THE VIKING ORBITER UPLINK COMMAND GENERATION AND VALIDATION VIA SIMULATION*

Maurice B. McEvoy
Jet Propulsion Laboratory, Pasadena, CA.

ABSTRACT

The Viking Orbiter spacecraft and its predecessor Mariner spacecraft are increasingly complex vehicles that are automatically controlled by on-board computers. These on-board computers are flexibly programmed, and periodically re-programmed during flight, to execute complex sequences of science and engineering events. Validation of these remotely controlled updates to the four Orbiter on-board memories is a difficult task that relies heavily on simulation of the effects of the updates prior to transmission. This paper describes the Orbiter uplink command-generation and validation process that is currently being simulated in Viking mission operations, and the simulator program design considerations, design features, level of detail, language selection, resource constraints, implementation, testing, calibration, performance, and experience to date.

INTRODUCTION

The Viking Project is the latest step in the National Aeronautics and Space Administration's continuing program to explore the planet Mars. Two Viking spacecraft, each consisting of an Orbiter and a Lander, are currently en route to Mars. Viking 1 was launched on 20 August 1975 and will arrive at Mars on 19 June 1976. Viking 2 was launched on 9 September 1975 and will arrive at Mars on 7 August 1976. Both spacecraft are operating normally in a cruise configuration, with their programmed sequences being updated once per week. Once in Mars orbit, activity will increase, with each Orbiter being updated on alternate days.

Navigation, command, and control of Viking are performed from the Space Flight Operations Facility at the Jet Propulsion Laboratory (JPL) in Pasadena, California. Uplink commands are generated at JPL and are routed to the spacecraft through one of the Deep Space Net (DSN) stations in Goldstone, California; Madrid, Spain; or Canberra, Australia. Downlink telemetry, tracking, ranging, and doppler data from both spacecraft are received by the DSN, which routes it to JPL where it is processed, displayed, and monitored by on-line analysts.

The design of the Viking Orbiter (Fig. 1) is derived from the Mariner 9 spacecraft that photographically mapped Mars between 14 November 1971 and 27 October 1972. The on-board computers,

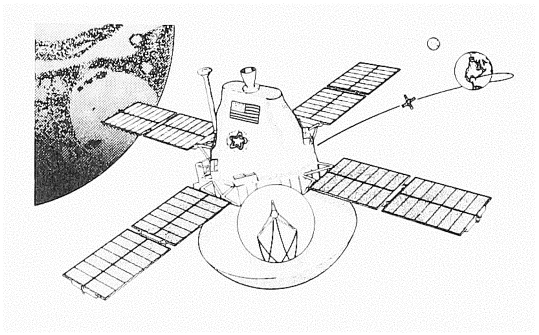
Figure 1. Viking Spacecraft

on-board data storage, ground-based sequencing software, and the spacecraft size and weight have been upgraded significantly.

The orbital nature of Viking and the associated long periods of intensive operations and data acquisition present unusual development and mission-operations organization problems. Design of the sequencing software was strongly affected by the need to rapidly adapt to changes or failures of on-board equipment during the eleven-month cruise and five months of planned orbital operations. The mission design must also be adaptable to unexpected changes in the external environment, such as the planet-wide dust storm that was present when Mariner 9 reached Mars. The Orbiter sequence-generation procedure evolved from and is largely patterned after the Mariner 9 operational mode (Ref. 1).

Verification of complex systems is difficult at best, but is increasingly difficult for systems whose operation is dependent on changeable software. System adaptability is constrained by available operational support capabilities; and the extent of subsequent re-verification, prior to its operational use, determines the associated risk. Simulation of uplink commands prior to their transmission can help detect and analyze erroneous or harmful memory loads.

This paper presents the Orbiter simulator's capabilities and considerations made during its design, implementation, testing, and use in supporting real-time, high-risk decision making in actual mission operations. The features of the simulation

*This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS 7-100, sponsored by the National Aeronautics and Space Administration. The Langley Research Center is Project Manager for the overall Viking Project and is responsible for Flight Operations; JPL is responsible for the Orbiter; MMA is responsible for the Lander.

software described in this paper result from the mission that it is designed to support, its planned operational use, its interaction with other programs, and the experience gained from previous JPL flight projects.

Viking mission requirements, hardware, software, and sequence-implementation techniques that affect the simulator's design and use as a validation tool are also described. A similar, but separate, set of software that supports the Viking Lander is not discussed.

REQUIRED ORBITER SEQUENCING

After separating from their Centaur launch vehicles, each Orbiter opened its solar panels and performed a controlled search for the Sun, using its inertial reference unit for spacecraft attitude reference. Once Sun-acquired, they performed a roll search for the star Canopus. Deep Space Net (DSN) tracking ranging and doppler data were used to determine required trajectory corrections. Each Orbiter was then maneuvered to align its rocket engine with the desired velocity to be gained, and its engine was fired until the desired velocity was achieved.

During early interplanetary cruise, the Orbiter's science instruments were checked out and calibrated, and the high gain antennas and the scan platforms were calibrated to improve their pointing accuracy. Calibration of the Visual Imaging Subsystem (VIS) began by recording and playing back star/star and Earth/star picture pairs. Six photopairs of the Pleiades are planned for additional geometric calibration. An engineering test was run on one Computer Command Subsystem processor in an attempt to diagnose the cause of an anomalous memory checksum error that was observed shortly after the spacecraft separated from its Centaur launch vehicle. A pre-encounter science sequence will obtain global distributions of temperature and water vapor. Beginning about five days before Mars arrival, the Orbiter will provide video images of the planet and optical navigation images of the Martian moon Deimos in preparation for the Mars orbit-insertion maneuver.

As the spacecraft nears Mars, the Orbiter will maneuver to point its engine for the approximately 40-minute burn that will insert the spacecraft into orbit around Mars. Inopportune approach-trajectory tracking geometry may necessitate a late update to previously loaded orbit-insertion parameters within hours of the automatic maneuver execution. The planned orbit has a Mars synchronous period of 24.6 hours, a periapsis altitude of 1500 km, and an apoapsis altitude of about 32,600 km. Once in orbit, each Orbiter will perform reconnaissance of preselected primary and secondary landing sites, using its scan platform-mounted science instruments. Approximately ten orbit trim maneuvers will achieve the desired synchronous orbit geometry for Lander separation, station-keeping to maintain Lander support, and period-trims for conducting Mars observations. The Orbiter will continue to orbit Mars throughout the primary mission, which ends at Earth-Mars conjunction in November 1976. During this time, atmospheric experiments and surface observations will be conducted along with radio science experiments.

Within 50 days after Mars orbit insertion, each Lander will be separated from its Orbiter and will begin its deorbit and landing. Each Lander will examine the structure and composition of the Martian atmosphere during descent and will begin a 60-day surface investigation following landing. The Orbiters will continue to use their instruments to acquire science data on the landing sites and their surroundings and to survey other areas of Mars to study the physical and dynamic characteristics of the planet's surface and atmosphere. Sun and Earth occultations may occur during the mission, depending on the geometry of the selected orbit. Orbital operations will be restricted for orbits with Sun occultations in order to avoid excessive battery discharge.

Flexible pairing of spacecraft and communication links and the large-scale redundancy of spacecraft components yield a high probability of success for the Viking mission as a whole. This flexibility will be valuable should trouble develop with any of the spacecraft. Use of this flexibility, however, requires adaptable ground-based sequence design, implementation, and validation software.

Science Instrument Sequencing

Three types of optical instruments will view the surface of Mars from a two-axis scan platform (Fig. 2), with their fields-of-view (Fig. 3) aligned along a common axis to facilitate correlation of observations. These are a pair of high-resolution television cameras, an infrared atmospheric water detector, and an infrared radiometer for atmospheric and surface thermal mapping. Together, the instruments will scan a swath on the surface below as the Orbiter swings around Mars. The infrared instruments will pinpoint warm, wet places which can then be checked visually (using photographs) for geological interest and suitability for landing.

The scan platform is used to point the instruments to permit the Orbiter to remain locked on celestial references, whenever possible; however, when the scan platform pointing range is insufficient, Orbiter turns are executed to achieve the required pointing. For low-altitude observations, single or multiple swaths of contiguous imaging photopairs are acquired, using Orbiter ground track motion to produce area coverage with any one swath

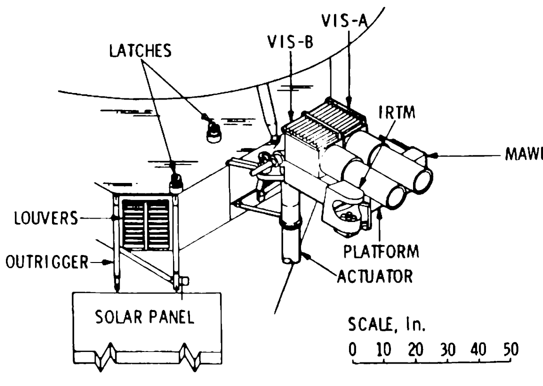

Figure 2. Scan Platform Mounted Science Instruments

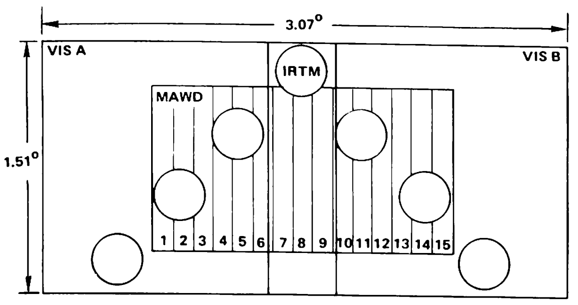

Figure 3. Science Instrument Fields-of-View

and using the scan platform to slew between each photopair to increase the coverage. At an altitude of 1500 kilometers, the cameras can photograph contiguous, non-overlapping squares (80 kilometers on a side) in swaths about 500 kilometers long, with resolution of about 40 meters. Once the Landers are on the surface, large-scale observations from the Orbiters may be correlated with fine-scale measurements obtained by the Landers.

The Visual Imaging Subsystem (VIS) consists of two identical 475-mm focal length telescopes, each with a 1.5-inch magnetically focused, slow-scan vidicon camera, mechanical shutter, and filter wheel. The cameras are operated alternately, with the capability to shutter one camera every 4.48 seconds.

When its shutter is open, an electrostatic image of the scene below is formed photoelectrically on its vidicon. The image is scanned by an electron beam, and the resulting image intensity is sampled and digitized into 1056 lines of 1182 7-bit "pixel" elements. These digital data are transferred to the Flight Data Subsystem (FDS), where it is multiplexed with appropriate identification, synchronization, engineering data, and other science data to form a composite data stream. This composite data stream is then transferred to the Data Storage Subsystem (DSS) and stored on magnetic tape for subsequent playback at a rate commensurate with communications link performance.

During actual operation, each camera takes one picture every 4.48 seconds. The cameras operate in response to 11-bit digital control words that are loaded into the FDS memories by uplink command. They specify exposure durations between 3.182 msec and 2.66 seconds, stepping of the filter wheel, light flood, amplifier gain state, and dc video offset. A zero-exposure duration indicates that no picture is to be recorded and can prepare the sensor for a low residual image.

The Mars Atmospheric Water Detector (MAWD) is an infrared spectrometer that will map the distribution of water vapor over the planet for landing site certification and will provide data on latitudinal and diurnal variations of water vapor.

Fifteen instantaneous fields-of-view (Fig. 3) are sampled sequentially during a 4.48-second interval. The samples measure radiation at five wavelengths in the 1.38-micrometer water vapor absorption band. Radiation from the planet is focused by a small (2.5 cm, f/5) input telescope which views the surface via an external scanning mirror.

Transitions between states can be made by means of Discrete Commands (DC) directly to the MAWD or by a Coded Command (CC) which resets a control word in the FDS memory.

The Infrared Thermal Mapper (IRTM) is a 28-channel infrared radiometer that will aid in landing-site selection, monitor the regions surrounding the Lander, and provide spatial and temporal distribution of surface and upper atmospheric temperatures and thermal balance. Some IRTM measurements will be taken at the same time and over the same region covered by VIS, while others will be independent of VIS.

Each of the four IRTM telescopes contains a field stop which defines seven separate fields-of-view, each with its own detector. Each telescope is aligned such that the seven fields-of-view are superimposed as shown in Fig. 3.

Infrared radiation from Mars, from deep space, or from the IRTM internal reference surface, is selected by positioning the scan mirror to one of its three discrete positions. The radiation is reflected from the selected source by the mirror into the cluster of four cassegrain reflecting telescopes. Three of the telescopes are f/4, 2.3-inch diameter, and the fourth is a f/5.3, 1.5-inch diameter. Optical elements within each of the telescopes select the spectral bands 6-8, 0-9.5, 9.5-13, and 18-24 micrometers, and measure atmospheric temperature at 16 micrometers and reflected radiation (albedo) at 0.3-3 micrometers.

ORBITER COMMAND AND CONTROL

Orbiter sequences are implemented by loading the Computer Command Subsystem's (CCS) memories via uplink commands (Fig. 4). The CCS flight software controls the hardware necessary to decode the four-bit-per-second uplink data into 50-word buffers (Fig. 5), perform single-bit error correction, and reject commands having more than one error. Discrete commands (DC's) and 14-bit coded commands (CC's) are issued to control Orbiter subsystems

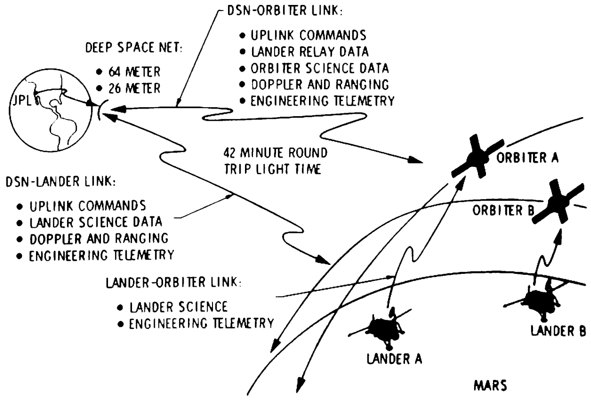

Figure 4. Communication Links

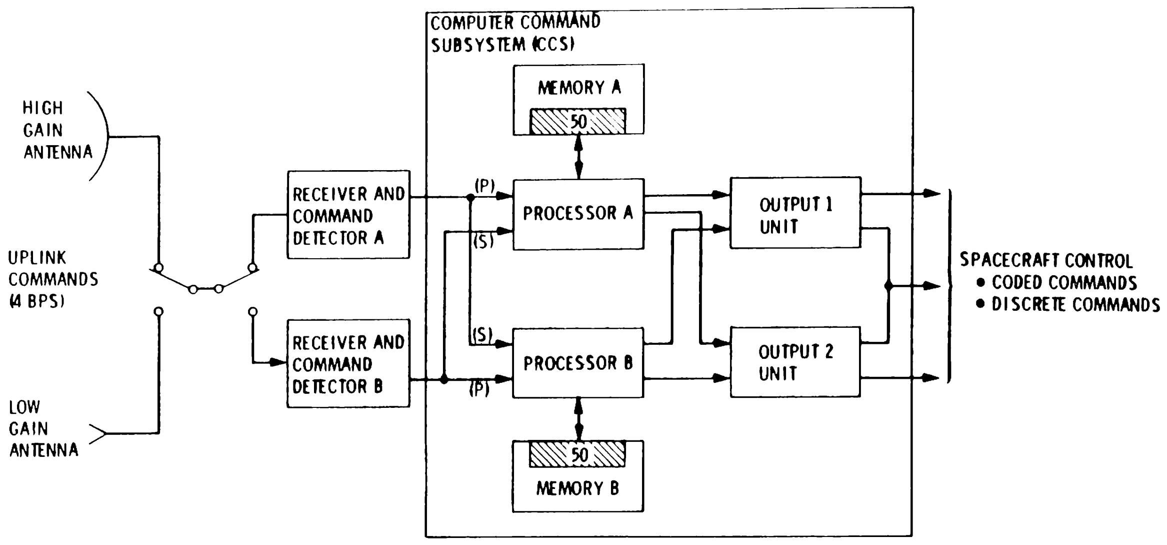

Figure 5. Orbiter Uplink Command Processing

under CCS program timed control (Fig. 6). Tables (Fig. 7) drive CCS routines to execute Orbiter sequences. Real-time uplink commands can also be used for emergency situations or for convenience.

Automatic CCS routines are provided to correct problems that may exist without requiring intervention from the ground. These routines are initiated by interrupts and are selectively enabled and disabled during different phases of the mission by modifying their entry linkages.

Single or block immediate commands are executed directly from the command buffer in CCS memory. Programmed blocks are used to load new sequences into CCS and FDS memories. Conditional execute blocks are used to activate previously loaded sequences if, and only if, all flight-software validity checks were satisfied during the update. Otherwise, a telemetry word is sent downlink to notify the command team that at least one block requires re-transmission.

The CCS can operate in individual, parallel, or tandem modes. In the individual mode, the

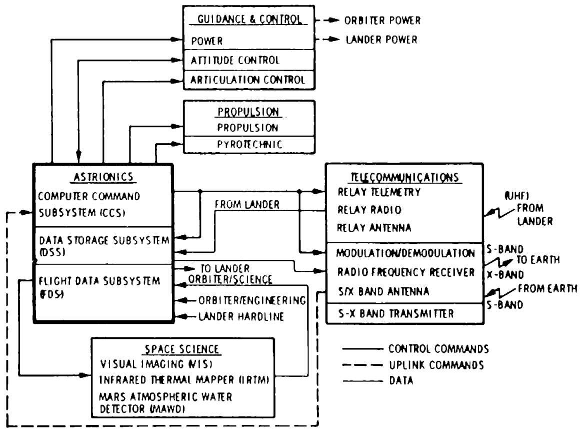

Figure 6. Orbiter Functional Block Diagram

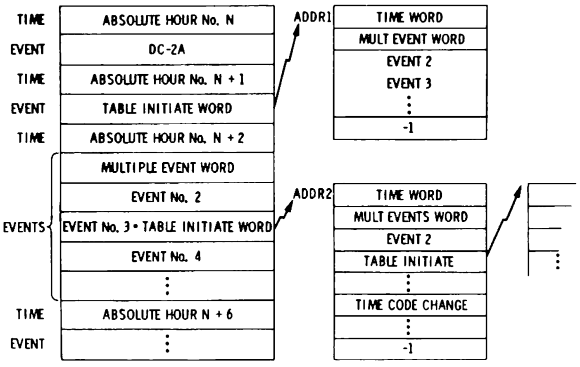

Figure 7. Computer Command Subsystem Sequencing Tables

processors operate independently, using available output units for command executions. The parallel mode is used when redundancy is required to insure that a command output occurs. Parallel operation in essentially the same sequence as that for the individual mode; however, both processors perform the same software routine, each accessing a different output unit. The tandem mode is used when redundancy is required to insure that a correct command output occurs only at the required time. Both processors and one output unit are required to execute a tandem event.

ON-BOARD DATA HANDLING

Figure 4 depicts possible telemetry links between the spacecraft and the Deep Space Net. Lander telemetry can be transmitted directly to Earth by the Lander, recorded by an Orbiter's Data Storage Subsystem (DSS) and subsequently relayed, or relayed via an Orbiter in real time. The Orbiter relay link is the primary method of transmitting Lander data back to Earth. Orbiters can record or relay data from the Landers whenever the planet's rotation carries the Landers to the side of Mars away from Earth. Data can be recorded on either of two identical and independent Digital Tape Recorders (DTR's) on each Orbiter. Each DTR is capable of storing 5.6×10^8 bits of Orbiter visual imaging data and 8×10^7 bits of IR, engineering, and/or Lander data. Viking science data return is limited by the usable downlink data rates. Bit error rates, when at Mars distance, limit the Orbiter's high-rate telemetry channel to 8 kbps for expected telecommunications performance and 4 kbps under worst-case conditions.

The Flight Data Subsystem (FDS) selects and samples engineering inputs (Fig. 8) in accordance with flexibly reprogrammable measurement identifiers in its memories. These engineering data are required to monitor the status and performance of the Orbiter and is continuously transmitted on the low-rate channel at either 8-1/3 or 33-1/3 bps. Engineering data can also be telemetered on the high-rate channel and/or recorded on the DDS at 1 kbps.

Orbiter science data and Lander relay data are sent downlink on the Orbiter's high-rate channel. Visual imaging data are recorded simultaneously on seven tracks of one of the digital tape recorders for later playback, one track at a time. About 40 minutes of playback time are required to return a single imaging frame at the 4-kbps data rate. Infrared science data are normally acquired whenever VIS data are obtained with the lower rate (1 kbps) IR data being multiplexed with the VIS stream. Interleaved 1 kbps IR data and 1 kbps engineering data can also be placed on the high-rate channel. IR science data constitutes the usual FDS high-rate output.

CCS memory readouts can be obtained by replacing engineering data with readout data; and FDS memories are continuously being read out whenever the high-rate telemetry channel is on.

SEQUENCING STRATEGY

Viking operational complexity is due both to its ambitious mission and to the many operational modes of each of its subsystems. The numerous operational modes are required to insure adequate work-arounds for possible failures during its 17-month primary mission. In fact, Viking Project Directive 6 states: "No single malfunction shall cause the loss of data return from more than one scientific investigation,... and no single malfunction shall cause the loss of all engineering telemetry data." This flexibility increases the resources required to operationally adapt to changes in a near real-time environment. In practice, command-file generation and validation are limited by available personnel, computer time, on-board memory space, and time required for validation. The Viking 75 Project Mission Rules (Ref. 2) and the Viking 75 Orbiter Block Dictionary (Ref. 3) documents were specifically written to describe constraints on the flight use and operation of the Orbiters.

The Project Mission Rules define and govern operational activity of ground and spacecraft systems. The rules were derived from analysis of mission equipment configuration, flight team procedures, operational strategies, critical periods analysis, tests, and Viking Project objectives. They are limited to those rules where a choice of action is possible and include responses to nonnominal operations. The rules include hardware, software and procedural constraints, or rules which limit the operation of the Viking spacecraft. The rules aid the Viking Flight Team in the complex decision-making process required to conduct Viking Flight Operations.

The mission design and all subsequent updates are implemented, using Orbiter Blocks. Except in

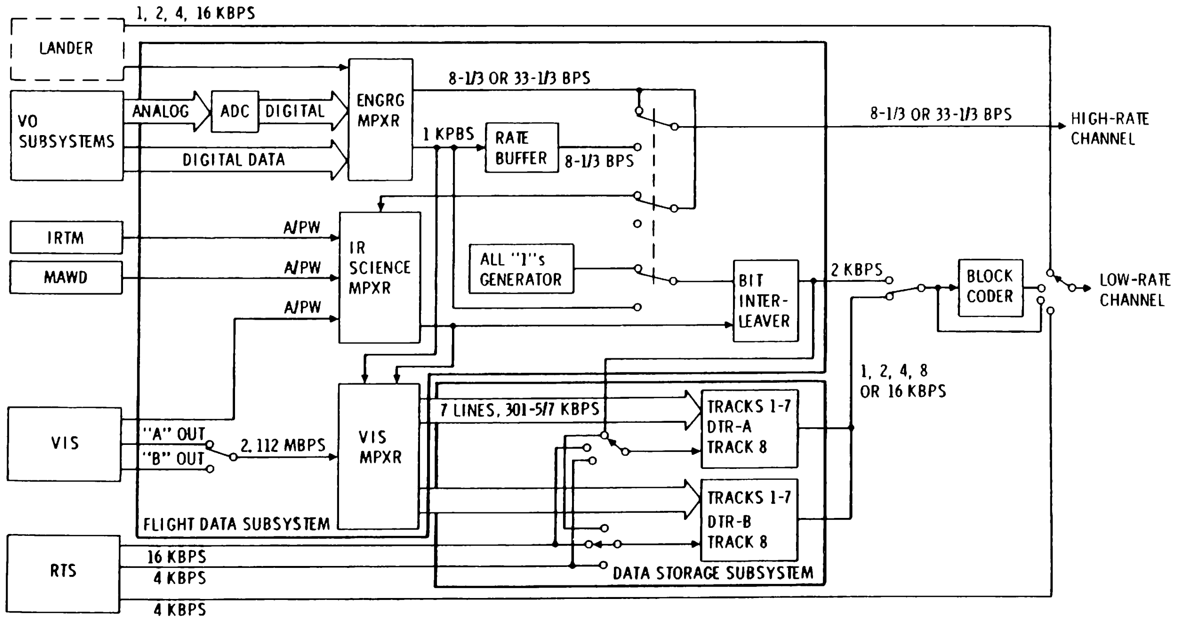

Figure 8. On-board Data Handling

Spacecraft/Orbiter emergencies, all flight-operational sequences are implemented, using validated Orbiter Blocks. Each Orbiter Block and all subsequent modifications are tested in a series of Block/Algorithm/Macro runs that are validated, using the simulator, prior to their use in flight.

An Orbiter Block (Fig. 9) is a group of Orbiter commands and/or events, with a well defined time interrelationship, that perform a single system-level function. Each Orbiter Block catalogs a tested Orbiter system capability that is available for performing a required mission function. Orbiter Block options are sequence or functional variations allowable within an Orbiter Block. Options may be selected for use with an Orbiter Block within specified constraints, which are the rules governing the implementation or application of an Orbiter Block. Subsystem states that are required prior to initiation of an Orbiter Block sequence of events are specified as required initial conditions. The Orbiter Block final conditions are the resulting

Step No.	Option	Interface		Event	Timing	
		CCS Command	Destination		Relative	Cumulative
1	g	DC3A	MDS	Cruise telemetry mode	T_0 - 3m	0.0
2	f	DC3C	MDS	FDS high-rate data select	T_0 - 3m	0.0
3	f	CC6C002	FDS	No change; no change; 2 kbps playback rate	T_0 - 3m	0.0
4	f, g	CC6BX0	FDS	Engineering data rate; no change	T_0 - 2m	1.0 m
5	c		CCS	Enable the ACS accelerometer calibration within CCS and place pulse counter data in CCS output register	T_0 - 1m	2.0 m
6	a	CC7B1210	ACS	IRU 1 on; IRU 2 off; IRU 1 enable; no change	T_0	3.0 m
7	b	CC7B2120	ACS	IRU 1 off; IRU 2 on; IRU 2 enable; no change	T_0	3.0 m
8	b		CCS	Change CC7B2212 to CC7B2222 in CCS ACE power changeover routine	T_0	3.0 m
9	c			Monitor CCS pulse counter data in the engineering telemetry measurements E-125/E-126		
10	d	CC7A1222	ACS	Inertial mode; stop roll turn; stop yaw turn; negative turn direction	$T_0 + \Delta T_0$	ΔT_0 + 3.0 m
11	e	CC7C2021	ACS	Gyro rate; no change; TVC inhibit; roll inertial	$T_0 + \Delta T_0$	ΔT_0 + 3.0 m
12	d	CC7A2222	ACS	Rate mode; stop roll turn; stop yaw turn; negative turn direction	$T_0 + \Delta T_0 + \Delta T_1$	$\Delta T_0 + \Delta T_1$ + 3.0 m
13	e	CC7C2022	ACS	Gyro rate; no change; TVC inhibit; roll rate	$T_0 + \Delta T_0 + \Delta T_1$	$\Delta T_0 + \Delta T_1$ + 3.0 m
14		CC7B2212	ACS	IRU 1 off; IRU 2 off; IRU 1 enabled; IRU auto control enabled	$T_0 + \Delta T_0 + \Delta T_1$ + 1m	$\Delta T_0 + \Delta T_1$ + 4 m
15	b		CCS	Change CC7B2222 to CC7B2212 in CCS ACE power changeover routine	$T_0 + \Delta T_0 + \Delta T_1$ + 1m	$\Delta T_0 + \Delta T_1$ + 4 m
16	c		CCS	Disable the ACS accelerometer calibration within the CCS	$T_0 + \Delta T_0 + \Delta T_1$ + 2m	$\Delta T_0 + \Delta T_1$ + 5 m

Figure 9. Sample Orbiter Block: Gyro Drift/Accelerometer Bias Calibration

subsystem states that have been modified by the Block sequence of events or selected per the required initial conditions. One continuous sequence of events is obtained by linking two or more Blocks together by providing single commands, as required, to insure that the final conditions of each Block are compatible with the required initial conditions of the following Block. The resulting merged-time event pairs and tables are normally loaded into and issued from the CCS Near Term Activity Table.

Orbiter Blocks were developed to implement all anticipated Orbiter flight sequences (excluding single event/commands). New and modified Blocks are developed (as required) to satisfy changing mission and operational needs. The detailed content of Orbiter Blocks is specified in Ref. 3. All changes and revisions of the Block Dictionary are under Orbiter change control.

Orbiter Block final conditions are designed to allow convenient linking to those other Blocks that will most frequently follow. For example, a Block involving the use of a Digital Tape Recorder (DTR) and/or scan platform ends with the scan platform post-positioned to the starting position of its next anticipated use. Similarly, the DTR in use is post-positioned to the location of its next anticipated use. This results in the platform and DTR final conditions of a science Block having the required initial conditions for the next science Block.

SEQUENCE DESIGN AND IMPLEMENTATION

Orbiter sequence design and implementation are divided into areas of technical specialties. Navigation, science instrument targeting, and maneuver design are performed first. Engineering functions, such as battery charging, temperature control, antenna and scan-platform pointing, and tape-recorder management are specified next. The sequence is then implemented by determining memory updates to the four on-board memories that, when loaded into the Orbiter's memories and activated, will cause the Orbiter computers to clock out the desired sequence of on-board intersubsystem control commands. These desired memory updates are then formatted into an uplink command file. The modularity of this process isolates and uncouples many areas such that technical specialists can analyze the impact of their specialty on the sequence, while suppressing extraneous data.

Mission Planning transposes project science requirements into valid Orbiter Mission Profile Events and associated profile specification parameters. Specific sequence of these events is input on a Mission Profile file. Examples are: tape playback, gyro calibration, propulsive maneuver, Earth occultation, and periapsis. Figure 10 depicts the sequence design and implementation processing that follows.

The Sequence Generator Program (SEQGEN) (6)

SEQGEN merges the Mission Profile with other files containing supporting engineering data. The resulting Sequence Development file is then processed by maneuver and science programs to produce the Maneuver and Engineering file and the Scan

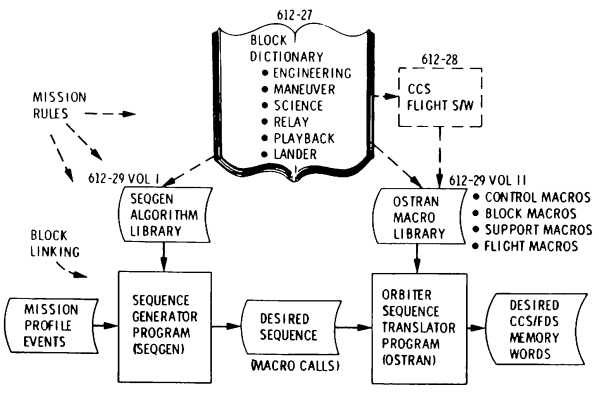

Figure 10. Sequence Design and Implementation

Specification Parameter file. This additional processing refines parameter values from approximate Mission Profile values into best-estimate values derived by considering telemetry feedback data and from checkpoint files from earlier runs. The final step in sequence generation is SEQGEN's processing of the Maneuver and Engineering file and the Scan Specification file to generate the Viking Orbiter Sequence (VOSEQ) file. This file contains the complete Orbiter sequence in the form of OSTRAN macro calls for Orbiter Blocks and single commands. Another file containing expected, time ordered, on-board commands is later compared with the simulator results.

As SEQGEN processes each event, selected constraints are checked to insure that no violations of hardware limits or procedural constraints have occurred. A library of SEQGEN Algorithms, which implement Orbiter Blocks, is maintained. These algorithms are driven by the Mission Profile Events, their associated profile specification parameters, and manual SEQGEN input to produce OSTRAN macro calls as output.

The set of OSTRAN macro calls, generated by SEQGEN, comprises the desired Orbiter sequence. This sequence description is at the system level; however, its detailed definition is controlled by the Orbiter Block Dictionary and the corresponding SEQGEN-Algorithms and OSTRAN-Macro implementations of each Block.

The Orbiter Sequence Translator Program (OSTRAN)(7)

OSTRAN processes the set of assembly language macro calls, with their associated parameters, from the VOSEQ file produced by SEQGEN. Manual inputs can be merged with the VOSEQ file input to override or supplement parameters generated by SEQGEN. OSTRAN processes the VOSEQ file input by expanding the macro calls as directed by pseudo-operations in the macro definitions and the arguments specified in the macro call. The Macro Definitions provide a standard implementation that will be used each time its corresponding Macro Call is encountered. Additional pseudo-operations direct the output to one or more of the four on-board memories, thus providing the link between Orbiter Blocks and the CCS and FDS Memories. The allocation of tables within both CCS and FDS memories is managed by the macros. CCS time/event tables and/or temporary routines are loaded into either current, even or odd overlap, or even or odd long-term event regions, depending on their completion time. Current events cover the period from the most recent update to the next update, overlap events are complete before the end of the following update period, and long-term events extend over two or more update periods. VIS parameters are loaded sequentially into dedicated table locations in alternating FDS memories. A Master or Absolute Hours Tables can initiate regions, in the event table regions, at specified times. SET symbols, such as those used to indicate table boundaries for memory management, are checkpointed at the end of each run to permit reinitialization for the next run. The results of the macro expansion are a series of 18-bit CCS memory words and 8-bit FDS memory words which are required to implement the sequence. The 18-bit CCS words include CCS pseudo-event words, time words, CCS instructions and data. The FDS memory updates typically consist of parameters to control the filter and exposure settings of the TV cameras and engineering measurement identifiers when changing an engineering telemetry commutation format.

The CCS Assembly language source-code output from the macro processor is then assembled and the Linkage Editor assigns memory addresses, and the CCS and FDS memory words are output onto the Desired Memory Words File (DMWF) which represents the memory contents (at a specific time) required to execute the specified sequence. The completed DMWF is then transmitted across an electrical interface between the Univac 1108 being used and the IBM 360/75 that will be used for the following simulator run.

Additional automation of the sequence design process is not without pitfalls. When programming general sequences, it is extremely difficult to program computers to recognize what you "meant" to say. Imperfect computer automation can propagate errors at very high rates.

SEQUENCE VALIDATION

The process of designing, implementing, and verifying sequences is complex and involves considerable manual interaction and control of the software. Specialists in attitude control, temperature control, power, science instruments, telecommunication performance, and science objectives provide input during sequence design and must review and concur with the final implementation. This process is strewn with potential pitfalls.

Orbiter sequences generally consist of one or more Orbiter Blocks which have been linked together. Each Block and its corresponding SEQGEN-Algorithm and OSTRAN-Macro implementations are validated, using the OCOMSM simulator, prior to their operational use. Validation of individual Blocks is necessary, but is not sufficient to guarantee a valid sequence of Blocks. This is largely due to the required overlapping of playback Blocks with other Blocks and the possible interaction that could result. In addition, it is desirable to check the on-board memory management, long-term events loaded previously, effects of possible manual editing of predecessor files, sequence-software errors, and human-input or control errors.

Sequence validation is accomplished in two stages. The sequence design phase (SEQGEN) automatically checks as many constraints as possible in an interactive manner with the SEQGEN operator. The final SEQGEN output is then manually verified to assure that, if implemented properly, it will accomplish the desired results. The validated design is then implemented and simulated with the resulting best-estimate control commands issued by CCS and FDS being automatically compared with SEQGEN's functionally generated control commands. The simulator's output products are designed to facilitate the remaining manual verification. Thus, the sequence is validated by first validating its design and then validating its implementation.

The Orbiters are controlled by the CCS, with FDS providing additional control of the science instruments as directed by CCS (Fig. 6). Specific sequences are defined by sequence tables (Fig. 7) in CCS memories, VIS control words in FDS memories, and CCS flight software routines that process the tables. The CCS closed-loop control of DSS defines when data is recorded and played back. Simulation of CCS, FDS, DSS, their relevant environment, and interactions between subsystems can provide best-estimates of all Orbiter control events.

A simulator of the on-board computers and data system provides additional visibility into on-board activity. The Orbiter's receipt, processing, and resulting control of other spacecraft subsystems is simulated for each sequence. The simulator output products are used to validate each sequence prior to its transmission, and also provides the on-line analysts with best-estimate predictions of on-board activity. On-line analysts compare actual telemetry data with the simulator predictions in order to rapidly detect and diagnose anomalous Orbiter operation. Visibility into status and activity on board the spacecraft is limited to those measurements that are observable via telemetry, but the simulator provides additional details that simplify the interpretation of actual telemetry. The simulator is also used to test various hypotheses as to how certain unplanned-for events would affect Orbiter operation.

ORBITER COMMAND SIMULATION (8)

The Orbiter Command and Simulation Program (OCOMSM) (Fig. 11) formats desired CCS/FDS memory words into an Uplink Command file, and simulates the transmission, receipt, and on-board processing of these commands; it also provides the detailed CCS hardware/software control response and their affect on FDS and DSS hardware. These control commands are automatically compared with the "planned" commands. Discrepancies in commands and/or timing are examined as a major validation tool. This file is also compared against real-time telemetry indications as an aid in detecting and understanding anomalous Orbiter activity. The simulator state is generally initialized from a previously checkpointed state, and simulation commences from the point at which it was previously suspended. Simulation continues until a specified END time is reached.

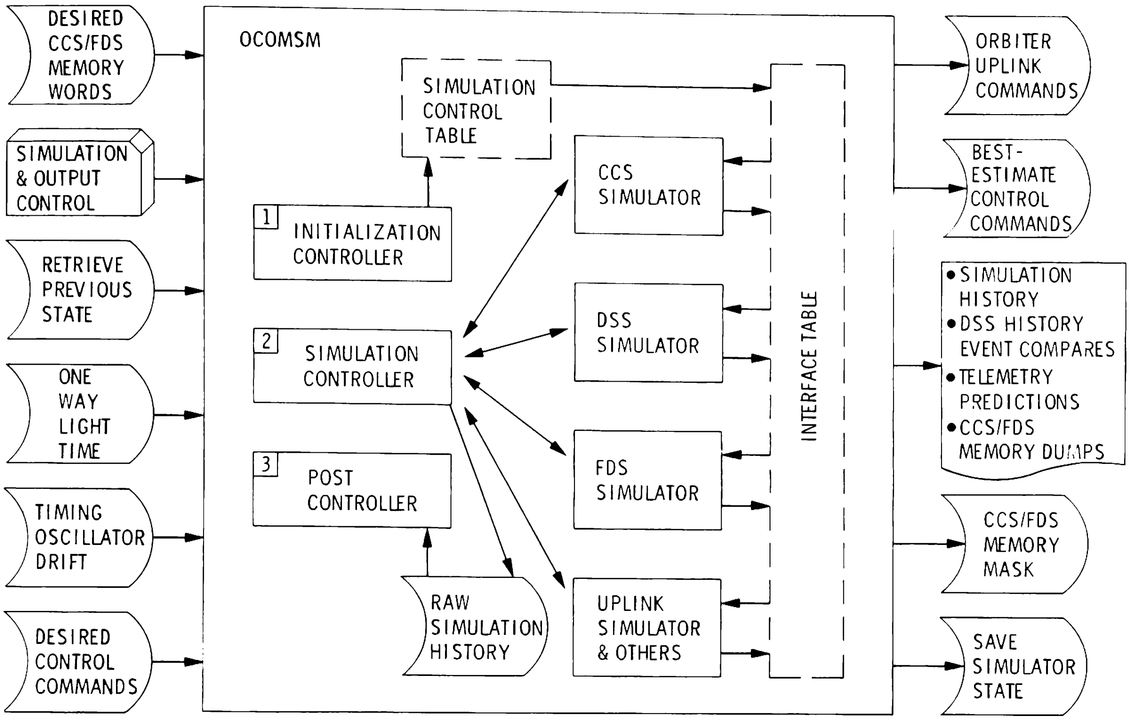

Figure 11. Simulator Functional Block Diagram

The main purpose of the simulator is to aid users in detecting erroneous, faulty, undesirable, or catastrophic problems in a sequence. The program must also be understandable, usable, flexible, adaptable, testable, maintainable, reliable, and tolerate user imperfections. Program development effort must fit within rigid resource constraints, such as available man-hours, computer time, and calendar time. The final program characteristics must be compatible with the available computer memory size and CPU time required per run and the anticipated number of runs per day (including necessary reruns), and must allow time for sequence validation. These factors are quite important, since the life-cycle costs of this program are about 20 man-years and about 1000 hours of IBM 360-75 CPU time.

The Orbiter hardware and flight software, that is simulated, was specially designed for the long-life and fail-soft requirements of the 15 month Viking mission.

One difficulty in modeling digital hardware is the different languages used in the hardware and software disciplines. Hardware description languages that describe hardware at the logic-gate bit-time register-transfer level provide a commonly understandable detailed definition that can bridge the gap. The APL language itself provides an excellent basis for describing digital hardware. Power-on-reset states and hardware idiosyncracies, that could affect a sequence, must be considered in the modeling stage. Any of these that could affect operation of the hardware, software, timing, or other subsystems should be modeled.

The level at which the hardware, and possibly the software, is modeled strongly affects the execution speed and ability to detect unexpected or anomalous timing relationships. The faithfulness (or fidelity) of the simulation must minimize the frequency with which the user has to disregard simulator predictions due to simulator simplification or shortcomings.

Users need the control to activate only those portions of the simulator that are needed to satisfy the purpose of their runs. Control should be provided to gain selective visibility into the detailed subsystem environment, work load, and response to selected important events. The simulator's usefulness to a user is affected by the precision with which they can request visibility into questionable simulator activity while suppressing unwanted output.

The design features of the simulator were tailored to support sequence validation during Viking flight operations. In general, the design is consistent with a Simscript implementation; however, some features required special tailoring in order to stay within the allocated computer resources and throughput times required for timely sequence validation.

Major portions of the simulator were implemented, using 360 Basic Assembler Language, in an effort to minimize execution time and the amount of memory used. The high-frequency bit manipulation associated with instruction decoding and modeling hardware flip-flops and registers required efficient use of the host computer's capabilities.

The program design is separated into three phases in an attempt to modularize the code and minimize the memory required for any one phase.

Simulation Phase

The event-scheduler design deviates slightly from a Simscript implementation due to the very high efficiency requirement. In this particular model, the internally generated (endogenous) events are orders of magnitude more frequent than the externally scheduled (exogenous) events. As a result, testing of endogenous/exogenous queues for the next event was collapsed to one endogeneous queue which contained a pointer to the next exogenous event being scheduled.

Double-precision, floating-point time variables were used to span the required nanosecond to two-year time range. The time unit of nanoseconds was chosen to avoid problems associated with machine representation of irrational numbers. The sign bit of time words is used to de-schedule obsolete events.

The Computer Command Subsystem (CCS) simulator (Fig. 12) accurately models the Block redundant CCS processors, output units, and memories at the instruction and register transfer level. The accuracy with which software programs (in the memories) are simulated is dependent primarily on the accuracy with which the bit-time clocks can be calibrated during flight and the timing accuracies of the 32 priority interrupts and 24 level indicators. Output simulation consisting of discrete commands (relay closures), coded commands, and telemetry data can be routed through either or both output units. A hardware and software self-test must be satisfied before access can be obtained to an output unit.

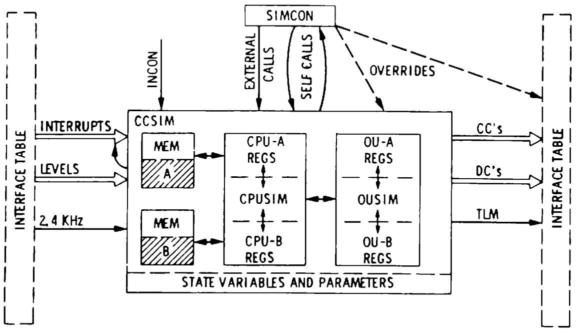

Figure 12. Computer Command Subsystem Simulator

The Data Storage Subsystem (DSS) simulator (Fig. 13) models the two digital tape recorders (DTRs) that can record and store one billion bits of Orbiter science and engineering data and data relayed from the Lander. The recorders have a wide dynamic range, with a 45-inches-per-second record speed and playback speeds as slow as 0.15 inches per second. DSS SIM receives and decodes 14-bit coded commands from CCS SIM and provides CCS SIM with increment/decrement interrupts for each three-inch movement of the tape and interrupts, signaling beginning and ending of the tape.

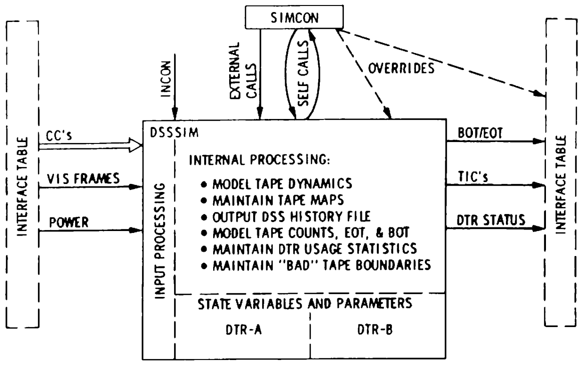

Figure 13. Data Storage Subsystem Simulator

The DTR discrete event model describes the tape dynamics for each possible state transition. The acceleration for each transition is a function of position or the tape and is calibrated separately for each DTR. The model maintains histograms that show frequency of tape passes as a function of tape position; a tape map is maintained, indicating what data have been recorded at each position on the tape; summary statistics indicate total usage and duty cycle; and a DTR history print shows all activity separately for each DTR.

The Flight Data Subsystem (FDS) simulator (Fig. 14) models the hardwired memory controller and two independent memories, each containing 1024 8-bit words. The memories are used to rate-buffer data, store control words for the TV cameras, and contain identifiers that specify which engineering measurement are to appear at each position in the engineering telemetry data stream. FDS SIM receives and decodes 14-bit coded commands from CCS SIM. These commands can modify FDS memory, direct FDS as to how it should control the science instruments, and control the data rates and routing of science and engineering telemetry.

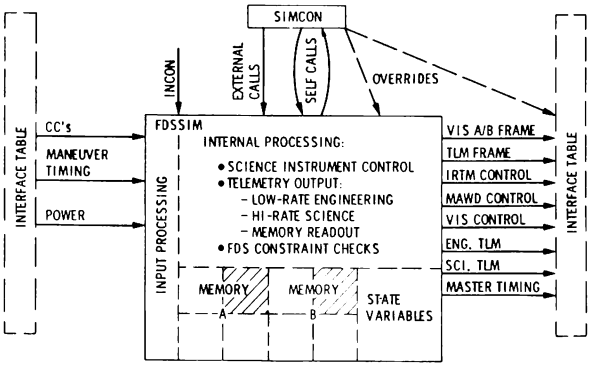

Figure 14. Flight Data Subsystem Simulator

Simulator Execution Time

Figure 15 illustrates the large effect of uplink command activity, CCS hardware and software self-test activity, engineering telemetry rates and formats, and CCS processors executing a maneuver in the tandem mode.

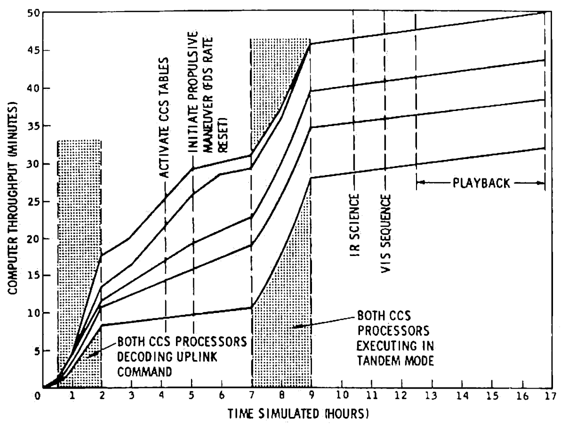

Figure 15. Simulator Execution Time

The lower curve reflects expected operations, while the higher curves illustrate the effect of different FDS data rates in a failure mode, where the FDS memories were lost and its fixed format backup had to be used. Degraded, but faster than the lowest curve, simulator performance could be obtained by de-configuring FDSSIM, with one input, and providing its frame timing and telemetry interrupt signals to CCSSIM by several repetitive override inputs which could provide the missing FDSSIM-CCSSIM interface signals. These overrides, however, would need to be modified whenever CCS altered FDS's engineering telemetry rate.

The large amount of time used to simulate uplink command decoding is due primarily to the CCS hardware and software self-test activity, which in turn is affected by the FDS engineering telemetry rate and format. The different slopes to the curves immediately following the uplink are due to the FDS simulator. The large amount of CPU time required to simulate the propulsive maneuver is due to the high activity in both CCS processors and output units.

During the simulation phase, real-time activity takes from 3% to 12% of the 360 CPU time, and the simulator uses all of the remaining time. Optional output products and file maintenance require little 360 CPU time, but can easily double the throughput time required due to their affect on initialization and post processing.

The portions of the sequence that occurs after completion of the propulsive maneuver (Fig. 15) are representative of orbital operations activity and represent an upper bound on cruise activity. The time used when simulating propulsive maneuvers is large, but amortizing approximately 10 of them over approximately 400 days of the mission does not bias the average run time very much.

CONCLUSIONS

The usefulness of the simulator described in this paper is in the visibility and insight it provides users into the internal operation, timing, and accuracy with which a given sequence implementation accomplishes its desired goals. Few features in the simulator itself are unique; however the design process, critical design decisions, implementation decisions, anticipation of most of the major pitfalls, and its operational use resulted in a computer program that satisfies its requirements, was developed on time, within its original budget estimate, executes within the time allotted, and almost fits within the desired memory size.

Murphy was right: "Things do go wrong, and often, and at the worst possible time." Complex software-controlled systems can be manageable and adaptable only if the operation software system provides for controlled adaptation, system reverification, and there are built-in checks to detect human or procedural imperfections that Christiansen (Ref. 9) aptly coined "Mushware." Operational software must provide for manual intervention and it must be overridable. Simulation of systems that are designed to tolerate hardware failures and be usable even in degraded operational modes requires: easy access to change the simulator's data base, the ability to enable and disable portions of the simulator, and the ability to provide unplanned-for signals on subsystem interfaces. A system simulator that faithfully models the portions of a system that a user can control can: provide a useful tool to train users, provide users with detailed visibility into selected portions of the system while suppressing unwanted or irrelevant data, provide a means of testing and verifying changes, and can help detect human errors and procedural problems.

Care must be taken during the simulator design stage to assess the affect on the simulator design and code of possible changes to the hardware and software being simulated. Functional simulators can easily be rendered useless by minor system changes; and it should be noted that hardware failures can look like hardware changes, even in "frozen" systems.

System designers must recognize the operational impact of designing systems whose operational appearance to a system user can be radically altered by late changes to its software. Flexibility and adaptability are not free and can totally invalidate prior system testing.

The Orbiter Block concept, the block implementation software, the management control it provides, and the organization it provides to validity-testing should be useful in a variety of other situations. The Blocks define system usage, while the detailed implementation is "soft" coded as sequence algorithms and assembly language macros.

The hard segmentation of simulator modules corresponding with hardware subsystems and their dedicated interface tables were identifiable with the hardware wiring interconnections. This segmentation facilitated engineer/programmer communication and reduced the coordination required between programmers who were implementing modules that had intercommunications.

Simulator credibility was enhanced by providing user access to all subsystem registers and inter subsytem interfaces. The ability to run each subsystem simulator in a stand-alone mode permitted staggered testing of modules, which also increased the need for early availability of initialization and input-processing portions of the program.

The usefulness of a simulator is dependent on the precision with which a user can request visibility into selected internal operation, while suppressing unnecessary or unwanted output.

Operating procedures, file-naming conventions, and standard output specification should be standardized and automated, but readily changeable.

The software described in this paper is being used daily in support of the two Viking Orbiters currently en route to Mars. It provides an excellent tool for analysts to examine and analyze what is going on inside a complex spacecraft, and to detect potential problem areas during sequence design and implementation; provides best-estimate telemetry predictions that permit rapid detection of real-time anomalies; aids in the diagnosis of their cause; and provides a tool to examine the affect of alternate hypotheses that attempt to explain anomalies.

Constraint checking that can be done must be done as early as possible during sequence development. The later problems are discovered, the more time that is lost and the more software that has to be rerun.

The recent dramatic increase in the performance/cost ratio of mini and micro computers and their associated peripherals suggest very different implementations of simulators, such as the one described in this paper. As an example, a network of micro computers could be used with at least one micro dedicated to the scheduling function, to each of the major simulator modules, and to parallel output processing. Modules such as the CCS simulator might have four micros, one for each processor and each output unit. Bipolar micro processors might be used where extra speed is needed. Parallel processing, where possible, could increase throughput. High level simulation languages, if available for a variety of micro computers, might simplify and speed the development process, while retaining acceptable speed with low additional memory cost. A standardized network communications protocol could simplify required input/output interfaces with other computers and data bases.

REFERENCES

1. Mariner Mars 1971 Project Final Report: Vol. III. Mission Operations System Implementation and Standard Mission Flight Operations, Technical Report 32-1550, Jet Propulsion Laboratory, Pasadena, California, July 1973.

2. Young, A. T., Viking 75 Project Mission Rules, Martin Marietta Aerospace Document PL-3720472, March 1975.

3. Stuart, J. R., Viking 75 Orbiter Block Dictionary, JPL Document 612-27, Rev. A, Jan. 1975 (JPL internal document).

4. Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description, JPL Document 612-28, Rev. B, Nov. 1974 (JPL internal document).

5. Viking 75 Orbiter Performance Analysis Group Sequence Software Macros, JPL Document 612-29, Rev. C, Oct. 1975 (JPL internal document).

6. Page, D., Viking 75 Project Software Requirements Document for the VO Sequence Generator Program (SEQGEN), JPL Document 620-30, Sept. 1973 (JPL internal document).

7. McEvoy, M. B., Viking 75 Project Software Requirements Document for the Orbiter Sequence Translator Program (OSTRAN), JPL Document 620-17, Rev. A, Sept. 1973 (JPL internal document).

8. McEvoy, M. B., Viking 75 Project Software Requirements Document for the Orbiter Command and Simulation Program (OCOMSM), JPL Document 620-15, Rev. A, Sept. 1973 (JPL internal document).

9. Christiansen, D., "Hardware, Software, and Mushware," IEEE Spectrum, p. 37, Oct. 1975.