |
| Frame escape | `0xdb` | `FESC` |
| Transposed frame end | `0xdc` | `TFEND` |
| Transposed frame escape | `0xdd` | `TFESC` |

KISS reserves several characters for its own purposes (Table 2.4). Key is the frame end character `FEND`, which delimits frames. Should the bit pattern corresponding to `FEND` occur in data it is replaced by the combination of the frame escape character `FESC`, followed by the transposed frame end character `TFEND`. Similarly, should `FESC` occur in the data, it is replaced by the combination of `FESC` followed by the transposed frame escape character `TFESC`.

A program that wishes to receive and process KISS transmissions must thus initially wait for an `FEND` character. If the following character is anything other than an `FEND` character it begins to accumulate a frame; otherwise, it ignores consecutive `FEND`s until a frame begins. The program handles the command function code as necessary and then proceeds to accumulate characters until the frame is complete, as signalled by a second `FEND`. During this processing it must recognize and process escape sequences. The frame is now complete: the checksum is validated and the frame is handed to the host for processing.

This type of "multi-mode" processing is routinely implemented with a *finite state machine*, which switches processing modes to follow the protocol. The finite state machine used for processing KISS frames is illustrated in Figure 2.9. Several of the programs described in Appendix C use this state machine, implemented in a routine called `Protocol()`.

## 2.5 Modems & Modulation

The function of a *modem*, short for **mod**ulator-**dem**odulator, is to convert digital signals into an analogue form that may be transmitted over a communications channel: in this case, a radio system.

The feature that drives many modem designs and implementations is bandwidth, and this will be discussed in some detail. Several modulation schemes presently in use on amateur satellites will be discussed and their features will be compared.

## 2.7 Notes and References

1. *AX.25 Link Access Protocol for Amateur Packet Radio, Version 2.2.* Newington: ARRL and Tucson: TAPR. 1997.
   **Internet:** `http://www.tapr.org/tapr/pdf/AX25.2.2.pdf`

2. S. Wicker, *Error Control Systems for Digital Communication and Storage.* Upper Saddle River: Prentice Hall, 1995.

3. T. McDermott, *Wireless Digital Communications: Design and Theory.* Tucson: TAPR. 1996.

4. *Universal Serial Bus Specification,* Revision 1.1, 23 September 1998.
   **Internet:** `http://www.usb.org/developers/data/usbspec.zip`

5. J. Miller, “9600 Baud Packet Radio Modem Design”, paper presented at ARRL 7th Computer Networking Conference, 1988.
   **Internet:** `http://www.amsat.org/amsat/articles/g3ruh/a109.txt`

6. *Spirit–2 Technical Reference Manual and Interfacing Instructions,* 3rd edition. Tampa: Pac-Comm Packet Radio Systems Inc. 1995.

7. T. Sailer, “DSP Modems”, paper presented at 11. Internationale Packet-Radio Tagung, Darmstadt, Germany, 1995. In German.
   **Internet:** `http://www.ife.ee.ethz.ch/~sailer/ham/ham.html`.

8. M. Chepponis and P. Karn, “The KISS TNC: a Simple Host-to-TNC Communications Protocol”, paper presented at ARRL 6th Computer Networking Conference, 1987.
   **Internet:** `http://people.qualcomm.com/karn/papers/kiss.html`

9. J. Romkey, *A Non-standard for Transmission of IP Datagrams over Serial Lines: SLIP.* RFC–1055, June 1988.
   **Internet:** `http://www.ietf.org/rfc/rfc1055.txt`

10. E. Lee and D. Messerschmitt, *Digital Communication,* 2nd edition. Boston: Kluwer Academic Publishers, 1994.

11. S. Haykin, *Communication Systems,* 3rd edition. New York: John Wiley & Sons. 1994

12. F. Kostedt and J. Kemerling, *Practical GMSK Data Transmission,* Winston-Salem: MX·COM, 1998.
    **Internet:** `http://www.mxcom.com/app_notes/apgmskr2.pdf`

13. S. Pasupathy, “Minimum shift keying—a spectrally efficient modulation”, *IEEE Communications Magazine*, 17:4, 1979.

14. A. Jones and and J. Gardiner, “Generation of GMSK using direct digital synthesis” in *IEE Colloquium on Implementations of Novel Hardware for Radio Systems*, 1992.

15. J. Miller, “The Shape of Bits to Come”, paper presented at ARRL 10th Computer Networking Conference, 1991.
**Internet:** `http://www.amsat.org/amsat/articles/g3ruh/a108.txt`

16. K. Murota and K. Hirade, “GMSK Modulation for Digital Mobile Radio Telephony”, *IEEE Trans. on Communications*, vol. COM–29:7, July 1981.

17. R. de Buda, “Coherent demodulation of frequency shift keying with low deviation ratio” in *IEEE Trans. on Communications*, vol. COM–20:6, June 1972.

18. M. Hodgart and J. Schoonees, “A Robust MSK Demodulator Through DSP” in *Proceedings of the 1992 South African Symposiom on Commmunications and Signal Processing*, 1992.