STUDENT NITRIC OXIDE EXPLORER (SNOE) TELEMETRY AND FLIGHT SOFTWARE DESIGN

Mark A. Salada*, Randal L. Davis†
Laboratory for Atmospheric and Space Physics
University of Colorado
Boulder, Colorado
Phone: (303) 492-5208
Fax: (303) 492-6444
E-mail: salada@mirza.colorado.edu

Abstract

This paper describes the telemetry design and the flight software design for the Student Nitric Oxide Explorer (SNOE) satellite. Sponsored by NASA and the University Space Research Association (USRA) under the Student Explorer Demonstration Initiative (STEDI), the SNOE satellite is being designed, built and operated by the University of Colorado's Laboratory for Atmospheric and Space Physics (CU/LASP), with substantial involvement of University students. It will be launched into a sun-synchronous 550 km orbit in March 1997. An early decision for the SNOE project sets forth a packetized telemetry data scheme based upon the recommendations by the Consultative Committee for Space Data Systems (CCSDS). The CCSDS is an international organization composed of representatives from the world's space agencies. Compared to a traditional time-division multiplexed telemetry system, packetized telemetry systems provide substantial design freedom: any number of different packets can be developed to hold the spacecraft's science, engineering and ancillary data. Generating packets sometimes involves complex sets of functions that are, just as all functions performed by the SNOE software, grouped into software modules. Since the SNOE software handles functions traditionally implemented in hardware, the software modules are called from a main loop that is rigorously deterministic in its timing.

I. SNOE Telemetry

SNOE Physical Communication Channels

There are two telemetry channels from the SNOE satellite. The first is a low-rate (512 bits per second) channel that will primarily be used for initial acquisition and trouble-shooting. The channel has a safety factor of approximately 9 dB, providing a communication link in even the most extreme conditions. The 512 bps bandwidth channel has room for only real-time data, primarily engineering telemetry, leaving science data and playback data for the high-rate channel. Even with the limited bandwidth, it is still possible to transmit both engineering and science data in real-time on the low-rate channel. The second telemetry channel is a high-rate communication channel that transmits at 128k bps. This channel has the bandwidth capacity to transmit engineering, science and playback data without concern. Both channels are NASA compatible PCM/PSK/PM.

SNOE Usage of CCSDS Recommendations

The choice to comply to CCSDS packet telemetry recommendations[1] derives partly from a mission requirement to be NASA compatible during flight. This requirement provides the added security of the already existing NASA ground station coverage for initial acquisition and any trouble-shooting necessary — a feature beneficial to any satellite regardless of size. Replacing the Time-Division Multiplexing (TDM) telemetry format with packets simplifies the ground station engineering/science separation while providing compatibility with new NASA ground stations. Packetized telemetry also suits the science mission very well.

Both telemetry channels utilize fixed-length transfer frames with variable-length source packets inside the frames. The transfer frame lengths on each channel are not equal, since each is derived from transmission rate

* Graduate Research Assistant
† Mission Operations and Data Information Systems Division Director

and desired frame update rate. The desired real-time frame update rate is nominally 2 seconds, a value based solely on the desires of mission operators. This equates to a frame length of 1024 bits on the low-rate channel. Frame length selection on the high-rate channel, discussed below, is more involved. Simple CRC error checking is performed on a frame-by-frame basis, as neither telemetry stream is Reed-Solomon encoded. Additionally, SNOE will not utilize the optional transfer frame secondary header. The transfer frame overhead then equates to a header of 48 bits, a 32-bit trailer for a Frame Acceptance and Reporting Mechanism (FARM)[2], and a 16-bit CRC code, for a total of 96 bits. This leaves a 464 bps real-time bandwidth for data on the low-rate channel.

Originally, SNOE designers intended to utilize the high-rate channel solely for playback data. The bandwidth analysis for the high-rate channel would then be identical to the low-rate. However, if the SNOE team used the low-rate channel as the only means to a real-time communication link, telemetering anything but engineering data on the low-rate channel would effectively choke the bandwidth and cause severely increased engineering data update delays for the ground crew. In other words, the operations team would have to settle for either engineering data only, science data with limited engineering data, or neither during a memory dump. Clearly, the limited bandwidth on the low-rate channel complicates the downlink; therefore, the SNOE team has 'stolen' part of the high-rate channel for real-time communication use.

Real-Time Bandwidth Expansion

Telemetry design for the high-rate channel is not as straightforward as design for the low-rate channel. If the SNOE team proceeded with the original design, frame update rate would not be an issue as there would be no real-time frames on the high-rate channel. Yet the need for additional real-time bandwidth is apparent. The SNOE team therefore chooses to multiplex real-time frames with playback frames on the high-rate channel. The question then becomes one of choosing the appropriate frame lengths and multiplex ratio. For simplicity, the playback and real-time frame lengths are chosen to be identical. Also for simplicity, the playback frame length is chosen to be about as large as the largest Source Packet (discussed below). This yields a frame length of 4096 bits. A 1/64 multiplex ratio accomplishes the desired 2 second real-time frame update rate. Fig. 1 illustrates the high-rate channel frame multiplexing. Now performing the bandwidth analysis as on the low-rate channel, the real-time data rate on the high-rate channel is 2000 bps, approximately four times greater than

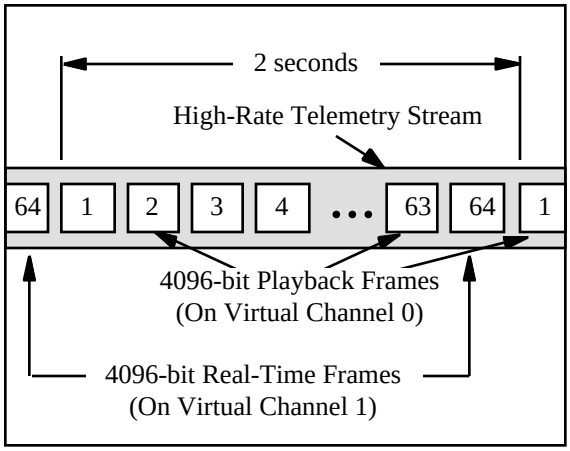

Fig. 1 High-Rate Channel Frame Multiplexing

with the low-rate channel alone. Also, the 1/64 frame multiplex ratio does not significantly impact the playback bandwidth, leaving approximately 123k bps. Clearly, utilization of the high-rate channel for real-time communication has expanded the SNOE real-time data capacity, allowing simultaneous engineering, science, memory dump, and playback data transmission.

The next question is a concern for separation of the playback frames from the real-time frames on the ground. A built-in CCSDS packet telemetry feature called "Virtual Channels" solves the ground separation problem. By Assigning different virtual channel ID's to playback and real-time frames, any ground service compatible with CCSDS packet telemetry can route any virtual channel or frame to the end users of those data. In this case, the real-time virtual channel is sent to the spacecraft command and control center while the playback channel is stored for later processing.

SNOE Source Packets

The available bandwidth for each channel is used to hold flexible data structures called "Source Packets." The SNOE telemetry system allows for packets to overlap frame boundaries in both the low-rate and high-rate channels. This unconventional design decision maximizes the bandwidth

use, eliminating the so-called 'slop,' or unused portion of a transfer frame. Most telemetry systems steer clear of this technique due to the fact that a loss of a single frame may affect data in adjacent frames. The SNOE team can afford this risk since loss of an occasional piece of data does not cripple the SNOE science objectives. This is a classic example of a small satellite trade-off that the SNOE engineering team has used to its advantage.

The flexible CCSDS telemetry system allows for customizing telemetry overhead to a certain degree. The source packet secondary header is an optional field in the source packet whose contents are user defined. The SNOE team uses the on-board timing reference, a 32-bit Vehicle Time Code Word (VTCW) as the secondary header, effectively time-tagging the construction of every source packet on-board. All source packets therefore contain a 48-bit primary header (as specified by CCSDS standards), a 32-bit VTCW, and finally, a variable length data field. The SNOE team divides all on-board telemetry into two categories: pre-defined packets and programmable packets.

Pre-Defined Telemetry Packets

Of the total 11 source packets designed for the SNOE mission, 9 are pre-defined. Four science packets and 5 engineering packets compose the primary telemetry structure. There are four science instruments aboard the SNOE spacecraft. Each instrument generates the data portion of a source packet, whose headers and additional information are then appended by the flight software. The engineering packets are constructed in an identical manner, with the exception of the flight software also taking responsibility for the contents of the data field.

Engineering packet design centers on telemetry sample frequency and availability. The SNOE satellite is a spinning satellite; many telemetry items on-board need updating only as fast as the spacecraft spin rate. Therefore, a packet is generated by the flight software once every revolution, updating all telemetry items only as fast as needed. The remaining engineering packets have been designed in a similar manner, with the exception of the Memory Dump packet whose contents mimic science packets. Table 1 lists the engineering packets with a description of the items within each. There are essentially three engineering packets whose contents comprise all engineering telemetry items on-board. The remaining engineering packets are simply reorganized duplicates for particular anticipated telemetry modes. As implied, all packets listed are pre-defined packets whose contents cannot be changed without re-programming in flight.

Packet Name	Contents	Update Period
Communication Status	Command verification and comm. link status	2 sec.
Subsystem Status	ADCS, instrument status, C&DH parameters.	Spin Period
General Engineering	Temps, power, C&DH parameters	60 sec.
Critical Engineering (Contains duplicate items from main three engineering packets for low-rate channel science telemetry mode)	Minimal command verification, comm. link status, power, ADCS, and C&DH parameters.	6 sec.

Table 1 Packet Contents

Programmable Telemetry Packets

Although a term typically viewed with trepidation, SNOE uses 'programmable' telemetry: items in some packets can be *selected* in a custom manner during flight. There are two such packets on-board the spacecraft: the Programmable Rapid Sample Packet and the Programmable Engineering Packet. Each telemetry item on-board is associated with an ID. These ID's are then used to populate the programmable packets through ground commands. Both packets limit the total number of items in each, but neither pre-define the contents of any 'slot' within a packet. The Rapid Sample Packet allows operations personnel to specify which telemetry ID's, and hence which telemetry items, to sample as fast as 10 Hz. Due to the high frequency capability, only four items are allowed at a time. When commanded, the flight software will then sample the four telemetry items and place ten consecutive

samples of each in the packet for transmission. The Programmable Engineering Packet allows for a larger set of items (up to 28) at frequency ranges much lower than the Rapid Sample Packet. These packets provide a flexible means of de-bugging problems on the spacecraft, as well as allowing for telemetry arrangements not anticipated by the pre-defined packets.

II. SNOE Flight Software

SNOE Flight Software Environment

Development Environment

The SNOE flight software will be developed using Borland C++ compiler, linker, and de-bugger on a 386 personal computer. An additional set of hardware that mirrors the flight environment is used to perform all testing prior to integrated testing with the flight computer and spacecraft. This additional set of hardware consists of an Intel 186 processor, referred to as the evaluation board, with complete memory and interface electronics. A simple RS-422 serial digital interface connects the development 386 personal computer and the evaluation board to the flight computer for loading code. Vendor supplied monitor programs on both the evaluation board and the flight computer provide the necessary de-bugging interface for testing. The code will be developed on the 386 PC, tested on the evaluation board, and finally loaded on the flight computer. A separate UNIX workstation using RCS, a standard revision control utility, maintains a copy of all current versions of code in order to control the flight software development. With a small development team, LASP does not expect security issues to impede the development, yet access to released code is strictly limited. The SNOE flight software team consists of a LASP professional, a graduate research assistant, and two or more undergraduate students as programmers.

Flight Environment

Small satellites often cannot afford the extensive effort required to develop a custom flight computer. The commercial-off-the-shelf (COTS) solution usually provides a less than perfect fit to a spacecraft, unless the commercial product allows for some custom expansion. The SNOE team is using and customizing a computer from Southwest Research Institute. The SC-4A Space Flight Computer is a 80C186 based computer with a 10 MHz clock frequency and the capability for bus expansion. The flight computer comes equipped with up to 24 Mbytes of mass data storage, 512 Kbytes static RAM, and 256 Kbytes EEPROM, all error detected and corrected. The SNOE team only utilizes 8 Mbytes of the mass data storage. LASP has also developed a custom daughterboard for interface between the spacecraft and the flight computer. LASP has chosen not to use the available 80C187 co-processor. Standard communication interfaces are built-in.

The Main Loop

The ‘main loop’ is simply a set of functions that repeats indefinitely. The SNOE flight software’s main loop is unique, however, in that it performs some functions not typically associated with software. For example, the SNOE flight software handles collection of telemetry where a hardware Digital Telemetry Unit (DTU) is traditionally used. This is a common small satellite approach due to the enormous expense of hardware development. Since this function requires rigorous timing, software that executes functions in a well defined manner is desirable. The key to good flight software design then is in how all the functions are allocated, and how the main loop handles the execution of those functions.

A Modular Approach

Spacecraft functions handled by the on-board computer include such tasks as processing commands, storing science and engineering data, and synchronizing spacecraft activity. A group of related functions can be handled by a single set of software code, i.e., a module. Separation of functions into modules opens the door to parallel development, test, and integration. These favorable qualities usually convince the casual reviewer as to the merits of modular design, while also satisfying the programmers that have to deal with complex systems. Addition of new functionality requires only the development of a new module or an addition to an existing module, not the re-design of the entire system. The SNOE team then separates functions into high and low levels, based on the type and frequency of a given function. High-level modules handle complex groups of functions such as assembling packets while low-level

modules perform simple but frequent functions such as memory access.

The Main Three High-Level Modules

The first 'division' of functions into modules effectively separates the uplink from the downlink. The SNOE team further divides the downlink into functions that are hardware synchronous and functions that are driven by events on-board. These two divisions result in three main modules: *Execute Commands*, *Build Packet*, and *Transmit Data*. *Execute Commands* handles all command processing while, together, *Build Packet* and *Transmit Data* handle the telemetry collection and transmission respectively. These main three modules comprise the majority of all functions performed by the flight software. Fig. 2 illustrates the three main modules with the relationship between the ground, the flight software, and the spacecraft.

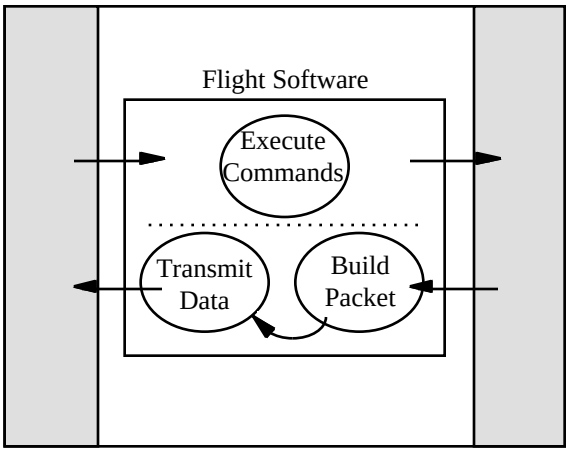

Fig. 2 Main High-Level Modules

The custom daughterboard handles all low-level functions for command reception and telemetry transmission such as bit sync and error detection, including a minimal command decoder. Still, the majority of all command processing and virtually all telemetry processing occurs in software. The conventional hardware Command Decoder Unit (CDU) and DTU have been replaced by software modules.

Other High-Level Modules

To complete the division of functions, the SNOE team uses an additional six high-level modules. These modules handle (among other things) all stored command processing, stored command execution, and closed loop attitude control. LASP intends to control the spacecraft attitude in an open loop fashion, but includes a closed loop attitude control module for safing. (The control algorithm employed is a "B dot" law that attempts to align a tumbling spacecraft orbit normal using the Earth's magnetic field.) As mentioned before, the flight software performs some functions usually implemented in hardware. Another case in point involves adjusting the instrument start delays when the roll error of the spacecraft exceeds an unacceptable threshold. The software literally sends itself a command to route to the instruments for re-configuration. This continual instrument adjustment lessens pointing requirements on attitude control while reducing the complexity of the instrument electronics. By allocating this set of functions to a module, the software can perform the auto-correction with minimal impact on the design.

A Deterministic Approach

One key to success has been satisfied with proper functional allocation to modules. The remaining key is to execute these functions-now modules-in a manner that absolutely ensures time and CPU resources are available. Typically, the modules are placed in the main loop encased with a dizzying array of flow control statements (e.g., if-then-else or while statements). This approach, however effective, is a cumbersome solution that is usually extremely difficult to debug. Additionally, concerns for CPU resources become a challenge to analyze without extensive testing. The SNOE team avoids these pitfalls with a vector approach. A main loop that simply steps through a table and uses a vector stored there to call the proper module after a given time interval turns the flight software into a deterministic system whose behavior is readily analyzed.

The Distribution Vector Table

Imagine that one second of CPU time is actually a set of 40 pieces of time, each 25 milliseconds in duration. Further, imagine a table of 40 "pointers," or vectors, to high-level modules. A main loop that, every 25 milliseconds, executes the high-level module at the location pointed to by the vector in the table fixes the start time for every function on-board. The SNOE team refers to this table as the distribution vector table (DVT). The main loop then steps to the next vector at the

next time interval. High-level modules that require greater than 25 milliseconds of CPU time are allocated more than one consecutive 'piece' of time. The SNOE main loop is guaranteed to execute all high-level modules within the DVT every second of CPU time. If a module requires an execution frequency greater than the one second duration of the

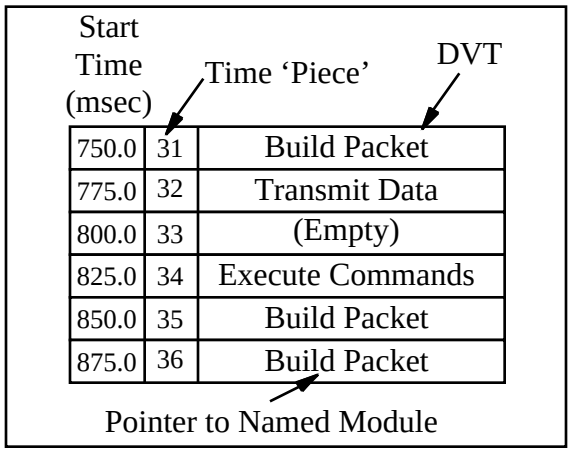

Fig. 3 Sample Distribution Vector Table

loop, repeated calls to that module are included in the DVT. Fig. 3 shows a sample vector table with start times and pointers to high-level modules. Each module checks whether or not that module should execute using a standard call to a low-level module (containing all flow control statements) based on the current configuration of the flight software. If a module called by the main loop is *not* supposed to execute, the main loop will *still* wait 25 milliseconds before stepping to the next vector in the DVT. If there is no module at the location (an *empty* vector call) the main loop also waits before executing the module at the next vector. This rigorously fixes the start time of each module, providing a means to analyze software timing issues with millisecond accuracy. Also, the vector approach makes replacement or re-ordering of the module sequence a simple matter of a vector change.

Execution of the high-level modules is now a matter of populating the DVT in an intelligent manner. The SNOE team will first identify the execution duration of each high-level module using the evaluation board. Current estimates are based on source lines of code (SLOC) and processing speed. Not all of the 'pieces' of CPU time in the DVT are used, leaving room for addition of new modules, or expansion of current modules. This deterministic approach provides a means to analyze the flight software time and resource usage prior to any actual programming. The explicit allocation of CPU time to module execution assures complete coverage of flight software functional requirements. The SNOE team plans to integrate flight software with the spacecraft in July 1996.

---

[1] CCSDS Packet Telemetry, November 1992, Blue Book Doc. No. 102.0-B-3

[2] CCSDS Telecommand, January 1987, Blue Book Doc. No. 202.0-B-1