PACSAT Data Specification Standards

Harold E. Price, NK6K
Jeff Ward, G0/K8KA

ABSTRACT

This document provides a standard way of describing PACSAT data formats in specifications, and provides certain assumptions for implementors.

Purposc

This document describes the standard format for PACSAT data.

Background

This standard is based on the following assumptions:

1) The spacecraft are the critical resources in the PACSAT/groundstation network. If a particular data representation can conserve memory space and CPU cycles in the spacecraft, all other items being equal, the representation that favors the spacecraft should take precedence.

2) The UoSAT and the AMSAT-NA PACSAT hardware are based on an Intel 80186-compatible device. Therefore, all internal multi-byte numeric data is stored with the least-significant byte in low-order memory.

3) The UoSAT and the AMSAT-NA PACSAT software is largely based on the Microsoft C programming language.

4) The UoSAT and the AMSAT-NA PACSAT software development systems are based on IBM **PCs** or compatibles.

Discussion

The primary decision to be made in PACSAT data formats is "big endian" (BE) vs. "little endian" (LE). Most network standards are defined as BE, meaning the Most Significant Byte (MSB) of multi-byte data appears in low order address space, and the Least Significant Byte (LSB) appear in high order memory. The UoSAT and Microsat spacecraft all use Intel 80186 or compatible CPUs, which store data with the LSB first, and are LE.

Multi-byte data appears in many places in PACSAT data? including the file headers and the control structures of the broadcast and FTL0 protocols. If these protocols were BE, the spacecraft software would need to swap byte order in several places. Whether done as in-line code or as function calls, these conversions use both CPU cycles and code space. It is clear that a native data representation will result in a more efficient utilization of the spacecraft CPU, and that the data format conversions, if any, should be done on the ground. Experimentation was done showing that avoiding byte swapping on thc spacecraft resulted in significant space savings.

This will not affect the actual high-level software code, as prudent programmers who wish to write transportable code that is applicable to BE and LE hosts will use macro calls to swap the byte order when moving data from an external source to local variables. By using the somewhat less common LE in the protocol specification, the macro will bc active on **BE** systems when it would normally be active on LE systems. In any case, the macros would still be present in the source file.

For example,

fnum = NETSWAP32(broadcast head.fnum):

would be the line of code to read in the file number from a broadcast protocol frame. This code will be the same no matter which order the protocol required the 4-byte integer field to be in.

Taking these assumptions into account, the standard to be used when defining data exchange formats between PACSAT and a ground station are as defined below.

Intended Applicability

This document is primarily intended to apply to shared file formats, such as the standard PACSAT File Header; and to PACSAT specific protocols such as the PACSAT Broadcast Protocol. It is not meant to infer that existing protocols, such as IP, are to have integers byte-swapped when transmitted to a PACSAT.

PACSAT Data Structure Specification Standard

1) All structure definitions in PACSAT standards documents should provide C structures wherever possible to describe data formats.

2) All structures are assumed to be packed; do not assume slack bytes are provided to align words and doublewords.

3) All multi-byte numeric data is assumed to be stored and transmitted with the Least Significant Byte first.

4) Where it is necessary to number bits, the least significant bit is zero.

5) The standard method for referring to hexadecimal constants will be the C standard Oxhh.

6) The assumed length of an unsigned or int type is 16 bits.

7) The "left" end of a string is stored and transmitted first.

8) "ASCII" characters are the printable ASCII characters 0x20-0x7f.

9) Times are represented by the UNIX 4-byte unsigned integer counting the number of seconds since 0000 UTC 1 January, 1970.