SOPHISTICATED SOFTWARE SYSTEMS FOR SMALL SELF-CONTAINED SPACE SHUTTLE PAYLOAD G285

Robert Burkhardt
Getaway Special Project G-285
University of Colorado, Boulder

ABSTRACT

The increasing development of small microprocessor systems has allowed the use of more advanced software in the area of control systems. This paper discusses the development of software for small Space Shuttle Getaway Special Project payloads using payload G285 as a case example. The development process behind a space related software package (as in any software package) is a major factor. The design process for G285 is discussed in some detail along with the general scheme behind data acquition and thermal environmental control for a space related payload. Additionally, key concepts in a software system concern the implementation of redundant systems, error detection, and error response. All of these factors are discussed within this paper.

INTRODUCTION

The University of Colorado's Getaway Special Program began in the Spring of 1984 when a group of students began to solicit the University community for experiment ideas applicable for investigation aboard NASA's space shuttle orbiters. Three experiments were selected for their feesability, applicability to current space operation work, and for their contribution to scientific advancement within the University.

PRECEDING PAGE BLANK NOT FILMED

The first experiment, a biological experiment, will study the theory of geotropism as it applies to a small fungus called phycomyces. Growth of this organism will be initiated several days prior to launch via a small preprogrammed clock. The fungus will be photographed prior to launch and during launch using a small camera. The second experiment deals with studies of the separation of gases and liquids in a zero-gravity environment. This experiment, utilizing a small centrifuge to separate the gasses and liquids, will be started within 24 hours after achieving orbit. The activity within the centrifuge will be recorded using a camera. The third experiment will examine the phenomenon of shuttle glow around the orbiter's tail section using an ultraviolet spectrometer. The spectrometer will be able to view the tail via an intricate mirror assembly constructed at the top of the payload. A Motorized Door Assembly (MDA) allows view outside the cannister. A fourth experiment was later added. This will use eight thermal sensors positioned throughout the cannister to examine the temperature fluctuations within an open GAS cannister.

MISSION SEQUENCE

Controlling all of the sybsystems with the exception of the phycomyces experiment (this is a completely isolated experiment) is a small National Semiconductor (NSC) MA2000 series microprocessor component system utilizing a Z80 instruction set as a base for all software. All data will be stored in bubble memory totaling 0.5 Megabytes for retrieval post-flight (see Figure 1 for data allocation of various experiments). A general timing sequence is needed for payload operation and this is provided by an MM58174A Microprocessor-compatible real-time clock external to the microprocessor.

Within 24 hours of achieving orbit, the astronauts will flip one of the three GCD switches which will activate the G285 payload. This will initiate internal power thus causing the microprocessor to bootstrap. A resistive heater will also be turned on near the microprocessor to help alleviate possible microprocessor problems due to cold temperatures. The power to the bubble memory will be turned on and a header written to the top section of memory to indicate a successful initiation of microprocessor control. All thermal sensors (eight) will be checked and data stored in the header section of the bubble memory for post-flight data analysis as to the initial characteristics of the cannister.

Once initialization has completed, the microprocessor will initiate power to the fluids experiment and a small camera. The microprocessor will be in complete control of the camera, sending a pulse to the device to take each picture. Approximately 300 pictures will be taken over the period of the next two hours at varying but predefined intervals. For each picture taken, three bytes of data will be stored that contain a time tag consisting of hours (0-??), minutes (0-59), and seconds (0-59) since payload activation by the astronauts. The clock will be read for each time tag and compared with the initial time of power up (stored within a main program variable and also within the header of the bubble memory). These 300 time tags stored over a period of two hours will actually be much greater then needed by fluids experiment specifications but will enable microprocessor personnel evaluate the computer's response to the severe environment and will also provide the best time correlation for data analysis. Temperature data will also be stored at a rate of eight samples (one sample per sensor) each minute.

CATEGORY	BYTES	FRAMES	%TOTAL
FLUIDS EXPERIMENT	**1900**	**10**	**0.36%**
CAMERA TIME TAGS	900	-----	0.17%
THERMAL DATA	960	-----	0.18%
BLOCK HEADER TIME TAGS	0	-----	0.00%
STATUS FLAGS	20	-----	0.004%
SYNC WORDS	10	-----	0.002%
UNUSED	10	-----	0.002%
INTER-EXPERIMENT GAP	**57**	-----	**0.01%**
SPECTROMETER EXP.	**522310**	**2749**	**99.62%**
SYNC WORDS	2749	-----	0.52%
STATUS FLAGS	5498	-----	1.05%
TIME TAGS	8247	-----	1.57%
THERMAL DATA	10996	-----	2.10%
EXPERIMENT DATA	494820	-----	94.38%
BUBBLE MEM. HEADER	**21**	-----	**0.004%**
TIME TAG	10	-----	0.002%
SYNC	1	-----	<0.001%
STATUS FLAGS	2	-----	<0.001%
THERMAL DATA	8	-----	0.002%
TOTAL	**524288**	-----	**100.00%**

Figure 1. *Data allocation for project G285*

At the conclusion of the two hours, the fluids experiment will be powered off and a sequence of bytes (i.e. 255 values) will be written to the bubble memory to indicate a gap in data prior to initiation of the next experiment. Following this, the spectrometer will be powered up and a pressure sensor will be checked to verify a safe pressure level. (The inside of the cannister will be pressurized to approximately 1 atm of argon. This is to avoid problems with corona discharge caused by high voltage in vacuum or near vacuum environments. The entire spectrometer will be within this sealed container with a field of view out through a small quartz window. In the event that the pressure drops to approximately 10 torr, corona discharge will take place. To avoid this problem, the microprocessor will automatically shut off high voltage power at approximately 50 torr.) Upon initiation of power to the high voltage power supply, the microprocessor will initiate an ultraviolet region wavelength scan with the MDA closed (used for calibration purposes).

Data in the form of data numbers (0 to 255) will be read from a Pulse Amplification Detector (PAD) counter. This counter is a ripple bit counter and as such, the line from the PAD to the counter must be disabled prior to being read by the microprocessor (see figure 2). Every 1/3 second, the microprocessor will disable the line, read the counter, reset the counter to zero, and then enable the line between the PAD and counter. A ninth bit of the counter is also checked on each read. This is an overflow bit that is latched upon a carry out of the eighth and most significant bit of the counter. In the event of a data overflow, a value of 255 will be written to memory to help post-flight data analysis.

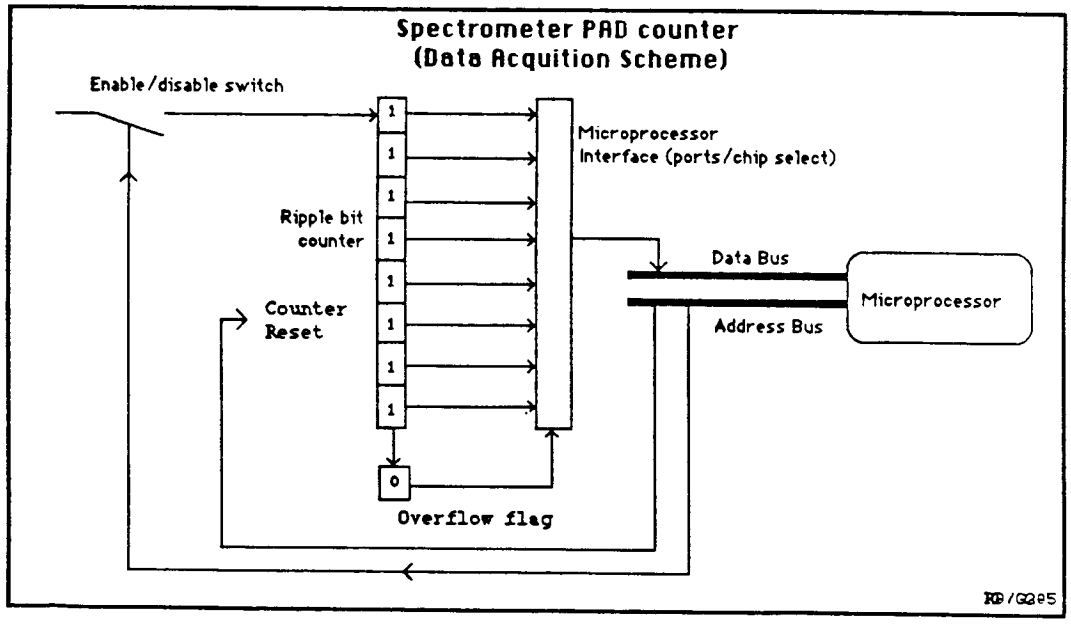

Figure 2. *Data acquition scheme*

After each data sample is read, the grating drive of the spectrometer will be stepped. A total of 180 grating positions will be used (to allow for a scan time of approximately 1 minute). After each complete grating drive scan, the mirror (constructed outside the quartz window) will be moved one position by activating a worm gear motor controlled by the microprocessor. This

will be repeated twelve times to allow for twelve different mirror positions. In this way, a complete scan across the tail section is achieved. (See figure 3 on pointing angles.) A photodiod is used as a Bright Object Sensor (BOS). This incorporates a field of view greater than that of the spectrometer to detect of the emergence of any bright object into the field of view (i.e. Sun, Moon, Earth, etc.). In the event of a triggering of the BOS, the power to the high voltage power supply is disabled and an interrupt is sent to the microprocessor.

Thermal data is also collected during the spectrometer portion of the mission at a rate of four samples a minute. This will result in a complete thermal data resolution of at least two minutes per sensor.

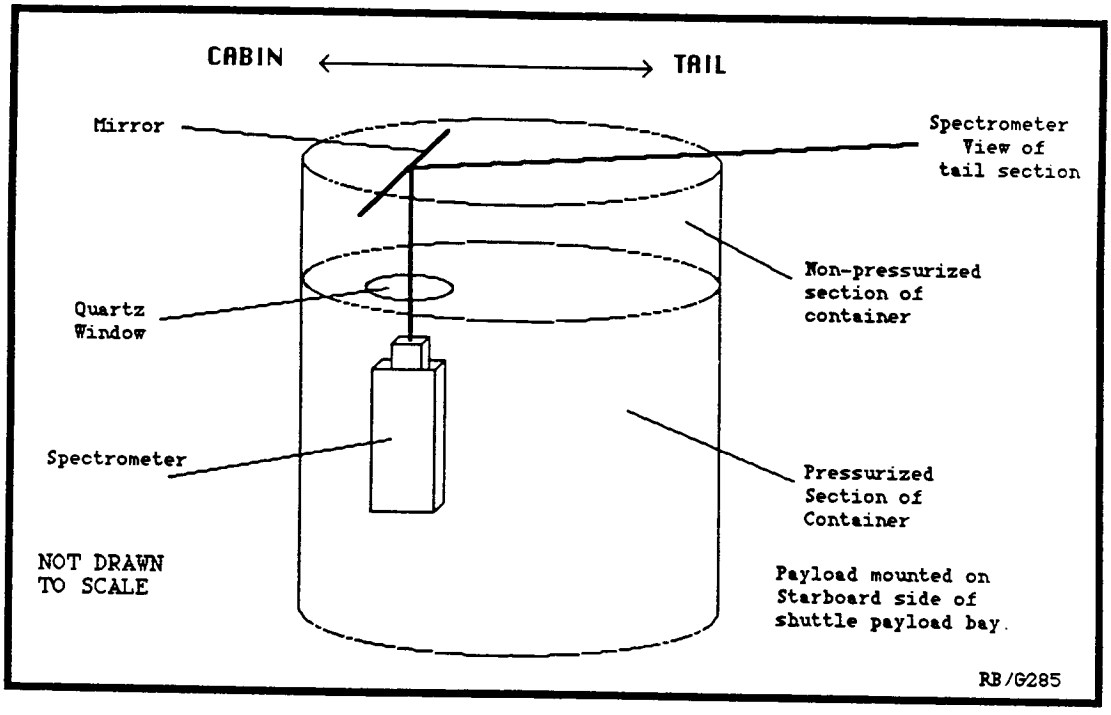

Figure 3. *Spectrometer pointing angle*

TELEMETRY FORMATTING

All data gathered during the mission, will be formatted into 190 byte blocks (see figure 4 on data block format). This number was chosen to allow for an approximate one minute resolution of data frame during the spectrometer portion of the mission. Each data frame consists of a one byte sync code, three bytes of time tag, two bytes of status flags, four bytes of temperature data, and 180 bytes of instrument data. This allows for a 94.7 percent raw data content which meets all mission requirements. During the fluids portion of the experiment, this 180 bytes consists of time tags (3 bytes) and temperature data to give a resolution of approximately 12 minutes per frame. During the spectrometer portion, the data frames are collected in groups of 12 (corresponding with the 12 mirror positions) to form a major frame (or block) consisting of 2160 bytes of raw data for each 2280 bytes of data stored. All data is buffered in RAM in 190 byte segments and then downloaded to bubble memory.

Early in the mission, a variable length data format scheme was considered. This incorporates a block counter indicating the amount of data stored in each frame. This number is then inserted into the block to aid in post-flight data formatting. The variable length format, however, has specific drawbacks. Post-flight data formatting will rely on the fact that the block counter is correct (i.e. not getting corrupted) and in a particular position of the block. To add safety checks, one would need to add long sync words (longer than is needed in a fixed length version) to the telemetry frame, thus diminishing the amount of data storage available to science data. For this reason, the variable length data format philosophy was discarded and a fixed length format was selected for its structure and time oriented basis (the frame structure). The 0.5 Megabyte memory will allow for a total of 2759 frames with a total of 78 bytes left over. These 78 bytes are used up in the initialization header and the inter-experiment gap written to the bubble memory between the fluids experiment and the spectrometer experiment.

The fixed length format also helps in developing plans for post-flight data formatting. Upon conclusion of the mission, all data will be downloaded to an HP64000 computer system and from there downloaded to a VAX 11/780 computer. Here all data formatting will be done and data inserted into the appropriate data bases. Utilizing the fixed length format, the 190 bytes can be masked and all data stripped out quite easily. In addition, an added sync detect can be put into the system by triggering sync not only on the fixed one byte sync word but on the unused bits of the minutes and seconds fields of the time tags (2 most significant bits). This increases the ability to perform redundant error checking post-flight.

REDUNDANCY OF SYSTEMS

Early in the planning for G285, the subject of redundant microprocessors was discussed. The dual microprocessor system was later discarded since the development of such a system created more problems than would be solved by this method (i.e. handshaking, "who is slave and who is master", etc.). Redundant microprocessors would however be of great use in larger systems in which failure of the payload could have disasterous effects. In the case of G285, the payload just did not fall into this category.

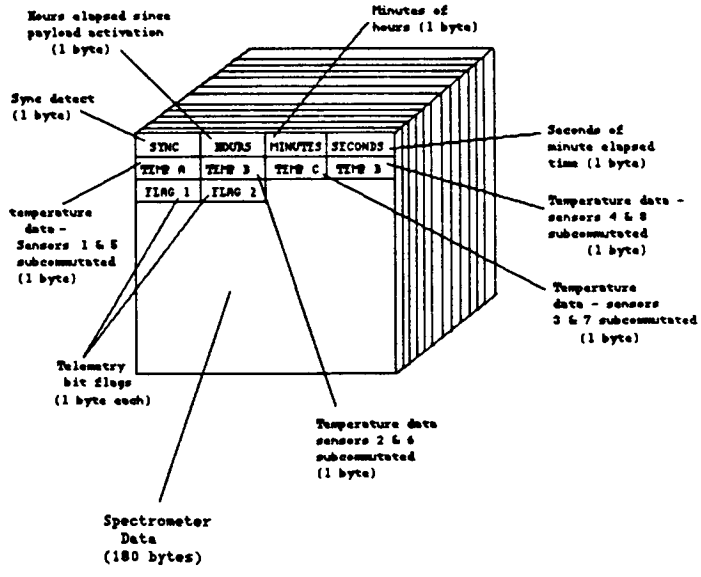

Figure 4. *G285 Data block format*

Some redundancy, however, was built into the software and hardware for project G285. In the case of the BOS, a hardware interrupt will disable power to the high voltage power supply. A BOS trigger will also cause the microprocessor to enter an interrupt service routine setting a software flag to true (BOS trigger has been activated). Thus the microprocessor vote will be in favor of turning off the power to the high voltage power supply. It takes only one vote to shut off the high voltage power supply but requires two votes to turn it back on. This is to protect against failure in the system causing the high voltage to be on during sunlight (This could cause a burn-out in the PMT tube.) A similar scheme is used with the pressure sensor to protect against corona discharge.

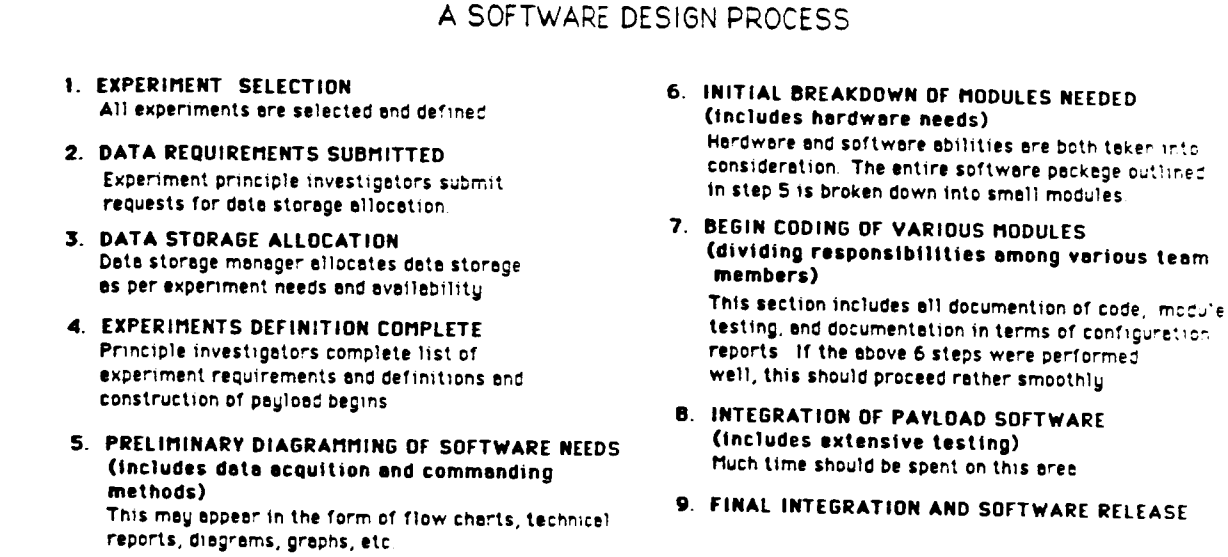

Figure 5. *GAS Software Development scheme*

Redundancy is also built into the thermal environmental control system. This consists of three resister heaters placed at strategic locations throughout the cannister. A temperature sensor is placed next to the heater to provide data to the microprocessor for imputing into the telemetry frame for post-flight data analysis. A second temperature sensor is set next to the first to control the heater. This simple circuit operates under the simple philosophy of "colder temperature yields higher heat output". In the event of a heater runaway, the microprocessor will disable the power line to the heater.

DEVELOPMENT OF A SOFTWARE SYSTEM

As in any software system developed for flight or ground use, the development process is a vital process. All hardware requirements need to be defined completely (or close to completely) before software work begins. This delaying action actually saves time in that it keeps a group from proceeding down many wrong roads. A simple diagram of a development process for a small shuttle payload is given in figure 5. The project starts out with documentation and diagrams

depicting exactly what is required of the software. This must then be examined within the whole context of the payload. How does a particular piece of software relate to the entire payload? This question must be answered at every step in the development process.

The development environment in which the programmer or engineer works must be kept organized. A software configuration process (complete with paperwork filled out on each routine completed or changed) is a definite MUST especially in systems in which many people are doing the coding.

ALL SOFTWARE CHANGES MUST BE DOCUMENTED BOTH INTERNALLY TO THE CODE AND EXTERNALLY IN CONFIGURATION REPORTS

Work should NEVER be performed directly on a released routine (i.e. in the event that some change is being made to a previously released routine). A separate copy should be made and extensively tested before being submitted for release. Backup tape copies and paper copies should be made as needed or as specified by the software manager. In all, a sense of organization is necessary. Disorganization will lead to mistakes and bugs.

The choice of a language is always a question that is discussed extensively in developing any software system. For project G285, the language being used exclusively is Z80 assembly. The software is developed on a VAX 11/780 using a Z80 assembler made for interface with an HP64000 computer. The software is downloaded to the HP64000 using normal computer network lines and then downloaded from the HP64000 to the flight microprocessor through RS-232 standard interfaces. Many computers (the HP64000 included) have cross-compilers that enable programmers to program in languages such as C or pascal and then convert the source code to Z80 (or some other instruction set). For project G285, it was felt that writing the code in Z80 directly would allow the programmer more ease and flexibility to do exactly what was required.

CONCLUSION

The increasing development of software systems for small microprocessors is causing an ever increasing complexity of jobs being delegated to small control systems. Computer scientists and engineers are developing the skill of putting into software what had been put into hardware only ten years ago (the programmer free to utilize many aspects and features of the software involved). In this way, more complex systems can be developed for both space and ground related software systems. The Getaway Special Project, with its great diversity, becomes a magnificant test bed for the programmer to develop sophisticated software. However, as in any project, the programmer must be driven by the ultimate goal – to develop a sophisticated software system and at the same time, achieve maximum simplicity.