6

Distributed Computing On Board Voyager and Galileo

Voyager and Galileo are two outer planetary spacecraft that carry extensive computing capability. In spectacular encounters with Jupiter and Saturn, Voyagers 1 and 2 returned science data and imaging that far exceeded results of previous planetary flybys. Uranus was the successful 1986 objective of Voyager 2, nearly 10 years after launch. Galileo is designed for a Jupiter orbiter and probe mission*. Both types of spacecraft carry multiple computer systems, distributing functions among several machines, rather than using one central computer system as on the Viking Orbiter and Lander.

Distributed computing on large unmanned spacecraft developed conceptually from several sources. In 1967, Marshall Space Flight Center commissioned a study by General Electric Corporation's Missile and Space Division in Philadelphia as part of preparation for a huge "Voyager" Mars lander to be launched on a Saturn V booster in the early 1970s. Marshall asked GE to compare the advantages of a central computer configuration versus separate computers for different subsystems. General Electric used a highly mathematical approach to develop power, size, and weight comparisons of the different proposals in light of reliability considerations. Computer physical limits were set as high as 100 pounds and 300 watts due to the large size of the booster. This would allow computers such as the IBM 4Pi series, Autonetics D26J, and IBM's Saturn Launch Vehicle Digital Computer (LVDC) to be considered. Planners expected that the functions that later showed up on advanced Mariners—such as accelerometers, programmable sequencers with 512 words of memory, and telemetry registers—would be part of the proposed computer's capabilities and responsibilities. However, GE found that economies gained by a central system were outweighed by reliability advantages intrinsic to a distributed system[1].

Another approach came from Edward Greenberg, a Jet Propulsion Laboratory (JPL) engineer who programmed for the Mariner VI and VII Central Computer and Sequencer and contributed to the Viking Command Computer Subsystem (CCS) design. In December, 1972, he proposed that the Viking computer be standardized as a multimission processor[2]. His intent was to reuse hardware and software development tools such as assemblers and simulators. Since one Viking computer could never handle all the functions needed on Voyager, several computers, each with a limited domain of functions, were needed.

Aside from the GE study and Greenberg's proposal, JPL developed

*Originally set for launch in the early 1980s, the mission slipped to May of 1986, but the grounding of the Shuttle fleet and cancellation of the Shuttle Centaur upper stage program in early 1986 led to an indefinite postponement and probably a change of launch vehicle.

an additional argument for distributed computing. Edward H. Kopf, Jr., a JPL engineer specializing in attitude control, pointed out that different sections of the Laboratory needed computers to perform their assignments on Voyager and Galileo. Each group wanted its "own" computer, so that it would not be constantly competing for resources with other groups[3]. Therefore, a distributed system would help keep the peace.

The attractions of distributing computing, reliability, potential reusability, and separation of tasks, proved true in the development of the Voyager and Galileo spacecraft. Each has a functionally distributed set of computers. Voyager makes use of two of the Viking machines and a third, custom-built, computer. Each concentrates on processing different functions, such as attitude control, data formatting, and commanding. Galileo has dual processors for attitude control and six in a network for command and data handling. Both spacecraft were designed for long-duration, autonomous flight, a goal difficult to attain without the use of distribution.

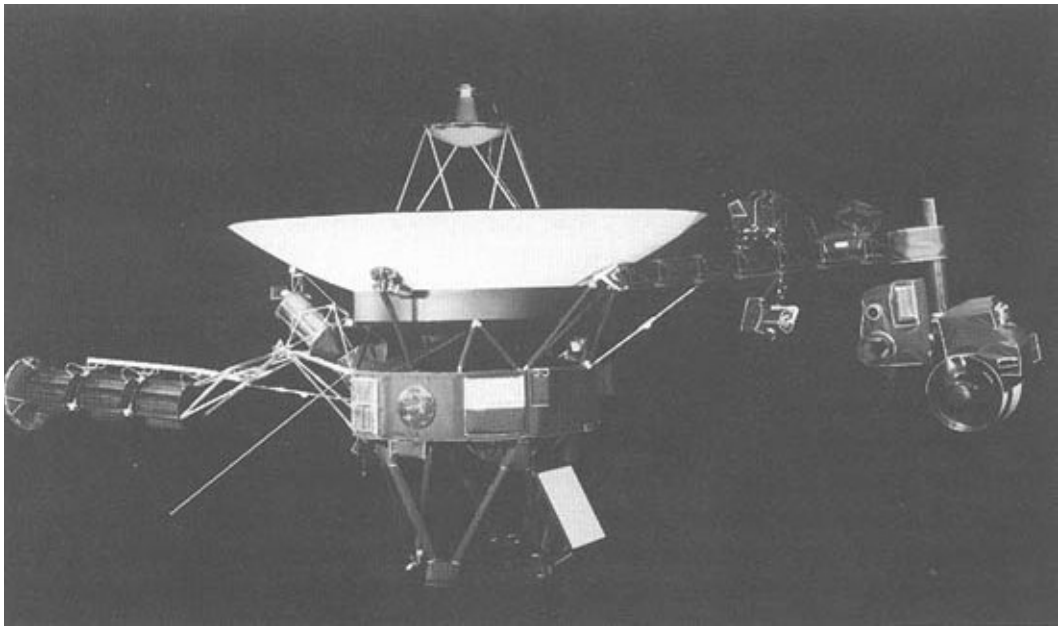

Figure 6–1. The Voyager spacecraft with the radioisotope generators on the left boom and the scan platform on the right boom. (JPL photo P10727B)

VOYAGER—THE FLYING COMPUTER CENTER

After the cancellation of the Thermoelectric Outer Planet Spacecraft (TOPS) project as such, JPL proposed, and NASA funded, a project called Mariner–Jupiter–Saturn 1977. It was given the name Voyager

in the mid-1970s. Although TOPS' original mission was to conduct the Grand Tour of the four gas giant planets, Voyager was limited to flybys of the innermost two, Jupiter and Saturn. However, favorable gravity assists and hardware longevity made it possible to plan for a Uranus flyby by the Voyager 2 spacecraft and, potentially, a Neptune encounter. After visiting Jupiter and Saturn, Voyager 1 is to travel out of the plane of the planetary orbits and leave the solar system.

Voyager employs three dual-redundant computer systems per spacecraft. The first, the CCS, is nearly identical to that flown on Viking, performing sequencing and spacecraft health functions along with new ones necessitated by the addition of the other computers. Telemetry data formatting and transmission handled by the Flight Data System are done on Voyager with the help of a custom-built computer. Attitude control and articulation of the scan platform are accomplished with the third computer system. One concept from the STAR computer proposed for the TOPS, applicable to Voyager, is dormancy. JPL's project staff believed that equipment would last longer if unpowered[4]. Although both CCSs are always powered, rarely are both Flight Data Systems running, and both attitude control computers are never turned on at the same time. Full bit-for-bit redundancy is not maintained in the dual memories. For example, "expended" algorithms, such as the deployment sequence executed shortly after separation from the booster, need not be maintained[5]. Both memories are accessed by the single active processor in each system. The Flight Data System keeps a copy of its instructions in both memories, but intermediate data and variables can be stored in either memory. This seemingly casual attitude toward memory duplication tightens up considerably near encounter periods, which is one time that both CCS processors are in tandem mode.

Since there are three computer systems on Voyager, JPL had to establish another layer of organizational control over its flight hardware and software development. Whereas Viking was assigned a single Cognizant Software Engineer, Voyager had three, managed by a Spacecraft Software Engineer. H. Kent Frewing of JPL assumed this position in early 1974 and sent out a series of organizing memos during the first half of that year.** Frewing's February 20, 1974 note set out his duties and a project time line through the summer 1977 launch dates[6]. Manpower estimates for software development ranged from one programmer in 1974 and 1977, with a peak of four full-time programmers in late 1975. The small group allowed most work to be done informally, easing communication. To provide some structure,

**He was replaced in early 1976 by Chnstopher P. Jones, who designed the integrated fault protection algorithms used on the mission, but Frewing laid the groundwork for management of the software.

Frewing established a Mariner–Jupiter–Saturn 1977 On-Board Software Design Team consisting of himself, Donald R. Johnson, the Flight Data System Cog Engineer, Stanley Lingon, the CCS Cog Engineer, and an Attitude and Articulation Control System representative[7]. They helped ensure the same close control of software development as on Viking, with good documentation and effective subroutine interfaces. The validation end of the software development process was handled by the Capability Demonstration Laboratory (CDL). Completed after the initial software was produced, it was a collection of either breadboard or flight surplus computer and science hardware, and its interfaces interconnected in the same way as those on the actual spacecraft. Its function is identical to that of the Shuttle Avionics Integration Laboratory (SAIL), in which both software and hardware changes could be tested to see if they functioned successfully[8]. Under this management umbrella, and with Cog Engineers constantly elucidating requirements from the science side and interpreting them to software engineers, each of the three computer systems took shape.

Voyager CCS: Parameters and Problems

NASA reeled from massive budget cuts during the 1970s. A changed political climate ended the Apollo era of near "carte blanche." Hampered by expensive Shuttle contracts as well as other factors, NASA management reduced its plans for unmanned exploration of the solar system. As Voyager developed under the new conditions, cost savings became a key ingredient in all engineering evaluations. JPL thus conducted a "CCS/CCS Memory Subsystem Design Inheritance Review" on January 17, 1974[9]. Held a year after Greenberg's proposal for standardizing the Viking computer, the Review resulted in the adoption of the Viking CCS as the Voyager CCS. The eventual hardware functional requirements document reads like a copy of the Viking document[10]. I/O interfaces with the new Flight Data System and Attitude Articulation and Control System computers are the major differences. Software such as the command decoder, certain fault processing routines, and others are fundamentally identical to Viking[11]. Here again, differences are related to the new computers. All command changes and memory loads for the other computers are routed through the CCS[12]. This required the addition of the routine MEMLOAD[13]. Another routine, AACSIN, was added to evaluate power codes sent from the Attitude Control computer as a "heartbeat" to inform the CCS of its health[14]. The frequency of the heartbeat, roughly 30 times per minute, caused concern that the CCS would be worn out processing it.

Mission Operations estimated that the CCS would have to be active 3% to 4% of the time, whereas the Viking Orbiter computer had trouble if it was more than 0.2% active[15]. As it turns out, this worry was unwarranted.

Part of the reason why the more complex Voyager spacecraft could be controlled by a computer with the same size memory as Viking is the ability to change software loads. In-flight reprogramming, begun when the programmable sequencers flew on Mariners, and brought to a state of high quality on Mariner X, was a nearly routine task by the time of Voyager's launch in 1977. Both the CCS and Flight Data System computer have been reprogrammed extensively. No less than 18 loads were uplinked to Voyager 1 during its Jupiter encounter. During long-duration cruise, such as between Saturn and Uranus, new loads are spaced to every 3 months[16]. As pioneered on Mariner X, a disaster backup sequence was stored in the Voyager 2 CCS memory for the Uranus encounter, and later for the Neptune encounter. Required because of the loss of redundancy after the primary radio receiver developed an internal short, the backup sequence will execute minimum experiment sequences and transmit data to earth; it occupies 20% of the 4K memory[17]. CCS programmers are studying ways to use some bit positions in a failed Flight Data System memory to compensate for the shortened memory in their system. A readout register in the Flight Data System has a failed bit, giving the impression that the entire memory has a one stored in that position in each word. Remaining "good" areas may be assigned to the use of the CCS[18].

Voyager Attitude Articulation and Control System Computer

JPL has been committed to three-axis stabilized spacecraft since it began designing probes in 1959. Attitude control systems maintain the proper pointing. The tasks assigned to the systems later expanded to include the actual operation of scanning platforms for imaging and other remote sensing instrument pointing. On the early Mariner missions the control systems consisted of analog circuits made up of hard-wired logic. By Mariner VIII, digital circuits replaced the analog electronics, and those were used on Mariner X as well as the Viking Orbiter[19]. Viking's Lander used the Honeywell central computer to run its independent attitude control system[20]. A landing craft engaged in a powered descent needed far finer pointing than a spacecraft in free flight, and the bandwidth of a hard-wired system was insufficient to provide such control[21].

Future probes, however, might need computer-controlled attitude electronics due to complex mission requirements or unusual spacecraft

configurations. NASA's Office of Aeronautics and Space Technology funded a study of extended life attitude control systems as the TOPS project wound down in 1972. The result was a combination analog and digital programmable attitude control system. Dubbed "HYPACE," for *Hy*brid *P*rogrammable *A*ttitude *C*ontrol *E*lectronics, it was a byte-serial processor with substantial power[22]. Using the same 4K, 18-bit-wide plated-wire memory from the Viking Orbiter computer, HYPACE added transistor-transistor logic (TTL) medium-scale integrated circuits to create a relatively fast (28-microsecond cycle) processor with index registers for addressing. Byte-serial architecture was possible because the TTL chips were designed for 4-bit parallel operation, so the 18-bit words could be moved around in five cycles instead of the 18 a serial machine would need, increasing overall speed. Index registering meant that the same block of code could be used for all three axes, reducing memory requirements. It appeared that the attitude control systems of future spacecraft would almost certainly benefit from such a computer.

Voyager was the first to do so, due to new requirements. One difference between Voyager and Mariner and Viking is that the latter two were fairly rigid in construction. Voyager's radioisotope thermoelectric generators, however, were mounted on a boom to keep radiation leakage away from scientific instruments. In addition, the magnetometer was boom mounted to avoid interference from spacecraft magnetic fields caused by motors, actuators, power buses, and electronics. Finally, the scan platform was also on a boom to give a better field of view. The extended booms made Voyager much less rigid in flight, with thruster firings and maneuvers causing the booms to flex, complicating the attitude control problem[23]. Additionally, the Titan III booster used for Voyager required a "kick stage" to successfully inject Voyager into the transfer orbit to Jupiter. Since the kick stage was kept simple, the spacecraft itself was required to do attitude control during firing, which entailed much narrower margins of control than the three-axis pointing in cruise[24].

JPL's Guidance and Control Section wanted to use a version of HYPACE as the computer for the Voyager. However, there was considerable pressure to build on the past and use existing equipment[25]. Greenberg proposed using the same Viking computer in all systems on the Voyager spacecraft that needed one[26]. A study showed that the attitude control system could use the CCS computer, but the Flight Data System could not due to high I/O requirements[27]. Wayne Kohl, the Viking computer Cog Engineer, thought that the Voyager project could save $300,000 by using the Viking machine for the attitude control function[28]. His division chief, John Scull, supported that idea, possibly because of budget pressure from NASA[29]. Raymond L. Heacock, as Spacecraft Systems Manager in the Voyager Project Office, and others from that organization were the key personnel involved in making the

final decision, influenced by the economy and feasibility of the idea[30]. Money could be saved in two ways by using the existing system: avoidance of new development costs and retraining of personnel.

Guidance and Control grudgingly accepted the CCS computer on the condition it be speeded up. Requirements for active control during the kick stage burn meant that real-time control programs would have to be written to operate within a 20-millisecond cycle, roughly three times faster than the command computer[31]. An executive for the attitude control computer differed in nature from those for either the command computer or the Flight Data System computer. Basically, the attitude control computer needed to run subprograms at different rates, requiring several cycles, as in Apollo, Skylab, and the Shuttle. Guidance and Control asked for a 1-megahertz clock speed but wound up getting about three quarters of that[32]. The attitude control engineers also added the index registers that proved so useful during the HYPACE experiment. Documentation for the system still refers to the attitude control computer as HYPACE, even though its heart was the command computer. General Electric, which built the command computer, naturally built HYPACE, but the rest of the attitude control system was constructed by Martin-Marietta Corporation in Denver.

Teoguer A. Almaguer was the hardware Cog Engineer for the attitude control computer, whereas H. Karl Bouvier led the software development group. Bouvier actually worked on an analysis team within the Guidance and Control Section, but the team members were afraid to use the word "software" in their name because their tasks might have been taken away and given to an existing software team in another division[33]. The programmers must have done an outstanding job, considering the slow processor and limited memory. At launch, only two words of free space remained in the 4K of plated wire[34]. Tight memory is now a problem because the scan platform actuators on Voyager 2 are nearly worn out, and software has to compensate for this during Uranus and Neptune encounter periods.

Box 6-1: Voyager HYPACE Operation

HYPACE had four execution rates. Scan platform stepper motors and thruster actuators were among the routines executed during the 10-millisecond cycle. Attitude control laws and thruster logic executed in the 20-millisecond cycle. Scanning control and turn execution were placed in the 60-millisecond group, and the command interpreter and heartbeat were 240-millisecond routines[35]. In operation, the standard 10-millisecond time interrupt would cause all 10-millisecond routines to execute. It was time for one of the 20-, 60-, or 240-millisecond routines to run, it would be scheduled. Sometimes if the computer got too busy, the 240-millisecond cycle slipped to up to 350 milliseconds, but routines in that cycle were less critical than a routine to shut off an engine on time.

One thing needed on Voyager that did not exist when only single computers flew on unmanned spacecraft was an interface between the machines. The command computer could directly request data from either of its partners. A primary function of the command computer was to check periodically on the health of the other computers. Programmers in the Guidance and Control Section originally intended to send a "heartbeat" to the command computer each second[36]. This was later raised to once about every 2 seconds, partly because of the command computer overload problem mentioned above. To carry the heartbeat, six direct input lines, similar to the 3-bit synchronization bus on the Shuttle, ran from the HYPACE to the command computer. A "power code" was the content of the 6 bits transmitted on those lines. For example, power code 37 was the simple heartbeat. Others related to passing information such as pointing commands. Power code 66, called "the Omen," told the command computer to save disaster parameters, because a failure was imminent[37]. Every eight 240-millisecond cycles the heartbeat was sent. Between times, the attitude control computer conducted its self tests. If it failed, the heartbeat generator was bypassed. After about 10 seconds passed with no heartbeats, the command computer would issue a switch-over command to the backup processor.

A switch-over to the backup attitude control computer took place on Voyager 2 16 seconds after separation from the solid rocket stage[38]. Separation was so rough that the spacecraft was sent off attitude. Simultaneously, the booms were being deployed by the command computer. A thruster configuration initialization involving the plumbing for the thrusters delayed their acting to correct the attitude error.

Since this was one of the mission-critical times that the command computer was turning in dual mode, the attitude control computer got *two* commands to initiate the plumbing. Executing the second command pushed back the attitude control recovery even farther. Soon the computer exhausted its options and voluntarily stopped the heartbeat. When the backup came on-line it had no record of the gyro readings. Not knowing how bad things were was a blessing, as it executed a simple orientation and stopped the spacecraft roll[39]. Here is an instance where maintaining bit-for-bit identical memories would have been disastrous, as the backup computer would also have tied itself in knots.

Developing Voyager's Flight Data System Computer

Flight Data Systems handle the collection, formatting, and storage of science and engineering data on spacecraft. If the data are to be transmitted directly, a high rate of input and output is needed so that nothing is lost. If data transmission is deferred because a spacecraft is occulted from the tracking station, then the Flight Data System sends the data to a magnetic tape recorder known as the Data Storage System (DSS). As JPL progressed through Ranger to Surveyor to Mariner and to Viking, the rates of the data-handling requirements went steadily upward. This was because of increased instrumentation, greater sophistication in the spacecraft engineering systems, imaging equipment with better resolution (thus needing higher bit rates), and improved communications equipment permitting faster transmission of data. These changes led away from hard-wired Flight Data systems. One big step was the use of a digital memory on Viking to store different sequences of data handling. It was much like the microprogram in a central processor and for a similar purpose: to save hardware[40]. From there it was a short step to a full-fledged computer.

TOPS feasibility studies refer to a Measurement Processor Subsystem, the first time a separate computer was considered for flight data[41]. Although the command computer had been suggested as a possible Flight Data System machine, JPL engineers soon realized that even though the processing part of the job was well within the power of the computer, the I/O rates precluded its use.

JPL commissioned the development of a new computer from scratch and assigned Jack L. Wooddell to the job. Wooddell prepared an unusual document to tell the story of his work on the computer: a paper for a graduate computer science course taught by Dr. Melvin Breuer at the University of Southern California. Written around 1974, the paper includes what appears to be the flight version of the design[42].

In it Wooddell lists the tasks he performed during the design period. He began by preparing a list of functions that the proposed Flight Data System was required to provide. These included sending control signals to sequence the science instruments, the ability to handle a wide variety of data rates and formats from the various instruments, potential for redesigning the mission in flight (as is now being done), monitoring engineering telemetry, and keeping to the reliability standard that no single failure result in loss of data from more than one scientific instrument or one-half the engineering sensors[43].

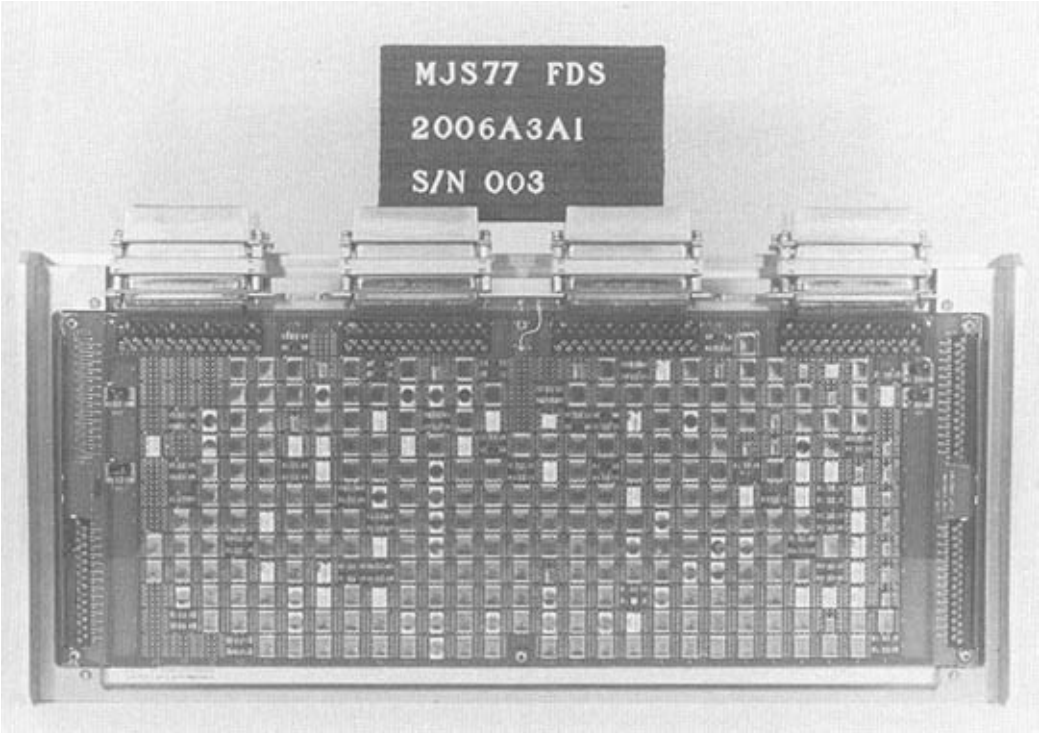

Figure 6–2. The Flight Data System hardware in its package. (JPL photo 360-751AC)

After determining requirements, Wooddell examined possible hardware and software tradeoffs. In an insightful memorandum, John Morecroft explained the concept of "soft logic" as a complement to the "hard logic" in the Flight Data System[44]. Writing in 1975, when the actual flight software began to be prepared, Morecroft pointed out that the program for the computer was actually a soft representation of hard-wired circuits. Conceptually, the memo stands as an explanation of the essential meaning of firmware in general. During the second phase of his work, Wooddell determined which functions could be handled by hardware and which should be left to the flexibility of software. With those decisions made, a preliminary instruction set and

logic design could be prepared.

Uniquely, Wooddell began working with a programmer in 1973, as soon as the instructions were ready[45]. Richard J. Rice of JPL began by developing software for a breadboard version of the data computer. The breadboard originally used the ubiquitous 4K memory of plated wire with 18-bit words and 150 of the same low-power TTL ICs used in other JPL machines[46]. Instruction execution times for this version ranged from 12 to 24 microseconds. Rice's prototype flight program, developed on the basis of what was then known about Voyager instrumentation and previous experience, showed that the processor speed should be doubled[47].

Two significant hardware changes solved this problem. One hardware modification added direct memory access circuits and provided for using them on *each* instruction cycle. Direct memory access capability meant that some data could be sent directly to the memory without having to go through the central processor. In other computers, direct memory access is permitted as a sort of interrupt and is often referred to as "cycle stealing" because it takes time away from instruction execution. In the data computer, it would have been foolhardy to do direct memory access in that way because the data rate was so high that the instructions might never get a chance to be executed quickly enough for time-critical sequencing. Wooddell solved this by adding a direct memory access cycle to those instructions that did not already have cycles in which the memory was accessed[48]. By adding that cycle all the instructions took the same time to execute regardless of direct memory access, making it easier to predict program run times and to guarantee the memory access rate[49]. Rice, who suggested the change, later said that his programming job would have been impossible without it[50].

The second hardware modification to Voyager's data computer led to a first in spaceflight computing: volatile memory. After the first round of prototype programs, an intermediate hardware design evolved using CMOS ICs[51]. This type of circuit is very low powered, fast, and can tolerate a wide range of voltages, making it excellent for space use. Early in the 1970s, CMOS was still relatively new, so it was with some risk that JPL chose the circuits. To go along with the new CMOS processor, the data computer group fought for CMOS memories as well. Trying to drive a slow plated-wire memory with fast CMOS circuits would have negated the attempt to speed up the computer. However, CMOS memories are volatile, in that if power is cut off, the data stored in them disappear. The designers of previous manned and unmanned spacecraft avoided volatile memories, fearing that power transients would destroy the memories at critical mission times. Voyager management had to be convinced that the risk was acceptable.

James T. Kinsey, a JPL manager, was instrumental in getting the semiconductor memory accepted because a method of providing backup power was devised[52]. Voyager's primary electricity is alternating current. The radioisotope generators produce direct current, which is converted. By running a separate power line from the direct current bus fed by the generators to the CMOS memories, the only way power would be lost is if a major catastrophe destroyed the generators. If that happened there would not be any need for a data computer anyway. Enough voltage is supplied to retain the information in memory and in the registers in the processor that contain the state vector[53]. Success with the CMOS memory led to the adoption of all CMOS circuits in both computer systems on the Galileo spacecraft. Along with the new chips, the memory changed with an expansion to 8K. Two "external" address bits were added to flag whether the top or bottom half of the memory is being accessed[54]. One bit is used to select the memory half used for data access; the other, for the half used for instruction access.

Eventually, the cycle of prototyping and interaction between Rice and Wooddell stopped as a final design was accepted. Wooddell wrote that the extensive use of breadboards instead of paper designs optimized the process[55]. His method, although not strictly "software first" was certainly software sensitive. Martin-Marietta's experiences with a software first philosophy as described in the previous chapter indicate that Wooddell had a clearer idea of his objective than did Martin. The job done on the Flight Data Systems computer is a good model of fine engineering practice in developing a total system.

Voyager Flight Data System Software

The original software development for the data computer has essentially been a two-man show since 1975, beginning when Edgar M. Blizzard joined Richard Rice to develop the flight version of the code. Others have been involved in testing and management, but these two JPL engineers have been the key programmers for the entire mission to date. They sit in the same area as the "Laboratory Test Set," an Interdata computer and peripherals that contain the software simulator of the data computer and the assembler and flight load generator. Across from them is the CDL, the loose conglomeration of hardware that represents the real spacecraft. From start to validation to release, their tools were within sight, and certainly hearing, since the room is filled with the constant hum of spinning disks, occasional clattering printers, and the undefinable sound of computers crunching numbers.

Rice characterized the unique nature of the data computer software this way: "We didn't worry about top-down or structured;

Box 6-2: Voyager Flight Data System Computer Architecture

Voyager's data computer is different from most small general-purpose computers in several ways. Its special registers are kept in memory, permitting a large number (128) of them. Wooddell also wrote more powerful shift and rotate instructions because of data-handling requirements. Despite its I/O rate, the arithmetic rate is quite slow, mostly due to byte-serial operation. This meant 4-bit bytes are operated on in sequence. Since the word size of the machine is 16 bits, it takes six cycles to do an add, including housekeeping cycles[56]. If all the arithmetic, logic and shifting were not done in the general registers, the machine would have been even slower. Reflecting its role, in addition to the usual ADD, SUB, AND, OR, and XOR instructions found on most computers, the data computer has many incrementing, decrementing, and machines instructions among the 36 defined for the flight version of the machine[57].

Overall, the Flight Data System requires 14 watts of power and weighs 16.3 kilograms[58]. Its computer needs just one third of a watt and 10 volts, less than the power required for a temperature sensor[59]! At first the estimated throughput required was 20,000 16-bit words per seconds[60]. By flight time, the instruction execution rate was 80,000 per second, with data rates of 115,000 bits per second, much higher than previous Flight Data Systems[61]. The dual processor/dual memory architecture of the command computer and attitude control computer is repeated in the data computer. There was no provision for automatic switch-over in case of failure. A command from the ground routed by the command computer is necessary for reconfiguration[62]. Note that the attitude control computer can be switched by the command computer without ground intervention because it is much more critical to retain orientation.

we just defined functions"[63]. One important function is the software's provision of basic timing for the entire spacecraft, not just itself. It is also required to provide the capability to read out the memories of all three computers, under orders of the command computer[64]. Don Johnson, the Cog Engineer, determined other requirements and interfaces with the scientific instruments. Rice called him"Mr. PDS," claiming that Johnson often knew more about the scientific instruments than the scientists themselves: "If someone forgot something, Johnson knew it"[65]. Raymond L. Heacock, Voyager Project Manager, said that Johnson was largely responsible for the overall success of the system, including the design[66]. Rice said that Johnson's ebullient style and competence worked well in the informal mode in which the data computer requirements were set, which was a fully iterative process. New software needs continued to be discovered during the mission, which is one reason why a programmable machine was chosen. For

example, at one point Rice and Blizzard were asked to create software to determine where the limbs of satellites were so that imaging could be started[67]. Development of some programs was deferred until after launch, such as the Saturn encounter program, when better data on the telecommunications rates and specific science requirements would be available[68].

Allowing for constant change mandated certain controls over the data computer's memory. A limit of 90% capacity was set in 1976 by Frewing, the Software Cog Engineer[69]. Though later abandoned, the constraint indicated the software management's early concern about memory overruns. Also, since the machine can directly address the lower 4K of memory, programs were to be kept there, with the upper portion for transient data[70]. Later, the flight configuration of the computer evolved to one processor accessing both memories. Therefore, a copy of the programs is kept in the lower portion of each memory, but both upper portions are usable by the single processor as a scratch pad[71]. If dual mode is required, the memories are separated. Experience has produced increased confidence in the memories. At first, complete loads had to be sent when an update was done; recently, pieces of software have been allowed to be inserted in the programs. Full redundancy between the memories is not now automatically maintained[72].

Box 6-3: Flight Data System Computer Executive

Like the command computer, the data computer has a simple executive. Time is divided into twenty-four 2.5-millisecond intervals, called "P periods." Each 24 P periods represent one imaging system scan line. Eight hundred of those lines is a frame. At the beginning of each P period, the software automatically returns to memory location 0000, where it executes a routine that determines what functions to perform during that P period[73]. Care is taken that the software completes all pending processes in the 2.5-millisecond period, a job made easier by the standardization of execution times once the direct memory access cycle was added.

Voyager's Future

Voyager software development continued into the late 1980s. Kohl, Wooddell, Greensburg, Deese, Johnson, Kopf, and others closely connected with the hardware of Voyager's computers were then on other projects, but Rice and Blizzard and their counterparts on the command computer and attitude control computer were still programming, prepar-

ing Voyager 2 for Uranus and Voyager 1 to discover the boundary of the solar wind. An increasing problem as the spacecraft recede from the earth is the reduction in the data transmission rate. The closer a spacecraft is to earth, the higher the bandwidth possible. Computer loads that once took minutes now take hours because error checking by retransmitting to earth is slowed. In the summer of 1984, a Flight Data System software load took 4 hours, and the situation cannot improve[74]. Voyager Project officials decided to use the Flight Data System in dual processor mode for the first time for the Uranus encounter to provide image data compression. Thus, the information content remained high even though the transmission rate was grossly reduced[75].

Voyager's computer system did not carry on to the next JPL project. Galileo combined the CCS and the Flight Data System into a single Command and Data System. This is logical from JPL's standpoint because both systems are the responsibility of the same Information Systems Division. Attitude control is provided by a separate computer. Whereas Voyager was a functionally distributed system with dual redundancy, Galileo's Command and Data System contains computers that do true distributed processing and use a new concept of redundancy. That system may be a model for the future, as it can impact designs aimed at complex spacecraft with extensive data processing needs, such as the Space Station and Mariner Mark II, both due in the 1990s.

GALILEO—TRUE DISTRIBUTED COMPUTING IN SPACE

Project Galileo began at JPL in the late 1970s with the objective of developing an orbiter and probe for further exploration of Jupiter. Galileo will proceed toward Jupiter, launching a probe 5 months before arrival. Plans are for the probe to enter the Jovian atmosphere at a relatively low angle, using an aeroshell braking system for entry followed by a parachute system for final braking and descent. Due to the nature of the entry, an antenna large enough to send data directly to earth cannot be carried on the probe. Instead, it has to relay the data to the orbiter, which will fly a parallel path thousands of kilometers above. At the end of the probe mission, expected to last 60 to 75 minutes until the probe is crushed by atmospheric pressure, the orbiter will execute an insertion burn. For the next 2 years, the orbiter will fly by the four Galilean satellites (hence, the mission name), using gravity assists to change its path after each encounter.

Great demands will be placed on Galileo's on-board computer systems, because of both the nature of the mission and the design of the spacecraft itself. First, Galileo is a one-shot mission. Interplanetary probes have been mostly launched in pairs for the obvious reason that a

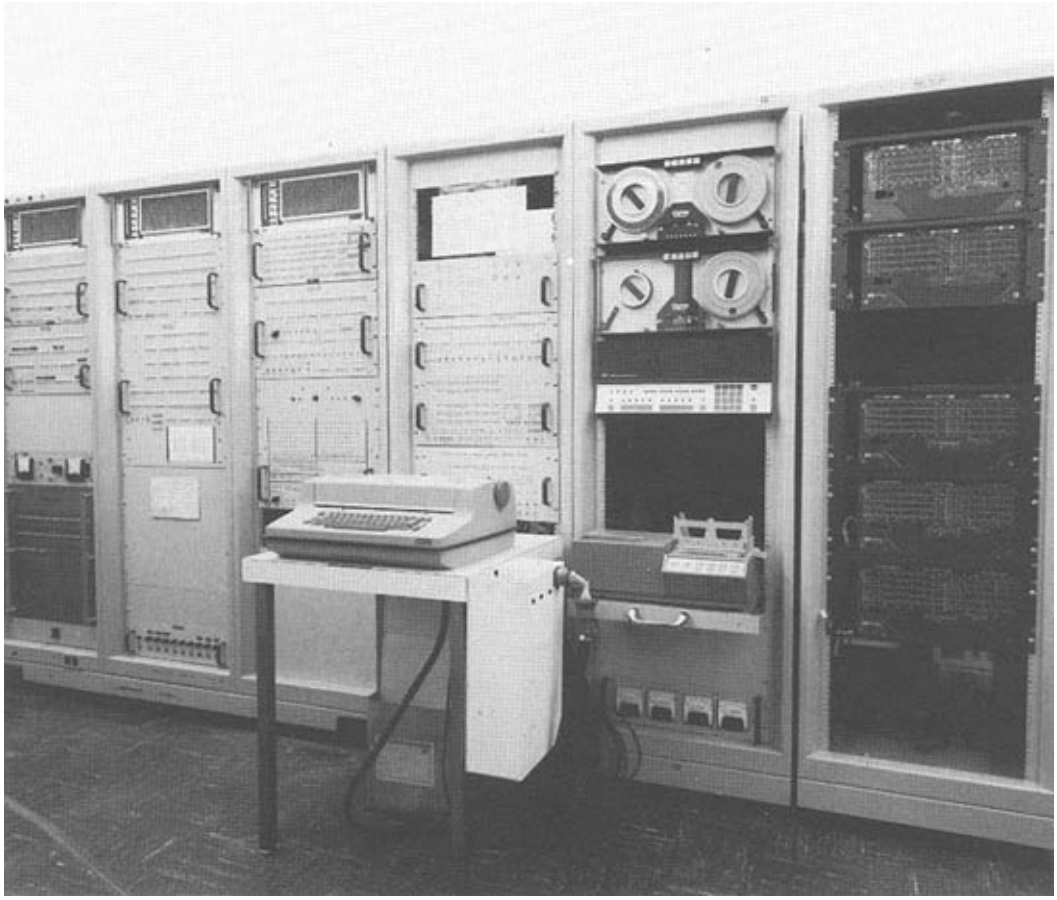

Figure 6–3. The Unified Data System: precursor to Galileo's Command and Data Subsystem. (JPL Photo 360-630)

full backup then exists. Such dual launches are generally cost effective, as a second spacecraft can be obtained for 15% of the price of the first one[76]. Budget constraints forced NASA to buy just one Galileo, so there was tremendous pressure to construct a highly reliable spacecraft. Additional pressure has been on the project because of changes in the launch date and booster rocket. Originally scheduled for a 1982 launch, delays in the Shuttle program and other factors caused rescheduling to 1984, then 1985 and, finally, 1986, when the grounding of the Shuttle fleet forced an indefinite postponement. At first, the Air Force's Inertial Upper Stage rocket was chosen for the booster. Later, the new "wide body" Centaur got the job. Centaur upper stages have flown on Atlas and Titan III boosters since the 1960s. The new "fat Centaur" would carry 50% more fuel than the earlier version. Other changes were made to adapt it to the Shuttle cargo bay. One JPL engineer said that it is "like Abe Lincoln's axe. The head broke and they replaced it and the handle broke and they put on a new one, but it's still Abe Lincoln's axe"[77]. However, NASA canceled the Shuttle version of the new Centaur in the spring of 1986 due to safety considerations, leaving Galileo without a ride to Jupiter. By early 1987, NASA decided to go

back to using the Inertial Upper Stage, but it has significantly lower lifting capability than the Centaur. As a result, the flight path has to be changed to include a Venus flyby and two Earth flybys to gain velocity by gravity assistance. Unfortunately, the total flight time to Jupiter will nearly triple to about seven years.

Spacecraft design also caused problems for the computer designers. All previous JPL probes have been three-axis inertial to gain advantages such as easing communication and providing a stable scan platform for imaging. Galileo has a fixed attitude area, called the "despun section," and also a "spun section" that rotates three times a minute. Fields and particles experimenters required a spin to help them differentiate local fields from external fields. Aside from the obvious increase in the order of magnitude of the attitude control problem on a dual-spin spacecraft, communication between the two parts is hampered by the need to transmit serially across a rotary transformer. Four hundred milliseconds are required to send a message between the spun and despun sections[78]. To overcome this time penalty and provide more real-time control, a fully distributed system of computers is mandated.

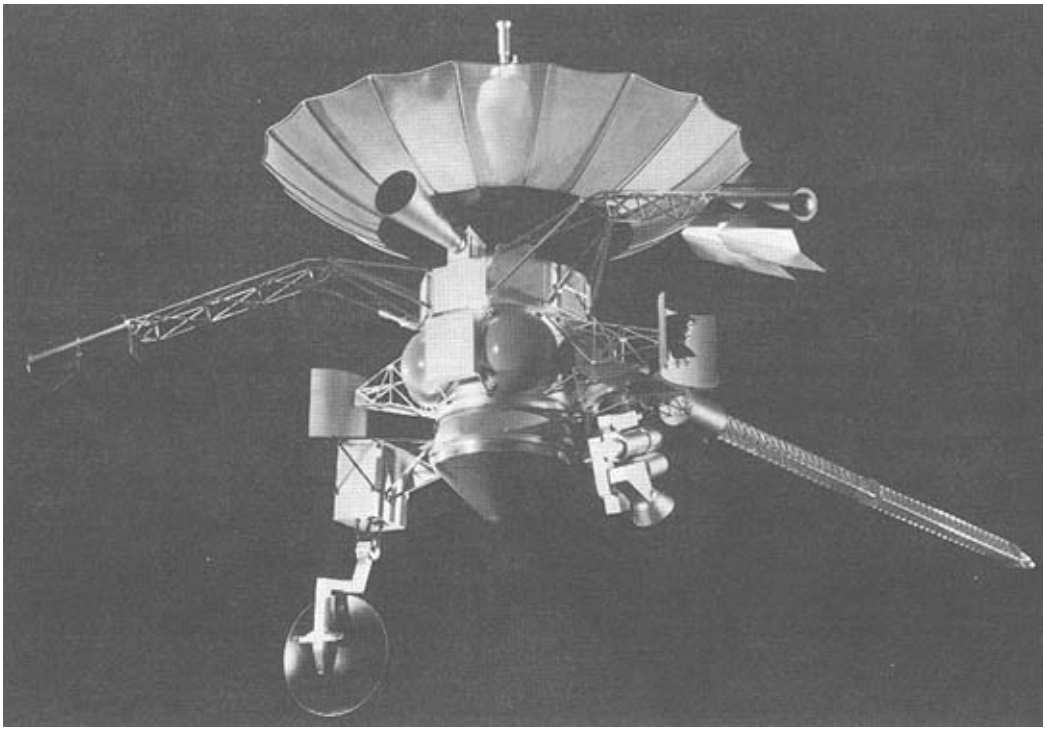

Figure 6–4. A model of the Galileo spacecraft with the probe visible at the bottom center. (JPL photo 230-222A)

Finally, computer system complexity is further increased by the number of science experiments on board and the fact that they are largely computer controlled as well. Eight of the nine instruments have microprocessors for control and data handling[79]. These have to communicate with the Command and Data System, itself containing six microprocessors. Attitude and Articulation Control has dual computers, and the probe also contains a dual microprocessor system. In all, Galileo contains 19 microprocessors with about 320K of semiconductor random access memory and 41K of read-only memory[80]. No unmanned spacecraft launched to date can approach Galileo in the power and size of its on-board computer network.

JPL took great care in the selection of the computer systems for Galileo. Procurement of the systems for the Command and Data handling equipment and the attitude control equipment proceeded separately but in a somewhat coordinated fashion. In the case of the Command system, a 1977 study examined using the existing Voyager computers, the National Standard Spacecraft Computer (NSSC-1), and some form of a microprocessor distributed system like the Unified Data System (UDS), then a research program at JPL aimed at complex long-duration space missions[81]. Although a lot of pressure was exerted on Galileo's builders to choose either the existing equipment or the NSSC-1, cost factors favored the UDS as the basis for the Command and Data Subsystem[82]. Similarly, the attitude control group was pressured to use the NSSC-1, but the desire for floating point and greater power defeated that idea. Since the star tracker is in the spun section and thus moving, complex coordinate transformations must be calculated, and the NSSC-1 was not up to it[83].

With new computers needed for both major controlling subsystems, JPL carefully explored memory requirements and software development prospects. Prototype programs were written in HAL/S and FORTRAN for the command computer and the attitude control computer[84]. Ideas for the content of the programs came from Voyager experience and the executive written for the NSSC-1. The project office originally specified that HAL would be used for programming all flight software. When irreducible inefficiencies appeared in the compiler bought for the command and data computers, HAL was abandoned for that system and replaced with "structured macros"[85]. HAL was retained for the other computer system. Although most microprocessors in the scientific experiments are coded in assembler, one is programmed in FORTH, so high-level languages finally appeared on unmanned spacecraft. The project office set a limit of 75% memory usage at launch (later raised to 85% for the attitude control computer only) and 85% load at Jupiter insertion. Officials hoped to avoid the tight memory problems associated with earlier missions and even asked the Shuttle software office for advice in setting these limits[86].

JPL considered software crucial to the success of the overall Galileo mission. As of 1985, Neil Ausman of JPL was in charge of all software, both flight and ground support, and he reported directly to project manager John Casani. Patricia Molko, also of the project office, wrote a standards document for software development. At one point, Howard W. Tindall, first introduced in the chapter on the Apollo computer systems, was brought in to serve as a consultant. He found that in some ways JPL was "going through the same problems that we did when we developed our original large programs"[87]. However, in some respects Galileo is more complex than Apollo because of the number of intercommunicating computer systems. Thus, size is not the only factor contributing to the difficulty of writing the software.

The Galileo probe mission is handled by NASA Ames Research Center, and the entry probe was assembled by Hughes Corporation. Even though it contains a dual microprocessor system, its function is primarily confined to sequencing, and its architecture is similar to systems already described on Mariner, Viking, and Voyager.

Galileo's Command and Data Subsystem Origins: The UDS

STAR was JPL's foray into ultrareliable computer research in the 1960s; the UDS was its 1970s counterpart. David Rennels, who had been instrumental in the STAR program, led in developing the system. Assisting him on the hardware side was Borge Riis-Vestergaard, a visiting scientist, and Vance C. Tyree. Frederick Lesh and Paul Lecoq did the software. One reason the UDS project started was the desire to develop a new architecture for flight computers that would reduce life cycle costs[88]. Another impetus came from 1973 studies of distributed systems done in support of Voyager and by the Air Force[89]. Distribution of functions among several computers on Voyager has been shown to be a natural result of requirements. The Air Force study found that avionics tasks are better handled by partitioning and using dedicated computers for specific functions. Since microprocessors became commercially available at about that time, they were recommended for use in such distributed systems.

Rennels' UDS project explored the difficulties in tying multiple computers together in a flexible manner. He defined an architecture using two levels of computers. Individual computers and associated memories at one level were called "high-level modules (HLM)." These computers controlled system-wide functions such as the data buses and fault detection. Other computers were located at specific subsystems and were called "terminal modules." They controlled one functional area such as engineering instruments or attitude and articulation. Each module had its own processor and memory.

Figure 6–5. Galileo and its booster being deployed from an orbiting shuttle. (P25722AC)

Communication between modules was accomplished on a bus that carried data from one memory to another. By using direct memory access for all intercommunication, processor resources other than for transfer commands were unaffected. HLMs did not have I/O capability other than to the terminal modules via the bus. All input and output to spacecraft systems and the ground was handled by the terminal modules.

Some influence from the STAR project can be noted in that Rennels kept critical functions highly redundant and simple. Reconfiguration after failures reflected STAR concepts[90]. In order to avoid a potential single-point source of failure, there was no central bus controller in the UDS. Each of the HLMs controlled a separate bus, but only one bus was needed to support all processors because any HLM

could transfer data between any two memories[91]. The breadboard built for the UDS project had three HLMs and three terminal modules, so it had three buses as well[92]. Failure of a HLM caused its functions to be accepted by the remaining ones. Reliability is obtained by such reallocation of functions to resources, making this a highly fault-tolerant system.

One advantage of distributed systems is that interfaces can be simpler than in a system using a central computer. Each local computer is responsible for its own timing and control. Only on and off commands and data transfers need be made between machines[93]. System-wide synchronization is accomplished by providing all processors with a real-time interrupt signal. By using a cyclic interrupt, the complexity attendant with priority interrupt systems is avoided[94]. Every 2.5 milliseconds, a signal is sent to all components. Basically, every processor has to be finished with its current processes before the next interrupt occurs[95]. Data transfers and scheduling of tasks can be timed using the periodic interrupt.

Key advantages of the UDS concept are that expanded requirements can be handled by adding terminal modules, software can be highly specialized and distributed, and fault tolerance is very high. Availability of flight-capable microprocessors in the mid-1970s made it possible for JPL to seriously consider the UDS as a competitor with NSSC-1 and the old Voyager equipment. Its flexibility and potential as a permanent architecture for space flight helped its case.

NASA chose RCA's 1802 microprocessor for the Galileo implementation of the UDS. A CMOS-type device, it was "nobody's favorite choice"[96] but at the time (c. 1977) was considered to be the only microprocessor suitable for spaceflight. Recall the use of CMOS components in the processors and memory in the Voyager Flight Data System. Similar advantages accrue with the use of CMOS microprocessors: low power requirements (30 milliwatts) and tolerance of a wide range of voltages[97]. However, some disadvantages had to be dealt with. CMOS chips are especially susceptible to damage from electrostatic discharges[98]. RCA 1802s are slow, with a 5-microsecond cycle time and an average of two cycles per instruction. In contrast, the discrete component Voyager CCS had a 1.37-microsecond cycle, making it faster for functions that did not require multiple cycles[99]. Speed has been a major constraint to the software development[100]. Carryover of the direct memory access cycle to Galileo from the Voyager Flight Data System alleviates this problem somewhat[101]. In general, making the six microprocessors "come together" has been much more difficult than originally expected[102]. Software is the most important component in achieving the success of the CCS.

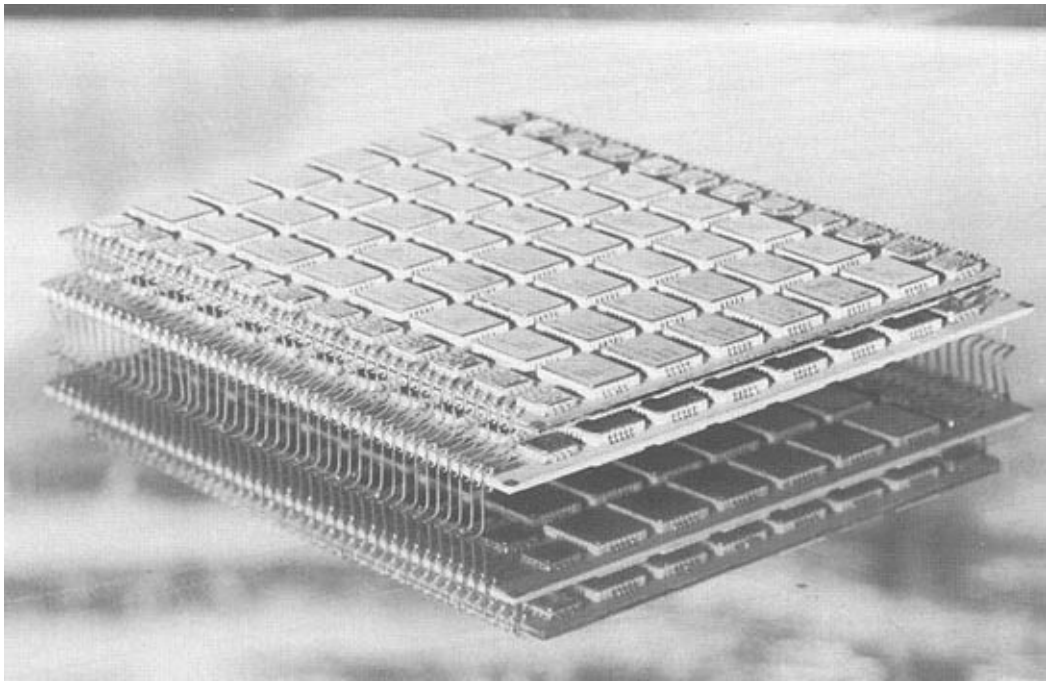

Figure 6–6. Circuit boards for the Galileo Command and Data Subsystem. (JPL Photo 360-1756)

Box 6-4: Evolution of the Command and Data Subsystem

Galileo's Command and Data Subsystem adapted UDS technology fairly directly. In 1978 designs, the Subsystem was shown as consisting of three HLMs and four low-level modules (LLMs), which were the realization of the tensional modules. Three buses were also present, each controlled by a HLM. Functionally there was one HLM dedicated to stored sequence control, one for real-time control, and the third as a spare. LLMs in that configuration handled sequencing, telemetry, status polling, and other subfunctions. Eventually one high-level processor was eliminated, but three buses remain, though one of them is used for test equipment only and will not function in flight[103]. Software architecture is now much different than the UDS.

One major difference from the UDS concept is the way the processors in the Subsystem are separated into redundant strings. Whereas reconfiguration after a failure was done by combining any of the remaining processors into a new control string in the UDS, on Galileo two basically identical strings are configured from the start, one backing up the other much like the backup processors on Viking and Voyager. Each HLM has 32K of memory and a bus controller associated with it. A LLM with a processor and 16K of memory for engineering control is connected to the string, along with a data bulk memory (DBUM) of 8K and a bulk memory (BUM) of 16K. These components are in the spun section of the spacecraft. Another LLM with 16K of memory is in the despun section, connected to the probe and the launch vehicle, among other functions[104]. This configuration of one HLM and two LLMs, a BUM and a DBUM is repeated in string B. Therefore, the total system consists of six microprocessors with 176K of semiconductor memory. About 12K of HLM memory is write protected and used for programs[105]. BUMs are used for auxiliary storage and buffering in that all new sets of commands are directly inserted into them from the ground and then redistributed by the software in the HLMs. Data memories serve as buffers for incoming science instrument data, again with direct memory access[106]. Commands and data are transferred on the buses using packets with three-word headers. Headers contain the code numbers of the source and recipient, the starting address in memory of the message, and fillers for timing. More than one address can be specified for a message, but usually there is only one recipient[107]. Data transfer is coordinated by the real-time interrupt. Odd-numbered real-time intervals are used for input; even numbered intervals for output.

As in previous missions, several operating modes are available for the Command and Data Subsystem. During cruise and other noncritical mission phases, one string is up and running and the other is in a quiet state. Otherwise both can be commanding in one of several ways. Dual string mode means that the strings are executing code concurrently and both send commands. Parallel mode is used for time-critical operations needing closer synchronization. Tandem mode is used during maneuvers. If a failure is detected in one string, the other halts the activity[108].

Developing the Command and Data Subsystem Software

Development of the software for this Subsystem consumed more time and labor than any previous unmanned spacecraft. Dr. John Zipse, the Cog Engineer for the Subsystem software, had an average of 12 full-

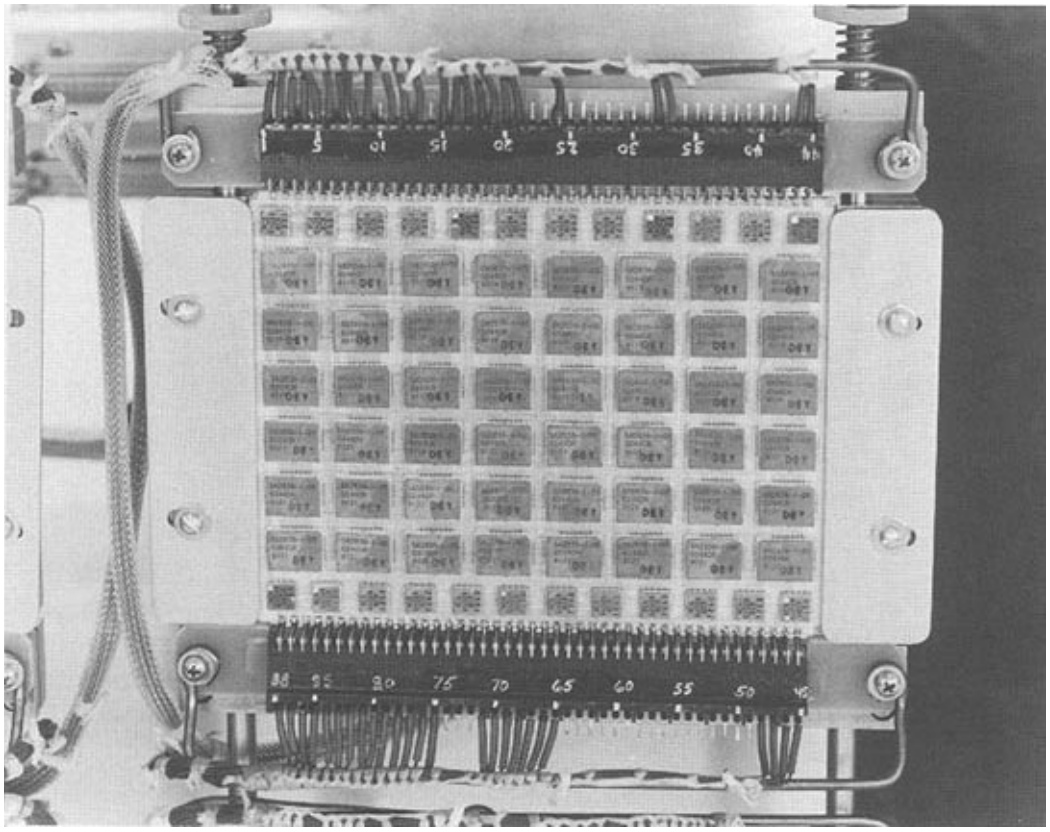

Figure 6–7. Memory modules for the Galileo Command and Data Subsystem. (JPL Photo 360-1704)

time software developers working under him at peak periods[109]. They shared five terminals hooked to an IBM System 370/158 computer on which the assembler and functional commands resided. Originally, HAL/S was specified as the programming language for the Subsystem[110]. A prototype compiler for the RCA 1802s was not successful, and HAL was dropped in favor of "structured macros"[111]. Called "functional commands" in the software documentation, they have names such as IF, ELSE, DO, ASSIGN, and others very similar to the statements of a high-level programming language[112]. These functional commands make up the "Virtual Machine Language" in which most of the software was written. Each command causes the execution of a prepared block of 1802 assembly code, much like a subroutine call. Project documents recognize three layers of language associated with the Subsystem: Level A is the hardware external to the 1802s that may provide input and receive output, Level B is the 1802 assembler, and Level C is the Virtual Machine Language[113].

Recognizing the complexity of the software, JPL instituted ever

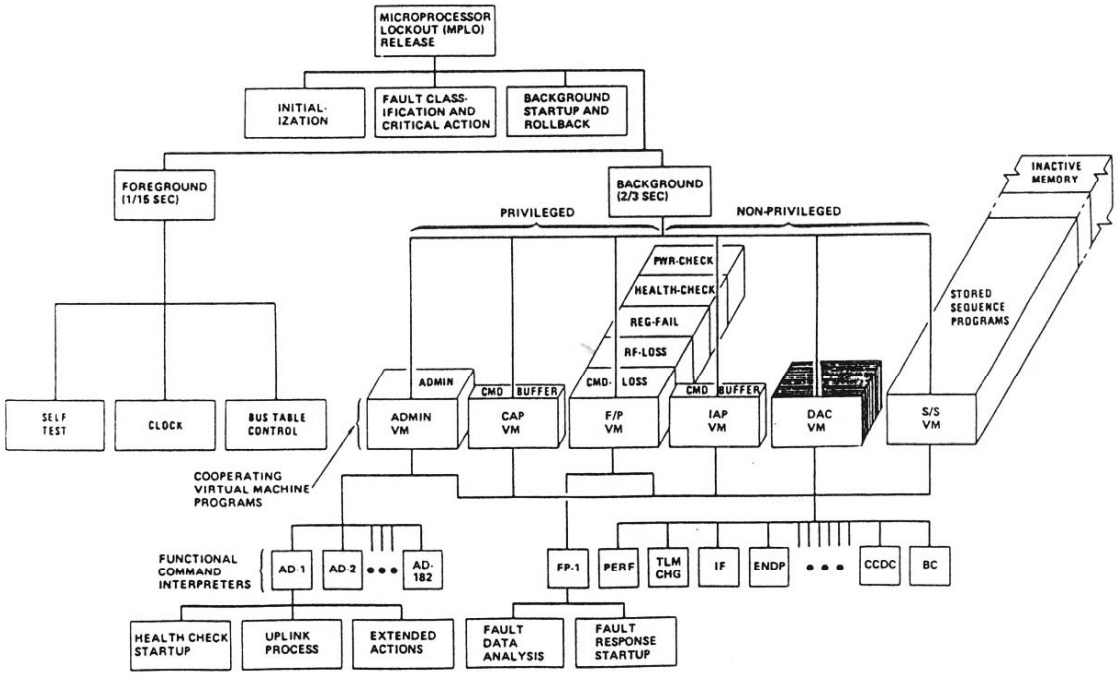

Figure 6–8. The software architecture of the Galileo Command and Data Subsystem High Level Modules. (From JPL, 625-340-006000)

more stringent development requirements. Preparation for the development process began with a "Galileo Software Thinking Group" which met in 1977–1978[114]. Programmers were ordered to keep software modules smaller than 150 assembly language statements and were reminded that simplicity was the highest priority[115].

Box 6-5: Command and Data Subsystem Software Architecture

Galileo's Command and Data software is considered a "hierarchical software architecture" and is divided into two sets of processes, with further divisions within them. Foreground processes are executed at each real-time interrupt, or every 1/15th second. They are a self test, a clock, and bus control. Background processes begin each 2/3 second, and are much more complex. Functions done in the background have been divided into six "virtual machines," a term of many meanings. In this case, three virtual machines are considered "privileged": the administration machine, the contingency action program machine, and the fault processing machine. These three machines consist of software that is always resident in the Galileo computers and is kept in the 12K of write-protected memory[116]. They are called privileged because the nonprivileged machines can be canceled if they do not complete processing before the end of the 2/3-second cycle, whereas the privileged machines can never be canceled. Nonprivileged machines include an immediate action program machine, a delayed action program machine, and a stored sequence program machine. So the nonprivileged machines are reserved for commands and sequence control, whereas the privileged machines are for executive and fault detection and correction. Nonprivileged software is to be updated about once a week in flight[117]. Originally, the immediate action programs were considered privileged, but with the addition of a contingency machine they were moved to nonprivileged status[118]. Software developers imagine that a "wall" exists between the privileged and nonprivileged machines. They consider that the nonprivileged software is more error prone because it is constantly changing, whereas the privileged software should have had a thorough exercise over several years in testing before the flight.

Execution of the virtual machine software is related to the 2/3-second interrupt. In each cycle the software goes through each virtual machine pending program stack and executes what is waiting. Many programs can be running in each virtual machine in each cycle, up to 10 in the administration and fault protection machines, for example. As mentioned above, the privileged machines always get to clear their pending programs, whereas the nonprivileged machines do what they can until the time is up.

Software described so far is resident in the HLMs. LLMs have specialized software for their particular functions, such as monitoring engineering instrumentation and talking to the launch vehicle. Data from those tasks needed by the virtual machines are passed on the buses during the 1/15-second interrupt.

Design of the software was done in a JPL-developed Software Design and Development Language that used statements similar to those in high-level languages[119]. Even with excellent documentation and tools, such as a hardware-based simulator for software validation[120], it takes (according to Zipse) a new programmer three months to be effective. Until Galileo has flown, no final evaluation can be made of the virtual machine architecture. Yet, future spacecraft requiring expandability and a high degree of flexibility could probably gain from using such an architecture as a complement to the UDS type hardware structure.

Galileo Attitude and Articulation Control Computer System

In terms of tasks, the Attitude and Articulation Control System has less to do than the Command and Data Subsystem, but it must perform its jobs faster and with more critical tolerances. During the discussion of the Voyager computers, it became clear that the attitude control system needed a fast-cycle, real-time software architecture running on a high-speed computer. Galileo's control requirements are much greater than Voyager's; the dual spin problem and more complex imaging equipment indicated from the beginning a need for a completely new computer system. The computer was to provide starbased attitude determination and control an inertially referenced, target-body-tracking scan platform[121]. Speed was the primary criterion for the new processor[122].

Kenneth Holmes, in charge of looking for the Galileo control computer[123], and the other engineers ran old Voyager attitude control programs on several processors. One of those processors soon proved itself superior: Itek's 2900 series[124]. Itek, now a-division of Litton Industries, built a computer known as the ATAC, or Advanced Technology Airborne Computer. Using 2900 series processors, each with 4-bit words, Itek assembled a 16-bit, low-power, flying minicomputer roughly equal in power to a Digital Equipment Corporation PDP-11/23. Navy aircraft use this computer, although its specific applications are classified[125]. ATAC's basic cycle time is 250 nanoseconds, or more than five times faster than the Voyager computer's cycle. However, the memory cannot cycle faster than 2 microseconds, so operations rates average 143,000 cycles per second[126]. Floating-point capability is a plus, and since it handles eight interrupts using microcode, there is no software overhead for real-time operation[127]. Another good feature is that its 16 registers are general purpose; none are dedicated as accumulators, program counters, address registers, and so on. Therefore, multiprocessing is made much easier. Further advantages to the computer are that special instructions can be added by the user for

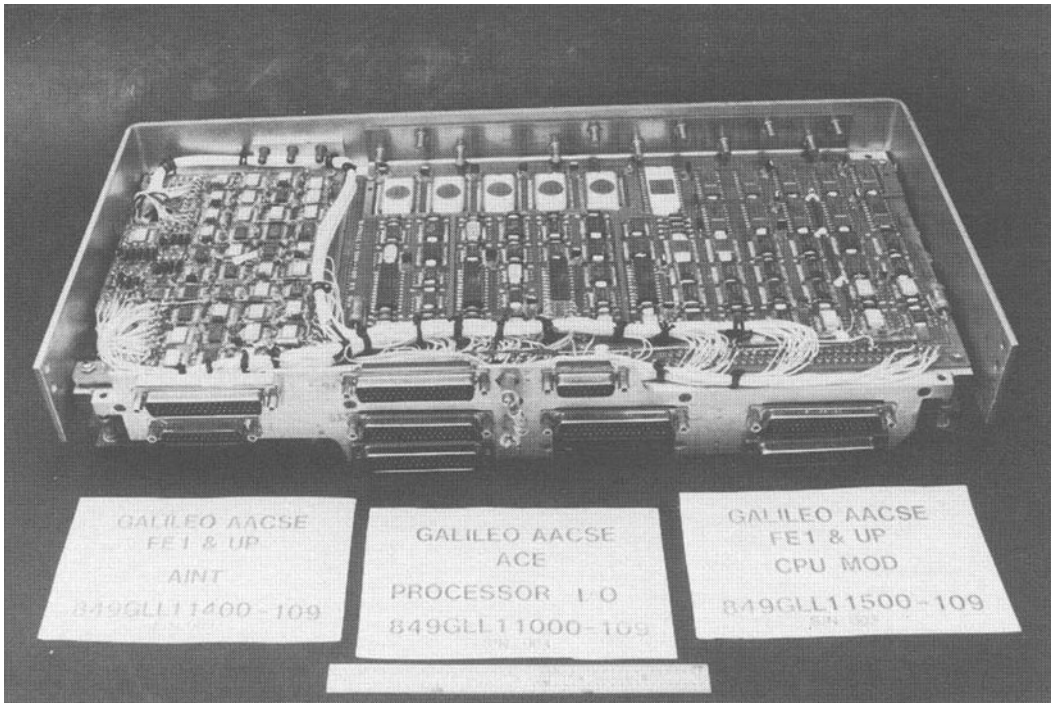

Figure 6–9. The central processor and input/output circuits of the Galileo Attitude and Articulation Control Subsystem. (JPL 230-1128bc)

specific applications. Four instructions added by the Galileo project saved over 1,500 words of code in the flight program[128].

The ATAC came with a considerable amount of software support. Target compilers were available for FORTRAN, BASIC, and HAL/S[129]. Galileo project management wanted a higher order language used in the coding, so HAL was adopted for the attitude control system. Unlike the Command and Data Subsystem, HAL was successfully adapted to the ATAC. Compilers developed by Intermetrics for the ATAC are about 12% speed inefficient, but 36% memory inefficient compared with assembler[130]. Apparently this was within acceptable limits, and the flight applications code is written in HAL, with the operating system in assembler. Edward H. Kopf, Jr. said, "We love HAL/S; we could never do it without HAL/S," even though he referred to it facetiously as "flight PL/I." Given the complexity of the resultant software, he was probably right that a high-level language was critical to success.

Attitude Control Electronics Software Organization

Implementing HAL/S on the ATAC involved creating a special operating system. Ted Kopf wrote GRACOS, or the Galileo Real-time Atti-

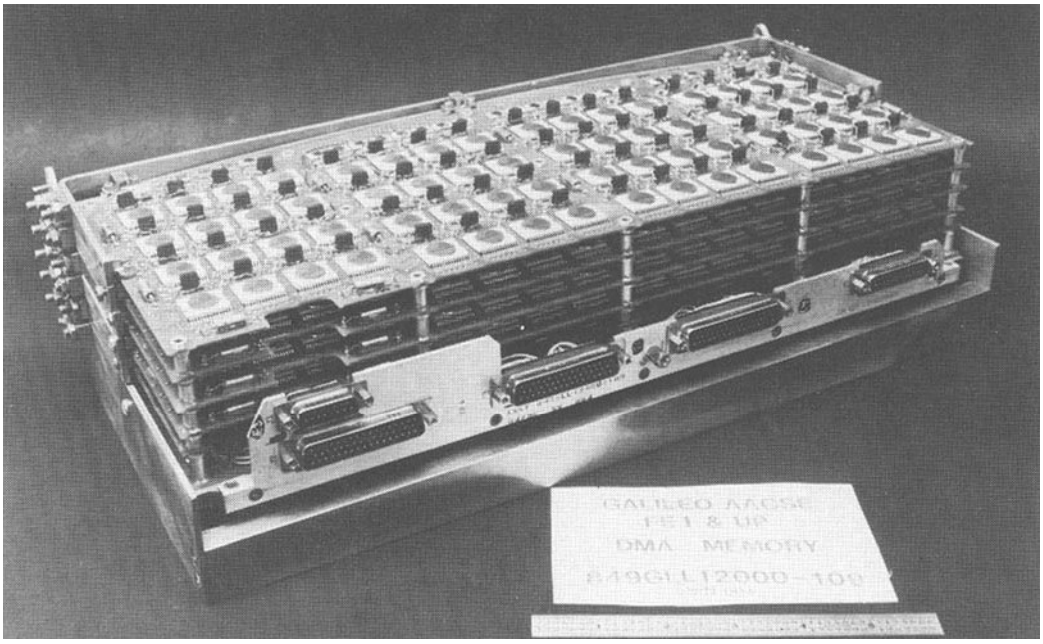

Figure 6–10. Memory modules of the Galileo attitude control computers. (JPL 230-1121bc)

tude Control Operating System, to accomplish that implementation[135]. GRACOS begins the operation of the software by scheduling a HAL/S module called STARTUP. Within STARTUP are statements that set up the concurrent processes necessary to do the attitude control and articulation tasks (a version of STARTUP is reproduced in the HAL/S appendix as an example of the language). Up to 17 concurrent processes may be running under GRACOS, with at least 10 up all the time[136]. STARTUP is given as much time as necessary to complete, and then the established processes begin at the next 1/15-second real-time interrupt[137]. Kopf wrote GRACOS to be mission independent. He avoided constraining the timing of rate groups and other things that would have been too specific[138]. Interestingly, if the fault-handling routine has to come in, it restores the registers of the failed HAL/S module where the fault occurred and tries again, very similar to the "roll back and try again" scheme in STAR[139].

Sanford M. Krasner, the Software Cog Engineer for the Attitude Control Electronics, reported directly to Brian T. Larman, the Spacecraft Flight Software System Engineer. Early in the project, five people were full time on the software development, several more than on Voyager[140]. The coding process was speeded up when HAL was used as the software design language[141]. To make development easier, Voyager structure was adopted whenever possible[142]

Box 6-6: Configuration of the Attitude Control Electronics

Galileo's Attitude and Articulation Control Subsystem necessarily has parts in both the spun and despun sections of the spacecraft. Most components are in the spun section, including the two redundant processors and 64K of memory. Called the ACE, for *A*ttitude *C*ontrol *E*lectronics, its despun partner is the DEUCE, for *De*spu*n* Se*c*tion *E*lectronics, a long way to go for an acronym. Communication between the ACE and DEUCE is across the rotary transformer. Since the transformer is not considered fully reliable, input or output to the DEUCE is not complete until an ACKNOWLEDGE interrupt reaches the ACE[131]. CMOS-type memories similar to those used in the Voyager Flight Data System are in the ACE. Sixty-six 1K chips are needed, two of which are actually permanent read-only memory. Those two contain the Memory Loss Recovery Routine written in ATAC assembler[132]. A 5-volt keep-alive current directly from the generators is constantly fed to the memories, but in the case of a destructive transient the Recovery Routine can restart the software after it has been repaired or replaced.

Planning for the system included careful memory sizing. Based on actual Voyager programs and extrapolations from them to handle the new requirements, a 1978 study thought 10K of memory to be sufficient. Using a policy of 100% margin and 75% limit at launch, 32K was eventually bought[133]. As noted above, a waiver to 85% full at launch was given. Ground computers can reprogram the ACE in flight by sending code to the off-line memory through the Command and Data Subsystem. As with its own LLMs and as pioneered in the UDS, the command computers can directly access the ACE memory. Commands can be placed in the active memory where they are "discovered" by the ACE software[134].

Ironically, the advanced nature of the new attitude control system and its control computer made it more vulnerable to space conditions than its predecessors. A potential disaster was averted when the "single event upset" was discovered and dealt with before launch.

The Single Event Upset Problem

Space environments are much harsher to electronics than the surface of the earth. Since circuitry essentially consists of hardware that moves electrons, creates and destroys magnetic fields, and emits waves of electromagnetic radiation, fields and particles of the types loose in space can affect the operation of electronic equipment. Ironically, the miniaturization of components has made electronics more sensitive to interference. One possibility that concerns computer designers is the

effect of highly charged particle impacts on memory cells. If a particle has sufficient energy to change the information stored in a bit, it can affect the software in a potentially disastrous way. Such a spurious change is called a single event upset (SEU). Sufficient numbers of particles can cause so many bits to change states that the software fails. Since primary and backup memories are equally vulnerable, simple redundancy is not a solution. Error-correcting codes that test for random bit flips exist but require storage and processing time not always available on a spacecraft. Additionally, bits in the processor can be affected during execution, so the problem is not limited to memory.

Galileo's processors and memories were chosen in 1977. Voyager had not yet reached Jupiter, so hardware decisions were based on 1973–1974 Pioneer data[143]. Nothing was known about SEU vulnerability, so no space for error detection and correcting codes and no provision for special shielding was made. Some incorrect imaging commands sent by sequencers in the Pioneers were later tagged as SEUs. Voyager's clocks were slowed by Jovian radiation so that the computers were forced out of synchronization occasionally[144]. By 1980–1981, the nature of the SEU problem became apparent. Sulfur ions from Jupiter's volcanic moon, Io, were being whipped up to high energy by the Jovian gravity. In 1982, Galileo Project Chief Engineer B. Gentry Lee was assigned the job of determining how bad the SEU problem could be and finding a solution. Lee arranged for cyclotron tests at the University of California's Berkeley campus in which computer and other electronic parts were submitted to bombardment by high-speed particles. Results indicated that the 2901 chips used in the attitude control computers were highly SEU sensitive, with 20% 50% of hits causing probable software failures[145]. RCA 1802s used in the Command and Data Subsystem were actually much less sensitive, being of older, and thus less dense, technology.

Attitude control engineer Kopf commented, "It is not worth flying the mission if you cannot get rid of the SEU problem." Failures were most likely at the most critical part of the probe mission when the orbiter is very near Jupiter. In order to avoid possible further delays in an already much postponed mission, Lee searched for solutions along two tracks. One solution would use a radiation-hardened processor built by Trecor called the RHEC—Rad Hardened Emulating Computer—1750A. Even though it is an emulator capable of imitating the 2901, a new retargeted HAL compiler would be needed. The cost of this solution would be $20 million. Lee's second solution was to contract with Sandia National Laboratories to custom make radiation-hardened 2901s. No software needed be changed, just new ICs were necessary, and they cost $5 million. Due to cost considerations and the inherent attraction of retaining the already created and largely validated software, the Sandia solution was chosen[146]. As a footnote, it is interesting

that if the Galileo had launched on time, a sufficient understanding of the SEU problem would not yet have been available, and a doomed spacecraft carrying an unknown time bomb would have been traveling toward an unfriendly Jupiter waiting to hurl ion thunderbolts at it.

FUTURE UNMANNED SPACECRAFT COMPUTERS

Distribution of computers aboard spacecraft has now been done several times. Both Voyager spacecraft inherited command computers from the Viking project. Computers for specific functions such as attitude control and data formatting were added in response to increased requirements. The result was a functionally distributed system of processors. Galileo's project managers also adopted the concept of functional distribution, assigning microprocessors to control attitude and, in the lower-level modules of the Command and Data Subsystem, to connect to engineering and other instruments, including the scientific experiments. Additional innovations on the Galileo spacecraft centered on the development of virtual machine software, which distributes functions over several processors.

Advancing microprocessor technology makes the continuation of the concept of single function computers more attractive. At JPL, plans are currently under way for the Mariner Mark II, which will be the deep space version of the Multimission Modular Spacecraft developed by Goddard Space Flight Center for earth orbital operations. Using the same concepts of a standard bus and modular equipment, JPL hopes to reduce mission costs to $400 million each, about half the price of Galileo[147]. The staff is exploring the use of the C programming language, a very powerful tool, for the new spacecraft. Future missions seem certain to use multicomputers, with internal networks similar to Galileo's. In the 15 years since the first primitive programmable sequencers flew with 128 word memories, JPL spacecraft have grown to carry *2,500 times* more memory. Progressing from simple counting to complex coordinate transformations in such a short time is remarkable, and the application of computer power to each spacecraft function will make for ever more remarkable gains.