The Athena System and Ground Computer Assignments

The Athena is a four-stage missile launched from Green River, Utah toward White Sands Missile Range, New Mexico, covering a ground range of 450 miles. It is designed to deliver a payload into White Sands with prescribed atmospheric re-entry velocities and angles, so that re-entry phenomena may be studied, and radar performance against re-entry bodies may be evaluated.

A ground computer (presently an IBM 7044) operates in real time during an Athena flight, with seven assigned tasks:

1. Providing instantaneous impact prediction data during first and second stage burning, for range safety use. This function is especially important since the Athena is flying over populated areas and, in case of missile malfunction, must be destroyed by safety personnel so as to land only along non-populated sections of its flight path.
2. Computing and transmitting midcourse (between second stage burn-out and third stage ignition) guidance correction commands to bring the missile into the proper re-entry orientation.
3. Evaluating telemetered data from the missile following the transmission of the guidance commands, to determine if all system functions are operating within prescribed limits.
4. Transmitting the third stage ignition command to the missile at the proper time, once it has been established that the third stage can be safely fired.
5. Transmitting precise pointing data to several tracking and measurement sites throughout the flight, to assist them in acquiring and following the missile.
6. Providing ground support personnel with data displays and flight status information throughout the mission.
7. Logging all computer inputs and outputs in real time for postflight analysis.

The following sections, while relating to real time philosophy in general, will at the same time show the development of the above tasks as a multiprocessor job under the real time constraints of input and output data demands.
MONITOR PHILOSOPHY

All real time programs within the Computer Directorate at White Sands operate under the control of an Executive Control Monitor which directs the flow of activity among various functional modules to assure that all program functions are carried out at the proper times and in the necessary contingency to related activities. The Monitor was originally adapted for Athena use from the Mercury project monitor, but has been revised to make it more applicable to specific support problems encountered at White Sands.

A multi-processing job is a programming task that can be subdivided into functional modules or processors, each of which executes a specific function in the overall program, and can be in various stages of completion throughout execution of the entire job. Each functional processor should have well defined inputs and outputs in relation to other processors in the system, and in relation to the outside (non-computer) world. The Monitor might well be referred to as a multi-processing monitor, since in controlling the relative flow of activity between functional processors, it monitors the state of completion and the dynamic requirements of each processor in the job throughout execution of that job.

A multi-processing job becomes a real time job when execution of various program functions becomes dependent on some regular or random time constraint, such as input or output data requests from a device operating dynamically during program execution. Thus, for a real time job an additional, or perhaps we should say a master, processor must be added to the other functional processors in order to service the real time demands and pass any pertinent control information on to the Monitor. For the Athena system this processor is known as the trap or interrupt processor, to which control is given immediately whenever a data input or output demand is recognized, interrupting whatever functional processor may be in process at that time. The trap processor examines the real time request, reads in or transmits the required data, and provides the Monitor with information regarding the input or output action just completed so that either execution of pertinent program functions may be initiated, or control will be returned to the interrupted functional processor.

Each real time processor can be classified in one of three time-oriented categories, and relative execution priorities can be assigned on this basis. First are those activities that must be executed immediately upon recognition of some event or time in the program. Examples in the Athena program are the transmission of the third stage ignition command within a critical time interval, or the plotting of range safety display data without undue delay after they have been computed. The second category consists of functions that must be executed within a given time interval after they are requested, such as the editing and processing of direct data messages before the next sample arrives at a remote terminal, or the calculation of regularly transmitted computer outputs within a time dictated computation-transmission cycle for those data. The third group of processors involves program functions whose execution can be deferred until some later point in the mission, when higher priority functions have been completed and computing time is available. Included in this category would be such functions as selecting a radar for later use on the basis of current data from several radars, or the calculation of guidance commands to be transmitted at some later point along the trajectory. The first class of functions described must, of course, have highest priority, since initiation of these actions usually cannot be delayed for any other purpose. Other processors will be assigned lower priorities as the importance of their roles and the judgment of the program organizer dictate. The trap processor, as mentioned before, will usually assume control whenever a data demand appears at a remote terminal, but even trap processing can be temporarily inhibited if an extremely critical function is in process at the time the demand occurs.

Figure 1 shows the relationship between functional processors, a trap processor, and the Executive Control Monitor. A simplified version of the Monitor is shown since this paper intends only to outline its control functions, omitting many of the details and system tools available. When an input or output data demand interrupts a functional processor, the contents of all machine registers are saved as control is given to the trap processor. When control is eventually returned to the processor at the point where it was interrupted, these machine registers can be restored and processing can resume. Numerous control tables and indicators link the functional processors to the Monitor. Some of these
Demand Functional Processor \( I = 1, 2, \ldots, N \) at time of interrupt from processor \( I \); return

Figure 1. Monitor-trap processor — functional processor linkage.

are set initially to define, among other things, the entry points and job priorities assigned each processor, while others are maintained dynamically for use in monitoring the status of each processor.

Among the indicators that change during job execution are three basic ones described here. First is a “request” indicator for each processor, turned on whenever execution of that processor is requested by another segment of the program. For example, the program might contain a processor which edits or evaluates radar data, in which case this function would probably be requested by the trap processor each time a new set of radar data enters the computer. When control is returned to the Monitor by the trap processor following the data interrupt, the Monitor would see the request indicator on for the radar processor, and would transfer control there as soon as any higher priority activities had been completed. The second indicator is a “suppression” flag for each processor, causing the Monitor temporarily to suppress execution of that processor even though its request indicator might be on. Any processor can be suppressed by any other processor, and can likewise be released by turning off the suppression indicator. The third indicator is the “in process” indicator, turned on whenever execution of the functional processor associated with it has begun but has not been completed, and turned off upon normal exit from that processor.

PROCESSOR QUEUEING AND ROUTING SHARING

Real time processes often encounter times of peak activity where processor execution requests occur more frequently than control can be given to the processor. In order to avoid losing any of these requests, a queueing facility is provided by the Monitor, which allows the stacking of requests until computing time is available for servicing them. A basic assumption, of course, is that overall program timing has been so established as to assure the execution of queued functions within some limiting time frame and before the queue tables for that processor have been filled.

Another problem occurs when two or more segments of a multi-processing job share the same subprocessor, resulting in a possible loss of indicators and temporary storage within that subprocessor. At least four solutions may be applied to this problem, all of which have been used at the White Sands facility with varying degrees of success. First, and perhaps most obvious, is to avoid the problem by loading duplicate subprocessors, though this is a space consuming procedure. Second is a modified queueing philosophy applied to subprocessors. This is ideal in some instances, but may tend to subvert the priority assignment given to processors unless handled carefully at the subprocessor level. Third is the disabling of data traps during execution of a subprocessor, thus forcing the completion of that subprocessor prior to its call by another processor. This should be done only for short subprocessors, and even then there is some danger of losing vital segments of data while in the disabled mode. Fourth is the saving of temporary cells and indicators for the subprocessor at the time a data interrupt occurs. This method respects the initial priority establishment, but is often wasteful of computer storage and execution time.

INPUT-OUTPUT AND INTER-COMPUTER COMMUNICATIONS

Inputs to the Athena real time program are radar tracking data (polar coordinates, site identification and tracking mode) from five radars; 15 channels of telemetry data from the missile in both a data and a calibrate mode format; signals such as lift off, stage ignition and burn out; and timing from four sources—a millisecond clock attached to a multiplex-
er control unit to provide timing pulses for radar interrupts, an astrodatal clock measuring range timing in millisecond increments, a range timing word attached to each sample of telemetry data, and an internal computer core clock with a 162\(^{1/3}\) millisecond resolution, to be used if other timing sources fail. Computer outputs include acquisition data (polar or rectangular coordinates and site identification) to eleven tracking sites; commands to the missile via a command transmitter station, including stage firing commands and midcourse pitch and yaw maneuver instructions; impact prediction, present positions and other information sent to seven plotter display boards; and numerous binary and decimal displays showing the present trajectory status, the results of dynamically changing computer decisions, and various data and error messages appearing on an on-line printer throughout the mission. Input/output formats and data rates may vary throughout a mission.

Since 7044 core storage and computer speed prevent loading or executing the ideal large scale real time program from core, any data handling or formatting that can be done outside the computer simultaneously with computer processing is always welcome. For this reason the personnel of the Real Time Data Center at White Sands have designed a Direct Data Buffering System to handle many of the computer inputs and outputs without tying up valuable computer storage and time.

Radar data are transmitted from the radar sites to the Real Time Data Center over telephone lines via kineplexes, one sample requiring 15 8-bit bytes. These bytes come from the Kineplex receivers every 3\(^{1/3}\) milliseconds, so that one complete frame of radar information is transmitted every 50 milliseconds with the data multiplexed for the five input radars, adding a time word from a master clock. Telemetry data and calibration words are transmitted over 14 pairs of telephone lines, formatted and buffered by the Direct Data Buffering System, and similarly sent to the 7044 on an interrupt basis about 11 times per second. The Direct Data Buffering System includes an output buffer to format and transmit acquisition data to the tracking sites via the kineplexes, and to transmit computer generated missile commands and display data to the proper device. Output data demands interrupt the computer every 50 milliseconds. The System displays the computer-buffer inputs and outputs dynamically, and records all real time inputs serially on analog tape as they enter the Real Time Data Center. Communication between the direct data buffer and the 7044 main computer is of the demand-response type through the direct data connection on Channel B of the 7044. Communication from the 7044 to the analog computer and plotting boards through the digital-to-analog converter is via the direct data connection on Channel C of the 7044.

A Direct Couple System (7044-7094) has recently been installed in the Computer Directorate, and the present Athena program is being rewritten for the new configuration. The new program will rely largely on the 7044 side of the system to edit and format data to and from input-output 7288 subchannels, replacing in large part the function of the present Direct Data Buffering System.

**DEVELOPMENT OF THE ATHENA PROCESSORS**

Figure 2 outlines the organizational structure of the Athena real time program. The solid lines indicate a direct transfer of control from a processor to a subprocessor, in the manner of standard deferred time subroutines. The dashed lines represent a request for the execution of a processor by another processor. The function of each program in the system is described below.

1. **Priority Processor #1:** transmits, at the proper time, a command to ignite the third stage of the missile.
2. Priority Processor #2: transmits data to display plotters via digital-to-analog converter and an analog computer.
3. Priority Processor #3: logs computer inputs, outputs, and other pertinent data on tape during a mission for post flight analysis.
5. Priority Processor #5: edits and processes raw telemetry data messages, and transmits midcourse guidance commands to the missile.
6. Priority Processor #6: edits and processes input radar data, monitors the tracking status of the radars, and transmits acquisition and display data.
7. Priority Processor #7: formats acquisition and display data, and controls the linkage and synchronization between the processors generating acquisition data, range safety plots, and various digital displays.
8. Priority Processor #8: evaluates telemetry data from the missile after midcourse guidance commands have been transmitted to decide whether all parameters are within prescribed limits.
9. Priority Processor #9: evaluates radar data after second stage burn-out, and chooses a radar from which input data for the guidance equations can best be obtained.
A. Subprocessor RADCOR: a subroutine of Processors 7 and 9, RADCOR performs coordinate transformations and generates acquisition data.
B. Subprocessor PRED: a subroutine of Processor 7, PRED generates Kepler extrapolated impact prediction and other plotting board data.
C. Subprocessor DIRSIT: a subroutine of Processors 7 and 9, and of PRED, DIRSIT generates smooth positions, velocities and accelerations from raw radar position data.
D. Subprocessor SUB86: a subroutine of Processor 8, SUB86 computes predicted payload impact points.
E. Subprocessor EXTRAP: a subroutine of Processor 9, EXTRAP performs trajectory extrapolations for guidance equations inputs.
F. Subprocessor MATRIX: a subroutine of Processor 9, MATRIX solves the guidance equations, computes missile attitude commands for midcourse guidance, and determines third stage ignition time.
G. Subprocessor DFXWRS: a subroutine of PRED, DRXWRS selects a radar for instantaneous impact prediction plots on the basis of digitally filtered position and velocity data from all input radars.
H. Subprocessor PLOT: a subroutine of PRED, PLOT scales and formats plotting board data for transmission to the digital-to-analog converter.
I. Subprocessor CHECK: a subroutine of PRED, CHECK is a limiting subroutine for output data.

Note the high priority given to routines to transmit third stage ignition command and plotting board data for range safety use. This indicates that these functions must be executed within critical time frames. The data editing and processing routines are generally given next highest priorities since they must be completed before the next data samples reach the remote terminals. Functions that may span longer time intervals for completion, and which may, to some extent, be completed as available time permits, are given lowest priority.

REAL TIME STORAGE LIMITATIONS

Most real time programs currently in use are of the medium to large scale variety, as is the Athena program at White Sands. An immediate problem arises when the entire executable program is too large to be accommodated by the immediately accessible core at one time. Three techniques are used by the Athena program to alleviate this problem, and may be worthy of mention here.

First, all data input and initialization programs are loaded into core and executed, and are then chained over or overlaid by the real time processors and subprocessors, the two program links sharing only the data and control storage common to each. The same procedure is followed after execution of the real time phase of the job, with post flight routines overlaying the real time processors, retaining only those data and control words common to both.

Second, all data which are generated during the real time mission but are intended only for post-flight use, are moved out of the computer immediately via a direct data connection or onto logging tapes, to preserve storage within the main cores.
The 7044 currently in use at the Real Time Data Center does not have a disc or drum storage unit, hence cannot use these facilities for conservation of core.

Third, much preprogramming, formatting, filtering, buffering, and functions of this nature are handled externally by the input-output Direct Data Buffering System and by the analog-to-digital link between the 7044 and the plotting boards. This eliminates large blocks of digital programming as well as conserving computer time. Future real time work at White Sands will probably move more into this field of hybrid operation as systems become larger and more complex.

CHECK-OUT PROCEDURES

Check-out of real time programs requires methods not usually necessary in deferred time programming. First of all, of course, each module or functional processor must be checked out individually, as far as possible. If a multiprocessing job has been properly formulated, with each processor assigned well defined functions and communications with other program parts, this check-out at the processor level will be greatly simplified.

Repeatability of input data is necessary in order to reproduce the results of computation processes during check-out phase. Unfortunately, real time processes dependent upon external timing and data demands cannot usually be reproduced with any degree of precision. However, the ability to record actual input data during a real time process, in such a manner that it can be played back into the program exactly as it was recorded, assists the programmer in his check-out diagnostics. A related process is the establishment of a data simulator which will provide the real time program with repeatable data demands at prescribed time intervals. Both procedures have been used extensively in checkout at White Sands.

The complexity of interprocessor and monitor interactions is frequently exceeded by a maze of intercomputer and data communications problems. Errors in data formatting and transmission are difficult to trace during or following the completion of a real time process, pointing up a need for thorough diagnostic check programs for all external devices and intercomputer links prior to the execution of a real time job. Even with thorough premission check-out, the possibility always exists that some linkage device or external module may fail during a real time mission. Such possibilities demand the use of a limited amount of equipment monitoring by the program during a real time process. Upon occasion facilities can be provided for the switching of modules during the run. In case errors are detected, thus endowing real time jobs with an additional feature not usually found elsewhere, that of dynamic check-out.

DOCUMENTATION

The preparation for and writing of a real time program requires the close cooperation of a number of persons, among them the program administrators, coders, equipment specialists, and those establishing the functional requirements and job specifications. This, together with the extreme complexity of most existing real time systems, compels a formalization and documentation of requirements and procedures not usually needed for a standard programming job.

Once the functional requirements for a job have been set down, the program management group should incorporate the system functions into an arrangement of processors and subprocessors which will form the backbone of the programming assignments. This initial work must be done with great care and imagination, since the establishment of time dependent linkages and relative priorities among processors is basic and can be altered only with great difficulty once the programming work has begun.

After the overall system documentation and flow charting has been completed (ideally before any other work has begun), program specifications and detailed flow charts can be drawn up for each functional processor. Specifications must be controlled by someone familiar with all parts of the program so that logical program linkages may be defined. Detailed flow charts should be primarily the responsibility of the individual programmer, and just as in deferred time programs, can be compiled before, during, or after the program has been written.

Standards for program management and documentation need to be set down early in the game, and followed rigidly even when the pressures of time begin to close in on the programming team. Not only will these standards aid the program man-
ager and his real time staff during the program writing stage, but also will establish a basis for future changes and program module rewriting when that becomes necessary. The competent real time programmer will be called upon to use all the ingenuity and creativity he possesses to solve the problems he faces, and poor standardization and documentation will make his task more difficult and lead to a substandard quality of work.

Most of the difficulties encountered in the Athena and other White Sands real time systems have been due to breakdowns in communication and documentation. The establishment, check-out and maintenance of a sound real time system is sufficient in itself to tax the patience of the most erudite team, and the use of such a conceptually simple yet basic tool as documentation cannot be overemphasized.

REFERENCES
