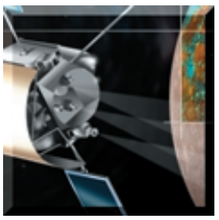

Flight Software in the Space Department: A Look at the Past and a View Toward the Future

Horace Malcom and Harry K. Utterback

Since the first use of general-purpose reprogrammable computers onboard the early Transit Improvement Program satellites, flight software developed within the APL Space Department has undergone an evolutionary process, from a collection of relatively simple control modules on single-processor custom-designed computers to systems that contain many megabytes of memory and numerous interconnected high-performance processors. At least one such system included over 50 loosely coupled processors. Today's flight software systems can be scripted to perform sophisticated closed-loop tracking and navigation, autonomous attitude control, and onboard processing of science data, and can greatly reduce the burden on ground operations in monitoring and exercising control over the spacecraft. In this article, we describe the nature of flight software, highlight some of the major changes that have occurred in the functions performed by the software, and examine changes in the way the software is specified, developed, and tested by the Space Department. (Keywords: Embedded systems, Flight software, Real-time systems, Satellites.)

INTRODUCTION

"Flight software" is the system of programs that reside in the memory of onboard spacecraft computers. These computer systems are also called embedded systems because they serve as the control mechanism for a larger system such as a spacecraft command or telemetry system. Before the Triad satellite, which was built by APL and launched in late 1972 as the first of the Transit Improvement Program (TIP) satellites, the command, telemetry, and other onboard subsystems were controlled by hard-wired logic circuits that responded to signals transmitted (uplinked) from the ground. Telemetry systems used voltage-controlled oscillators to modulate transmitters in sending housekeeping and other data to the ground (downlinking). Since then, not only have the command and telemetry systems come to depend on programmable computers and flight software, but so has almost every other subsystem onboard a typical satellite, including the attitude and guidance control systems, the power management systems, and most science instrument systems.

With the launch of Triad, APL was among the first, if not the first, to use onboard general-purpose

reprogrammable computers and uploadable flight software systems.[1] The Triad computer was based on a custom-designed and custom-built 16-bit processor. No suitable commercial processor was available. The Intel 4004 was not introduced until 1971 and was only a 4-bit processor. After the release of the Intel 8080 8-bit microprocessor in 1974, the Laboratory became one of the pioneers in the use of commercial microprocessors by incorporating an 8080 into the Seasat-A altimeter launched in 1978.

By today's standards, APL's early satellite computer systems would be considered primitive. However, the basic functions that they performed can still be found in the systems currently being designed and built by the Laboratory. These functions include executing uplinked and stored commands; monitoring, recording, and downlinking telemetry data; controlling sensors and managing power; and determining and controlling attitude. Where the current systems differ from the earlier ones is in their higher level of complexity, greater degree of autonomy, and greater number and variety of subsystems and experiments that must be controlled. Today's systems rely more on the ability to upload software to extend their capability and to correct for errors that may not be detected until after launch. The early systems, with their relatively low-power 8- and 16-bit processors, had limited memory capacities compared with current systems based on 32-bit processors. These newer processors employ pipelining, multimegabyte memories, and instruction and data caches and are frequently configured in multiprocessor arrangements.

Rather than attempting to chronicle the development of every flight software system used on APL spacecraft and instruments, we highlight in this article events in the evolution of flight software in the Space Department, beginning in the late 1960s and early 1970s and extending to the present.

THE NATURE OF FLIGHT SOFTWARE

Flight software is an example of a real-time embedded system in which the operational code resides initially in some form of nonvolatile read-only memory (ROM) such as electrically erasable programmable ROM (EEPROM). Such systems are self-starting because at power up, a ROM-resident boot program is automatically initiated, which in turn performs a series of self-tests on the processor and memory. If these tests reveal no abnormalities, then the boot program typically copies the operational code into read/write random access memory (RAM) and starts the execution of the operational code. Any errors detected as a result of the tests performed by the boot program are normally reported in the output telemetry stream and result in the performance of contingency actions. These contingencies may simply correspond to the boot program waiting for commands to specify the proper action to take in responding to the problem, or they may involve switching control to a redundant backup system.

By its very nature, flight software must respond in real time to events that usually occur asynchronously so that it can affect the progress of the processes under its control. This real-time response is achieved through the use of hardware-generated interrupts. These interrupt signals indicate, for example, the arrival of new commands, the expiration of timers, the extraction of telemetry data, the production of sensor data, the existence of error conditions, etc.

Most of our early software systems used a "foreground/background" design in which the main processing (e.g., formatting science and telemetry data, executing commands) was performed in a continuous background-polling loop. Within this loop, various flags were tested to detect the arrival of new commands or new data, which were then processed before the loop was repeated. The flags associated with the arrival of commands through the command interface or the data acquired from an experiment or sensor were set by software, "running in the foreground," in response to hardware interrupts triggered to signal such events. The interrupts would temporarily suspend the execution of the background loop and cause control to be transferred to an interrupt service routine, which was said to run in the foreground. There were almost always multiple competing interrupt sources that were handled on a priority basis. The timely response to such interrupts was critical in providing a software system that would behave in a deterministic fashion.

Custom-designed executive or kernel programs had to be written for these early systems since, as noted earlier, no suitable commercial operating systems were available. The need for such custom systems was especially true for the APL-designed processors on satellites such as the TIP/Nova series, as well as for the commercial processors (e.g., Intel 8080, 8085, and 8086; RCA-1802; etc.) used in the late 1970s and early 1980s. These systems tended to have memories with storage capacities from just a few kilobytes to at most tens of kilobytes and ran at speeds of around 1 MHz. Some of the flight processors used to control experiment subsystems had extremely limited memory (e.g., 4 KB of ROM and 2 KB of RAM for the Energetic Particles Detector [EPD] onboard the Galileo spacecraft).

Beginning in the 1980s, APL initiated a multitasking approach in the design of many flight software systems. With this approach, the software is organized as a system of units (called tasks or processes), each implementing some different functionality (telemetry formatting, science data acquisition, command processing, etc.), and each executing as a separate thread of control. Each task or process can communicate with other tasks through services (e.g., message queues,

mailboxes, semaphores, event flags, etc.) provided by an operating system. This approach greatly simplifies the design of the software system; it relies on the operating system to resolve contention for resources and to handle the details involved in task scheduling and intertask communications. A good example of this scheme is the one used for the Tracking Processor System (TPS) on the Midcourse Space Experiment (MSX) spacecraft, which contained 18 separate tasks written in the Ada programming language.

An important factor in flight software design is the nature of the environment in which it must operate. Very early on, it was understood that the behavior of solid-state circuits was affected by exposure to the types of radiation common in space (see the article by Ebert and Hoffman, this issue). The consequences for flight software include the possible alteration of instructions in memory due to single-event upsets and double-bit errors. To cope with such problems, the hardware normally includes error-detecting and error-correcting circuitry and watchdog timers (clock counters that must be periodically reset by the software to avoid a processor restart when the timer counts down to zero). Error detection and correction (EDAC) circuitry can correct single-bit errors in memory and detect double-bit memory errors. The flight software must perform "scrubbing" by continually cycling through memory, reading the contents of each location, and rewriting the data at those locations found by the EDAC circuitry to contain a single-bit error. This procedure corrects single-bit errors before a second error can occur in the same location.

Some of the Space Department's early satellites employed magnetic core memories that offered some degree of immunity to radiation and allowed for reprogramming. However, to achieve a higher level of radiation hardness, many other early flight systems used masked ROMs or programmable ROM (PROMs), which could not be changed after the flight software (i.e., firmware) was deposited into them. These ROMs often required that the flight code be completed and fully tested much earlier than would be necessary if some form of memory whose contents could be erased and rewritten had been used. The flight code had to be deposited into masked ROM during the manufacturing process, which, for these systems that relied on this technique, could require as long as a 6-month turnaround time. Such long turnarounds contributed to tight schedules for developers.

For example, as noted earlier, the EPD instrument on Galileo resided in a 4-KB masked ROM and thus could not be changed. This problem was overcome by using a technique based on a series of "patch hooks" in which the ROM-resident code would jump, at strategic points, to locations in RAM. These locations would normally contain an instruction that immediately returned to the proper location in ROM to continue execution. However, by using the Spacecraft Command System, instruction sequences could be loaded at the patch-hook locations within RAM, which would perform some new functionality before returning and resuming execution of the code in ROM.

Good software design tools and techniques can increase the likelihood that changes made in one part of a system will not have a ripple effect on other parts.

The wisdom in deciding to use uploadable systems was amply demonstrated with TIP II and III. Having a reprogrammable computer onboard these satellites provided the power and flexibility to work around early deployment problems while still achieving many of the mission's goals.[2] The adaptability and flexibility of uploadable software have been proven time and again over the years on a number of subsystems and spacecraft built by APL. It was through a patch-hook scheme that the EPD System on Galileo was able to salvage the primary science objectives despite the greatly reduced downlink data rate that resulted from the failure of the spacecraft's main antenna to fully deploy.[3] More recent examples include the Near Earth Asteroid Rendezvous (NEAR) spacecraft for which reprogramming allowed the elimination of cross talk between the X-Ray and Gamma-Ray Spectrometers (XGRS). No matter how extensively and exhaustively the flight hardware and software systems are tested, cases almost always exist in which flaws, however minor, go undetected until the spacecraft integration phase or even after launch. In such cases, the usual remedy is to modify the flight software to correct or compensate for the problem.

Unlike ground-based computer systems, early flight software systems did not have access to large-capacity online disk storage units. Instead, output data products were placed into a downlink telemetry stream or stored temporarily in a nonvolatile memory. The TIP systems used a ROM-resident 64-word bootstrap loader program, which in turn read in a more powerful loader transmitted from the ground.[1] These systems also used a 32-KB magnetic core memory to accommodate the application software and to serve as temporary storage for output data. Later spacecraft used magnetic tape recorders ranging in capacity from 5 megabits to 54 gigabits. More recent spacecraft used dynamic RAM (DRAM) chips to implement solid-state recorders, as first used on NEAR[4] and also on the Advanced

Composition Explorer (ACE) and Thermosphere-Ionosphere-Mesosphere Energetics and Dynamics (TIMED) spacecraft. The solid-state recorder devices contained in these higher-performing systems provide multiple-gigabit storage capacities but, as demonstrated on NEAR, may cause a loss of data when they are shut down owing to power emergencies.

Increased Complexity

Although many of the functions performed by the subsystems onboard today's spacecraft can be found in very early versions, their scope and level of complexity have increased significantly. The Triad Command System could handle 70 commands, and the onboard computer could be programmed to cause the execution of stored commands at prescribed times or in response to specified conditions within the recorded data.[5] Current systems comprise multiple subsystems and instruments, each containing one or more computers and each with its own discrete set of real-time and delayed commands which, taken together, can number into the hundreds.

Along with the escalating level of complexity in the functions provided by command and data handling (C&DH) systems, and by the attitude control and onboard experiments, has come a commensurate increase in the scope and complexity of the software required to manage and support those functions. There is also a much greater reliance on software to accomplish these more ambitious missions for which long communication time delays and extended periods of noncontact with ground control stations make commanding and monitoring operations by ground-based operators less feasible. In addition to the basic functions typically performed, the newer flight software systems allow for onboard image processing, more sophisticated data compression techniques, closed-loop tracking and navigation, and onboard data editing through preprocessing of science data.

One technique for achieving a higher level of autonomy is illustrated by the Far Ultraviolet Spectroscopic Explorer (FUSE) Instrument Data System (IDS), which uses an onboard scripting language (SCL) to provide more flexible control and to specify command sequences in response to various conditions. The Attitude Control System (ACS) as well as the C&DH Systems on MSX and NEAR contain autonomy rules designed to handle potential hardware or software faults. These systems can detect such problems and respond by achieving a "safe" spacecraft state.

With the increasing scale and scope of spacecraft missions, and with each subsystem and science instrument usually requiring its own set of flight software, the timely completion of such projects can only be ensured through concurrent development by separate teams. Until recently, these teams tended to work independently but required a clear, complete, and precise specification of the interfaces between subsystems. On missions such as NEAR, with a compressed development schedule, greater efforts were made to share software designs and components among teams to cut costs and meet the schedule. For example, a common boot program was developed and used for several instrument systems on NEAR. This program also served as a model for the Ultra Low Energy Isotope Spectrometer (ULEIS) instrument on the ACE spacecraft. The same concept was carried over to the TIMED Project, in which the C&DH System, the Guidance and Control Computer, and the GPS (Global Positioning System) Navigation System all share a common boot program module.

The Processors

The typical spacecraft computer lags several generations behind current commercial products, mainly because of the need to limit power consumption and the need for radiation hardness for the onboard central processing units (CPUs) and memories. Achieving radiation hardness is an expensive and time-consuming process. The market for radiation-hardened processor systems is relatively small when compared with the market for commercial computer systems and thus provides less incentive for commercial companies to develop them. Consequently, the computational power and speed of these onboard systems usually cannot compare to the power of a fairly inexpensive desktop computer. In many cases, these less powerful processors limit the types of functions that can be easily performed by flight software. As mentioned earlier, the unavailability of low-power radiation-hard computers resulted in the custom design by APL of the computers used on the Triad, TIP, and Nova satellites.

With the advent of microprocessors in the early 1970s, the Space Department began evaluating these new devices for use in onboard computers. The Intel 8080 microprocessor, introduced in 1974, was used as the basis for the Seasat Radar Altimeter's Controller/Tracker.[6] The launch of that altimeter in 1978 marked the first use of a commercial microprocessor in space. In 1979, Magsat, built by APL, was launched with the first command and attitude systems to use a microprocessor[7]—the radiation-hardened RCA-1802 that applied low-power CMOS technology. Both the Intel 8080 and RCA-1802 were 8-bit processors with clock rates of about 1 MHz. By the time these systems were launched, more powerful systems (e.g., Intel 8085 and 8086) had been commercially released but were not yet available in radiation-hardened form.

Because of its relatively high immunity to radiation and low power consumption, the RCA-1802 became

the processor of choice for instrument control systems and command and telemetry systems for a number of years. In addition to being used for Magsat, it was exploited for the Ion Composition Telescope (ICT) provided by APL for the Firewheel satellite, the Laboratory's EPD instrument on Galileo, the Medium Energy Particle Analyzer (MEPA) instrument and command and telemetry systems on the Charge Composition Explorer Active Magnetospheric Particle Tracer (CCE/AMPTE) Project, Geosat, the Ulysses Hi-Scale instrument, Hilat, and systems as recent as the Electron, Proton and Alpha Monitor (EPAM) Experiment on ACE.

In addition to the RCA-1802 and Intel 8080, other 8-bit processors such as the Intel 8085 were used for several flight systems. The U1 instrument on Delta 181 and the U2 on Delta 183 were both controlled by data processing units (DPUs) based on 8085s, as was the Geosat-A Altimeter launched in 1985.

As the demand for processing power increased, 16-bit processors (Intel 8086, Military Standard 1750A, RTX2000, RTX2010, etc.) came into use. The 8086s were used as target tracking processors onboard the Delta 181 (Thrusted Vector) and Delta 183 (Delta Star) spacecraft, as well as for the Topex Altimeter and the Star Tracker (START) System. The MSX spacecraft used five 1750As to implement the target tracking, attitude determination and control, data handling, and image processing functions for the Ultraviolet and Visible Imagers (UVISI) suite of instruments. The UVISI Image Processing System used a 1750A in conjunction with a digital signal processor (the ADPS-2100) in a simple processor pipeline arrangement. Other systems that incorporated digital signal processors included the special-purpose, inexpensive Satellite Radar Altimeter, which used the ADPS-2100 together with an 80C86RH processor.

The first 32-bit processor to be used was the FRISC (Forth Reduced Instruction Set Computer) chip, designed and built by APL,[8] and later licensed to a commercial vendor which marketed it under the name SC32. This processor was used in the ground support equipment for the Topex Altimeter and for the first time in space to control the Magnetic Field Experiment (MFE) flown on the Swedish Freja satellite. Since then, other instruments and spacecraft have also been based on 32-bit processors, including the IDS based on a Motorola 68020 processor onboard FUSE, and more recently the Mongoose V processor based on the 32-bit million instructions per second (MIPS) R3000, which was used by the C&DH System, Guidance and Control Computer, and dual-processor GPS Navigator/Tracker System on TIMED.

Tables 1 and 2, respectively, illustrate the variety of processors used on APL spacecraft and instrument systems. They are meant to show the relative magnitudes of the flight software systems based on memory usage and lines of code. Although not every spacecraft or instrument is listed, the tables provide a representative sample. The code size entries in some cases are approximate, and for some of the systems implemented in the Forth language, the number of Forth "words" (similar to small subroutines or procedures in other programming languages) is given. Figure 1 compares the processing power of CPUs onboard some of the APL spacecraft to illustrate the progression in their processing power and to contrast the capabilities of current and older system. The fact that a single instruction on the 16- and 32-bit machines usually accomplishes more than a single instruction on an 8-bit processor must be taken into account when considering the MIPS ratings listed in the tables.

The Development Environment

Again, owing to the limited memory resources and relatively low performance provided by the available processors, coupled with ever-increasing demands made on the flight software, programming of these systems tended (especially early on) to rely on assembly language to maximize run-time efficiency. However, this level of programming is inclined to be more tedious and error-prone than it is with higher-level languages such as C or Ada. In addition, there were few, if any, choices in development tools such as language translators, linker/loads, debuggers, etc., for these early systems.

Custom-designed processors required in-house development of the supporting programming tools. The processor used for Triad and the TIP satellites, for instance, required a custom-designed assembly language.

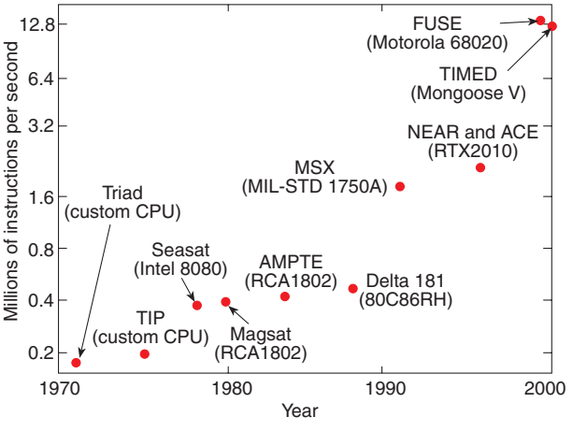

Figure 1. Relative processing power of flight processors. Electrical power limitations and the need for radiation hardness make it difficult to obtain and use the latest processors. However, a definite trend is evident toward more computationally powerful processors over the years.

Table 1. Flight software on APL spacecraft.

Project	Launch	Processor	Speed (MIPS)	Memory capacity	Language/code (lines)
Triad	1972	Custom-built 16-bit computer	0.25 max., 0.005 avg.	4K × 16 core, 64 words of ROM	ARTIC and assembly/ 5630 diagnostic
TIP II	1975	Custom-built 16-bit computer	0.25 max., 0.005 avg.	32-KB core, 64 words of ROM	Assembly/5700
TIP III	1976	Custom-built 16-bit computer	0.25 max., 0.005 avg.	32-KB core, 64 words of ROM	Assembly/5750
Magsat Telecommunications System	1979	RCA-1802	0.5	4 KB ROM, 1 KB RAM	Assembly/2800
Magsat ACS	1979	RCA-1802, 8-bit CPU	0.5	4 KB ROM, 1 KB RAM	Forth/1890
Hilat Magnetometer	1983	RCA-1802, 8-bit CPU	0.5	4 KB ROM, 2 KB RAM	Forth/432
CCE/AMPTE Command System	1984	RCA-1802, 8-bit CPU	0.5	4 KB ROM, 2 KB RAM	Assembly/3200
CCE/AMPTE Telemetry System	1984	RCA-1802, 8-bit CPU	0.5	4 KB ROM, 2 KB RAM	Assembly/3879
Geosat-A Command System	1985	RCA-1802, 8-bit CPU	0.5	4 KB ROM, 1 KB RAM	Assembly/1157
Polar BEAR Magnetometer	1986	RCA-1802, 8-bit CPU	0.5	4 KB ROM, 2 KB RAM	Forth/432
Delta 180	1986	Intel 8086A, 8087, 8089	0.5	192 KB RAM	Pascal/8212, assembly/5020
Delta 181	1988	80C86RH, Southwest Research SC-1	0.5	192 KB RAM	Pascal/8609, assembly/4818
Delta 183	1989	80C86RH, Southwest Research SC-1	0.5	192 KB RAM	Pascal/8310, assembly/8708
MSX C&DH	1996	MIL-STD 1750A 16-bit CPU	2	512 KB RAM	Ada/6188, assembly/2929
MSX C&DH DH and Serial I/O Controller	1996	Intel 8085 (2)	0.1	2 KB PROM	Assembly/6200
MSX TPS	1996	MIL-STD 1750A, 16-bit CPU	2	2 KB PROM, 256 EEPROM, 512 KB RAM	Ada/11689, assembly/10760
MSX AFC	1996	MIL-STD 1750A, 16-bit CPU	2	2 KB PROM, 256 EEPROM, 512 KB RAM	Ada/39642, assembly/16157
MSX UVISI Image Processor	1996	MIL-STD 1750A, ADSP-2100	2	2 KB PROM, 256 K EEPROM, 512 KB RAM,	Ada/3357, assembly/7946
				364 KB RAM for ADSP	Assembly/5120 ADSP
MSX UVISI Serial I/O	1996	Intel 8085	0.25	2 KB PROM	C/8963, assembly/11283
MSX Beacon Receiver	1996	80C86RH, ADSP-2100 (3)	0.5	8 K × 24 bit EEPROM, 16 KB RAM	C/3182, assembly/878
NEAR C&DH	1996	RTX2010	3	64 KB	Forth/13500
NEAR AIU	1996	RTX2010	6	128 KB	C and Forth/1300
NEAR Flight Computer	1996	MIL-STD 1750A	1.7	512 KB	Ada and assembly/18000
ACE C&DH	1997	RTX2010	3	64 KB	Forth/13500
FUSE	1999	Motorola 68020	14	128 KB PROM, 1 MB EPROM, 1 MB RAM, 48 MB bulk RAM	C/41000, assembler/1200, SCL/20000
TIMED GPS Navigation Processor	2000	Mongoose V	12	2 MB SRAM, 4 MB flash 16 KB DPRAM	C/31000, assembler/1960
TIMED GPS Tracking Processor	2000	Mongoose V	12	2 MB SRAM, 8 KB DPRAM	C/22599, assembler/600
TIMED C&DH	2000	Mongoose V	12	2 MB SRAM, 4 MB flash, 16 KB DPRAM	C/25000, assembler/1200
TIMED AFC	2000	Mongoose V	12	2 MB SRAM, 4 MB flash, 16 KB DPRAM	C/30000, assembler/1200
TIMED AIU	2000	RTX2010	3	128 KB RAM	C/17000, assembler/6000

Note: AFC = Attitude Flight Computer, AIU = Attitude Interface Unit, DPRAM = Dual-Port RAM, SRAM = Static RAM; all other acronyms can be found in the text.

Table 2. Flight software systems for APL science instruments.

Instrument	Spacecraft	Launch	Processor	Speed (MIPS)	Memory capacity	Language/ code (lines)
RadarAltimeter	Seasat-A	1978	Intel 8080	0.2	4 KB ROM, 2 KB RAM	Assembly/2100
ICT	Firewheel	1979	RCA-1802	0.5	4 KB ROM, 2 KB RAM	Assembly/3560
EPD	Galileo	1981	RCA-1802	0.5	4 KB ROM, 2 KB RAM	Assembly/4870
MEPA	CCE/AMPTE	1984	RCA-1802	0.5	6 KB ROM, 2 KB RAM	Assembly/3680
START	Spartan	1985	80C86RH	0.5	64K ROM, 512 KB RAM	Pascal/5680, assembly/1210
U2	Delta 181	1988	Intel 8085	0.25	16 KB PROM, 8 KB RAM	C/3064, assembly/480
Topex Altimeter	Topex	1989	80C86RH	0.5	32 KB ROM, 32 KB RAM	C/4714, assembly/4139
Hi-scale	Ulysses	1990	RCA-1802	1	6 KB ROM, 3 KB RAM	Assembly/4080
HUT	Astro-1	1990 and 1996	AMD-2900	0.5	2K PROM, 48K RAM, 128K Image RAM	Forth/12000
MFE	Freja	1992	SC32	4	64K × 32 RAM, 64K × 32 EEPROM, 8 KB PROM	Forth/4455
SSUSI ECU	DMSP Block 5D-3	1994	RTX2000	6.25	32K × 16 RAM, 64K × 16 EEPROM, 4K × 16 PROM	Forth/4374
SSUSI DPU	DMSP Block 5D-3	1994	RTX2000	6.25	24K × 16 RAM, 2K × 16 PROM	Forth/732
UVISI Instrument Control Unit	MSX	1996	8085 (2)	0.25	16 KB PROM, 8 KB RAM	C/3631, assembly/5151
UVISI sensors	MSX	1996	8085 (9)	0.25	16 KB PROM, 8 KB RAM	C/8964, assembly/11283
XRGS	NEAR	1996	RTX2010	6	96K × 16 RAM, 32K × 16 EEPROM, 4K × 16 PROM	Forth/9545
Multispectral Imager	NEAR	1996	RTX2010	6	96K × 16 RAM, 32K × 16 EEPROM, 4K × 16 PROM	Forth/5926
NIS/Magnetometer	NEAR	1996	RTX2010	6	96K × 16 RAM, 32K × 16 EEPROM, 4K × 16 PROM	Forth/3019
NLR	NEAR	1996	RTX2010	6	96K × 16 RAM, 32K × 16 EEPROM, 4K × 16 PROM	Forth/2946
MIMI CPU	Cassini	1997	RTX2010	6	448K × 16 RAM, 64K × 16 EEPROM, 4K × 16 PROM	Forth/14785
MIMI EPU	Cassini	1997	RTX2010	6	448K × 16 RAM, 64K × 16 EEPROM, 4K × 16 PROM	Forth/13689
ULEIS	ACE	1997	RTX2010	6	96K × 16 RAM, 32K × 16 EEPROM, 4K × 16 PROM	Forth/3762
HENA	Image	1999	RTX2010	6	640K × 16 RAM, 128K × 16 EEPROM, 4K × 16 PROM	Forth/8892
GUVI	TIMED	2000	80C86RH	1	48 KB PROM, 256 KB EEPROM, 320 KB RAM	C/10000
GUVI	TIMED	2000	RTX2010	6	16 KB PROM, 256 KB RAM	Forth/1357

Note: Acronyms not defined in the text are as follows: DMSP, Defense Meteorological Satellite Program; ECU, Electronics Control Unit; EPU, Event Processing Unit; GUVI, Global Ultraviolet Imager: HENA, High Energy Neutral Analyzer; MIMI, Magnetospheric Imaging Instrument; NIS, Near Infrared Spectrometer; NLR, NEAR Laser Rangefinder; SSUSI, Special Sensor Ultraviolet Imager.

The result was a cross assembler, called ARTIC,[5] which was implemented on an IBM mainframe computer (360/91) in the PL/1 programming language. Later, the cross assembler was rewritten to execute on the Xerox Sigma 3 computer in assembly language.

The cross assembler and loader used in developing flight software for many of the systems based on the RCA-1802 microprocessor were implemented in-house and written in a language called APL (A Programming Language). These tools ran on the IBM 3033

mainframe and generated files containing the executable code image that could then be downloaded from the host IBM computer over a communications link into a PROM programmer device.

For years, this use of cross-development tools hosted on a mainframe was the standard means of generating flight code. Commercial development systems based on the Digital Equipment Company's PDP-11 were purchased and used for the development of the Geosat and CCE/AMPTE Command and Telemetry Systems. Today, systems still rely on cross-development tools, but they are typically obtained from commercial vendors, are much more powerful and convenient to use, and usually run on workstations or desktop computers. In many cases, the generated software can be downloaded from a remote host over a network, with the target system capable of being executed and debugged or monitored remotely over the network. This approach was followed in developing flight software for the MSX spacecraft. Software modules written in Ada or assembly language were translated by a cross compiler and a cross assembler running on a VAX host computer to produce object code to be downloaded into the 1750A processor systems for the Attitude Control, Tracking Processor, and UVISI Image Processor Systems.

Several of the systems on the NEAR spacecraft relied on the Forth programming language, which provides either an interpretive or compiled mode of operation. The former allows the software to be run interactively to facilitate the debugging process. Once debugged, the source code can be compiled into an executable image for loading into RAM or EEPROM. Forth systems have been implemented for commercial systems such as the RCA-1802, the RTX2000, and the RTX2010, as well as for custom chips (FRISC and SC32) designed at APL. Some of these systems directly execute the high-level Forth language and have proven to be particularly well-suited for use in space instrumentation on "lightsats," i.e., small satellites with limited power and weight allocations.[9] Despite the relatively small size of these systems, they must perform increasingly complex data processing functions under real-time multitasking control.

The Seasat-A Radar Altimeter marked one of the first projects to use an in-circuit emulator (ICE), in this case for the 8080 microprocessor.[6] Such devices allow the operation of the processor to be simulated and monitored from an attached host system based on a workstation or PC. The development and checkout of the embedded software can consequently proceed even if the actual target processor is not yet available. The use of such tools is especially important given that the hardware and software are usually developed in parallel.

To observe the operation of the code after it has been loaded into memory, a logic analyzer was often used, when an ICE was not available, to provide a trace of execution and to monitor data and addresses placed onto the system bus. Unfortunately, tools of this sort often require dealing with the hexadecimal representation of the machine instructions and data. This approach is even less appealing for the more recent and powerful processors that use instruction pipelining and cache memories for instructions and data. On these systems, instructions fetched from cache do not appear on the data bus and are thus not captured by the logic analyzer. The Mongoose V processor, which is based on a MIPS R3000 core, is an example of this type of processor. The TIMED spacecraft uses four Mongoose V processors.

Support for multitasking can be provided by either a real-time executive or by facilities built into the language. Both the Forth and Ada programming languages include such facilities. For languages without them (e.g., C, Pascal, assembly) a real-time multitasking operating system (RTOS) must be used. The first commercial RTOSs on APL spacecraft were for the flight systems onboard the Star Tracker (START) Project and the Delta 181 and 183 Projects in 1985–1989. These systems were implemented using a combination of assembly language and Pascal modules, and were designed as a collection of cooperating, concurrently executing tasks that used synchronization and intertask communication services provided by the operating system.

Making any software system easier to develop, test, and maintain requires a good set of tools. Bus analyzers, visualization tools, and source-level debuggers can help in diagnosing problems and in understanding the system's behavior. Good software design tools and techniques can increase the likelihood that changes made in one part of a system will not have a ripple effect on other parts. Configuration management systems can help capture and record changes in the software and facilitate the re-creation of previous versions of the software when necessary. Flight software developers increasingly rely on such tools to create robust, reliable code.

The Development Process

Although valiant efforts were made to employ structured and modularized designs for the early software systems, the resulting systems were often difficult to understand for anyone other than the original developers. This was due in large part to the complexity of the systems, combined with the fact that assembly language was often the only viable choice for implementation. Thus, these systems were difficult to maintain or extend and usually required the original developer to make any needed changes.

The original EPD software, completed in 1981, was designed to output science and telemetry data at a 91-bps data rate. The Galileo spacecraft was waiting for a

May 1986 launch when the Challenger accident occurred, resulting in an extended postponement of space shuttle launches. By the time Galileo was actually launched in October 1989, several software changes were required owing to improvements made in the hardware system during the intervening period. Other changes to the software, after launch, were necessary because of the failure of the spacecraft's main high-gain antenna. The secondary antenna had to be used, thus effectively cutting the output data rate available to the instrument down to only 5 bps. By using the patch-hook capability, the EPD software was modified to apply a different data collection and readout scheme to cope with the reduced output data rate. By the time Galileo arrived at Jupiter in December 1995, 14 years had elapsed since the EPD ROM-based flight software had been completed. The lack of complete reprogrammability and the need to patch around ROM-based code forced the modified software to take on the appearance of "spaghetti code."

Many other examples exist, including the HUT (Hopkins Ultraviolet Telescope) software system,[10] in which the original developer had to make a number of changes since the same instrument being used on one mission was used on a second mission. Such changes may span several years for long-term projects.

Recall that another significant factor in the flight software development process is that the software is usually developed simultaneously with the hardware. This situation causes software developers problems above and beyond the usual, i.e., ferreting out real requirements, defining interfaces, creating an elegant and robust design, planning tests, etc. Since the hardware is new, the developers must "learn" the new machine and its interfaces to the rest of the spacecraft. Generally, hardware engineers need help in determining if their new computer and related hardware is working properly, a need that usually draws the software developers into the effort of designing and implementing diagnostic software. In addition, new hardware often adds complexity to software development, e.g., the use of flash EPROM adds a burden of complexity to both the flight and ground software systems to accommodate the special loading and modification needs of that type of memory.

To generate code containing fewer faults and to facilitate maintenance and modification by the original developer and others, the Space Department initiated a more formalized development process. The first major software project to undergo a formal design review process was the AMPTE ground system in 1983. Since then, the software development process has progressively embraced all levels of design review. Today, software development plans, requirements reviews, preliminary design reviews, detailed design reviews, and code walkthroughs are common steps in the process. Their purpose is to determine whether the designs adequately reflect the science and engineering objectives specified in the requirements. Software configuration management became a formal part of the development process during the software implementation for MSX; software problem reports were used to track errors and enhancements in both the ground and flight systems for the first time, and regular meetings of a configuration control board were convened to examine the problems and their solutions. A commercial source code control system is being used on TIMED to track the software configuration and to maintain software problem reports.

The Testing Approach

Considering the expense of spacecraft failures in terms of dollars and lost scientific data, thorough testing of flight software systems is critical. With the added complexity and functionality of the newer systems, however, adequate testing is becoming much more expensive and time-consuming.

Developers must perform unit testing of software components before integrating them into higher-level systems. This often requires embedding the component to be tested into the larger system, which frequently is not possible since the various system components are being developed concurrently. To cope with such situations, the design of the system must allow for components to be "stubbed out" and replaced with simpler modules that provide equivalent functionality.

In general, flight software testing must proceed even though the actual spacecraft flight hardware is not available during the software development phase. Therefore, the use of simulators is essential in identifying and correcting faults before the spacecraft-level testing phase. In the case of Triad and TIP, a flight computer simulator was built before the flight hardware was available. This simulator emulated the flight computer and its interfaces to the spacecraft. It proved very useful in developing and testing the software for those systems. A software-based test-bed simulation was used for the Delta 181 software development effort to facilitate testing. This effort was extended to develop and test software for the Delta 183.

By using ground support equipment along with hardware and software simulators, many closed-loop-tracking experiments were conducted for the MSX Project before integration with the spacecraft.[11] These simulations enabled the input of preset image sequences into the UVISI Image Processor System. Within these images, the image processor would identify candidate objects of interest and pass a prioritized list of such candidates to the Tracking Processor, which in turn would issue commands to the Attitude Control System to track and follow the selected object.

Such autonomous control systems raise difficult verification and validation (V&V) issues, however. V&V techniques must be employed to increase the level of confidence in these decision-making systems.[12] They can also aid in ensuring that the science objectives are achieved. A private firm was given the Independent V&V task beginning with Delta 181 and again for Delta 183, MSX, and NEAR.

Today, software is included in integration and test plans in the early phases of development, and various levels of testing are conducted throughout the development schedule. No longer can the attitude be taken that it is "just software." The same disciplined engineering approach followed for hardware systems must be taken for software design, development, and testing.

FUTURE DIRECTIONS

Beginning with the NEAR Project, as the first spacecraft in NASA's Discovery Program, emphasis shifted away from developing large, complex, expensive systems that required long production schedules. Instead, attention turned to the development of systems that were lower in cost and could be designed and built in less time without sacrificing quality or functionality. Making such systems feasible has entailed a corresponding shift in the approach taken for flight software development.

Unlike the early days when a unique software system was developed for each subsystem, often resulting in duplication of effort and functionality, current systems attempt to reduce cost by using suitable commercial off-the-shelf software systems and reusable components. Such systems must mesh well with the environment and the requirements of the intended applications and should not require extensive customization. Software development techniques have progressed from the use of custom tools written in-house to the use of commercially available integrated tool sets and a greater reliance on high-level languages and scripting systems, which facilitate the implementation of complex computational and control algorithms. No longer are custom-designed real-time executives the norm. Instead, commercial real-time operating systems are used that allow multitasking with prioritized and preemptive scheduling of tasks to better cope with the dynamically changing and event-driven scenarios that occur in space missions.

The future will require an even greater use of such systems, perhaps coupled with tools that can automatically generate code as was done for the Attitude Flight Computer on TIMED.[13] Based on descriptions of the system requirements, constraints, and resources, these automatic code generators can help to shorten development schedules. A greater reliance on object-oriented languages and development techniques should facilitate and promote software reuse. To further reduce cost, future systems may reverse the trend toward the use of numerous relatively low-performance processor systems and emphasize the use of one or two high-performance systems that allow the disparate functions required of the spacecraft to be combined into a single flight software system. The feasibility of such software systems will depend on the availability of sufficiently powerful computers that are capable of withstanding the conditions to which they will be subjected in space and on a willingness to accept the lack of redundancy that results from concentrating more and more functions within a single system.

Future systems will also have to include a greater degree of autonomy to better cope with faults and unanticipated situations as well as to reduce the costs associated with close monitoring and control by ground operation teams. The designs of these systems, in addition to reducing cost, will have to maximize flexibility and allow for reconfiguration so as to take advantage of unexpected opportunities to deliver additional science data. An example of such flexibility is the changes made on NEAR after launch to extend the science objectives by including the ability to sense the arrival of gamma ray bursts and to obtain imagery during a flyby of the asteroid Mathilde.

CONCLUSION

Not only is software one of the most important risk and reliability areas, it is also rapidly becoming one of the largest cost items in spacecraft development. Reuse of hardware will enable the reuse of software, which will reduce overall software costs. Additionally, the adherence to a well-defined and rigorous software development process will help to ensure that the science objectives of future space missions will be faithfully carried out through the timely production of what are becoming more and more complex, software-intensive systems for spacecraft.

REFERENCES

[1] Perschy, J. A., Elder, B. M., and Utterback, H. K., "Triad Programmable Computer," *Johns Hopkins APL Tech. Dig.* **12**(4), 12–24 (1973).

[2] Jenkins, R. E., and Whisnant, J. M., "A Demonstration of the Value of Spacecraft Computers," *Johns Hopkins APL Tech. Dig.* **5**(3), 225–237 (1984).

[3] Williams, D. J., "Jupiter—At Last!" *Johns Hopkins APL Tech. Dig.* **17**(4), 338–356 (1996).

[4] Burek, R. K., "The NEAR Solid-State Data Recorders," *Johns Hopkins APL Tech. Dig.* **19**(2), 235–240 (1998).

[5] Utterback, H. K., Whisnant, J. M., and Jenkins, R. E., "A System of Software for the TIP Spacecraft Computer," in *Proc. Symp. on Computer Techniques for Satellite Control and Data Processing,* Slough, England (11–12 Oct 1977).

[6] Perschy, J. A., "The SEASAT-A Satellite Radar Altimeter Spaceborne Microcomputer," *J. Br. Interplanet. Soc.* **32**, 9–14 (1979).

[7] Lew, A. L., Moore, B. C., Dozsa, J. R., and Burek, R. K., "The MAGSAT Telecommunications System," *Johns Hopkins APL Tech. Dig.* **1**(3), 183–187 (1980).

[8] Lee, S. C., and Hayes, J. R., "Development of a Forth Language Directed Processor Using VLSI Circuitry," *Johns Hopkins APL Tech Dig.* **10**(3), 216–225 (1989).

[9]Henshaw, R., Ballard, B., Hayes, J., and Lohr, D. A., "An Innovative On-Board Processor for Lightsats," in *Proc. AIAA/USU Conf. on Small Satellites*, AIAA (Aug 1990).
[10]Ballard, B., "Forth Direct Execution Processors in the Hopkins Ultraviolet Telescope," *J. Forth Applic. Res.* **2**(1), 33–47 (1984).
[11] Wilson, D. S., "A Testbed for the MSX Attitude and Tracking Processors," *Johns Hopkins APL Tech. Dig.* **17**(2), 161–172 (1996).
[12]*Space Department Software Quality Assurance Guidelines*, SDO-9989, JHU/APL, Laurel, MD (22 Sep 1992).
[13]Salada, W. F., and Dellinger, W. F., "Using MathWorks' Simulink, and Real-Time Workshop, Code Generator to Produce Attitude Control Test and Flight Code," in *Proc. 12th AIAA/USU Conf. on Small Satellites*, AIAA (Aug 1998).

ACKNOWLEDGMENTS: We would like to thank the people who provided information in preparing this article, including J. A. Perschy, R. E. Jenkins, R. C. Moore, J. R. Hayes, J. O. Goldsten, J. D. Boldt, S. F. Hutton, S. P. Williams, P. D. Schwartz, B. K. Heggestad, and W. F. Salada. We also thank the editors and reviewers whose helpful comments resulted in improvements to the article.

THE AUTHORS

HORACE MALCOM received B.S. and M.S. degrees in physics from Emory University and an M.S. degree in computer science from the Pennsylvania State University. Mr. Malcom is a Principal Professional Staff Physicist in APL's Space Department, specializing in real-time embedded flight software for satellites. Since 1980, he has taught courses in computer architecture and systems software in the JHU G. W. C. Whiting School of Engineering Part-Time Programs in Engineering and Applied Science, for which he also serves as a member of the Computer Science Program Committee. He has served as lead software engineer on a number of systems including the ICT and EPD on the Firewheel and Galileo spacecraft, the spacecraft telemetry system and MEPA instrument on AMPTE, the Star Tracker System on START, the UVISI Image Processor System on MSX, and the ULEIS instrument on ACE. Currently Mr. Malcom is engaged in developing flight software for the GPS Navigation System on the TIMED spacecraft. His e-mail address is horace.malcom@jhuapl.edu.

HARRY K. UTTERBACK received an A.B. in mathematics from Gettysburg College in 1957 and an M.S. in computer science in 1975 from The Johns Hopkins University. Mr. Utterback is a member of APL's Principal Professional Staff assigned to the Space Reliability and Quality Assurance Group of the Space Department. He also teaches in the JHU G. W. C. Whiting School of Engineering Part-Time Programs in Engineering and Applied Science. Mr. Utterback has developed real-time software systems for various spacecraft and ground systems since joining APL in 1969. He has served as the Chairman of the IR&D Committee on Software Engineering. He is a member of the AIAA, an emeritus member of the AIAA Technical Committee on Software Systems, and also a member of the IEEE Computer Society and the American Society for Quality. His current interests focus on quality assurance and improvements to the software development process. His e-mail address is harry.utterback@jhuapl.edu.