SOFTWARE DESIGN REVIEW OF MAY 1993 AND DOCUMENTATION STATUS

L. D'Addario
2 June 1993

I. SUMMARY OF THE SOFTWARE DESIGN REVIEW

A comprehensive review of the software design of the Green Bank OVLBI Earth Station was held in May 1993, including a meeting on May 13 that was attended by JPL representatives and by NRAO staff from outside the project. The review included both the real-time and offline software components, as well as all external interfaces. Not all of the design was complete, with the main omissions being in the offline software for output data; the latter has not yet been given much attention, in view of the obvious need to give higher priority to input data processing and real-time control.

A large amount of progress had occurred since the limited internal review of December 1992 (see OVLBI-ES Memo No. 37) and since the Critical Design Review of October 1992 (see "Report on the Detailed Design Phase," 29 Oct 1992). The top level design of both offline programs for input data processing (the orbit data converter and the schedule converter) is now complete, and some algorithms have been prototyped. The structure of the real-time control software is now clear, with all internal data structures designed and each of the major components (schedule dispatcher, monitor/checker, antenna control, MCB communication, and timing) existing in at least a prototype form. The precise ways in which the VLBA code can be reused have been decided, and methods of code control have been adopted.

The details of the current design have been described in a series of design documents, the current versions of which are attached to this memo. Further information about our system of documentation is given in section II, below.

The agenda of the May 13 meeting is given in Appendix A, and a list of attendees is given in Appendix B.

Most of the discussion during the meeting concerned clarifications of small details of the design. There were a few major points made on general issues:
(a) There is concern about the continued lack of precise definition of some external interfaces, including the global schedule (input) file and the correlator timing corrections (output) file. The overall design attempts to isolate these definitions as much as possible, so that work on most of the system can proceed, but we are getting close to the point where final definitions will be needed. [Ulvestad]
(b) We have not shown convincingly that the project can be completed on schedule with the available manpower, which seems lean compared with other software projects of similar scope. [Petrie]
(c) It would be helpful to have quantitative measures for gauging progress, such as a count of new C functions required and completed. [Petrie]

I want to thank Ron Heald of the NRAO staff in Socorro for spending several days in Green Bank including this meeting, resulting in valuable advise about the use of the VLBA code and the establishment of more organized methods of code sharing; and Ken Sowinski, also of NRAO in Socorro, for providing written comments on the design documents. I also thank Bob Petrie and Jim Ulvestad of JPL for attending and providing useful comments.

II. DOCUMENTATION SYSTEM

Here I want to summarize the existing and planned documentation of the OVLBI ES software, and to provide information on its availability outside the OVLBI group in Green Bank. Some of this information applies to hardware design documentation also.

There are several levels of documentation being maintained during development, as described below. In addition, a set of Technical Manuals will be produced at the end of the project to describe the "as built" system and provide for its maintenance. The ongoing documents have the following characteristics:

A. OVLBI-ES Memo Series. This series consists of informal, unrefereed contributions from anyone on subjects related to the design of earth stations for orbiting VLBI. Each contribution is distributed to a world-wide mailing list of those who have requested to be on it. Contributions from outside NRAO are accepted, and the contents need not be specific to the Green Bank station. The Green Bank staff attempts to include in this series discussions of those aspects of our design that are likely to be of interest to people outside our group. Generally, these memos are *not* definitive descriptions of design decisions and may include tentative results. Each contribution is numbered and indexed. A separate mailing list is maintained for the index only; this is intended for those who want to be kept aware of results and who may request copies of certain issues, but who do not need every one. Finally, the current index and most of the memos themselves are available over the Internet by anonymous ftp; this is done by running the ftp program on your own computer and connecting to

sadira.gb.nrao.edu (192.33.116.115)

with username "anonymous" and any password. You should then change directory to "ovlbi/memoseries" and get the file README for further instructions.

B. Design Documents. This is a more formal series of documents which *do* represent definitive design decisions. They are specific to the Green Bank OVLBI Earth Station. At present they cover only software designs, but some hardware designs may be included in the future. Although they describe a definite design as of their issue dates, they are subject to change during development. For this reason and because of their specialized nature, they are not automatically distributed outside our group. Interested persons may request copies by sending e-mail to the responsible development team member or to me (ldaddari@nrao.edu). In addition, we intend to make copies available by anonymous ftp to sadira (as above) in ftp directory "ovlbi/doc." At the end of development, the final versions of these documents will be incorporated into the Technical Manuals.

C. Technical Notes. This is a very informal memo series of mostly internal technical information that we want to have recorded for reference. It may consist of laboratory notes, test results, memoranda of discussions on technical issues, and planning papers. Again, these documents will not be automatically distributed outside the Green Bank group, but copies will be sent to interested persons upon request. However, the items will be numbered and indexed, and the index will be available by anonymous ftp (as above) in ftp directory "ovlbi/notes." The text of some memos may also be available there, but it is not likely to be as complete as the other on-line document sets.

D. Source Code. The most definitive documentation of the software is contained in the code itself, in the form of commentary. All of the code is accessible over the Internet, but making it available by

anonymous ftp is considered too much of a maintenance burden for now; besides, making significant use of it would require more computer privileges than we are willing to grant anonymously. Anyone outside NRAO who would like access to these files can be given an individual user account on the Green Bank computers.

APPENDIX A: Meeting Agenda

GREEN BANK OVLBI EARTH STATION
SOFTWARE DESIGN REVIEW, 6 MAY 1993

AGENDA

0830-0930 Design Overview — D'Addario

1. Top level requirements
2. Separation into real-time and non-real-time components
3. External interfaces (non-real-time components)
4. Real time control system
   - 4.1 Operating system and environment
   - 4.2 Hardware configuration
   - 4.3 Top level data flow; internal data structures
   - 4.4 Operator interface
   - 4.5 Re-use of VLBA code
5. General issues: code control; schedule; current status

0930-1030 Input Data Processing

1. Orbit geometry calculations	Meinfelder
2. Schedule conversion (off line)	Langston
3. Command dispatcher (real time)	Meinfelder/Langston

1030-1100 (Coffee Break)

1100-1200 Real Time Control System -- Part I

1. Initialization	Varney
1.1 Normal cold start	
1.2 Emergency shutdown and automatic restart	
2. Timekeeping and task synchronization	Varney
3. Control functions that run as separate tasks	
3.1 Orbit geometry data handling task	D'Addario
3.2 Antenna pointing control task	D'Addario
3.3 Two-way timing control task	D'Addario

1200-1300 (Lunch Break)

1300-1430 Real Time Control System -- Part II

4. Control functions that run as Dispatcher subroutines	
3.1 Tape Formatter and Recorder control	Varney
3.2 Front ends and downconverters control	Varney
3.3 Two-way timing system setup	D'Addario
3.4 Demodulator and Decoder setup	Varney/Escoffier
5. Monitoring, checking, and logging task	Varney

1430-1500 Output Data Processing

1. Status File (near real time)	Varney
2. Timing Corrections File	Langston
3. Doppler File	Langston
4. Log Files	D'Addario

1500-1530 (Coffee Break)

1530-1630 Questions and Discussion

1630-1730 Software Demonstrations (at 45 foot antenna)

APPENDIX B: MEETING ATTENDEES

Dave Burgess, NRAO
Mike Balister, NRAO (Charlottesville)
Larry D'Addario, NRAO
Ron Heald, NRAO (Socorro)
Glen Langston, NRAO
Edmond Meinfelder, NRAO
Bob Petrie, JPL
Bill Shillue, NRAO
Jim Ulvestad, JPL
Doug Varney, NRAO

APPENDIX C: LIST OF CURRENT SOFTWARE DESIGN DOCUMENTS

Author(s)	Title
D'Addario	"Software Overview"
Langston	"Off-line Scheduling Software"
Varney	"Initialize Function"
Langston	"Real-Time Command Dispatcher"
Varney	"Monitor and Check Task"
D'Addario	"Log Writer Task"
D'Addario	"Geometry Task"
D'Addario	"Pointing Task"
D'Addario	"Two-Way Timing Control Task"
Varney, D'Addario	"Status Task"
Escoffier	"Interface Protocol Between Decoder and Station Computer"

(Copies of all of these documents are attached to this report immediately following this page.)

File overview.doc, version 1.1, released 93/05/09 at 11:08:53

GREEN BANK OVLBI EARTH STATION: SOFTWARE OVERVIEW

L. D'Addario
6 May 1993

1.0 EARLIER WORK

The software design of the Green Bank OVLBI Earth Station was described in the preliminary design report of July 1991 [1] and the detailed design report of October 1992 [2]. While some significant changes have been made since these reports were published, the overall structure and top-level requirements have been stable. Therefore, some familiarity with this earlier work will be assumed here.

2.0 TOP LEVEL REQUIREMENTS

The software system must implement automatic operation of the station on the basis of two externally supplied files, one giving the ephemeris of a satellite to be tracked and the other giving a sequence of events (schedule) to be followed. In principle it is sufficient if the schedule merely specifies the starting and ending times of each tracking pass, the name of the satellite, and (in some cases) a number specifying the satellite's operating mode; but we expect the actual schedules to contain much more detail than this. The station then produces several external outputs: a wideband data tape of astronomical signals downlinked from the satellite; processed log files containing records of events that occurred during the pass and needed by various other mission elements, including correlators; a satellite time correction file, needed by correlators; a file containing integrated Doppler tracking data, needed by orbit determination centers; and file that summarizes the current status of the station.

Some processing of the external input and output files is allowed to occur in non-real-time before and after a scheduled tracking pass, and this processing may involve some manual intervention. But during a tracking pass operation should be completely automatic except for the scheduled changing of tapes. Nevertheless, an operator will be on duty to deal with emergencies, so the real time software must provide him with current status information and must activate alarms in the event of serious difficulties.

The external inputs and outputs must conform to interface specifications that have been agreed upon by the relevant providers and users, respectively, of the data. At this point, not all details of these agreements are in place. Our present understandings and assumptions are given in a series of NRAO specifications [3]. In particular, the formats of the global Schedule File and of the Correlator Input Log are completely unspecified, although we have a good understanding of their main logical contents. The other interfaces are rather well defined.

If either the wideband data link or one of the timing links fails or has its signal-to-noise ratio drop below a predefined threshhold, we say that a link dropout has occurred. The station is required to respond to this automatically by following fixed algorithms; generally, data recording continues to the extent possible. If the lost signal is later restored, the station must respond by executing a fixed recovery algorithm based on the type and

duration of the dropout.

If commercial external power to the station is lost, spacecraft tracking and data recording cannot continue. Nevertheless, sufficient power will be maintained on critical circuitry to effect a quick and orderly recovery when external power is restored. This will be done by a battery-based uninterruptible power supply connected to the station computer, computer communication (LAN) hardware, and station timekeeping hardware. Upon restoration of power, recovery is required to be automatic (without operator assistance). The station computer must execute a fixed restart algorithm which includes initialization of all hardware subject to the power loss, establishing the station in the new desired state (not necessarily the same state as it was in when power was lost), and acquisition of signals from the satellite being tracked (if any).

3.0 OFFLINE VS. REAL-TIME SOFTWARE

Figure 1 gives an overview of the software system. Except for the wide-band data tape (not shown), all external interfaces will be implemented as electronic transfers of files over the Internet. (A backup mechanism is also specified; see A34300N001.) For several reasons, including a desire to keep the design of the critical real-time software independent of the interface specifications, most of the interface files will not be directly read or written during a tracking pass. Instead, the input files Schedule and Predicted_Orbit are processed by the offline programs SchedCo and Geometry to produce internal files that control the real-time operation; and the output files TimeResiduals and LogFile are processed by other offline programs to produce several files needed by the other mission elements.

In addition to keeping the real time system independent of the interfaces, this reliance on offline software components allows several simplifications. For example, the processing of the internal command file can be purely sequential, with no need for loops or for looking ahead; any actions needed in preparation for future events in the Schedule File can be inserted into the Command File by SchedCo. Also, some of the output file processing may require data that is not readily available in real time, such as the results of ionosphere monitoring measurements used as corrections to the timing residuals.

4.0 REAL-TIME CONTROL SYSTEM

Figure 2 is a top-level data flow diagram of the real time software system. Omitted from this diagram are tasks that provide interfaces for a human operator, since these run only when requested and then they run in parallel with and independently from the automatic tasks.

4.1 Hardware Configuration

The Station Computer consists of a single-board VME computer (Motorola MVME147S) with a 68030 processor, floating point coprocessor, 16MB of RAM, and various I/O controllers. Most of the earth station electronics -- including the receivers, transmitter, antenna servo, and recording system -- are connected to the Station Computer by a fast serial bus known as the VLBA Monitor/Control Bus (MCB). This is accomplished through a VME "intelligent controller" board (Motorola MVME331) that resides in the same VME chassis as the CPU board and is driven across the VME backplane. There is one major hardware module that is not connected via the MCB; this is the Decoder, which is a special purpose digital signal processor constructed on two VME cards that are also installed in the same chassis as the station computer. The Decoder is therefore controlled

and monitored directly over the VME backplane. Finally, two rather simple (from the software viewpoint) pieces of commercial hardware, not yet acquired, must be connected to the Station Computer, probably through RS232 serial lines; these are the GPS timing receiver and the Uninteruptable Power Supply.

The computer has access to disk file systems on other computers in Green Bank via the fiber optic ethernet LAN, using the ethernet interface built into the MVME147S. During operation, the required input and output files will reside on a local hard disk (not yet acquired) connected to the MVME147S via its built-in SCSI bus controller.

Several of the major electronics modules contain imbedded microcomputers that give them significant local "intelligence." These are the Two Way Timing subsystem (which uses a Digital Signal Processor), the Formatter, the Recorder, and the Decoder. The internal software considerations for these devices will not be considered here.

4.2 Operating System and Software Environment

The Station Computer runs the VxWorks real-time operating system. A description of the features and characteristics of this OS is beyond the scope of the present report, but the major points are these: It is a multi-tasking system with excellent real-time performance and predictable timing. Extensive support of TCP/IP network functions is included. The system is single-user and supports few development utilities in the target computer, since it is intended that development work will be done using cross-compilers and debuggers on a remote workstation. Extensive libraries of C-callable functions are provided, including functions that conform to standard C and UNIX conventions. Although much VxWorks code is similar to UNIX, there are some major differences. The chief one is that all static and external variables in VxWorks are global and available to all tasks, as is all of physical memory. This makes real-time programming more convenient and efficient than in UNIX, but it demands more discipline to avoid surprises.

VxWorks also includes a user shell program, which runs as its own task. From the shell, object modules can be loaded and any of them can be called for execution either within the shell's environment or spawned as separate tasks.

4.3 Organization by Tasks

As Figure 2 indicates, the real time system is organized into several separate tasks. Once spawned, these all run independently. Here we give a brief and simplified description of the initialization process and of each task.

Initialization: At power-up, a boot ROM causes the VxWorks kernel and shell to be loaded and a startup shell script to be executed. The shell script loads all of our application code and then calls the initialize() function. Initialize sets up certain internal data structures, spawns the major tasks that must run continuously, then exits.

Timekeeping task: The first task, tic, provides global timekeeping. It is activated by the OS at each system clock interrupt, normally 64 times per second. This runs from a crystal oscillator on the CPU board, but it is kept synchronized to absolute UTC by separate interrupts from the maser-driven 1 Hz pulses. Tic does two things: it maintains precise date, UTC, and local sidereal time in a global structure; and it broadcasts two semaphores for synchronizing the execution of other tasks. The semaphores are called

slowSem, nominally issued once per second, and fastSem, nominally 16 times per second.

Dispatch task: This is perhaps the most complex task, in that it forms the master controller of the station. Its job is to execute commands from the Command File at their scheduled times. These commands include the setting up of all station hardware for a satellite tracking pass. As events occur due to command executions, Dispatch keeps track of the currently intended state of the station in an internal structure, stationState. It also causes each event to be recorded in the Log File.

Geometry task: This task reads the internal Orbit File and calculates geometrical parameters needed for pointing the antenna and for controlling the up- and down-link synthesizers of the Two Way Timing subsystem. It passes the results of these calculations to other tasks via an internal structure. Geometry is spawned by Dispatch in response to a TRACK command, and runs each time that slowSem is given.

Point task: This task computes the actual pointing angles of the antenna and sends them to the antenna's servo via the MCB. This includes applying the pointing correction model, with refraction. It is spawned by Dispatch under a TRACK command, and it runs each time that fastSem is given (nominally 16 Hz). Frequent execution is needed to ensure accurate tracking. The angles computed by Geometry at less frequent intervals are interpolated.

TWTControl task: Here the synthesizer phases for the two way timing are computed and sent to that system's local processor, based on range and range rate data passed by Geometry. In addition, the measured down link phase residuals are collected and written to the Timing Residuals File. This task is also spawned by Dispatch and runs each slowSem.

MonChk (Monitor and Check) task: This is a rather complex task that handles all monitoring of the station hardware, verification of proper operation, and routine logging of measurements. It operates by referring to an internal structure, monchkList, that specifies a sampling rate and logging rate for each quantity of interest, as well as the address of a function that will obtain the measurements from the hardware and verify their correctness. When an anomaly is discovered, an entry in an internal Anomalies array is updated and a record is written to the Log File. Anomalies can have several levels of seriousness, from "warning" to "emergency"; the more serious ones also generate alarms to the operator. When an "emergency" occurs, MonChk calls the shutdown() function, which puts the station into a safe state and discontinues operation; if the emergency is later cleared, MonChk calls the restart() function, which causes automatic resumption of operation.

LogWriter: The actual writing of records to the Log File requires a separate task since only one task can open a given file. The records are passed to it via a message queue. Time stamps are added, and they are written to the file on a FIFO basis.

Status: This task provide a near-real-time interface to the station monitoring for external users. It runs about every 5 to 10 minutes at low priority. Its job is to take a "snapshot" of the station's status by examining the stationState, Anomalies, and Monitor internal data structures and producing an annotated summary as a human-readable ASCII file. The file is written first to the real time system's disk and then copied to another file system on a publically-accessible computer.

5.0 OPERATOR INTERFACE

We have adopted a set of operator interface routines written for the VLBA station control system called SCREENS. This allows the display on any VT100-type terminal of several "windows" into the operation of the real time system. Typically, an individual "screen" allows the display and sometimes the control of elements of a particular hardware module. Several "screens" can be run at once. Each runs as a separate task and does not disturb operation of the automatic control system described above.

We are using without modification many VLBA software modules that implement screens for the Recorder and Formatter, and we will adapt others for which we have similar hardware (e.g., front ends and cryogenics). We are also using the underlying support routines common to all screens, and this allows us to create new screens to support earth-station-specific hardware like the Decoder.

6.0 CODE RE-USE

In addition to use of the VLBA screens package, a large number of monitor and control functions can be used without modification. These include especially those associated with the Formatter and Recorder. Others can be used with slight modifications.

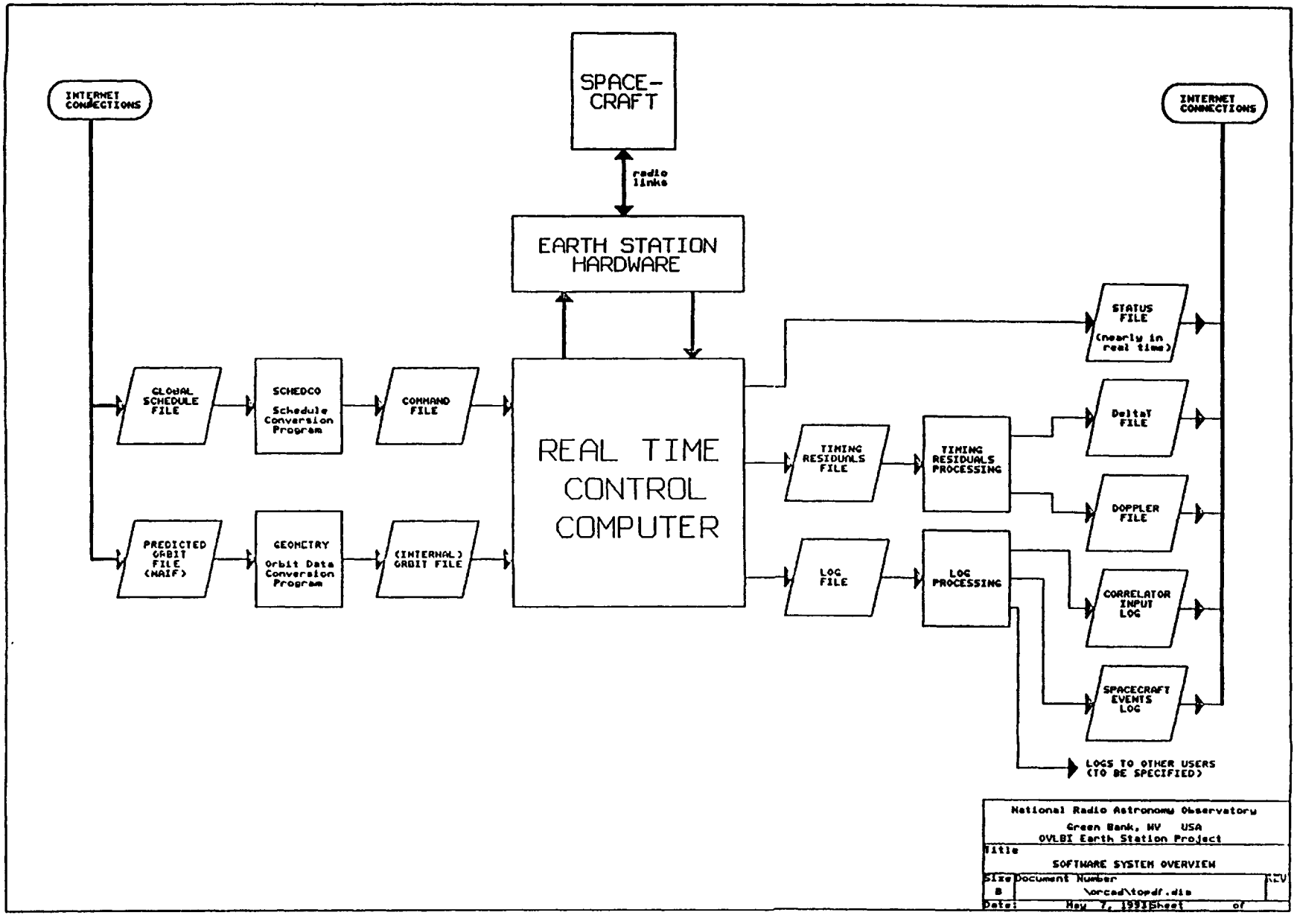

INTERNET CONNECTIONS
SPACE-CRAFT
radio links
INTERNET CONNECTIONS
EARTH STATION HARDWARE
GLOBAL SCHEDULE FILE
SCHEDCO Schedule Conversion Program
COMMAND FILE
REAL TIME CONTROL COMPUTER
STATUS FILE (nearly in real time)
TIMING RESIDUALS FILE
TIMING RESIDUALS PROCESSING
DeltaT FILE
DOPPLER FILE
PREDICTED ORBIT FILE (NAIF)
GEOMETRY Orbit Data Conversion Program
(INTERNAL) ORBIT FILE
LOG FILE
LOG PROCESSING
CORRELATOR INPUT LOG
SPACECRAFT EVENTS LOG
LOGS TO OTHER USERS (TO BE SPECIFIED)
National Radio Astronomy Observatory
Green Bank, WV USA
OVLBI Earth Station Project
Title
SOFTWARE SYSTEM OVERVIEW
Size B
Document Number
\orcad\topdf.dia
REV
Date: May 7, 1991 Sheet of

FIGURE 1

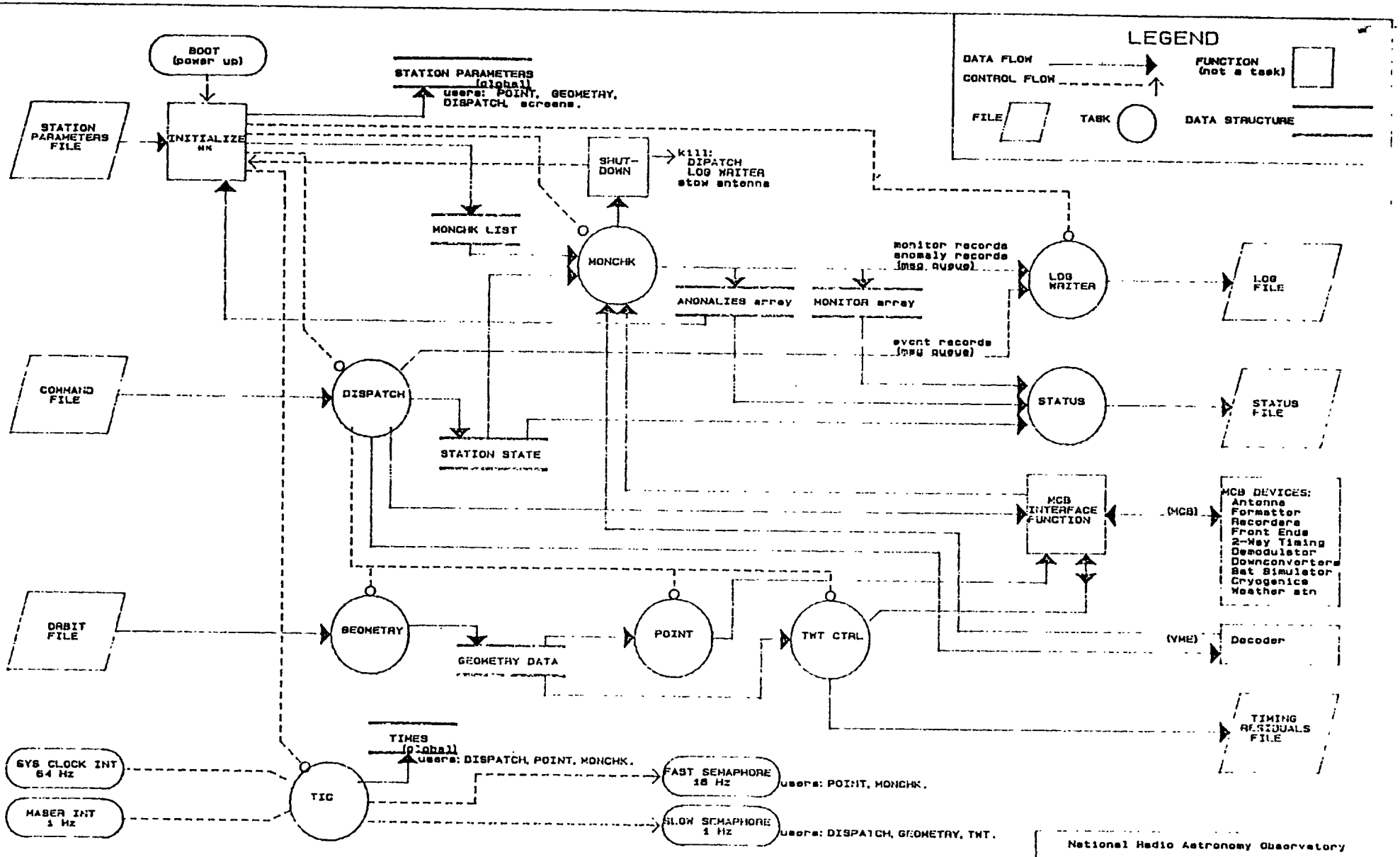

LEGEND
DATA FLOW
CONTROL FLOW
FUNCTION (not a task)
FILE
TASK
DATA STRUCTURE
BOOT (power up)
STATION PARAMETERS (global) users: POINT, GEOMETRY, DISPATCH screens.
STATION PARAMETERS FILE
INITIALIZE ##
SHUT-DOWN
kill: DIPATCH LOG WRITER stow antenna
MONCHK LIST
MONCHK
monitor records anomaly records (msg queue)
LOG WRITER
LOG FILE
ANOMALIES array
MONITOR array
event records (msg queue)
COMMAND FILE
DISPATCH
STATUS
STATUS FILE
STATION STATE
MCB INTERFACE FUNCTION
(MCB)
MCB DEVICES:
Antenna
Formattor
Recorders
Front Ends
2-Way Timing
Demodulator
Downconverters
Sat Simulator
Cryogenics
Weather stn
ORBIT FILE
GEOMETRY
POINT
TWT CTRL
(VME)
Decoder
GEOMETRY DATA
TIMING RESIDUALS FILE
TIMES (global) users: DISPATCH, POINT, MONCHK.
SYS CLOCK INT 64 Hz
MASER INT 1 Hz
TIC
FAST SEMAPHORE 16 Hz
users: POINT, MONCHK.
SLOW SEMAPHORE 1 Hz
users: DISPATCH, GEOMETRY, TWT.
When INITIALIZE is called from SHUTDOWN, it runs as a separate task while awaiting clearance of emergency conditions.
NOTE: Operator interface tasks not shown.
National Radio Astronomy Observatory
Green Bank, West Virginia, USA
ORBITING VLBI EARTH STATION PROJECT
Title
REAL TIME SOFTWARE SYSTEM - DATA FLOW
Size B
Document Number \orcad\dataflow.dia
REV
Date: May 11, 1993 Sheet of

FIGURE 2

Off-line Scheduling Software Design Document

Glen Langston

Draft 93 May 7

Overview

This document summarizes the off-line scheduling conversion (SCHEDCO) software for converting the global schedule into the local commands used by the OVLBI GB Earth Station real-time system. The major function of SCHEDCO is to select the global schedule requirements relevant to the GB Earth Station and expand them into sequences of timed commands for the real-time system. SCHEDCO will determine on which wrap of the azimuth mount the antenna should start the observations. SCHEDCO will calculate the satellite range during the tracking pass in order to estimate the appropriate receiver gain settings. SCHEDCO will also generate commands for all station operations that are not specified in the schedule file, including tests. The ASCII command file format is specified in the Command DISPATCH design document.

SCHEDCO will output to the tape inventory database the serial numbers of tapes to be used during recording and the tape status at the end of tracking. Another output of SCHEDCO is a list of all significant events during an observation, such as tape changes, etc. This output will be used by the operator and station manager to schedule their activities.

Design Philosophy

The schedule file format has not yet been determined, so in order to progress, a schedule file format must be assumed. It is assumed the global schedule will not specify the details of the operation of the GB Earth Station beyond A) indicating which satellite will be tracked, B) the start and stop times for tracking, transmitting and recording, C) satellite events (tone transmission, changing observing frequency and data rates), and D) destination (and format) of down-link data. It is also assumed that one global schedule file may be provided for all mission elements (stations, satellites and correlators), and SCHEDCO will select schedule events relevant for the GB Earth Station. The details of the schedule file format should not be important for the overall design of SCHEDCO.

Determination of the current status is an important input for creating the proper sequence of real-time system commands. A SCHEDCO design goal is to allow user input of all station, satellite and tape states, but the normal mode of operation will be to determine these states from local databases. The scheduled status database will be updated by SCHEDCO. The actual status will updated by the the real-time system, the tape inventory system and the station operator.

Times

It is assumed that all tracking station parameters (slew times, etc) will be unknown to the global scheduling process. This implies that schedule start and stop times should correspond to the successful begin and end of the relevant tracking, transmitting and recording operations. SCHEDCO must schedule initialization, tests, and other preparations (ie. tape movement) before scheduled start times.

The GB Earth Station specification A34300N002 requires that events be recognized with a resolution of 1.0 sec; that is, events less than 1.0 sec, apart may be treated as occurring at the same time. Further, events will be implemented with an absolute accuracy (relative to UTC) of 0.1 second. The SCHEDCO processing of commands will be accurate to at least 0.01 seconds, in order to avoid compromising the design goal. Internally, all times will consist of two components *1) mjd*, the Modified Julian Day number, and *2) utc*, the UTC time, in units of radians (range 0 to $< 2\pi$).

Inputs

It is expected that several global schedule files will be input to SCHEDCO, and that SCHEDCO will parse all files to create commands for the requested time period. User inputs to SCHEDCO are A) the range of times for which commands should be generated, B) state of the GB Earth Station at command start (usually "stow"), C) (optionally) the schedule directory where schedule files are kept; all files are examined to determine commands for the relevant period, D) (optionally) the list of un-available tape drives, E) (optionally) the serial number of the tape for recording data, and F) (optionally) the satellite tracking parameters (Keplarian orbital parameters). The user inputs are discussed in term.

A The schedule files are anticipated to contain plans for observations for long periods, but SCHEDCO will create command files for any desired time period.

B Besides the GB Earth Station "stow" state, other states include "tracking VSOP" and "tracking ASTRON", to allow switching between satellites.

C The schedule file will be placed in a central directory before execution. The central directory will be named `$GBES_HOME/schedules` , where `$GBES_HOME` will be a directory name defined in the unix environment.

D Either of the two GB Earth Station VLBA tape drives can be used in a recording session, but if a tape drive is not functioning, the other may be used. If only one drive is functioning, different procedures will be executed during tape changes.

E The serial number of the tape for data recording will normally be chosen via tape inventory software. This software will determine what tapes are available and whether data may be written onto a partially filled tape. The SCHEDCO user may over ride this software and specify the tape and start position on the tape.

F In order to determine the optimum azimuth "wrap" to start tracking the satellite, the satellite azimuth and elevation as functions of time must be available to SCHEDCO. In normal operation, functions of the GB Earth Station geometry/orbit software will be used to access the orbit data (which uses the NAIF package). Optionally, the Keplarian

orbital parameters for the satellite may be entered by the user to calculate the azimuth and elevation. The geometry/orbit software is discussed in a separate Orbit Conversion (ORBITCO) design document. The range to the satellite will also be calculated in order to determine appropriate receiver gains. SCHEDCO will minimize the number of receiver gain changes during tracking passes.

The input schedule file is assumed to be an ASCII human readable file written in free format (blanks, tabs will be correctly parsed). The schedule file will contain items tagged by time, tracking station pointing location (orbit). It is expected that the schedule file will be fairly terse. The types of schedule items are: [1]

Track Start: Start tracking pass for specified spacecraft: causes the antenna to be pointed and receivers to be tuned so as to be ready to *receive* signals from the spacecraft at or before the specified time. Thereafter, acquisition of down-link signals will occur automatically, and all signal normal processing will occur *except* that the up-link transmitter will not be turned on and the tape recorders will not be started.

Transmit Start: Turn on up-link transmitter, begin transmitting timing reference signal, verify acquisition by spacecraft, and begin recording two-way time residuals. When this has been accomplished, perform timing initialization sequence.

Record Start: Begin wide-band tape recording of down-link data. Recording will actually begin when valid data is being acquired or at the specified time, whichever is later. Thereafter, recording continues until a "Record Stop" event is specified, even if the data becomes invalid. Gaps in recording may be caused by tape reversals and tape changes; such tape management issues are regarded as internal to the ES and not part of the schedule file (however, such events will be logged by the ES and will be known post facto).

Spacecraft Mode: Specify a change in the operating mode of the spacecraft's receivers, digitizers, or other parts of the pay-load. It is primarily a direction to the spacecraft controllers, but if this information is available at the GBES it will be used to check the status bits in the down-link headers against the intended state; if they disagree, that fact will be logged.

Record Stop: This does not imply any particular tape management other than stopping the tape motion. Tracking and transmission may continue.

Transmit Stop: All transmission to the spacecraft will cease, although tracking and recording may continue.

Track Start: ES is no longer assigned to track this spacecraft, and may do something else until the next relevant event in the schedule file (including maintenance, tape changing, or sitting idle). It will force "Transmit Stop" and "Record Stop" if these have not already occurred. If a new "Track Start" occurs first, then "Track Stop" is done automatically before starting the new pass.

Event Tone Start: The satellite will transmit a tone at a specified frequency. The GB Earth Station will change mode in order synchronously detect this tone.

Event Tone Stop: Signals an end to satellite tone transmission mode and requires transition back to normal GB Earth Station data processing.

Event Freq. Set: The satellite will set it's frequency of observation to a specified frequency.

Tape Format: Several possible tape recording modes are allowed. The global schedule file must indicate which mode is appropriate for each observation.

A hypothetical format is shown in table 1.

[1] In the schedule items list, "reasonable" default states are assumed. For example a tape format default for VSOP will be assumed if not specified. Also certain actions will be assumed; if record start is scheduled, but transmit start has not, transmission will not start. However, if track stop is encountered before transmit and record stop, transmit stop and record stop will be scheduled.

UTC Time	Station(s)	Satellite	Action	Parameter
1993 April 13 1:05	GBES	VSOP	Tape Format	< *Parameters* >
1993 April 13 1:00	GBES	VSOP	Track Start	
1993 April 13 1:05	GBES		Transmit Start	
1993 April 13 1:10	GBES		Record Start	
1993 April 13 1:15	GBES		Event Tone Start	< *EventParameters* >
1:20			Event Tone Stop	
2:00			Event Tone Start	< *EventParameters* >
2:05			Event Freq. Change	< *EventParameters* >
2:05		VSOP	Event Tone Stop	
2:40	GBES	VSOP	Event Tone Start	< *EventParameters* >
2:45	GBES	VSOP	Event Tone Stop	
2:50	GBES	VSOP	Record Stop	
1993 April 13 2:40	DSN-X	VSOP	Track Stop	
1993 April 13 2:55	GBES	VSOP	Transmit Stop	
2:55	DSN-X		Transmit Start	
1993 April 13 3:00			Record Start	
1993 April 13 3:00	GBES	VSOP	Track Stop	
1993 April 13 4:00	GBES	ASTRON	Track Start	
1993 April 13 4:10	GBES	ASTRON	Transmit Start	
1993 April 13 4:15	GBES	ASTRON	Record Start	
1993 April 13 5:55	GBES		Track Stop	
8:55	DSN-X		Track Stop	

Table 1: Hypothetical Schedule file, showing example values for elements.

The SCHEDCO design assumes that if an element of a schedule file item is not included (ie satellite or station), then the previous value is assumed. For example, in the hypothetical Schedule file, the first 12 items are assumed to refer to GB Earth Station. The first 17 items refer to satellite VSOP and the last 5 items refer to Radio Astron.

The components of the Event Parameters are not yet know, but are assumed to be a list of a fixed number of numeric values.

Outputs

For each schedule file, one output command file will be placed in the directory `$GBES_HOME/commands` and given the name `commands.<UTC DATE>` . The `<UTC DATE>` part of the name will correspond to the earliest command time in the command file. This will typically be earlier than the times in the schedule file.

Also an operator summary of significant tracking station events will be placed in the directory `$GBES_HOME/operator` . The file name will be `operator.<UTC DATE>` . The `<UTC DATE>` will match the command file `<UTC DATE>` . The operator summary will note tape changes and other activities which may require human intervention.

The output command file will be in human readable ASCII format. The command file is specified in the command dispatcher design document. Commands will be optionally tagged by a start time for the operation, or the time tag for the previous command will be assumed. The command lines will be sorted into increasing time order. It is assumed that no command in the file will be executed until its appropriate start time *and* the previous commands in the file have been executed.

Command Expansion

The expansion of each of the schedule file item into a command file commands will involve three steps *a)* determining the current tracking station state, *b)* a table "look-up" of required command expansions and *c)* determining the times required to execute each command.

For each schedule item, a list of real-time system commands will prepared for each possible station status. Each real-time commands have an associated timing structure. The timing structure will consist of two parts; *1) execution time*, the time required for the function to be full filled, and *2) advance time*, the time between completing execution and the time before the command must be complete. The *execution time* is needed to allow proper sequencing of commands. The *advance time* will allow certain commands to be completed well before required start time, in case problems are detected during execution.

An example of a command needing a significant *advance time* is the tape test command, this test will take only a few minutes to execute, but should be done well before ($\sim$ 1 hour) recording, so that if problems occur, they may be fixed.

Most of the real-time system commands need very short times ($<$ 1 second) for execution. Commands requiring longer timer are certain formatter commands needing ($\sim$ 7 seconds) and telescope movement commands. The telescope slew rate is 40°/minute and the worst case azimuth movement from one wrap to another can require 9 minutes. (This also shows

the importance of being on the correct antenna warp at the start of a tracking pass.)

Antenna Wrap and Satellite Range

The antenna wrap will be determined by a simple modeling method. Upon finding a tracking start command, a wrap will be chosen and the azimuth and elevation will be calculated until tracking stop command is found. If a wrap condition is detected during the pass, the alternate wrap for start of tracking will be chosen.

During the modeling process, the range to the satellite will be calculated. From the range, the received power will be calculated as a function of time. If the receiver dynamic range exceeds tolerances, SCHEDCO will schedule commands to change gains during a tracking pass. The number of scheduled gain changes will be minimized.

Appendix A: Naming convention

It is proposed that all control files be kept in directories in a central directory called `$GBES_HOME`. Upon receipt, the schedule files will be placed in the directory `$GBES_HOME/schedules`, and given names with the format `schedule.<UTC DATE>`. The `<UTC DATE>` format for file names will be `YYMMMDD.hh:mm:ss` , where `YY` is the calendar year and `YY` < 50 implies `20YY`, `MMM` is a three character month (ie `JAN`) and `DD` is day of the month. The Time format is `HH:MM:SS` and consists of `HH` integer hours (range 0-23), `MM` integer minutes (range 0-59) and `SS` seconds (range 0-59). The command file for April 13, 1993 at 1:00 UTC would be named `command.93APR13.01:00:00`.

File initdesign.doc, version 1.1, released 93/05/12 at 19:21:46

INITIALIZE DESIGN DOCUMENT

D. Varney
10 May 1993

1.0 OVERVIEW

1.1 INITIALIZE must set the initial schedules for logging and monitoring of station hardware. A critical element of this routine will be to recover from power failures of various degrees of severity. INITIALIZE is also responsible for setup of the STATION PARAMETERS structure and spawning the tasks that rely on semaphores (TIC, DISPATCH, MONCHK, LOG_WRITER and STATUS).

2.0 DEPENDANCIES

2.1 This function is dependent on the STATION_PARAMETERS input file.

2.2 The INITIALIZE function will be called from the boot script and the SHUTDOWN function.

2.3 INITIALIZE will be passed a start parameter, either WARM_START from MONCHK or COLD_START from the boot script.

3.0 DESIGN CONSIDERATIONS

3.1 It will be necessary to distinguish the following starting conditions:

a. cold start after normal shutdown,
b. cold start after computer failure,
c. or warm start (e.g., station computer on battery).

3.2 The function must initialize the STATION_PARAMETERS structure and spawn all normal tasks including TIC, DISPATCH, MONCHK, LOG_WRITER and STATUS.

3.3 A method of automatically restarting the station after an automatic shutdown must be provided.

4.0 ALGORITHM DESIGN

4.1 In COLD_START mode, INITIALIZE copies station specific data (including pointing correction coefficients, tape recorder calibration data, and similar paramters) from the Station Parameters file to the STATION_PARAMETERS structure, and it copies the monitoring and logging intervals from the Station Parameters file to the appropriate elements of the MONCHK_LIST array. Then it spawns each of the station operation tasks in the order TIC, MONCHK, LOG_WRITER, DISPATCH, and STATUS. (The order is important because of the interdependencies of these tasks.)

4.1.1 The STATION_PARAMETERS structure is globally accessible and contains information required by several tasks the real-time system. It will contain substructures station and tapeparm[] that comform as closely as possible to the corresponding VLBA structures in order to facilitate the porting of some VLBA functions.

4.2 In WARM_START mode, INITIALIZE assumes that all

initializations have already been performed but that tasks DISPATCH and LOG_WRITER have been killed by the SHUTDOWN function (running under the MONCHK task) in response to detection of an emergency-level error. INITIALIZE then enters a loop that repeatedly scans the ANOMALIES array until it finds that emergency-level errors no longer exist. It then restarts LOG_WRITER and DISPATCH, passing the WARM_START parameter to DISPATCH.

4.3 The functionality of the INITIALIZE routines is described below.

```
-------------------------------------------------------------------------

function INITIALIZE(startup_mode)

IF COLD_START                                   -- start from boot script
  read STATION_PATAMETERS file                  -- read the file and initialize
                                                -- the STATION_PARAMETERS
                                                -- memory structure
  start tasks                                   -- spawn startup tasks
                TIC,
                DISPATCH(startup_mode),
                MONCHK,
                LOG_WRITER,
                STATUS
OTHERWISE                                       -- if a WARM_START; computer up
                                                -- but other hardware down,
  read ANOMALIES data                           -- get current error condition

  WHILE (EMERGENCY condition)                   -- if EMERGENCY, then
    delay one second                            --   allow other tasks to run
    read ANOMALIES data                         --   repeat error check
  ENDWHILE

  CALL RESTART                                  -- restart station operations
-------------------------------------------------------------------------

function RESTART                                -- resume station operation

    resume tasks
                DISPATCH(startup_mode),         -- wake up suspended tasks
                LOG_WRITER

-------------------------------------------------------------------------
```

Real-Time Command DISPATCH Design Document

Glen Langston

Draft - 93 May 7

Overview

This document summarizes the command dispatcher (DISPATCH) for the GB Earth Station real-time system. The major function of DISPATCH is to read commands from the command file and execute them at the correct UTC time. DISPATCH is started by the INITIALIZE function. DISPATCH initializes many of the real-time system functions. DISPATCH initiates the GEOMETRY. POINT and TWOWAYTIME tasks as well as TAPE functions. DISPATCH directly activates the formatter and tape drives. The antenna motion is directed both by DISPATCH and POINT. DISPATCH directly activates motion to STOW the antenna and set the breaks, while normal antenna tracking is performed by the POINT task.

Only the DISPATCH task will update the STATION_STATE structure which describes the commanded state of the GB Earth Station. The MONCHK task compares STATION_STATE with the actual state of the GB Earth Station and logs discrepancies.

DISPATCH will *not* perform any direct output to disk files. If DISPATCH error conditions are found, these errors are recorded via the LOGWRITER task.

Design Philosophy

The DISPATCH design philosophy assumes that no control loops should be closed; if an error condition is detected, DISPATCH should only raise error conditions and log them, but continue executing the commands. DISPATCH will run independently of any input other than the command file.

Times

The current station time is maintained in a global structure by the TIC task, and this structure is used by DISPATCH to control the actual time of execution of commands. [1]

The INITIALIZE function ensures that TIC is running, but DISPATCH also checks that the TIC task is running before beginning command execution. Internally, all times will consist of two components *1) mjd*, the Modified Julian Day number, and *2) utc*, the UTC time, in units of radians (range 0 to $< 2\pi$). DISPATCH checks whether to execute commands at 1 second intervals, by taking the 1 Hz semaphore. Upon taking a 1 Hz semaphore, DISPATCH executes all commands whose time has come. DISPATCH continues execution until encountering a command whose time has not yet come. If a future time command is

[1] DISPATCH uses `extern struct times time` to determine the current time. The values in this structure are maintained by the TIC task.

encounter, DISPATCH waits for the next 1 Hz semaphore before again checking whether the time has come for the next command. If the time for the next command has not come, no commands are executed and DISPATCH again waits for the 1 Hz semaphore. Note the reliance on the 1 Hz semaphore implies that the time resolution in DISPATCH is limited to 1 second.

Inputs

The inputs to DISPATCH are via command-line arguments and a command file. At startup, the default action of DISPATCH is to search a (hard-coded) file for the name of the command file for the current time. This command file is opened and the commands are executed until end of file is reached. In order to change the command file used by DISPATCH, DISPATCH must be halted and re-started. DISPATCH also takes the command file name as an argument, to allow overriding the default command file name. If necessary, the antenna operator will change the command file name by halting the DISPATCH task and restarting DISPATCH with a new file name. It will be possible to restart DISPATCH via the SCREEN package.

DISPATCH is designed to allow recovery from three types of fault conditions; Cold Start, Warm Start and Hot Start. These types of starts are described in the INITIALIZE design document.

Each DISPATCH command file line can be thought of as having four components; *a)* optional Date, *b)* optional Time, *c)* the command Name, and *d)* command Parameters. The Date, Time, Name and Parameters are separated by blanks. One and only one command is on each command file line. The input Date, Time and Name are case-insensitive and extra blanks between these items are ignored.

Upon encountering a command without a Time, DISPATCH executes it immediately. Until DISPATCH encounters a command with a valid Date, the time of the command is interpreted as the time on the command line *plus* the time when the command file was opened. (Ie. the times commands are *relative* to the time execution began.) Upon encountering a valid Date, DISPATCH executes that and all following commands assuming the time on the command line is exactly the *utc* time when the command should be executed. (Ie, it is an *absolute* time.)

If the command has a time but no date, the date of the previous command is assumed. A sample command file is shown in table 1. This command file INCLUDEs a second command file shown in table 2.

DISPATCH contains one set of subroutines which parse the Date, Time and Name of the command. Because different commands have very different parameters, the parameters are parsed by a function specific to each command.

		RESET	
		STARTUP	
		INCLUDE	./command.macros
		AUTOTEST	
		SATELLITE	VSOP
		TRACK	./ORbit.File
93apr13	20:30:00	ACQUIRE	100
		PEAKUP	
93apr13	20:35:13	TRANSMITTER	0.857
		CALTONE	ON 1,3456 2,888 3,333
93apr13	20:35:15	MCBWRITE	ADDRESS=10001 DATA=42
		MCBWRITE	ADDRESS=10002 DATA=-5
93apr13	20:40:00	TAPE	SPEED=135 HEADPOSN=42 WENABLE=1111 DRIVE=1
93apr13	23:30:16	TAPE	SPEED=0
93apr13	23:30:16	STOW	

Table 1: Sample of a simple command file.

MACRO	RA-MODE
RFMODE	RA
TWTMODE	RA
DATAMODE	RA0
AUTOTEST	
END	MACRO

Table 2: Example of a macro definition within a command file.

Outputs

The fundamental outputs of DISPATCH are to the station hardware. This includes writing to the Monitor Control Bus and directing the tape drives to record data. DISPATCH records the intended results of command file instructions in the STATION_STATE structure.

The processing of each command is an event recorded in the station log. DISPATCH also logs command file errors via the LOGWRITER task. DISPATCH makes no direct output to any file.

Command Groups

The commands can be put into 4 groups *1)* Antenna Commands (move, stow), *2)* Satellite/electronics (frequencies etc), *3)* Tape (position, change), *4)* Meta-commands (include, define,.macros). Normal antenna tracking commands are executed by the POINT task. The satellite/electronics functions are executed by DISPATCH using OVLBI specific functions. The tape functions are executed by DISPATCH, but are done mostly with VLBA software modules. To make the Earth Station interface easier to use, three facilities are added to the DISPATCH's command syntax: DEFINE, MACRO and INCLUDE. DEFINEs allow definition of strings in command parameters, for example a satellite type, a variable called STYPE could be defined as RA or VSOP and used in MACRO definitions. MACROs are sequences of commands which are executed as a single unit. INCLUDEs are instructions to start execution of another command file before execution of the rest of the current command file. Control returns to the current command file when the end-of-file is reached in the INCLUDE file.

Below the commands executed by DISPATCH are listed.

RESET: Reads the station parameters file and sets all station hardware to a default state. This is also done automatically at boot-up.

RFMODE: <RA|VSOP|test> Sets switches in receiver and transmitter to Radio-Astron or VSOP or test positions; sets all tunable oscillators to appropriate frequencies.

TWTMODE: <RA|VSOP|test> Sets two-way timing subsystem to appropriate mode.

DATAMODE: <RA0|RA1|RA2|VSOP|test> Sets up the demodulator and decoder. The three Radio-Astron modes cover its three possible data rates: 144, 72, and 36 Mb/s, respectively.

SATELLITE: <RA|VSOP|other?> Combines RFMODE, TWTMODE, DATAMODE, and possibly other commands to set up entire station for a particular satellite.

AUTOTEST: Performs station test sequence. Includes various subsystem tests, each of which also has its own command. Results go to log file. Failures generate operator alarms.

SPACECRAFTMODE:<code> Informs checker of switch settings on spacecraft, for comparison against mode codes in down-link header.

PEAKUP: Causes antenna pointing to be scanned around the nominal position in order to peak up on the satellite signal. Computes pointing error and applies appropriate pointing correction until next PEAKUP or RESET.

ACQUIRE: <timeout> Begin attempt to automatically acquire down-link signals from satellite. Includes execution of PEAKUP. Once adequate signal is being received, executes tracking pass initialization sequence (mainly clock setting). If not accomplished within timeout seconds, send alarm to operator.

TRANSMITTER: <power, W> Turn on transmitter at specified power level. Frequency should have been set up earlier. Zero or negative turns transmitter off.

CALTONE: <ON|OFF> [<channel,frequency>, <channel,frequency>...] CALTONE ON signifies begin detection of test tones in specified channels at specified frequencies. CALTONE OFF signifies end of detection of test tones; write results to log.

STARTUP: Turn on power to antenna servo and cause computer to take control. Send position commands equal to the present position. Ensure that brakes are set.

MCBWRITE: ADDRESS=<address#> DATA=<data#> Allows direct addressing of the monitor control bus.

ALARM: <level> <message> Allows informing the antenna operator of significant events.

MESSAGE: <level> <message> Inserts a text string as an event record in the log file.

TRACK: [FILE=<orbit file path>| RADEC=<right ascension,declination>| KEPLER=<value1,value2,value3,value4,value5,value6>| AZEL=<azimuth,elevation>] Drive antenna according to the tracking mode. Normal operation uses the **orbit file**. DISPATCH also spawns the two way timing task to set the up/down-link synthesizers accordingly. Antenna must have previously been given a STARTUP command, otherwise it won't move. Antenna brakes must be released before spawning POINT.

STANDBY: Stop antenna at present position and set brakes. Tracking computations continue if TRACK command has been executed.

STOW: Drive antenna to stow position and set brakes.

SHUTDOWN: Includes STOW, TRANSMITTER 0, and turning off power to antenna servo. May include other things, like closing files and copying them to other computers.

TAPE: SPEED=<ips> HEADPOSN=<microns> WENABLE=<abcd> [DRIVE=<drive>] Sets recording head-stack to given position, enables specified head groups for writing, and starts tape motion at given speed. If SPEED <0, goes in reverse direction. If SPEED is 0, the tape motion is stopped. If **DRIVE** is not specified, the drive from the previous TAPE command is assumed.

TAPE: MOVE=<feet> [DRIVE=<drive>] Positions tape at specified distance from beginning of supply reel.

FORMATTER: <VSOP|RA> MODE=<VLBA|MKIII> Setup Formatter.

INCLUDE: <filename> include and executes a command file specified by <filename>. (Note the space between command and file name, not "=" as is in the "Purple Book")

MACRO: <MACRO-NAME> defines all the following commands as a macro with name <MACRO-NAME>. The macro is terminated with the END MACRO command. (Note the space between command and macro name, not "=" as is in the "Purple Book")

END MACRO: Terminates the macro defined by the previous MACRO command.

DEFINE: <new variable>=<value> Allows definition of strings to be substituted into the command parameter lists. The DEFINEd variable list is only used when determining values for command parameters.

STARTTIME: [CURRENT | <date> <time>] Indicates that all commands in the file should be skipped until reaching commands starting after a specified time. By default the no commands are skipped. If **CURRENT** is specified, then all commands are skipped until reaching a time after the current time. Then DISPATCH waits until that command and begins normal execution. This function is used to implement selection of one command file out of several scheduled for future execution. Functions implementing this command are also used to handle certain restart functions.

The file specified by the INCLUDE command can also contain INCLUDE commands, but this recursion has a depth limit of 10. Upon starting execution of each command in the file, the times of commands are interpreted in the manner described earlier. When returning from the INCLUDE file, the interpretation of the time remains as it was before executing the INCLUDEd file.

When encountering a MACRO command, DISPATCH stores the following command file lines in dynamically allocated memory until reaching an END MACRO command. For simplicity of implementation, the user is not allowed to re-define a macro. (Ie. once the command MACRO FOO is encountered, an error will be raised if MACRO FOO is encountered again.)

Initialization

The INITIALIZE function spawns DISPATCH with a parameter indicating one of four hardware initialization modes:

A. Cold start after normal shutdown

B. Cold start after computer failure (e.g., loss of CPU power)

C. Warm start (CPU has been running, but station was shut down for some emergency, e.g. external power loss)

D. Hot start, some observing parameter (such as the command file name) needs to be changed, causing temporary suspension of function. It is assumed that Hot start requires as rapid restart as possible, with relatively little change to the system.

For case A, DISPATCH automatically executes a RESET, which sets the hardware into its default state, loads data from the station parameters file, and initializes STATION_STATE. It then looks for a certain file (whose full path can be hard-coded) called TASKLIST that contains a list of command files and the time range for which each is valid. If this file is found, it is read and used to determine the command file appropriate to the current time; the command file is then opened and commands are executed from its beginning. If TASKLIST is not found, then DISPATCH looks for a default command file (hard-coded path), and uses it if found; if not, then DISPATCH exits, leaving the hardware in the default state.

During normal execution, information about the state of the system will be recorded to disk at regular intervals. This state log should include a copy of STATION_STATE, which will include the name of the command file then in use, among other things. This "checkpoint" file will have a predetermined location, so for case B, DISPATCH can find it without any help. DISPATCH will then call a special recovery procedure which attempts to put the station into the state that it should have now. The exact algorithm for this recovery procedure is still to be determined.

Startup Case C assumes some outside anomaly was gracefully handled and that DISPATCH is required to only re-initialize a few hardware systems. The anomaly data structure includes a severity parameter for each possible error; the severity can be NONE, WARNING, ERROR, or EMERGENCY. Each anomaly entry has a parameter for each error. One of the things that can cause an EMERGENCY condition is loss of external power, which should be reported to the computer by the UPS. Another possible EMERGENCY error is very high wind speed.

When restarting, DISPATCH calls a recovery routine (in general different from the case B recovery routine, but possibly the same). Once again, the exact algorithm is TBD, and it could possibly be the same as a normal cold start (case A). But it is probably possible for it to just undo the hardware commands executed by SHUTDOWN and then continue in the command file from where it left off. If DISPATCH was tracking a satellite, it will also have to execute ACQUIRE.

Case D occurs when a new command file is needed during a tracking pass. In this case, only limited initialization is needed and the command file should be advanced to the current time before starting command execution.

Appendix A. Date and Time input formats

The Date format is `YYMMMDD`, where `YY` is the calendar year and `YY` < 50 implies `20YY`, `MMM` is a three character month (ie `JAN`) and `DD` is day of the month. The Time format is `HH:MM:SS` and consists of `HH` integer hours (range 0-23), `MM` integer minutes (range 0-59) and `SS` seconds (range 0-59). The seconds field is optional, defaulting to 0 seconds. There must be no spaces between components of the Date or Time. (ie. 93aPr13 is a legal date, but 93 Apr 13 is not.) [2]

[2] DISPATCH uses the VLBA **str2mjd** and **str2rad** routines for converting date and time to internal values. These routines somewhat "user friendly" concerning input formats. Many other formats are also accepted. The date could be the mjd as a 5-digit number, and the time could be in radians, since these are also accepted by str2mjd and str2rad.

File monchkdesign.doc, version 1.1, released 93/05/09 at 11:08:52

MONITOR AND CHECK TASK DESIGN DOCUMENT

D. Varney
7 May 1993

1.0 OVERVIEW

The MONCHK task is responsible for acquiring all monitor point data in the station according to a pre-set schedule, checking for anomalous conditions and updating the anomaly data structure with the results of such checks. It is also responsible for writing log file records of routine measurements and of anomalies. If an emergency condition is detected (e.g., high winds or power loss), it must place the station in a safe state and notify the operator.

2.0 OMISSIONS

Log record formats have not yet been specified. The procedures for monitoring the GPS receiver and UPS have not yet been established.

3.0 DEPENDANCIES

The design of the computing section of this task is independent of most task designs. However, the station state structure, STATION_STATE, is accessed by MONCHK, and its design will effect how MONCHK interacts with the state structure. The data structures MONITOR data, MONCHK_LIST, and ANOMALY data, which will be explained in this document, could affect the data consumer tasks INITIALIZE, STATUS, and LOG.

4.0 DESIGN CONSIDERATIONS

4.1 The MONCHK task must be activated periodically to acquire data for a set of monitor points. By using the fast semaphore from TIC, it should be possible to run MONCHK at 16 Hz; this determines the fastest available sampling rate. Most monitor points will be sampled musch less frequently, and each will be assigned an interval in the range of 1/16sec to 1hr.

4.2 The majority of monitoring requests will be via the Monitor and Control Bus (MCB). NRAO Specification A55001N001, VLB Array Memo 682 and the VLBA Technical Report No. 5 explain the details and use of this interface. Many functions to support MCB communication and collection of monitor information via the MCB have been successfully ported from the VLBA station control code; some details are given in Appendix A.

4.3 The Decoder module is connected to the station computer across the VME backplane. A draft communication protocol is given by R. Escoffier in the report "Interface Protocol Between Decoder and Station Computer" dated 30 April 1993.

4.4 A Global Positioning System (GPS) receiver and backup power supply (UPS) are planned but not yet acquired. We assume that they will each require monitoring via RS232 serial links, but the details have not yet been established.

5.0 ALGORITHM DESIGN

5.1 Operation of the MONCHK task is controlled by an array of structures called the MONCHK_LIST. Each entry in this array describes the monitoring, checking and logging of an elementary monitor point in the hardware. The entry includes a pointer to a function that actually acquires the necessary data from the hardware. Some functions are quite simple and merely return the status value of a primitive monitor point while others are more complex and make several primitive measurements that are combined to derive a quantity of interest. Each array entry (below) also includes a code giving the frequency with which the monitor function is to be called and the frequency with which the result is to be logged, along with several other parameters:

```
struct mp_functions                 Earth Station monitor point function list
{
        CFUNCPTR func               pointer to the function (all return ptrs)
        int      unit,              unit ID for subsystems with more than one
                                    copy of a device (e.g., 2 recorders),
                                    otherwise -1
                 MD_index           the starting index into the MONITOR data array
                                    for this monitor function...there may be
                                    several data quantities returned.
                 AD_index           the starting index into the ANOMALIES array
                                    for this monitor function...there may be
                                    several possible anomalies.
        BOOL     sampled            was this sampled last interval ?
                                            0 no,
                                            1 yes
        int      monitor,           monitoring interval.
                 log                logging interval.
        float    low,
                 high               lower and upper check limits; could be
                                    used for bit checks by type casting.
}
```

A sample of the contents of the first element of mp_functions[] is detailed below:

```
static struct mp_functions  mon_points[number_of_monitor_pts] =
{
         function  unit                        sampled?          log          high
             |        |  MD_index  AD_index      |     monitor    |     low    |
             |        |     |          |         |       |        |      |     |
        {recrdenb,  24,     0,         0,      FALSE, ONESEC, NOLOG, 0.1, 1.2},
        {...},
        ...
}
```

5.2 MONCHK has a main loop which is executed once each time that the fast semaphore is given by the timing task TIC (nominally 16 times per second); this determines the maximum sampling rate for any monitor point. For each such execution, all entries in MONCHK_LIST are examined. For those whose sampling interval has elapsed, the corresponding monitor function is called. Each function returns a single numerical value which is then stored in the MONITOR data array, detailed below

```
struct mon_list                     instantiated:  monitorlist[number_MD_pts];
                                    the size of this array may be larger than
                                    the number_of_monitor_pts.
{
        char     vartype            signifies the type of the returned variable
                                            'f' for float,
                                            'd' for double,
```

```
                                        'x' for hex long
        union varsize                 provides a method to hold different data
                                      types; machine and compiler independent.
        {
                long   lval
                float  fval
                double dval
        }
}
```

5.3 Most functions also check the validity of the returned value and return an error message pointer if there is any abnormality. In that case, MONCHK's main loop updates the appropriate element of the ANOMALIES array [see 5.6, below] and sends an ANOMALY record to the LOGWRITER task. For those entries whose logging interval has also elapsed (the logging interval must be a multiple of the sampling interval), a MONITOR record containing the returned value is passed to the LOGWRITER task.

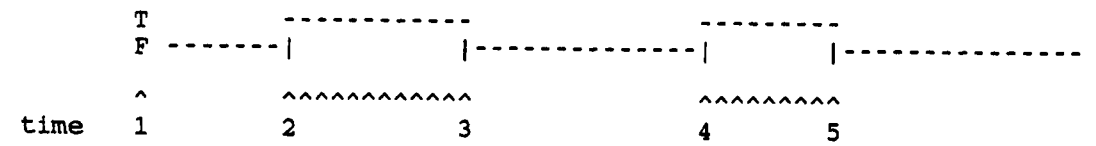
5.4 Many monitor points in the hardware (including all those measured by the reused VLBA routines) are accessed through the VLBA Monitor and Control Bus (MCB). We are also reusing the VLBA routines that support MCB communication, and this imposes further restrictions on the structure of MONCHK's code, including the need to call each monitor function twice. (Details are explained in Appendix A.)

5.5 Most of the MONCHK_LIST array is initialized at compile time, but the sampling and logging intervals for each entry are read in from a file when the MONCHK task is first spawned.

5.6 The applied test will flag suspicious values in the ANOMALIES array (below), where each element will have a time of the occurrence, a status indicator to denote the severity of the error, the returned value and some messages regarding the error type for use by other tasks.

```
struct anomaly                        instantiated:
                                           anomalies[number_of_unique_errors]
{
        int     status;               Status for this error type
        char    *errcode,             Short pneumonic describing error
                *message;             One-line error message for use in screens
                                      and status file
        int     val;                  returned value for mcb transactions
        time    firsttime,            Time of first occurrence:
                                         time CHECK first discovered the error
                lasttime;             Time of most recent occurrence: this is the
                                         last time that CHECK discovered the error
}
```

5.6.1 The time of an error will be recorded according to the following set of rules.



In the diagram above, let F[alse] indicate that no error is being detected and T[rue] that there is currently an error. Then at the indicated times (1..5),

1 initialization; no errors have occurred and the time variables

'firsttime' and 'lasttime' are set to NEVER and undefined, respectively;

2 an error for this monitor point has just been noted; 'firsttime' and 'lasttime' are set to the current time;

3 the error has been removed; 'lasttime' retains the time of correction. From t2 to t3, 'lasttime' will be updated to contain the time of the sample until t3;

4 the error has reoccurred, see [2];

5 the error has been corrected, see [3].

5.7 The current state of the station, contained in the STATE data structure, will be read by the chk[...] functions to determine what the state should be with respect to the monitor data. As major equipment will have some entry in the station state, it is required that the appropriate checking function compare the state with the returned data for compliance and flag appropriately in the ANOMALIES data array.

5.8 The sequence of actions for the MONCHK task to operate is outlined below.

```
-------------------------------------------------------------------------

task MONCHK ::

set task priority                          -- high priority task
allocate structures and initialize         -- init variables for MCB comm.
read monitor and logging schedules         -- get monitor pt. skeds

LOOP (indefinitely)                        -- task loop, spin indef.
  wait indefinitely on semaphore:fastSem-- wait for next 1/N sec sem
    CALL check(sixteenHzTic+1)             -- pass interval
                                           -- increment tic count
ENDLOOP


function CHECK ::

LOOP (pass:0..1)
  if pass 1                                -- if second pass,
    if message_count > 0                   --   if messages to send
       send message string to MVME331      --     transfer string to '331
    otherwise done                         -- no messages, return to caller

  LOOP (0..number_of_monitor_points)       -- for each sampled point
  set unit,message_count,mcb_message_ptr-- init some variables
    if function's interval                 --   set flag; this mp sampled
      CALL FP(^function,index)             --   pass a function pointer and
                                           --   index into function list
      if pass > 0
        CALL chk[...]                      --   for this group of points,
      if error                             --   check for error conditions
                                           --   an error occurred
        enter into ANOMALY_DATA            --     write results to anomaly

      message_count + nMCBmessages         --   add nMCB this func to count
    otherwise                              --   no sampling
      function not sampled                 --     clear flag in func list
  ENDLOOP
ENDLOOP
-------------------------------------------------------------------------
```

Appendix A: CODE REUSE

This design hides the details of the hardware in the low level monitoring functions, while providing a uniform infrastructure for their execution. Changes to individual hardware modules can be accommodated by changing only the corresponding access function. In particular, it allows reuse of code written for the VLBA for the monitoring of the Formatter and Recorder, which are particularly complex [ref 4.1.3]. The design does place certain restrictions on the monitoring functions: they must all use the same parameter list (i.e., [unit,] returned value), and they must all have the same function-return type. However, there is no restriction on how a monitor function behaves internally; for example, it can make additional entries in the MONITOR and ANOMALIES arrays and pass additional records to the LOGWRITER before returning.

The useage of the GET functions requires that the calling function actually reference the GET function twice. A value in the MONCHK task variable is set to either zero or one. Before each call the MCB message pointer must be set to the location the function is to use for its MCB message string. On exit this pointer is incremented by the number of locations used, leaving it ready for the next function call. This organization allows creation of long strings containing many MCB messages which can be processed more efficiently by MCBIO. The task variable for MONCHK points to the following structure for access to STATE and related MCB information,

```
struct mcbmsg                   used by MCB "get" and "check" functions
{
        long tic                monitor time; tics since start of monitoring
        int pass                GET function parameter (extern),
                                        0-build message,
                                        1-process monitor data

        char *pfirstmsg,        pointer to first byte in MCB message string
             *pmsg              pointer to current byte in MCB message string

        struct observ *pobs;    pointer to STATION_STATE structure, included
                                only for VLBA compatibility.
}
```

On the first function call (pass = 0), the function places the necessary monitor point addresses in the message. Before the second call (pass = 1), the caller should use MCBIO to send the message on the MCB. This reads the requested data into the message. Inside each GET function, a check of the MCB status byte returned with each monitor point is made. Function-specific code then extracts the data and converts it to appropriate units. In some cases this code also checks the value for correct range and/or compares it against the value given in the STATE structure.

File logwriterdesign.doc, version 1.1, released 93/05/09 at 11:08:51

LOG WRITER TASK DESIGN DOCUMENT

L. D'Addario
6 May 1993

1.0 OVERVIEW

The Log Writer task handles the creation of records in the log file of the real time control system. A separate task is necessary in order to allow several other tasks to generate log records. Log Writer merely accepts log record strings from each of the other tasks, attaches a date/time stamp, and writes the resulting string to a previously-opened disk file. The records are handled on a first-in, first-out basis.

At present, three types of log records are envisioned (although the design does not preclude additional types being defined later). These are

1.1 Monitor Records, containing processed monitor data that is logged according to a pre-determined schedule, generated by the MONCHK task;

1.2 Anomaly Records, containing data on the first occurrence or discontinuance of a detected abnormality, generated by the MONCHK task; and

1.3 Event Records, containing information on a commanded change in the station state, generated by the DISPATCH task.

2.0 DEPENDANCES

The syntax and content of each type of record is determined by the design of the originating task. LogWriter should be able to be implemented independently. Starting an stopping of LogWriter is controlled by the Initialize, Shutdown, and Restart functions.

3.0 ALGORITHM DESIGN

LogWriter is spawned at system startup by the Initialize function, and thereafter runs continuously. If an emergency level error occurs, Log Writer might be suspended by the Shutdown function in order to avoid filling the log with large numbers of unnecessary Anomaly Records; it would then be resumed by the Restart function when the error condition is cleared.

When first spawned, LogWriter opens the log file. The path and file name can be passed by Initialize, but normally it will use a default directory and will construct a default file name from the current date. If the file exists already, it is positioned for appending to its end; otherwise a new file is created. It then creates a message queue for receiving log records from other tasks and drops into a continuous loop that reads messages from this queue. For each message read, the date (times.mjd from the global times structure), time (times.utc) and the contents of the message are written to the log file. When the message queue is empty, LogWriter is automatically suspended by the operating system until a new message is placed in the queue by another task.

If LogWriter is suspended by another task (such as MonChk via

Shutdown), then the message queue may become full. Additional messages are then lost.

A provision is also needed to terminate LogWriter completely when performing a complete shutdown, such as at the end of a tracking pass or during a more serious emergency (like imminent loss of all power). This is implemented using vxWorks signals. If SIGUSR1 is raised, then control is passed to a signal handler within LogWriter; it will close the current log file and delete the message queue before finally deleting the LogWriter task.

File geomdesign.doc, version 1.2, released 93/05/12 at 19:18:28

GEOMETRY TASK DESIGN DOCUMENT

L. D'Addario
93/03/19, revised 93/05/07

OVERVIEW

This document describes the GEOMETRY task in the real time control system. Its purpose is to compute, periodically, the data needed by the antenna pointing task (POINT) and the two-way timing control task (TWT) and to pass this data to those tasks. To do this, it needs access to the orbit data file. See the top level data flow diagram, Figure 1.* The GEOMETRY task may be regarded as a server to the client tasks POINT and TWT.

OMISSIONS

The present version of this design does not specify the content or format of the orbit file, nor does it specify the calculations needed to transform data in the orbit file to that needed by the client tasks. It is possible that the orbit file will contain exactly the required data, so no calculations are needed in GEOMETRY; in that case, the transformations from the external orbit file (NAIF) will have been done by an offline program. It is also possible that the orbit file accessed by GEOMETRY will be the unmodified external file; in that case, all calculations are done in real time and GEOMETRY will need an extensive set of routines to accomplish this, including some ported from NAIF. It is likely that the final design will split the calculations between the offline and real time systems.

This document specifies the logical structure, data paths, and timing for GEOMETRY, leaving the above issues to be decided later.

DEPENDANCIES

This design has strong implications for the designs of the POINT and TWT tasks.

DESIGN CONSIDERATIONS

The GEOMETRY task must run periodically so as to provide data to its clients at fine enough time resolution to allow them to interpolate to sufficient accuracy. At present it appears that an interval of T=5sec will be adequate, so that value is assumed here. But nothing in the design (aside from CPU loading) prevents the interval from being changed.

The output required are:

to TWT:	T1	= predicted uplink delay
	T1dot	= derivative of T1
	T2	= predicted downlink delay
	T2dot	= derivative of T2
to POINT:	az	= antenna's true azimuth angle
	el	= antenna true elevation angle

Since the client tasks will be interpolating these values between updates from GEOMETRY, results must be supplied somewhat ahead of the current time.

*Figure 1 is the same as Figure 2 in "SOFTWARE OVERVIEW"

ALGORITHM DESIGN

1. Data will be passed to the client tasks via shared memory structures, as follows.

```
struct geom_data
  {
  double updelay;       /* uplink delay  */
  double updelayDot;    /* uplink delay derivative */
  double downdelay;     /* downlink delay */
  double downdelayDot;  /* downlink delay derivative */
  float  azimuth;       /* true azimuth */
  float  elevation;     /* true elevation */
  };

struct geom {
  struct geom_data *pgeom_start;
  struct geom_data *pgeom_end;
  int geom_data_interval; /*interval between updates */
  int geom_start_time;    /*time of data at start of interval */
  /* pgeom_start  is a pointer to data for the last update
        preceding the current time,
     pgeom_end    is a pointer to data for the next update
        following the current time,
     geom_data_interval and geom_start time are measured in
        system clock ticks.
  */
  } geometry;
```

It is the responsibility of the GEOMETRY task to ensure that the two pointers always point to data that brackets the current time. The client tasks will rely on this and will interpolate between the data in the two structures.

Internally, GEOMETRY maintains three data structures as a circular buffer, and exports pointers to two of them to the clients while it is working on filling the third with new data. At the end of the interval, the pointers are updated so that they always point to structures that bracket the present time.

During the very brief time that GEOMETRY is updating the pointers, the client tasks must not access them; yet the client tasks must not be delayed, because they actually update the hardware and must do so on time. Note that the client tasks also run periodically; the POINT task, at least, runs much more often than GEOMETRY (every 1/16 sec in the present design). To provide proper synchronization, we use two semaphores:

```
geomSem = semBcreate(...)
/* Created by the CLOCK task and passed to the GEOMETRY task.
   Given by  tic  routine inside CLOCK every update interval
   (5 sec), and taken by GEOMETRY. */
geomSyncSem = semCcreate(...)
/* Created by GEOMETRY (during its initialization) with an
   initial count of 2 and passed
   to GEOMETRY, POINT, and TWT tasks.  POINT and TWT each
   take/give this semaphore when executing code that accesses
   geom_data.  GEOMETRY takes it *twice* before updating the
   array, and then gives it twice. */
```

This method of mutual exclusion ensures that if tic releases GEOMETRY (via geomSem) and also POINT and TWT (via other semaphores) all on the same clock tic, then POINT and TWT will both complete their accesses to the structures before GEOMETRY updates them. It is then necessary

for GEOMETRY to complete its update before the next activation of POINT and TWT, else the clients will be late. Priority tuning might be needed to ensure this. Just in case, POINT and TWT should include checks that they are on time.

Having completed the pointer update at the beginning of its activation, GEOMETRY then continues by working on calculating the new data. It has until its next activation (5 sec) to complete this. It involves reading the orbit file for data associated with the time two intervals from now (current time + 10 sec, rounded down to a multiple of 5 sec), and possibly doing extensive calculations. (If the geometrical calculations are all done in the real time system, then multiple accesses to the orbit file are likely to be needed to get the satellite position at the uplink and downlink times.) Since the orbit file is accessed sequentially, a large buffer should be allocated (and possibly a double buffer) to ensure that physical access to the disk does not delay the completion of GEOMETRY.

File pointdesign.doc, version 1.1, released 93/05/09 at 11:08:54

POINTING TASK DESIGN DOCUMENT

L. D'Addario
22 March 1993
(rev. 29 April 1993)

OVERVIEW

The POINT task is the one that directly drives the antenna during automatic operation. (The only other way to drive the antenna from the station computer is manually via the operator screen "ACU," which must disable POINT before allowing manual control.) It is activated by the DISPATCHER task in response to a command in the command file. In normal satellite tracking operation, it determines where to point the antenna from data passed to it by the GEOMETRY task; but to support testing, two special modes are also provided. In one mode, the antenna is caused to track fixed celestial coordinates (ra and dec) given in the command; and in the other it tracks a satellite whose Keplerian orbit parameters are given in the command. POINT performs a linear interpolation between updates of the position by GEOMETRY. POINT also computes the pointing corrections according to the coefficients in the pointing_parameters structure and the meteorological data in the monitor structure, sending the corrected coordinates to the antenna. POINT must run periodically and at precisely known times in order to update the antenna control unit with sufficient accuracy.

DEPENDENCIES

This design affects the design of the GEOMETRY, DISPATCH, MONCHK, and INITIALIZE tasks.

DESIGN CONSIDERATIONS

POINT is the most time-critical task in the control system because the antenna is the only time-critical hardware that does not have some buffering of its commands and hardware-determined timing. (For example, the Formatter and the Two Way Timing hardware each receive a precise 1 Hz timing reference, so control signals from the station computer can be executed at precise times if they arrive any time within the right 1-sec window, but the antenna has no such capability.) The antenna control unit (ACU) accepts only position commands, and it includes a Type II servo loop to keep the antenna at the commanded position. The servo has a bandwidth of about 1 Hz, so position updates at a rate of 5 to 10 Hz should be adequate. The fastest drive rate of the antenna is 40 deg/min in each axis, and sometimes the satellite motion is this fast (or faster, but then we cannot track it). The beamwidth at 15 GHz is about 0.1 deg, so at the fastest drive rate the position will move through one beamwidth in 0.15 sec. Therefore, to stay pointed within 1/10 beamwidth (a reasonable criterion), we must send each update to the antenna within .015 sec of the correct time. These minimum specifications -- update at > 5 Hz with a timing error of < 15 msec -- should be substantially exceeded if possible.

We cannot allow two copies of POINT to be running at once, else the antenna would get confused. Thus we must guard against the possibility that the command file inadvertantly contains two commands to activate tracking without an intervening command to terminate tracking.

We cannot allow manual control of the antenna by an operator while point is running, for the same reason. Thus, we need an interlock mechanism that disables POINT while the antenna is in manual control and allows it to resume when the operator releases the antenna from manual control.

DESIGN SPECIFICATIONS

1. Activation and mutual exclusion. POINT will be spawned as a task in response to a TRACK command to the DISPATCHER. The track command will specify a "mode" argument, which will be passed to POINT to specify one of the three available sources of pointing data: GEOMETRY task output, fixed celestial coordinates (right ascension and declination), or fixed Keplerian orbit parameters. In the latter two cases, the values must be included in the command and passed to POINT when it is spawned. In the first case, DISPATCHER must also spawn the GEOMETRY task.

POINT begins by attempting to take a mutual-exclusion semaphore that is designed to prevent multiple copies from running. If the semaphore is not available, POINT does not block but returns immediately with an error code. The semaphore should have been created in INITIALIZE, and should have the option SEM_DELETE_SAFE to prevent task deletion while it owns the semaphore. If it is necessary to change the mode of POINT or to terminate its operation entirely, the task must be deleted and (possibly) re-spawned. To support this, POINT will have a signal handler that, when triggered by a calling task, gives back the mutual exclusion semaphore and then taskDelete's itself. (It is a design decision within DISPATCH, not covered here, whether to include an UNTRACK command that does this, or always to begin a TRACK command by killing any POINT that might be running.)

2. Timing. After initialization, POINT falls into a FOREVER loop that begins by taking a binary semaphore that is given 16 times per second by tic. This is the time-critical point, and priorities must be set so that POINT actually runs within a few milliseconds of the system clock tick that resulted in the semaphore being given. POINT then proceeds to send the position commands to the antenna that were pre-computed during its previous activation. It does this by calls to the appropriate routines in setacu.c, which in turn queue MCB commands through mcbio(). It then begins computation of the pointing commands that will be used next time, branching to the appropriate one of three cases, depending on its mode.

3. RA-DEC mode. This is the simplest mode. POINT obtains the current local sidereal time from the global times structure, subtracts it from the ra to get hour angle, and then transforms to true az-el using the station latitude from the station_parameters structure. Finally, it drops into the pointing corrections routine common to all modes.

4. Keplerian orbit parameters mode. Here POINT will get the current mjd and UTC from times and pass them to a routine (stolen from somewhere TBD) that returns true az and el. It then drops into the pointing corrections routine.

5. GEOMETRY structure mode. Here POINT interpolates linearly between the positions in the two structures pointed to by the pointers from GEOMETRY. It reads the system clock to see how much time has elapsed since the beginning of the update interval. The interpolated values are the true azimuth and elevation, which are passed to the pointing corrections routine.

6. Pointing corrections. POINT evaluates the pointing

correction formulas for az and el using coefficients in the pointing_parameters structure and (for the refraction term in the elevation correction) data from the weather station in the monitor data structure. It is important to know if the weather data is valid; if it is not, then POINT will use an a priori model for the meteorological parameters. The validity will be checked by reference to the appropriate element of the anomolies[] array. Finally, the computed corrections are added to the true az and el, and the results are stored for sending to the ACU at the beginning of the next loop.

7. Manual control. A global flag word:

```
int antenna_manual_mode;
```

will be set to TRUE by any task other than POINT that wishes to take control of the antenna. Only the "ACU" screen is expected to need this capability. POINT will check this flag immediately before sending the corrected az and el to the ACU, and will actually send the data only if the flag is FALSE. Except for this, POINT continues to run normally. It is the responsibility of the check routine in the MONCHK task to compare the flag against data in the station_state structure; if DISPATCHER has put the antenna into a tracking mode but someone else has taken manual control, then this is an error that MONCHK must record in the log file and in the anomolies[] array.

File twtdesign.doc, version 1.2, released 93/05/12 at 19:21:05

TWO WAY TIMING CONTROL TASK DESIGN DOCUMENT

L. D'Addario
11 May 1993

1.0 OVERVIEW

The Two-Way Timing System includes the uplink reference transmitter, which requires continuous and precise tuning of its fine synthesizer in order to effect accurate Doppler compensation; and the downlink reference phase detector, which also requires continuous and precise tuning of its reference generator as well as the acquisition of filtered samples of the measured residual phase. The hardware is under the control of a local digital signal processor (DSP); this device handles the high-speed control of the two synthesizers (several thousand updates per second), the rapid sampling of raw phase measurements (about 5000 samples/sec), and the processing and filtering of those samples to a lower rate (10/sec). The design of the DSP code will be covered in separate reports. The station computer task described here must provide updates of the parameters used for synthesizer control about every 5 sec, and it must collect all of the filtered phase samples and write them to a disk file.

2.0 DEPENDENCIES

This task is dependent on the geometrical data supplied by the GEOMETRY task, which in turn is based on data in the Orbit File.

3.0 OMISSIONS

Some details of the number formats and units in which synthesizer control data is to be supplied are not yet determined. The format of the timing residuals file is not yet specified.

4.0 ALGORITHM DESIGN

4.1 Synthesizer Control

From the predicted orbit, the GEOMETRY task will periodically provide values of the predicted delays on the uplink and downlink paths, as well as the derivatives of these delays. The TWT_CONTROL task will convert these to phases and phase rates at the uplink and downlink frequencies, then add the nominal uplink and downlink synthesizer frequencies to the phase rates, giving the instantaneous phase and phase rate (frequency) for each synthesizer at each update time. For the interval between two such updates, the TWT hardware's DSP will evaluate a cubic polynomial in the phase. The TWT_CONTROL task computes the four coefficients of this polynomial from the two phases and two rates at the ends of the interval, and sends the resulting coefficients to the DSP prior to the start of the interval.

Tests have shown that an update interval of about 5 sec will provide accuracy less than 1 psec, and this is considered adquate. To allow precise timing of the updates in the hardware, the DSP will receive a 1 Hz interrupt that is accurately tied to UTC. Therefore, each update must apply to an integral UTC second mark. The DSP_CONTROL task will signal this by sending a command to the DSP during the 1 sec prior to the 1 Hz interrupt that marks the next

update. Thus, absolute timing for execution of TWT_CONTROL must be accurate to well under one second. If necessary, the update interval could be shortened from 5 sec to 1 sec.

4.2 Residual Phase Recording

There will be 10 filtered phase residual measurements per second, but these can be buffered in the DSP. TWT_CONTROL will run once per second, collecting the 10 measurements of the preceding 1-sec interval. The DSP code will be arranged so that the first measurement in the buffer corresponds to a sampling time precisely on the UTC second mark, and that succeeding samples are at 0.1 sec intervals. TWT_CONTROL will obtain the full date and time from the global "times" structure (maintained by the TIC task) and construct a record of the form

```
struct phaserecord {
  long date   = times.mjd;
  double time = times.utc - 2.0;
  long phases[10];
}
```

These records are then written to the disk file in binary form, using the buffered I/O facilities of the operating system to avoid actual disk accesses each second. Note that the residual phase numbers are 32-bit fixed point; they are likely to be in the form s23.8, although this is not finally decided.

4.3 Initialization

When TWT_CONTROL is first spawned, it sets up the two way timing system hardware according to data in the Station State structure. This includes the transmitter nominal frequency, transmitter power level, phase detector input switch (X or Ku band), and second LO synthesizer frequency (conversion from IF to baseband). It then forces a reset of the DSP, which causes it to load its firmware from ROM and do its own internal initializations. When this is complete, it drops into a continuous loop that is executed once each time that the slow (1 Hz) semaphore is given by the TIC task.

4.4 Procedure Details

```
Spawn TWT_TASK:
  set transmitter band switch and coarse tuning bits
  set transmitter power level
  set phase detector input switch
  set 2nd LO synthesizer
  reset DSP
  FOREVER {
    wait for 1 Hz semaphore
    if synthesizer update is due {
        get data from geometry structure
        convert to synthesizer units
        send to DSP
    }
    read 10 measured phases from DSP
    assemble phase residuals record and write to file
  }
```

4.5 Other Considerations

The initialization routines will be available as separate functions that are callable outside the TWT_TASK context. This will manual control for test purposes, as well as the construction of automatic test procedures run in the context of the DISPATCH task. It will also allow DISPATCH to adjust parameters, such as transmitter

power, in response to commands that are executed after the start of a tracking pass.

The TWT_TASK should be spawned only from DISPATCH as part of its implementation of the ACQUIRE command. If the task is already running, it should be killed and re-spawned.

All communication between the Station Computer and the TWT system hardware is via the Monitor and Control Bus.

File statusdesign.doc, version 1.1, released 93/05/09 at 11:08:55

STATUS TASK DESIGN DOCUMENT

D. Varney and L. D'Addario
5 May 1993

1.0 OVERVIEW

STATUS is a program that creates a human-readable ASCII file summarizing the present state of the station. It does so by examining data stored by the MONCHK task in the ANOMALIES and MONITOR internal data structures and by the DISPATCH task in the STATION_STATE internal data structure. The summary file is first written to a disk drive that is mounted on the real time control system; for security reasons, this disk is not accessible to outside users. The file is then copied via the Green Bank LAN to a separate file system on a publically accessible computer, from which it is made available to all interested persons over the Internet.

STATUS will be executed periodically, about every 5 to 10 minutes, as a low priority task of the real time system.

2.0 OMISSIONS

The exact contents of the status summary file are not specified here. Indeed, they may be changed from time to time during operation according to experience and the desires of users. An example of a typical status file is given in Appendix A.

3.0 DEPENDANCES

This design is dependent on the designs of the internal data structures ANOMALIES, MONITOR_DATA, and STATION_STATE. As this is strictly a consumer task, it is dependent on the producer tasks MONCHK and DISPATCH.

Since the resulting file will be made publically available, it is the subject of an interface specification, NRAO A34300N007.

4.0 DESIGN CONSIDERATIONS

4.1. The STATUS_FILE must be formatted for reading by humans. Thus, whenever appropriate, the data displayed should be in physically meaningful units and properly annotated.

4.2. The STATUS_FILE should be displayable on any 80-column text terminal or printer. It should not be very long (a few screens), so no attempt will be made to provide a complete set of monitor information; it is a summary only.

4.3. To prevent the reporting of inconsistent data, some mutual exculsion mechanism may be needed to ensure that STATUS does not access the data structures while they are in the process of being updated by MONCHK or DISPATCH.

4.4. The local copy of the STATUS_FILE is intended as a backup in case of failure of the LAN. In the latter case, a mechanism is needed to report the error without disrupting real time system operation.

4.5. In addition to running periodically, STATUS should be callable

from DISPATCH in response to a WRITESTATUS command. This allows taking a snapshot of the station at specific special times, such as just before shutdown.

5.0 ALGORITHM DESIGN

task STATUS:

```
set task priority to low                -- low priority task; will run only if
                                        more critical tasks complete
set interval count                      -- init status writing interval
LOOP (indefinitely)                     -- task semaphore wait loop
  wait indefinately on semaphore:slowSem-- wait for the 1Hz sem release
  increment status interval count
  if interval to write status           -- if wait period over (5..10m)
    open status                         -- open local file for status summary,
                                        overwriting previous version
    write status                        -- write local status file
    close status                        -- close file
    copy status to public:status        -- copy to publically accessible
                                        computer
ENDLOOP
```

Appendix A: SAMPLE STATUS FILE

```
GREEN BANK OVLBI EARTH STATION STATUS       updated 27Feb1993 11:12UT

--------------GENERAL-----------------------------------------------
Station mode is TRACKING           Satellite ID is 1 (Radioastron)
Acquisiton occurred at                  27Feb1993 01:07:19
Number of dropouts since acquisition    0
Received signal power X band            -107.3 dBW
Received signal power Ku band           -113.1 dBW
--------------ALARMS------------------------------------------------
--none--
--------------ANTENNA-----------------------------------------------
Position:       Azimuth 135d07m32s      Elevation 058d32m17s
Tracking error:           0d00m09s                  0d00m11s
--------------TIMING TRANSFER---------------------------------------
Total residual phase since initialization     81935.127 cycles
Residual phase rate                              78.2 Hz
Satellite 1Hz - ES 1Hz                      0.29782312 sec
Transmitter power setting                        0.5 W
--------------WIDEBAND DATA-----------------------------------------
Data rate mode                                   128 Mb/s
Parity error rate per bit                    2.2E-05
Missed sync rate per frame                   3.1E-03
Total re-syncs    3               Net bit slips     -5
Total bad frames due to parity rate   31  sync loss   3
Recording:  drive number        02       tape speed  90 ips
            tracks enabled      32    head position  82 microns
            direction      FORWARD
            time remaining on this tape pass         08 min
            total time remaining on this tape        257 min
--------------DOWNLINK HEADERS--------------------------------------
Last frame number                                131
Total power by baseband channel  1 2045   2 2132  3 2001  4 2543
(in spacecraft units)    5 0071   6 0083  7 0091  8 0069
Receiver mode code      07      (expected from schedule:  07)
--------------ANOMOLIES---------------------------------------------
0023  WARNING  2cm dewar temperature is high    20.1 K
0091  WARNING  500 MHz reference level is low   0.97 mW
```

File decoder-protocol.doc, version 1.1, released 93/05/09 at 11:08:48

INTERFACE PROTOCOL BETWEEN DECODER AND STATION COMPUTER

R. Escoffier
30 April 1993

I. HARDWARE DESCRIPTION

The Decoder has a local 87C51 microprocessor to support the Decoder satellite frame recovery functions. The Decoder is housed in a VME crate and the Decoder microprocessor interfaces with a VME station computer via the VME bus.

The 87C51-VME computer interface is accomplished by providing a common 16 Kilobyte memory. This common memory is mapped into both parties memory space and the interface supports both byte and word addressing by the VME station computer. Each party can read from or write to the memory at will and no contention for access to the memory will occur.

Only the lower 16-bits of the VME address space are decoded in the Decoder-VME interface logic and hence the Decoder is intended for use in the short addressing VME mode. Any 16K byte block of addresses within the 64K byte range of the short address mode can be allocated to the Decoder but present assumptions are that the Decoder VME common memory will be mapped into VME addresses $0000 thru $3FFF.

Two blocks of 256 VME addresses within this address range are allocated in the Decoder interface logic for special hardware functions. Write operations by the station computer to memory locations within the address space $3E00 to $3EFF will result in the generation of an interrupt to the Decoder 87C51 (if the 87C51 interrupt is enabled). This feature will allow a one way interrupt driven communication mode between the station computer and the Decoder (the Decoder 87C51 cannot interrupt the station computer with the present logic). Priority of the VME interrupt is selectable within the 87C51 to one of three levels.

The second special VME address block is the address space 3F00 to $3FFF. A write by the VME station computer to any word or byte within this address range will cause the Decoder 87C51 microprocessor to receive a hardware reset.

Communication between the VME station computer and the Decoder 87C51 microprocessor is done according to the detailed protocol description below. In general, *commands* to the Decoder consist of a single-byte code written to address $3E00. This will generate an interrupt to the 87C51 microprocessor, provided that its interrupts are enabled; otherwise, the 87C51 must poll address $3E00. The microprocessor then acknowledges receipt of the command by writing -1 to $3E02, indicating that it is busy. When the command has been completed, it writes zero to $3E02 and a return code to $3E04; the latter will be zero for success, and non-zero in case of some error. (Note that *reading* the special addresses $3E00--$3FFF will not cause interrupts or resets.) The station computer should check that $3E02 is zero (not busy) before issuing a new command; the Decoder might be able to accept new commands while busy, but this is not guaranteed.

Many commands also require the passing of numerical data to the Decoder. In these cases, the appropriate data must have been placed in the command buffer portion of the common memory before the command code is written to $3E00. The command buffer consists of

addresses $2600--$3DFF (5,632 bytes). The format of this data depends on the individual command, and is detailed below. In a few cases, the command results in some returned data; that data will also be put into the command buffer, overwriting anything that was placed there by the station computer.

The Decoder also generates monitor data for the station computer. This data is automatically placed in the monitor buffer portion of the common memory, which is $0000--$25FF (9,984 bytes). The detailed format is given below. Much of this information is updated each downlink frame (minimum 2.5 msec), but some is updated at slower rates. To prevent the station computer from reading an inconsistent set of data (by attempting to read a block while the Decoder is in the process of updating it), address $0000--$0001 is reserved as a semaphore: the station computer must check that this word contains zero, and if so write -1 to it, before reading the monitor buffer; and the Decoder must do the same before updating any information in the buffer. If the Decoder cannot wait, it will simply skip the current update, in which case some monitor information may not be reported.

II. COMMAND SYNTAX

Command Name	Code $3D00	Parameters $2600-$3DFF (up to 5632 bytes)	Frequency
NOP	00		TEST
WRITE MEMORY	01	AA AA CC CC DD -- DD	INIT/TEST
READ MEMORY	02	AA AA CC CC	INIT/TEST
CHECKSUM REQUEST	03	AA AA CC CC (2 BYTES RET)	TEST
PERFORM MEMORY TEST	04	0X	TEST
LOAD XILINX PERSONALITY	10	XX DD -- DD (5544 BYTES)	INIT
FILL FRAME SEQUENCER	11	0X DD -- DD (4096 BYTES/XFER)	INIT
CHECKSUM REQUEST	12	(2 BYTES RETURNED)	TEST
FILL TEST FRAME MEM	13	0X DD -- DD (4096 BYTES/XFER)	INIT/TEST
CHECKSUM REQUEST	14	(2 BYTES RETURNED)	TEST
SET DECODER MODE	15	XX	INIT
SET TEST GENERATOR RATE	16	XX	TEST
SET MISC BITS	17	XX	TEST
RESET COUNTERS	18	XX	RUN
SET UTC CLOCK	19	0T TT TT TT TT TT	

Detailed Descriptions

1. NOP, MAIN MEMORY

```
NOP:                    00
WRITE MEMORY:           01 AA AA CC CC DD -- DD
READ MEMORY:            02 AA AA CC CC
CALCULATE CHECKSUM:     03 AA AA CC CC [2 BYTES RETURNED]
```

where
AA AA is a 16-bit start address and CC CC is a number of bytes with msbyte first.

The checksum returned is a 16-bit sum of all bytes in the command range. The checksum need not be separately requested after a write memory command since the checksum will automatically be written to the first two bytes of the command buffer, but the function code is provided for the verification of a multi-transfer memory write.

The results of a READ MEMORY command are written to the common memory starting at $2604, leaving the starting address and length in

$2600--$2603. The maximum length of a MEMORY READ or MEMORY WRITE is 5,326 bytes; it is suggested that long transfers be done in 4096-byte blocks.

2. PERFORM MEMORY TEST: 04 0X

where

0X = 1	microprocessor will test its main RAM memory
0X = 2	microprocessor will test the VME common RAM memory
0X = 3	microprocessor will test the frame buffer RAM
0X = 4	microprocessor will test the sequencer RAM memory
0X = 5	microprocessor will test the test frame RAM memory

In each case the Decoder microprocessor will loop 16 times testing the target RAM. In each loop it will first fill the target with a pseudo random sequence and then verify the contents. The total number of errors found (saturating at a count of 65535) will be left for the station computer at word location $2600. In the case of a frame buffer test two 16-bit error counts are left for the station computer starting at location $2600 for the I buffer and $2602 for the Q buffer.

3. LOAD XILINX PERSONALITY 10 XX DD -- DD (5544 BYTE XFER)

where

XX = 00	causes loading of Radioastron setup from ROM
XX = 01	causes loading of VSOP setup from ROM
XX = 02,...	causes loading of special setup from ROM
XX = FF	causes loading of the next 5544 bytes from command buffer.

4. FILL FRAME SEQUENCER

FILL FRAME SEQUENCER:	11 XX DD----DD (4096 BYTES DATA)
FRAME SEQUENCER CHECKSUM:	12 [2 BYTES RETURNED]

where

XX goes from 00 to 07 for the 8 transactions required to fill the frame sequencer RAM.
DD----DD indicates 4096 bytes of data per transaction. The sequencer RAM is 4-bits wide and each DD byte carries two nibbles. Total RAM locations filled is 8192 X 8 or 65536.
Check sum is made from stored information in the entire RAM.
XX = 10 causes loading of Radioastron sequence from ROM.
XX = 20 causes loading of VSOP sequence from ROM.

5. FILL TEST FRAME GENERATOR RAM

FILL TEST FRAME RAM:	13 0X DD----DD (4096 BYTES DATA)
TEST FRAME RAM CHECKSUM:	14 [2 BYTES RETURNED]

where

0X goes from 00 to 07 for the 8 transactions required to fill the test frame RAM.
DD----DD indicates 4096 bytes of data per transaction. The test frame RAM is 4-bits wide and each DD byte carries two nibbles. Total RAM locations filled is 8192 X 8 or 65536.
Check sum is made from stored information in the entire RAM.

6. SET DECODER MODE: 15 XX

where

```
XX=00   sets decoder for Radioastron at 72 MHz,
XX=01   sets decoder for Radioastron at 36 MHz,
XX=02   sets decoder for Radioastron at 18 MHz,
XX=03   sets decoder for VSOP (64 MHz).
```

This is a high-level command that causes many things to be initialized. The Xilinx personalities are loaded from ROM, the frame sequencer is loaded from ROM, and the sync detection method and error tolerance are set to appropriate defaults. Also, software switches are set to call the appropriate header processing subroutine. It is expected that this will be the command normally used during operation, but low level commands are also provided to allow control of the individual elements.

7. SET TEST GENERATOR RATE 16 XX

where

```
XX = 00  sets 72 MHz rate
     01  sets 36 MHz rate
     02  sets 18 MHz rate
     1x  sets 64 MHz rate.
```

8. SET MISC BITS 17 XX

where XX = abcd edgh

```
a  is input switch setting:  0 = input from receivers
                             1 = input from test generator
b = 0 for sync probation mode (delayed decision on validity)
    1 for immediate validity decision  (0 is normal)
c = 0 check parity in window (normal)
    1 disable parity check during window
d = 0 sync valid only if detected in I and Q
    1 sync valid if detected in I or Q
ef= {00|01|10} sync error tolerance, I channel
gh= {00|01|10} sync error tolerance, Q channel
```

9. RESET COUNTERS 18 XX

XX = abcd efgh

```
a  total frames processed counter
b  number of re-syncs
c  net bit slips
d  invalid I frames
e  invalid Q frames
f  I channel parity errors
g  Q channel parity errors
h  (spare)
```

Causes those counters whose bits are set to 1 to be cleared so that they will be zero at the start of the frame that begins with the next satellite 1 Hz tick.

10. SET UTC CLOCK 19 0T TT TT TT TT TT

The UTC clock is set with the 44-bit binary time code supplied by 0T--TT on the next ground one Hz tick.

III. MONITOR BUFFER

As stated above, the Decoder microprocessor keeps a number of items in the common memory so the VME station computer can have ready access to them. The syntax is given in the following table.

LOC (hex)	FUNCTION
0000	Semaphore byte (see discussion in Section I).
0001	Unused
0002 to 0003	Decoder status word (includes current mode, whether in sync or not, etc. -- details TBS)
0004 to 0007	(32b) Total number of frames processed counter
0008 to 0009	(16b) Number of re-syncs counter
000A to 000B	(16b, signed) Net number of bit slips
000C to 000D	(16b) Number of invalid frames counter
000E to 000F	Unused
0010 to 0013	(32b) Parity errors in I channel (Radioastron only)
0014 to 0017	(32b) Parity errors in Q channel (Radioastron only)
0020	Frame RAM fullness
0021	Unused
0022 to 0027	(6 bytes) UTC clock reading at last satellite 1 Hz tick.
0028 to 0059	(50 bytes, 400 bits) Frame status bit map for preceding satellite 1 sec period. Each bit that is set to 1 indicates a frame that was out of sync and hence was replaced by PRN, or whose parity error count exceeded a pre-defined threshold.
005A to 00FF	Unused (for future assignment)
0100 to 011F	(32 bytes) Raw I channel header, latest frame.
0120 to 013F	(32 bytes) Raw Q channel header, latest frame. [Radioastron uses the first 30 bytes and VSOP uses the first 6 bytes of each of the two raw header blocks.]
0140 to 01FF	Unused (for future assignment)
0200 to 25FF	(9216 bytes) Processed header data. Format TBS, different for each satellite.