National Defence / Défense nationale

REAL-TIME DATA STORING PACKAGE FOR SKYNET TRIALS

by

Capt E.R. Boudriau

DTIC ELECTE DEC 13 1994 S D B

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 94-3

Canada

April 1994
Ottawa

19941205 041

National Defence / Défense nationale

REAL-TIME DATA STORING PACKAGE FOR SKYNET TRIALS

by

Capt E.R. Boudriau
MILSATCOM Group
Radar and Space Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 94-3

PCN
D6470

April 1994
Ottawa

Abstract

A series of week-long experiments on an EHF downlink were conducted by the MILSATCOM groups at CRC and DREO. The software developed for these experiments included a reliable and robust data logging package to record the setup and measurement data required for post experiment analysis. The software provided the Skynet EHF Trials with a data logging package written in "C" that would interface with the existing serial communication software. The package was used as a real time storing device for recording experimental data such as hop power, noise power spectral density and pointing angles. In addition to data logging, the software had to provide several date and time references using the zulu time provided by a GOES satellite clock as the Trials' time reference. The software was loaded in a computer called Data Logger and was responsible for opening and closing all high and low level communications with the processors linked to it.

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Résumé

Les groupes MILSATCOM de CRC et DREO ont effectué une serie d'expériences d'une semaine sur une liaison satellite-terre EHF. Le logiciel développé pour les expériences comprenait un système fiable et robuste d'enrigistrement de données. Le logiciel fournissait aux Essais EHF Skynet un système d'enrigistrement de données écrit en "C" et étant compatible avec le logiciel de communications sérielles existant. Le logiciel d'enrigistrement de données était utilisé comme l'unique méthode d'enrigistrement des différentes données expérimentales des essais tels que la puissance des sauts de fréquence, puissance de la densité spectrale du bruit et les angles de positionnement. Le logiciel fournissait aussi plusieurs références de date et d'heure, utilisant l'heure zulu de l'horloge satellite GOES comme référence pour tous les essais. Le logiciel d'enrigistrement des données fut implenté dans le processeur principal dénommé "Data Logger" et fut ainsi responsable de commencer et terminer les communications avec tous les processeurs qui s'y rattachent.

EXECUTIVE SUMMARY

The MILSATCOM groups at DREO and at CRC have been investigating a typical EHF satellite communications downlink using a Skynet 4 satellite.

The trials consisted of five separate week-long access periods from May to October of 93 during which several tests were conducted such as synchronization, evaluation of ephemeris prediction pointing angles and the characterization of both antenna patterns and long term link stability. The need for a real time, reliable and robust data logging package arose from the quantity of experimental data required to complete the post experiment analysis.

The data logging software was implemented in the main processor, a DELL 486D/50 MHz computer called Data Logger, as the only experimental data recording device. Using the in-house developed serial communications software, it accepted status, configuration and result messages from various other processors. The Data Logger also provided the other experiment processors with a date and time reference taken via a serial interface with a GOES satellite clock.

The Data Logger routines were designed to minimize the amount of data that could be lost should the program end prematurely. Overall real-time performance of the logging software was enhanced by avoiding the normal use of buffering and caching by the operating system. The logging package performed successfully throughout the Trials.

Future work could include a more comprehensive error detection scheme, and sorting capabilities that would improve the post-experiment data analysis.

CONTENTS

DTIC QUALITY INSPECTED 6

LIST OF FIGURES AND TABLES

DOCUMENT CONVENTION

This paper uses the following typographic conventions:

Example	Description
STDIO.H	Uppercase letters indicate filenames.
writelog()	Bold type indicate routine name. Note that calling conventions and parameters associated with a routine name are not included unless absolutely necessary. They are listed in a separate Appendix.
prefix	Words in italics indicate placeholders for information you must supply, such as the prefixes on a message to be logged.
~~	These subscript characters each represent 1 space to be supplied. In this example, 5 spaces are needed.
`main()`	This font is used for coding examples and for all listings.
<LF>	Angle brackets are used to indicate a control character <line feed> here as an example.

1 - INTRODUCTION

The MILSATCOM groups at the Defence Research Establishment Ottawa (DREO) and Communications Research Canada (CRC) conducted a series of five week-long trials on EHF satellite communications using a Skynet 4 satellite. Several tests were conducted such as the evaluation of ephemeris pointing angles, synchronization, Bit Error Testing (BER) testing of commercial Binary Phase Shift Keying (BPSK) and the evaluation of Frequency-hopped/Digital Phase Shift Keying (FH/DPSK) BER. A considerable amount of data was recorded to facilitate data analysis and receiver characterization. In preparation for the experiments, a parallel hardware/software development was required and was facilitated by the use of several separate computers. It was found desirable to control the experiment simulators and processors and to record the required data from a single location. The data collection and storing package used the serial communications software that was developed to control all other processors.

The objective of this paper is to describe the Data Logging Package that was developed for the Skynet 4 Trials. The Data Logger was used as a single source for data recording and also as the experiment date and time reference for several other processors used in the experiment.

The data logging routines were written in "C" for compability with already existing routines and used the serial communications software that was separately developed to accept status, configuration and result messages from the many processors used in the Skynet EHF Trials. The routines were designed to minimize the amount of data that could be lost should the program end prematurely and special considerations were given to avoid the normal use of buffering and caching by the operating system. The use of the GOES satellite clock as experiment date and time reference is also described in this paper.

2 - EXPERIMENT SETUP

2.1 System Overview

The parallel hardware/software development was facilitated by the use of multiple computers. The Data Logger was developed to provide the unique recording device used for data collection using the already existing serial communications software. The software was installed in the main processor called Data Logger and provided other processors with the experiment data and time reference.

A simplified block diagram of the experiment setup is illustrated in Figure 2.1. Serial communications links between the Data Logger, the GOES satellite clock and other processors are shown by dashed lines.

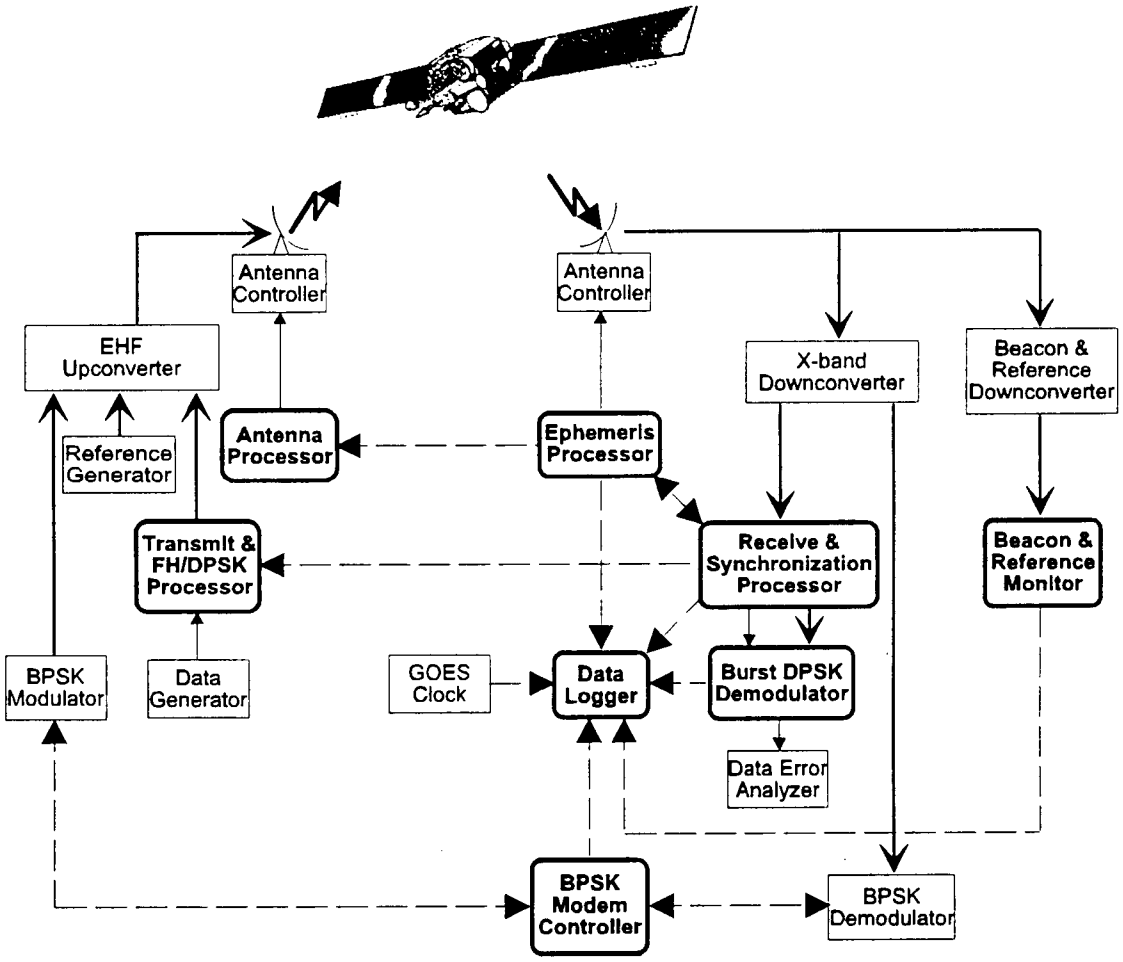

Fig. 2.1 Block Diagram of the Experimental Setup.

Computers and in-house developed equipment are highlighted by bold boxes. The EHF Upconverter, the X-band Downconverter and off-the-self equipment is identified by normal rectangles.

A Data Generator provided pseudo-random data to the Transmit & FH/DPSK Processor which performed differential encoding, hop waveform generation and local or remote control of a variety of continuous wave (CW) and FH/DPSK signals which would be converted by the EHF Upconverter and then fed to a 1.8m parabolic dish.

A BPSK Modem Controller configured both the BPSK Modulator and Demodulator and supplied the Data Logger with BER measurements.

A British Skynet 4 satellite provided the satellite link for the experiment. It filtered, amplified and translated the EHF signal from a nominal uplink frequency of 44.590 GHz to a 7.625 GHz downlink signal which was then transmitted via an earth coverage horn along with an onboard generated beacon signal.

As the downlink X-band signal was received and converted to a desirable IF signal and fed to the different processors, a number of messages were being sent to the Data Logger to be recorded. Envelope and quadrature detection, analog signal processing, signal capture and digital signal processing were provided by the Receive and Synchronization Processor. This processor then sent receiver noise spectral density and antenna scan amplitude measurements to the Data Logger for recording. For some experiments, the dehopped signal was demomulated by the Burst DPSK Demodulator which provided the Data Logger with BER measurements.

The power, frequency and nearby noise spectral densities of the reference, beacon, and beacon-squared signals were continually monitored by the Beacon and Reference Monitor which received the signal from the Beacon and Reference Downconverter. Each measurement was then sent to the Data Logger to be recorded.

The GOES satellite clock provided the experiment date and time reference. This reference was converted by the Data Logger and then fed to the different processors attached to it at the start of each experiments. Further description of each processor and preliminary post-experiment results can be found in [1].

The use of multiple processors for the trials required the addition of a multiple serial communication ports to the PC hosting the software. The DigiCHANNEL PC/8 RS232, an 8 serial port PC extension board from DigiBoard Inc. was installed into the Data Logger and configured as shown in Appendix A.

Figure 2.2 shows in more detail the relation between the Data Logger hosting the software and the stations that send it data to be stored.

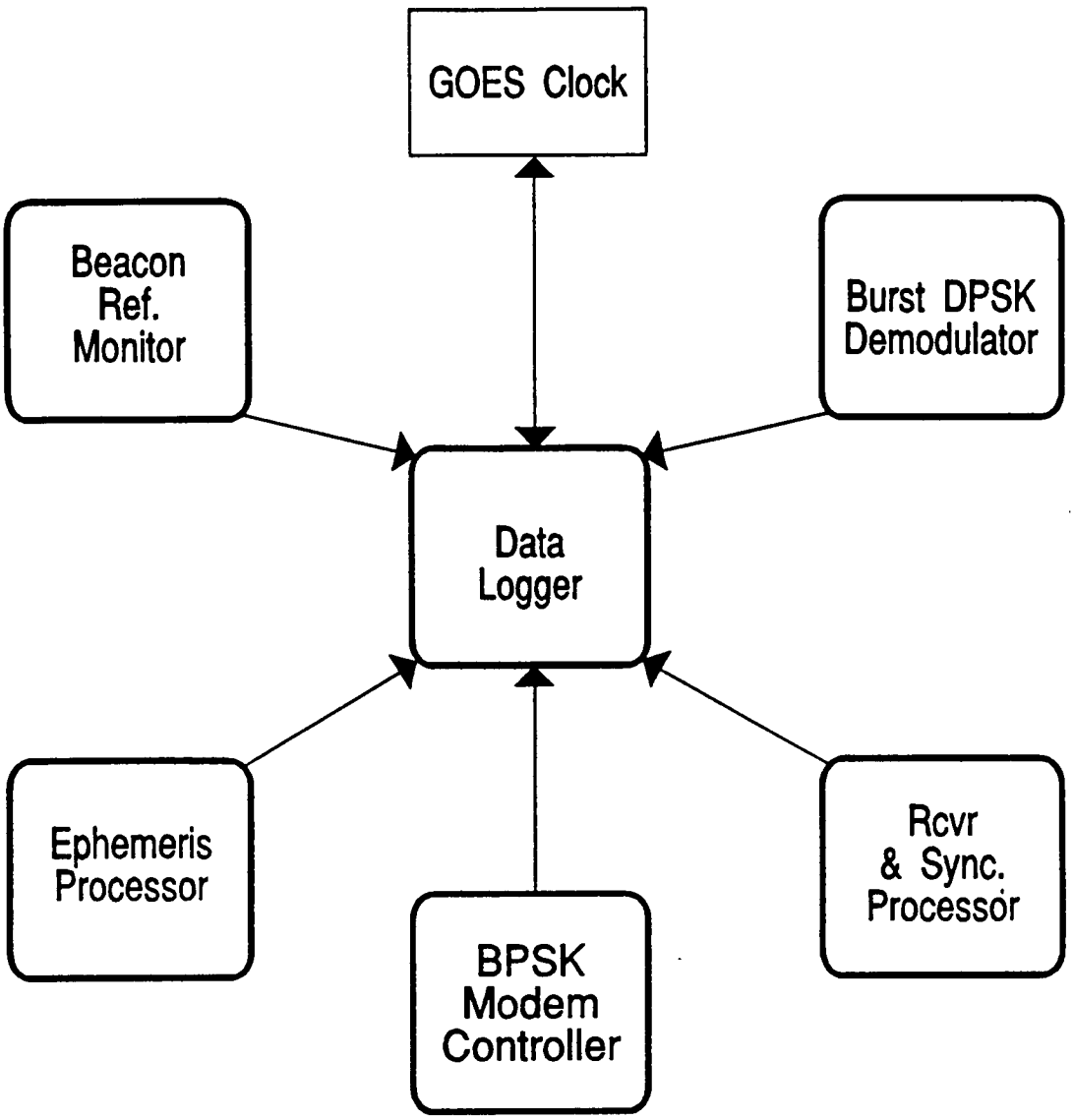

Fig. 2.2 Interconnecting Diagram between Data Logger and Other Processors

The serial communications software described in [2] allows all the processors shown in fig. 2.2 to communicate with the Data Logger via the DigiCHANNEL PC/8 serial board and the two serial ports in the PC hosting the software. The serial communication software described in [2] was responsible for handling serial ports.

Table 2.1 shows the frequencies and types of messages to be recorded by the Data Logger.

PROCESSOR NAME	TYPES OF MESSAGES	SENT EVERY
Beacon Monitor	- Receiver power & frequency of Reference Signal, Beacon Carrier and Beacon squared. - Rcvr, Beacon & Beacon squared noise spectral density	37 sec
Burst Demodulator	- BER error measurements	30 sec to 20 min
Ephemeris Processor	- Ephemeris prediction - Pointing angles with bias corrections - Nominal pointing values	10 sec
Synchronization Processor	- Receiver noise spectral density - Antenna scan amplitude of estimate power of CW	15 min 1 hr
BPSK Modem Cont.	- BER measurements	30 sec to 20 min
GOES Satellite Clk	- Date and Time reference	only on request

Table 2.1 Frequencies and types of messages to be recorded.

The baud rates and other relevant configuration information are shown in table 2.2.

DEVICE NAME	PROCESSOR NAME	BAUD RATE	DATA BITS	STOP BITS	TIME OUT (SEC)	PARITY
COM1	Beacon Monitor	2400	8	1	30	N
COM2	Burst Demodulator	9600	8	1	10	N
COM4	Ephemeris Processor	9600	8	1	10	N
COM5	Synchronization Processor	9600	8	1	10	N
COM6	BPSK Modem Controller	9600	8	1	10	N
COM10	GOES Satellite Clock	9600	8	1	-	N

Table. 2.2 Serial Port Configuration.

3 - SYSTEM DESCRIPTION

The data logging software is used to store ASCII data from a number of users on individual files located on the hard disk. It also supplies several date and time references. For the Skynet Trials, the software package was written in standard C using Microsoft C version 7.0 and was installed on a Dell 486D/50 MHz computer with 8 Mb of RAM and a 320 Mb hard disk. The system operates under DOS version 5.0. Microsoft C version 7.0 was chosen specifically for its capability of by-passing the operating system cache when storing data on the hard disk. This special option is described in more detail in section 5.3.2.

The software package is divided into several routines as shown in Figure 3.1.

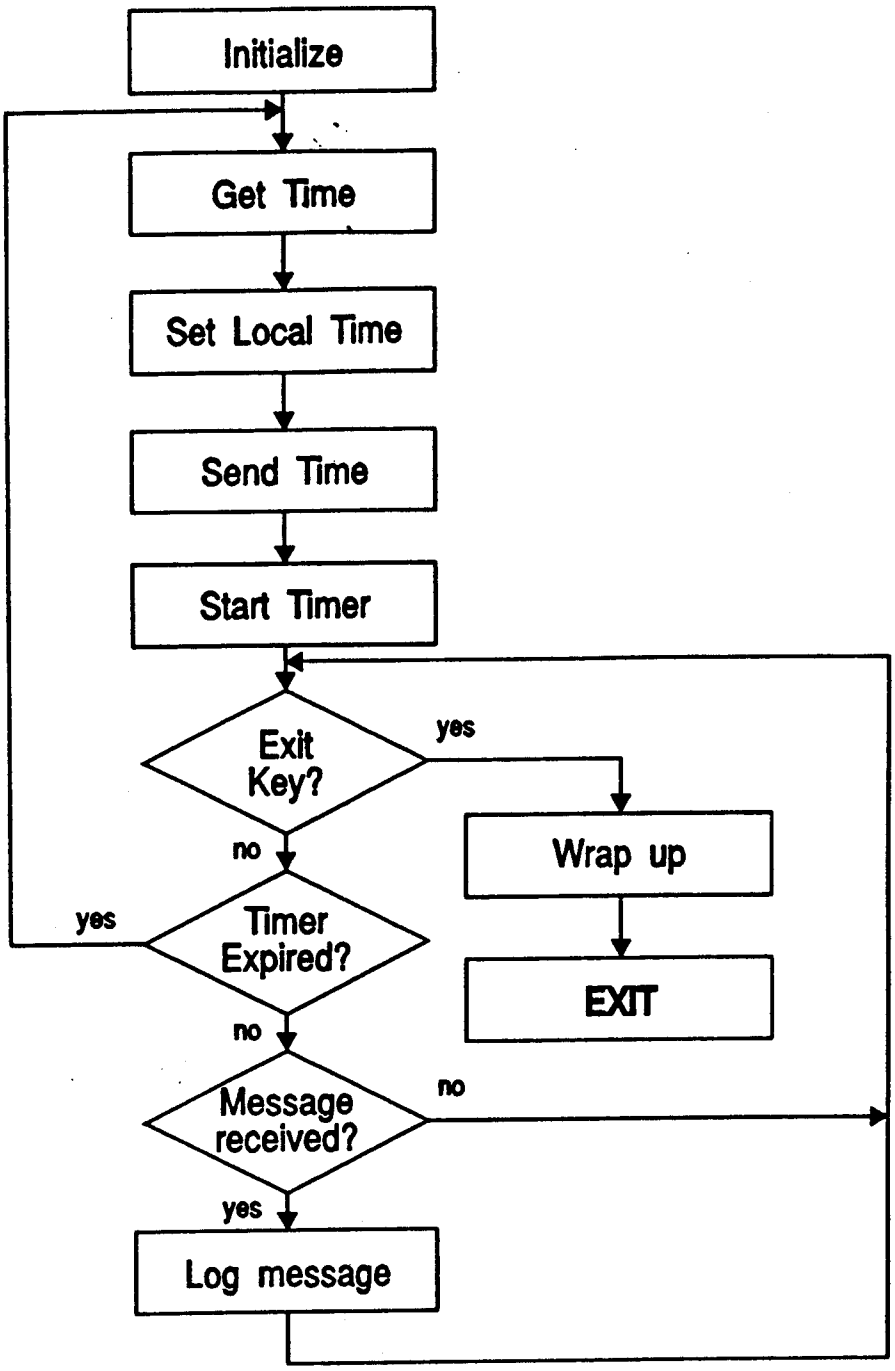

Figure 3.1 Data Logger Software Outline

The data logger routines were implemented in the main processor and were therefore responsible for opening and closing all high and low level serial communications links between the Data Logger and the various processors connected to it. The routines used in the Data Logger are described in the following section.

4 - LOGGER MAIN PROGRAM

4.1 General

The Logger main program was responsible for interfacing the data storing routines described in the section 5 with the existing serial communications software.

4.2 Description

The routines that compose the Logger main program are taken directly from the **com.c** package described in reference [2]. The main program can be divided into three separate small sections:

a) Initialization;

b) Data Logging; and

c) Shut down.

The initialization section is responsible for opening all communications using the **open_com()** function, and for initializing the Data Logger with the GOES clock's date and time that was used as the experiment reference. It also reads the time reference configuration file and starts the periodic timer.

The Data Logging section is responsible for storing the required data as described in section 5. If the periodic timer has expired, it also writes an interim date and time reference in the opened file.

The Shut down section is responsible for closing all opened files using **closelog()** described also in section 5 and finally for closing down the communications using the **close_com()** routine. The program then exits back to DOS.

5 - DATA LOGGER ROUTINES

5.1 General

The following sections describe the routines that are responsible for storing the data received from the other processors. Calling conventions and required parameters are detailed in Appendix C. Definitions can be found in Appendix D.

5.2 openlog()

5.2.1 Description

This routine is called during the initialization period by providing the integer **mlocal** representing the logger station number. **openlog()** is responsible for assigning and opening the proper files. File names are automatically assigned in accordance with the following format: *FILENAME.EXT* where:

FILENAME: User Name as returned by the function **stnstr()**, from [2], (4 characters long). In the Skynet EHF Trials, the station name calling **openlog** is **dlog** (Data Logger).

EXT: 000 - 999 (The actual file number)

5.2.2 Automatic File Name Generation

The large number of experiments which require data logging necessitated a chronological and automatic way of naming each file in order to facilitate the post-experiment data analysis. To keep the process as simple as possible, the same file name is used for every experiment and the **openlog()** routine is responsible for assigning a different extension number to each file.

The routine starte by looking on the hard disk for the first instance, if any, of the file **dlog.*** using the DOS routine **_dos_findfirst()**. If no dlog.* file exists, the integer **max** representing the extension number is set to be 000. If at least one instance of the file dlog.* is found, the process is repeated using a while loop to find the next instance of the file dlog.* until the last one is reached, incrementing the integer **max** every time an instance is found. When the while loop is terminated, the integer **max** represents the new file number to be opened. The maximum number of files is limited at 1000 (000 to 999). However, should an overflow occur, the **openlog()** routine will open another file called DEFAULT.(000 to .999). The function will then return a warning message with an audible alarm.

The subroutine uses the stream function **fopen** using the following access modes:

"a+" : Opens for reading and appending, creating the file first if it doesn't exist;

"b" : Open in binary (untranslated mode); and

"c" : Enable the commit flag. This option will be discussed in greater details in section 5.3.2. Also note that this option is a new feature offered with the C version 7.0.

A structure of type **FILE** defined in the Header File **STDIO.H** is associated with the opened file. All subsequent operations to the opened file are done by referring to this file pointer. A null pointer value indicates an error. The routine starts by looking on the disk for the last file stored by the logger.

5.3 writelog()

5.3.1 Description

This routine is responsible for storing on a disk file the ASCII stream that represented the message to be logged. It also provides several date and time reference options. Interim date and time references are added to the file every 5 minutes. A complete description of the interim time reference and other date and time options can be found in section 6.0.

5.3.2 Robustness

One of the critical factor throughout this work was the robustness of the software or its ability to minimize the amount of data lost if the program should terminate abnormally. The data logging routines had to be designed to minimize the loss of data. The routines were written in C for better compatibility with other existing routines.

Two different types of input and output (I/O) functions were considered:

a) Unformatted functions such as **_read** and **_write**; and

b) Stream formatted functions such as **fscanf** and **fprintf**.

The **writelog()** routine was written using the stream I/O functions because of their formatting capabilities. Their buffering capabilities were found undesirable and had to be overcome. Although buffering can significantly improve I/O performance by transferring a large block of data in a single operation rather than performing many smaller operations each time a data item is read from, or written to a stream, it causes some concerns. In write operations, such as **fprintf**, data is collected in an

intermediate storage location, or buffer. The output buffer's contents are written to the disk only when:

a) The buffer is full or flushed to the operating system;

b) The stream is closed; or

c) The program is terminated normally.

If the routine "hangs up" or causes the program to terminate abnormally, it will result in a loss of data since the output buffers may not be flushed. Using the stream function **setbuf** or **setvbuf** allows the programmer/user to specify the buffer size according to the importance of the experiment data since the buffer size would represent the amount of data that would be lost if a problem occurred. The buffer size for the Skynet EHF Trials was determined to be one string of a maximum length of 220 characters.

It is possible to further reduce the chances of losing data. The stream I/O function **fflush()** causes the output buffer's contents to be flushed to the operating system which can cache the contents before writing to the disk. In the case of a system failure, all the data cached by the operating system would be lost. This problem was eliminated by forcing the buffer's contents to be written directly to the disk. As stated in [3], two options were available:

a) Link the file COMMODE.OBJ to set a **global commit flag** since the default setting is "no-commit"; and

b) Set the **"c" commit flag** with **fopen()**. This option is a new feature available only to MS-C version 7.0 or later.

The second option was chosen. Only the files opened with the option "**c**" will be affected, preventing any error that could occur if another routine is also working with its own files. Since the files are opened for appending, the file pointer is automatically positioned at the end of the file before each write operation, allowing the user to call the routine several times to store data without having to reposition the file pointer.

5.3.3 Data string Format

The string received from any processor for logging must be terminated by a null ('\0') character. The string must also be preceded by a two character long prefix and two spaces. This specification was implemented to simplify the post-experiment analysis. The two character long prefix is the station identification or the

measurement performed. The string format is therefore as follows:

prefix^^message to be logged (max length is 220 char)<NULL>

The following figure shows the prefixes used for the Skynet EHF Trials:

PREFIX	DESCRIPTION
BE	BER measurements from the BPSK Modem Controller
EL	Bias Ephemeris angles (local, T-85 antenna) from Ephemeris Pro.
EP	Predicted Ephemeris angles from the Ephemeris Processor
ER	Bias Ephemeris angles (remote, CRC ant.) from Ephemeris Pro.
FM	Frequency measurements from the Beacon Monitor Processor
FS	Frequency status from the Beacon Monitor Processor
LO	Logging messages from the Data Logger
RX	Receiver messages from the Synchronization Processor

Fig. 5.1 Prefixes used in Skynet EHF Trials.

5.4 closelog()

5.4.1 Description

This routine writes the last date and time reference marking the end of the experiment and closes all files previously opened by the routine **openlog()**.

6 - DATE AND TIME STAMP ROUTINES

6.1 General

The Data Logger was responsible for providing the other processors with a date and time reference to aid in the data analysis. The time reference is provided by a GOES Clock through a serial link between the clock and the Logger. The logger

initializes its own time and then sends it to the other processors. Several options are available and the individual time requirements for each stations/processors can be implemented by simply editing an ASCII file called LOG.CFG which is read during the initialization phase of the Logger main program as stated in section 4.2.

6.2 readlog()

In addition to providing the user with data storage capabilities, the Data Logger program also provides each user with the following date and time reference options:

a) **BEGIN**: Time stamp is sent only at the beginning of the experiment;

b) **PERIOD**: Time stamp is updated every 2 hours. This option was added on after noticing a variation in the system time of some of the processors during the long term trial; or

c) **NONE**: Time stamp is not sent.

These time flags can be selected by editing an ASCII file called LOG.CFG which is read at the start of the program by **readlog()** and cannot be changed once the program is started. Blank and comment lines starting with the char ';' are ignored.

The routines written to provide all the date and time requirements for the experiments are described below. A sample configuration file for the Data Logger can be found in Appendix B.

6.3 gettime()

The Data Logger initializes its own system date and time by interfacing at 9600 baud with a GOES satellite synchronized clock via a serial RS-232 link and the low level communication routines **puts_low()** and **gets_low()**. The low-level communications routines are preferred here since the satellite clock does not require any handshaking.

Initially, the Data Logger sends a string to configure the satellite clock into the "T" mode to stop the satellite clock from sending its string representing the time stamp every seconds and consequently jamming one serial port. As described into reference [4], the satellite clock, when placed in "T" mode, responds only upon request from the Data Logger by sending a string representing the Julian calendar date

followed by the zulu time of the day as shown in Fig. 6.3.1.

<SOH> DDD:HH:MM:SS.SSS <CR> <LF>	
<SOH>	Or *<CTRL-A>*, Start of Header Character
DDD	Julian Calandar date
HH	2 digits representing the hours
MM	2 digits representing the minutes
SS	2 digits representing the seconds
SSS	3 digits representing the milliseconds
<CR>	Control Return Character
<LF>	Line Feed Character

Fig. 6.3.1 Time String Format as received from GOES clock.

This string is then converted into the proper DOS format needed by the functions **mktime**(), **_dos_setdate**() and **_dossettime**(). It is stored into each opened file to mark the start of the experiment and then sent by the Logger to all user stations using the routine **sendtime**().

Note that a "check for exit key" loop was added during the waiting loop in case the GOES clock aborts abnormally or does not respond to a time request. This loop enables the operator to bypass the satellite clock time initialization. The Logger will then supply its own system time to all other processors.

6.4 sendtime()

This routine initially sends a string representing the experiment system time reference to all the processors connected to the Data Logger. Each time the two hour-long periodic timer expires, it then utilizes the time flags read by **readlog**() to select only the serial link of the stations that have requested the periodic time update. The integer "p" used in the routine (as seen in its listing in Appendix E) helps differentiate between the initialization period which corresponds to "begin" flag and the normal stage of operation. All time flags can only be changed prior to the start of the Logger program by editing the ASCII file LOG.CFG.

6.5 Periodic Timer

The functions **time()** and **difftime()** were used in the logger main program to calculate a two hour long period. This option was used for some of the experiment processors to ensure the synchronization between all internal clocks. This simple method enables the Data Logger to re-initialize each processor's system time. This was accomplished by adding the following code in the main loop of the Logger program:

```
if(( elapsed_time = difftime( finish, start )) >= PERIOD {
   p++;     /* integer "p" incremented to differentiate   */
            /* between initialization and operation phase */

   gettime();          /* re-initialize Data Logger system   */
                       /* time using GOES clk time reference */

   sendtime();         /* only sends time to station with    */
                       /* periodic flag enabled              */

   time(&start);       /* reset periodic timer               */
   }
```

For increased flexibility and ease of testing, the flag "PERIOD" is placed into the file DEFINE.H, which can be edited to set the period to the desired amount of time.

6.6 Interim Time Stamp

A time reference is added to each opened file every five minutes to aid in the post-experiment data analysis. The five minute period is set using the routine **set_time()** provided with the **serial.asm** program. As documented in [2], the timer single "tick" occurs every 1/17 second. The timer is set for 5100 "ticks" which is equivalent to 5 minutes. When the function **chk_time()**, also provided with the **serial.asm** program, reads 0, the five minute period has expired and an interim time stamp is added to the currently opened file.

7 - TESTING PROCEDURES

7.1 General

The complete Data Logging package was designed so each section could be tested seperately. Tests were performed using an HP4952A Protocol Analyzer with its HP18179A RS-232C/V.24 Interface and a Y shaped connector. The flexibility of the Protocol Analyzer allowed it to be configured to simulate the GOES clock and to

monitor at the same time the link between the Data Logger and one peripheral station.

7.2 Testing Procedures

The Data Logger station was first tested with the Protocol Analyzer only which was configured using the "simulate" portion of its pull-down menu. Several tests were carried out using the hookup schematic shown in Fig. 7.1.

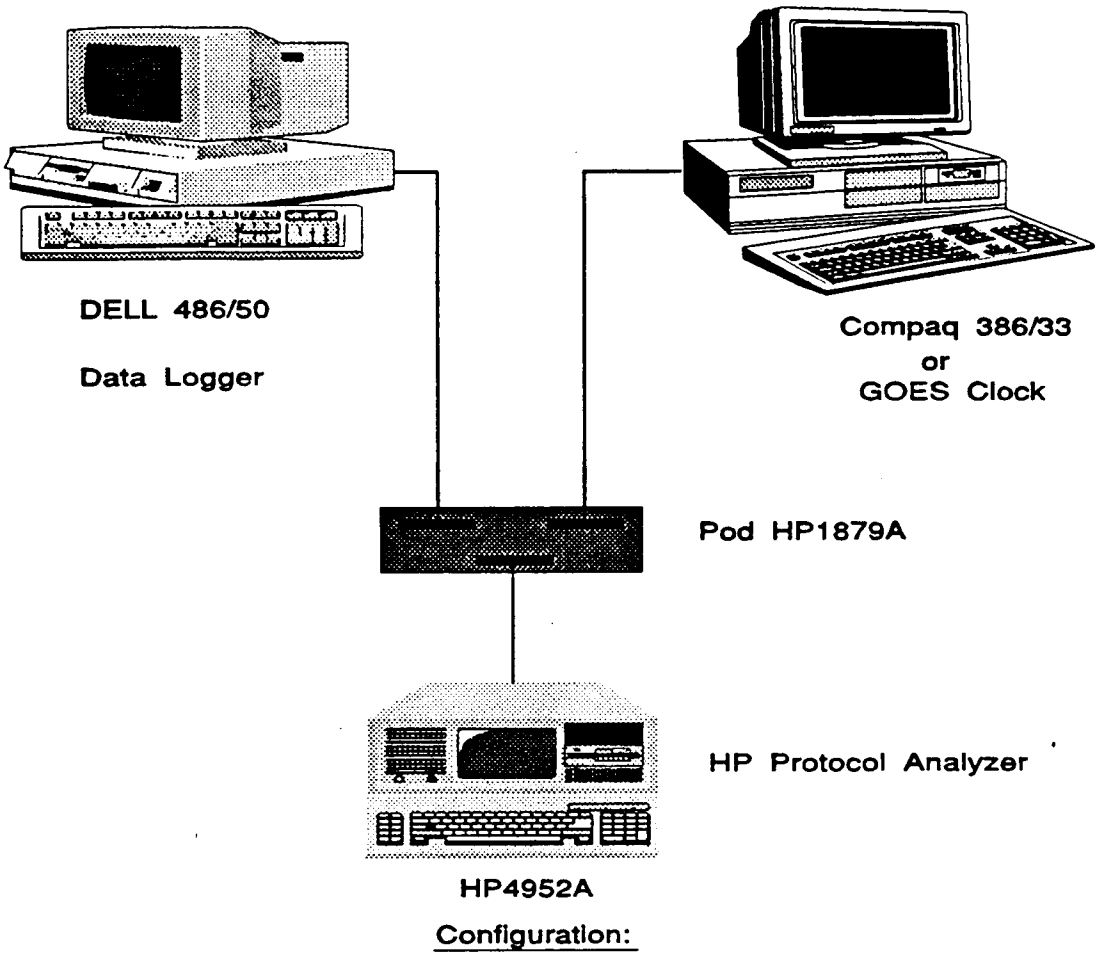

Fig. 7.1 Hookup Schematic for Testing Procedures

Hookup procedures and coding as documented in [5] for the HP4952 is explained in the following sections.

7.2.1 Simulating and Testing the GOES Clock

The first part of the testing was conducted to verify the compability between the GOES clock and the Protocol Analyzer configured to simulate a Data Terminal Equipment (DTE). The HP4952A was simply connected to the Goes clock and the following code was used:

```
Simulate DTE

Block 1:
Set Lead DTR On
    and then
Set Lead RTS On
    and then
Go to Block 2

Block 2:
Send T  <CR>  <LF>    /* Put GOES clock in Time on */
                      /* request only              */
Stop
```

With the HP4952A configured to match the serial requirements of the satellite clock (9600 baud, no parity, 1 stop bit) and using the "two Line Display Mode", the string representing the time reference coming back from the clock was displayed in reverse video on the HP4952A screen. Having recorded the time string and its format, the Protocol Analyzer was then hooked up to the Data Logger and using the same code but by replacing the T command by the actual time string received, the link between the Data Logger and the satellite clock was established and tested. A NULL Modem adapter was connected on the link between the Data Logger and the Protocol Analazer to ensure proper connection.

7.2.2 Simulating and Testing Data Logger Capability

Using the setup in Fig. 7.1, the Data Logger was connected via the HP4952A to a Compaq 386D 33 MHz computer representing the station Burst Demodulator. With the Protocol Analyzer configured to the "monitor" menu, it was verified that indeed a message was being sent from the Burst Demodulator processor to the Data Logger processor. The Burst Demodulator was using a simplified version of its program using only the modified **checkkey()** routine to send various test messages to the Data Logger. Resulting opened files in the Logger were then edited to verify if the message recording was actually performed.

Several tests were conducted which included the interim time reference verification. The modified **checkkey()** routine used can be found at the end of Appendix E.

As seen in Table 2.2, the baud rate between the Beacon Reference Monitor and the Data Logger had to be reduced to 2400 baud. Several communications problems were encountered between the two stations. The Beacon Reference Monitor processor was implemented on a PC AT-286 computer. Its slow processing speed could not handle a data rate hignher than 2400 baud. No other problems were encountered between the Logger and any other processor.

7.2.3 Test on Robustness of Logging Software

As stated in the section 4.2.2, one of the critical factors in designing the Logging package was to minimize the amount of data loss should the program terminate abnormally. Microsoft C version 7.0 allowed the option of being able to dump and store data onto the hard disk by skipping the operating system cache routines. This feature was verified by running the program under Microsoft Windows version 3.1 using the trace function to execute only one line of code at a time. A DOS window was also opened to verify that the data was stored immediately following the command **fflush(fp)** rather than only when the program terminated as would be the case without the version 7 option.

Following the successful completion of this test, several other tests were performed, from shutting the power off to disconnecting the serial link between the two stations. The amount of data lost never exceeded one string of data.

8 - CONCLUSION

The Data Logger software was successfully implemented on a Dell 486/50 MHz processor and proved to be very reliable. The routines were simplified as much as possible to speed up the process but could have easily been modified to include additional error handling capabilities.

Future work on the Logger package should concentrate on recording error messages from the Serial Communications routines as they occurred. The **writelog()** routine could be modified to perform a series of tests to verify the authenticity of the incoming data from each processor.

The following tests could have been performed:

a) Message Existence;

b) Validation of message; and

c) Error condition and issue of warning messages (see section C.2.1 in the Appendix C).

To simplify and speed up the storage process, the above tests were left out but could easily be added to the existing software as part of future experiments.

9 - REFERENCES

[1] Update on DREO/Skynet 4 EHF Trials, R. Addison and W.R. Seed, Defence Research Establishment Ottawa, November 1993.

[2] Real-Time Interprocessor Serial Communications Software for Skynet EHF Trials, Robin Addison, Defence Research Establishment Ottawa, April 1994.

[3] Microsoft C/C++ Run-Time Library Reference, Microsoft Corporation, one Microsoft Way, Redmond WA, 1991.

[4] Satellite Synchronized Clock Model 468-DC, Operating and Service Manual, Kinemetrics, True Time Division, Santa Rosa, CA, 1982.

[5] HP4952A Protocol Analyzer Operating Manual, HP Colorado Telecommunications Division, Colorado Springs, CO, 1989.

APPENDIX A

DIGICHANNEL PC/8 - 8 SERIAL PORTS

BOARD CONFIGURATION: DIP SWITCH AND JUMPER SETTINGS

JUMPER SETTINGS:

There are several jumpers on the PC/8 serial port board. This is the recorded configuration used in the Skynet EHF Trials:

JUMPER NUMBER	SETTING - DESCRIPTION
J1	PIN 1-2 : Odd Interrupt
J2	PIN 1-2 : Odd Interrupt
J3	PIN 1-2 : Odd Interrupt
J4	PIN 1-2 : Odd Interrupt
J5	PIN 1-2 : Odd Interrupt
J6	PIN 1-2 : Odd Interrupt
J7	PIN 1-2 : Odd Interrupt
J8	PIN 1-2 : Odd Interrupt
J9	PIN 2-3 : Board ID # 0
J10	PIN 2-3 : Board ID # 0
J85	ON - IRQ3
J86	OFF - (IRQ5) - Disabled
J87	OFF - (IRQ7) - Disabled
J88	OFF - (IRQ6) - Disabled
J89	OFF - (IRQ4) - Disabled
J90	OFF - (IRQ2) - Disabled

DIP SWITCH SETTINGS:

There are several dip switches on the PC/8 serial port board. The Dip Switch 1 (DS1) is used to set the board's status port address. The following settings were used for the SKYNET IV Trials:

DS1 SWITCH NUMBER	SETTING - DESCRIPTION
1	ON - Switches 1 to
2	OFF - 7 are used to set
3	ON - the board's status
4	OFF - address to 140h
5	ON -
6	ON -
7	ON -
8	ON - Enable board
9	ON -
10	ON - Enable Status Port

The following switches are used to set the I/O port address. Unless specified, all switches are set to the ON position. These are the exceptions:

DIP SWITCH NUMBER	SWITCHES THAT ARE OFF
DS2 (COM1)	2 (100h)
DS3 (COM2)	2,7 (108h)
DS4 (COM3)	2,6 (110h)
DS5 (COM4)	2,6,7 (118h)
DS6 (COM5)	2,5 (120h)
DS7 (COM6)	2,5,7 (128h)
DS8 (COM7)	2,5,6 (130h)
DS9 (COM8)	2,5,6,7 (138h)

APPENDIX B

DATA LOGGER TIME STAMP CONFIGURATION FILE

B.1 Time Stamp Configuration File

This file contains all the time stamp options for each of the processors linked to the Data Logger. It is read once at the start of the program by the routine **readlog()** and cannot be changed while the program is running.

The configuration file is an ASCII text file that can be edited using any text editor. Blank lines and comment lines starting with ';' are ignored. Several comment lines were added at the start of the LOG.CFG file to help the user on how to fill in the file.

B.2 Time stamp Options

The following options are available:

a) **BEGIN** : Time stamp is sent only at the beginning of the experiment;

b) **PERIOD** : Time stamp is updated every 2 hours. This option was added on after noticing a variation in the system time of some of the processors during the long term trial; or

c) **NONE** : Time stamp is not to be sent.

B.3 Sample Configuration File

Page B - 2 shows a sample of the file LOG.CFG as used in the Skynet EHF Trials.

LOGGER SAMPLE CONFIGURATION FILE

Fig. B.1 shows an example of the file LOG.CFG.

```
;Data Logger Configuration File

; Select desired option by entering 1 below the proper flag.

; none flag must always be 0 unless both the begin and period flags are 0,
; then it must be 1.

; Station/Computer name are as defined in the COM.H file.

;Station Name      begin flag     period flag    none flag

BEACON_MON          1              0              0
BURST_DEMOD         0              0              1
EPHEM_PRO           0              1              0
SYNC_PRO            1              1              0
```

Fig. B.1 LOG.CFG Sample Configuration File.

APPENDIX C

CALLING CONVENTIONS FOR DATA LOGGER ROUTINES

C.1 Calling Conventions

C.1.1 int openlog(int origin)

The **openlog(int origin)** routine is called by simply providing the routine with the integer representing the Data Logger station number as defined in the program header COM.H. This integer is provided by the **look_com()** routine provided by the COM.C program. Since the files are only opened during the experiment initialization, only the Data Logger is responsible for opening and closing the files. Users only need to send a LOG message (**type = log**) with the string to be stored to the Data Logger.

C.1.2 int writelog(int type, char *data, int origin, int exist)

The **writelog()** routine can be called by passing the following parameters:

a) int type : integer representing the message type defined in COM.H;

b) char *data : pointer to the data string to be recorded; and

c) int origin : integer representing the station number.

d) int exist : integer to indicate if there is a message to be logged.

C.1.3 int closelog(void)

No parameters required for calling this routine.

C.2 Expected Returns

C.2.1 int openlog()

The following integers defined in the header file COM.H can be returned by the **openlog** subroutine:

a) QUIT : Error and the file could not be opened; and

b) ALL_OK : No error . File was opened successfully.

C.2.2 int writelog()

The following integers defined in the header file COM.H can be returned by the **writelog()** routine:

a) COMM_ERR : Returned if error, invalid message or originator is detected; and

b) ALL_OK : Data was properly recorded. No error occurred or was detected.

C.2.3 int closelog()

The following integer defined in the header file COM.H can be returned by the **closelog()** routine:

a) ALL_OK : No error; and

b) QUIT : Error occurred. Some files may still be opened.

The Logger routines served only as a data recording device for the duration of the EHF Skynet Trials. In order to simplify the routines as much as possible, the expected return values were seldom verified or tested and no immediate actions were taken as a result of one of those return values. Future work could include the reading of the return values to simplify the debugging process should the program end abnormally.

APPENDIX D

HEADER FILES USED IN LOGGER SOFTWARE

DEFINE.H

The header files DEFINE.H and COM.H contain all definitions and decleration used in the Data Logger package. COM.H is the header file of the serial communications software that can be found in [4].

```
/*
Return Definitions
------------------  */

#define ALL_OK          0           /* No problem occurred          */
#define OVERFLOW        5           /* Number of files exceeded     */
#define MAX_SIZE_NAME   30          /* Max lenght for file name     */
#define DEFAULT_NAME    "default"   /* File name used if overflow */
#define DIR             ".\\"       /* storage directory            */
#define PERIOD          2000        /* time updated every 2 HR      */

/*
Include Files
-------------  */

#include "stdio.h"
#include "time.h"
#include "stddef.h"
#include "dos.h"
#include "stdlib.h"
#include "string.h"

/*
String return functions
-----------------------   */

char *stnlstr(int n,char *string);
char *messtr(int n,char *string);
char *stnstr(int n,char *string);


/*
Integer return functions
------------------------  */

int openlog(int origin);
int writelog(int type, char *data, int origin, int exist);
int closelog(void);
```

```
/*
Declaration of External Variables
---------------------------------   */

extern int msg_type, msg_existence, originator;
extern FILE *fp;
extern char name[15], text[];
extern char mtype[15];
```

COM.H

This is the header file of the serial communication software that can be found in [4].

```
#define HEAD_VERSION   "V02Jun93.01"

/*
 Station name definitions
--------------------------- */

#define BAD_STATION    -1   /* Station name or number not valid      */
#define UNKNOWN_ID      0   /* Station name garbled or not sent      */
#define DATA_LOGGER     1   /* Data Logger & Exp. Controller         */
#define BEACON_MON      2   /* Beacon & Reference Monitor            */
#define BURST_DEMOD     3   /* Burst DPSK Demodulator Host           */
#define TX_PROC         4   /* CRC Transmit Processor                */
#define EPHEM_PROC      5   /* Ephemeris Processor                   */
#define SYNC_PROC       6   /* Synchronization Processor             */
#define CRC_ANTENNA     7   /* CRC Antenna Controller Host           */
#define T85_ANTENNA     8   /* T85 Antenna Controller Host           */
#define NSTATION        9   /* Number of valid stations              */
#define LENSTN          4   /* Length of station name field          */
#define LOW_BASE       20   /* Base number used for low-level ports */
#define SNAMES  "unkn"," dlog", "beac", "bdem", "txpr", "ephm",
                "sync","crca","t85a"
#define LNAMES  "unknown", "data_logger", "beacon_mon",
                "burst_demod", "tx_proc",\
                "ephem_proc","sync_proc","crc_antenna","t85_antenna"

/*
 Receiver status definitions
------------------------------ */

#define NO_MESSAGE      0   /* No message ready received             */
#define VALID_MSG       1   /* Valid message received                */
#define COMM_ERR        2   /* Communications error occurred         */
#define QUIT            3   /* Exit program requested                */

/*
 Message type definitions
--------------------------- */

#define BAD_MESSAGE    -1   /* Message is invalid                    */
#define ACK             0   /* Ack message                           */
#define NAK             1   /* Nak message                           */
#define COMMAND         2   /* Command message                       */
#define CONFIGURE       3   /* Configuration message                 */
#define LOG             4   /* Log message                           */
#define STATUS          5   /* Status message                        */
```

```
#define POINT           6   /* Initial pointing information          */
#define MOD_POINT       7   /* Modified pointing info                */
#define TIME_STAMP      8   /* Time stamp                            */
#define ERROR           9   /* Error condition message               */
#define NMESSAGE        10  /* Number of message types               */
#define LENMSG          6   /* Length of message type field          */
#define MNAMES  "ack    ","nak    ","comd  ","config",
                "log    ", "status","point ","modpnt",
                      "time  ","error "

/*
 Error check definitions
------------------------- */

#define NO_ERROR        0   /* No error occured                      */
#define TOTAL           1   /* Too many total errors occurred        */
#define CONSEC          2   /* Too many consecutive errors on 1 port*/
#define BREAK           3   /* Control-Break or Control-C occurred   */


/*
 Low-level "get" return definitions
----------------------------------- */

#define BAD_DEST       -2   /* Destination number is invalid         */
#define NO_DATA        -1   /* No data is available                  */
#define ALL_OK          0   /* Normal return                         */

/*
 High-level communications routines
----------------------------------- */

/* open_com opens all high and low level communications          */
/* get_com gets one message, if available, returns status        */
/* send_com sends one message                                    */
/* look_com determines port number given station name            */
/* ready_com checks to see if port is ready to send message      */
/* config_com overrides default SERIAL.CFG name                  */
/* flush_com resets errors on a channel                          */
/* close_com closes all high and low level communications        */

int open_com(void);
int get_com(int *ctype, int *cfrom, char *cdata);
int send_com(int dest,int mtype,char *string);
int look_com(char *stn);
int ready_com(int dest);
void config_com(char *string);
int flush_com(int dest);
void close_com(void);
```

```
/*
 Low-level communications routines
---------------------------------- */

/*  These routines need high-level "open_com" and "close_com"    */
/*  before use                                                   */

/* getc_low gets one character                                   */
/* gets_low gets one string terminated by the parameter          */
/* putc_low puts one character                                   */
/* puts_low puts one string                                      */
/* look_low determines destination number given station name     */

int getc_low(int dest);
int gets_low(int dest,int term,char *string);
int putc_low(int dest,int c);
int puts_low(int dest,char *string);
int look_low(char *stn);

/*
 String return functions
------------------------- */

/*  In all cases the function points to string containing       */
/*  the answer                                                   */

/* stnstr returns the station string for the given ID number    */
/* stnlstr returns the long station for the given ID number     */
/* messtr return the message type for the given type number     */

char *stnstr(int n,char *string);
char *stnlstr(int n,char *string);
char *messtr(int n,char *string);
```

APPENDIX E

LOGGER ROUTINE LISTINGS

```
/*============================================================*/
/*                      LOGGER VERSION 2.0                    */
/*                            JUNE 93                         */
/*                                                            */
/*------------------------------------------------------------*/

/*
Include Files
-------------  */

#include <bios.h>
#include "com.h"
#include "define.h"
#include <dos.h>
#include <time.h>
#include <conio.h>

/*
 Local Routines
---------------- */

void pabort(char *msg);  /* Print message, close file, and exit   */
void gettime(void);      /* Get Date & Time Stamp from Goes Clock */
void sendtime(void);     /* Send time to stations requesting it   */
int checkkey(int mdest); /* Check and action key presses          */
int checkmsg(void);      /* Check for receive messages and others */
int readlogcfg(void);    /* Read "LOG.CFG" file to set up time    */
                         /* stamp requirements                    */



/*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - - - - - - -  LOGGER VERSION 2.0 - JUNE 93 - - - - - - - - -*/
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

struct tm jday, today;
struct _dostime_t newtime;
struct _dosdate_t newdate;
time_t now, clock_time;
int txdest;
int begin[10];           /* time stamp flag = time at beginning   */
int period[10];          /* time stamp flag = periodic update     */
int none[10];            /* time stamp flag = no time stamp req   */
int p;                   /* integer index variable for time update*/
char user[200];          /* station names in "log.cfg"            */
```

```
void main(void)
{
    int mlocal;             /* Station number of local station       */
    int ndest;              /* Port number for desired destination   */
    char string[220];       /* String buffer to hold station name    */
    int t, clock_setting, clock_return;
    char time_stamp[50], format_ack[50];
    char time_buffer[100];
    char date_buffer[40];
    char control[4];        /* To store the Control char "SOH"       */
    time_t start, finish;
    double elapsed_time, result;
    unsigned loop;

    printf("LOGGER V2.0\n");

      /* Open all communications */

    if ((mlocal=open_com()) == BAD_STATION)
      pabort("Error in com_open");
    printf("Local station is %s\n",stnlstr(mlocal,string));

      /* Read "LOG.CFG" to set up all time stamp requirements */

    if( readlog() != 0 ) pabort( "Error reading log.cfg" );

      /* Send time and date stamp from Goes to stations */

    p = 1;                  /* BEGIN stage only                      */
    gettime();              /* Get time from Goes Clock              */
    sendtime();             /* Send it to stations requesting it     */
    time( &start );         /* Start periodic time update timer      */

      /* Open log */

    if ((t=openlog(mlocal)) != ALL_OK)  {
      printf("Crash in open_log:  value=%d\n",t);
      close_com();
      exit(1);
    }

      /* Check keypress (send msg to 'ndest') and receive msg     */

    while (checkkey(ndest) == 0) {
      time( &finish );
      if(( elapsed_time = difftime( finish, start )) >= PERIOD ) {
          p = p + 1;        /* PERIOD stage only                     */
          gettime();        /* Get time from goes Clock              */
          sendtime();       /* Send it to stations requesting it     */
          time( &start );/* reset periodic time update timer      */
          }
      if (checkmsg() != 0) break;
    }
```

```
      /* Check log */

    if ((t=closelog()) != ALL_OK) {
      printf("Crash in close_log:  value=%d\n",t);
      close_com();
      exit(1);
    }

    close_com();
    exit(0);
}


/*=============================================================*/
/*                          checkkey                           */
/*                                                             */
/* Description: Checks to see if a key has been pressed and    */
/*              performs the necessary action such as sending  */
/*              various messages or exiting                    */
/*        Uses 'bios_keybrd' to see if a key has been pressed. */
/*        Control-C/break is not done here, but reported by    */
/*        checkmsg                                             */
/*                                                             */
/* Returns:    (int)          0 for normal return              */
/*                            1 for exit from main program due to */
/*                            keypress                         */
/* In:         (int ndest)    destination port number for msg  */
/* Out:        -                                               */
/*-------------------------------------------------------------*/

int checkkey(int ndest)
{
    int c;                  /* Character from the keyboard     */
    int d;
    char msg[220];

    if (_bios_keybrd(_KEYBRD_READY) != 0) {/* Was a key pressed ? */
      c = _bios_keybrd(_KEYBRD_READ) & 0x0FF; /* Get the char  */
      switch (c) {
            case 'm':
            case 'M':
              d=0;
              while ((d<1) || (d>4))
           {
             printf("\n Enter TX_PROC CW mode selection\n"
              "  1=2MHz_Low   2=2MHz_Ctr   3=60MHz_Ctr   4=:CW_Off :");
             scanf("%d",&d);
           }
             sprintf(msg,"%d",d);
           (void) send_com(txdest,CONFIGURE,msg);
           printf("\n\n  TX_PROC CONFIGURE mode %d sent\n",d);
           break;
```

```
            case 'e':
            case 'E':
            case 'q':
            case 'Q':
            case 'x':
            case 'X':
            return(1);                          /* e,E,q,Q,x,X = quit */
            default:
            break;                              /* Otherwise ignore */
        }
    }
    return(0);
}


/*=============================================================*/
/*                          checkmsg                           */
/*                                                             */
/* Description: Checks with communications routines for:       */
/*              - receive messages from any link               */
/*              - communications errors                        */
/*              - aborts from control-C/break                  */
/*                                                             */
/* Returns:     (int)     0 for normal return                  */
/*                        1 for exit from main program due to comm */
/*                        errors or control-C/break            */
/* In:          -                                              */
/* Out:         -                                              */
/*-------------------------------------------------------------*/

int checkmsg(void)
{
    int mstat;              /* Message status - valid, error or quit */
    int mtype;              /* Message type number                   */
    int mfrom;              /* Message from station number           */
    char mdata[220];        /* Message data                          */
    char string[220];       /* String buffer used for name, type or  */
                            /* error                                 */

    mstat = get_com(&mtype,&mfrom,mdata);     /* Check for message */
    if (mstat == VALID_MSG) {                 /* Message available */
      writelog(mtype,mdata,mfrom,VALID_MSG); /* Log message       */
      printf("   from %s ",stnstr(mfrom,string));
      printf("(%s):  \"%s\"\n",messtr(mtype,string),mdata);
    } else if (mstat == COMM_ERR) {           /* Comms error       */
      printf("-- Communication error with %s: %s\n",
             stnlstr(mfrom,string),mdata);
    } else if (mstat == QUIT) {               /* End main program  */
      if (mtype == TOTAL) {                   /* Too many errors   */
          printf("Too many communication errors\n");
      } else if (mtype == CONSEC) {           /* Too many in a row */
          printf("Too many consecutive communication errors with
                 %s\n", stnlstr(mfrom,string));
```

```
    } else if (mtype == BREAK) {            /* Control-C/Break   */
        printf("Break detected\n");
    }
    return(1);
  }
  return(0);
}

/*=================================================================*/
/*                            pabort                               */
/*                                                                 */
/* Description: Print error message, close file and abort          */
/*                                                                 */
/* Returns:     -- no return, aborts --                            */
/* In:          (char *msg)    Pointer to error message string     */
/* Out:         -- no return, aborts --                            */
/*-----------------------------------------------------------------*/

void pabort(char *msg)
{
    printf("%s\n",msg);
    close_com();
    exit(0);
}
```

```
/*=============================================================*/
/*                          openlog                            */
/*                                                             */
/* Description:  Automatically name, number and open proper file */
/*                                                             */
/* Returns:     integer corresponding to one of the return     */
/*              definition                                     */
/*                                                             */
/* In:          (int n) station number                         */
/* Out:    (int n) corresponding return message                */
/*                                                             */
/*-------------------------------------------------------------*/

/*
Include Files
-------------  */

#include "com.h"
#include "define.h"

int msg_type, msg_existence, originator;
FILE *fp;
char name[15];
char mtype[15];

int openlog(int origin)
{
  char full_name[MAX_SIZE_NAME], buffer[MAX_SIZE_NAME], **bidon;
  char base_name[MAX_SIZE_NAME];
  char dbuffer[9], tbuffer[9];
  struct _find_t c_file;
  int max;

  set_time(10, 5100);        /* start timer # 10 - 5 minutes       */
  stnstr(origin, name);      /* get user name                      */
  strcpy(full_name, DIR);    /* file path                          */
  strcpy(base_name, name);   /* user name is file name             */
  strcat(full_name, base_name);
  strcat(full_name, ".*");

    /* look if file [user].000 was previously stored by this user */

  if( _dos_findfirst(full_name, _A_NORMAL, &c_file) == 0 )
     max = strtol(strchr(c_file.name, '.') +1, bidon, 10);
                              /* if yes, new file is [user].001   */
  else
     max = -1;

    /* look for last file number opened by this user  */

  while(  _dos_findnext( &c_file) == 0 )
  if( strtol(strchr(c_file.name, '.') +1, bidon, 10) > max )
```

```
    /* new file number is last file number + 1 */

max = strtol(strchr(c_file.name, '.') + 1, bidon, 10);

    /* check for overflow, sound alarm if yes  */

  if( max >= 999 )  {
     printf("\aMaximum number of file reached for
             base name: %s.\n", base_name);

    /* if overflow, file name becomes [default].000  */

     printf("using base name: %s\n", DEFAULT_NAME);
     max = -1;
     strcpy(full_name, DIR);
     strcpy(base_name, DEFAULT_NAME);
     strcat(full_name, base_name);
     strcat(full_name, ".*");
     if( _dos_findfirst(full_name, _A_NORMAL, &c_file) == 0 )
        max = strtol(strchr(c_file.name, '.') +1, bidon, 10);
     else
        max = -1;
     while(  _dos_findnext( &c_file) == 0 )
     if( strtol(strchr(c_file.name, '.') +1, bidon, 10) > max )
        max = strtol(strchr(c_file.name, '.') + 1, bidon, 10);
     strcpy(full_name, DIR);                  /* file path          */
     strcat(full_name, DEFAULT_NAME);         /* filename = default */
     sprintf(buffer, ".%03d", ++max);
     strcat(full_name, buffer);
     printf("%s\n", full_name);

     /* opens default file */

     if(( fp = fopen(full_name,"abc+")) == NULL ) {
        printf( "cannot open log file\n");
        return(QUIT);
        }
     fprintf( fp,"\nLO  Initial Time Stamp is:  %s
              %s\r",_strdate(dbuffer), _strtime(tbuffer));
     return(ALL_OK);
     }
  else {
     strcpy(full_name, DIR);          /* file path                  */
     strcpy(base_name, name);         /* filename = user            */
     strcat(full_name, base_name);
     sprintf(buffer, ".%03d", ++max );
     strcat(full_name, buffer);
     printf("\nfile opened: %s\n", full_name);
```

```
    /* opens proper file */

     if(( fp = fopen(full_name,"abc+")) == NULL ) {
         printf( "cannot open log file\n");
         return(QUIT);
         }
     fprintf( fp,"\nLO  Initial Time Stamp is: %s
              %s\r",_strdate(dbuffer) ,_strtime(tbuffer));

     return(ALL_OK);
     }
}
```

```
/*=============================================================*/
/*                          writelog                           */
/*                                                             */
/* Description:  Store data given the pointer to a string      */
/*                                                             */
/* Returns:      integer corresponding to one of the return    */
/*               definition                                    */
/*                                                             */
/* In:   (int n)          message type number                  */
/*       (char *string)   pointer to data string              */
/*       (int n)          station number                      */
/*       (int n)          message existance                   */
/* Out:  (int n)          corresponding return message        */
/*                                                             */
/*-------------------------------------------------------------*/

/*
Include Files
-------------  */

#include "com.h"
#include "define.h"

int writelog(int type, char *data, int origin, int exist)
{

    char dbuffer[9], tbuffer[9];    /* date & time stamp buffer    */
    int i,j,n;                      /* integer index variables     */

    /* check for timeout           */

    if( (chk_time(10) == 0 )) {
      fprintf( fp,"\nLO  interim time stamp is: %s
               %s\r",_strdate(dbuffer), _strtime(tbuffer));
      set_time(10,5100);            /* reset timer to 5 min        */
      }
    i = exist;
    j = type;
    n = origin;

    /* verify if msg to be logged */

    if( i == VALID_MSG ) {          if( n == TX_PROC ) {
         fprintf( fp, "\n%s\r", data );
         fflush(fp);
         return(ALL_OK);
         }

    /* verify if not error msg    */

    if ( j == ERROR )  {
         fprintf( fp,"\nE1**********************************************
                  ********************\r");
```

```
        fprintf( fp,"\nE2  Error Time Stamp is:   %s %s\r",
                 _strdate(dbuffer), _strtime(tbuffer));
        fprintf( fp,"\nE3  Error was made by station: %s\r",
                  stnlstr(origin,name));
        fprintf( fp,"\nE4  Message was:  %s\r", data);
        fprintf( fp,"\nE5 ****************************************
                ******************\r");
        fflush(fp);
        return(COMM_ERR);
        }

    /* Log all other messages */

      fprintf( fp, "\n%s\r", data );
      fflush(fp);
      return(ALL_OK);
      }

}
```

```
/*=================================================================*/
/*                              closelog                           */
/*                                                                 */
/* Description:  Provide last time stamp and close all opened      */
/*               files                                             */
/*                                                                 */
/* Returns:     integer corresponding to one of the return         */
/*              definition                                         */
/*                                                                 */
/* In:          -                                                  */
/* Out:    (int n) corresponding return message                    */
/*                                                                 */
/*-----------------------------------------------------------------*/

/*
Include Files
-------------  */

#include "com.h"
#include "define.h"

int closelog()
{
    char dbuffer[9], tbuffer[9];

    fprintf(fp,"\nLO  Finish time stamp is : %s
            %s\r",_strdate(dbuffer), _strtime(tbuffer));
    fprintf(fp,"\n%s \r","  ");

    /* close all opened files  */

    if( _fcloseall() == 0 )  {
       printf("\nError occured.  Some files might still be opened.");
       return(QUIT);
       }
    return(ALL_OK);         /* files were closed successfully      */
}
```

```
/*=============================================================*/
/*                          gettime                            */
/*                                                             */
/* Description: Get date & time from Goes Clock. Converts it and */
/*              stores it into time_stamp                      */
/*                                                             */
/* Returns: -                                                  */
/* In:      -                                                  */
/* Out:     -                                                  */
/*                                                             */
/*-------------------------------------------------------------*/

/*
Include Files
-------------  */

#include "com.h"
#include "define.h"

char time_stamp[50];
struct tm jday, today;
struct _dostime_t newtime;
struct _dosdate_t newdate;
time_t now, clock_time;
int txdest;

void gettime(void)
{

    int ndest;
    char string[220];       /* String buffer to hold station name  */
    int t, clock_setting, clock_return;
    char time_buffer[100];
    char date_buffer[40];
    char control[4];        /* To store the Control char "SOH"     */
    time_t start, finish;   /* Periodic time update variables      */

    /* Select the link to 'goes_clock"  (COMA)  */

    if(( ndest = look_low("goes_clock")) == BAD_STATION ) {
      printf("System date and time will not be updated. \n");
      }
    else if(( ndest = look_low("goes_clock")) != BAD_STATION ) {

    /* Request time from Goes Clock using the "T" command */

      if(( clock_setting = puts_low( ndest, "T" )) == ALL_OK )
          printf("\nT clock trigger was sent to goes clock.\n");
      while(( clock_return = gets_low( ndest, '\n', time_stamp ))
            != ALL_OK ) {
          printf("\rWaiting for Goes_Clock to return time stamp...");
```

```
/* Check for exit key while waiting for time clock */

      if( checkkey( ndest ) != 0 ) {
      close_com();
      exit(1);
      }
  }
}

/* Converts time stamp to proper DOS format and set DOS        */
/* time and date                                               */

/* Converts the Julian date to find equivalant date and month */

sscanf( time_stamp, "%1s%d:%d:%d:%d:%2d", &control,
       &jday.tm_yday, &newtime.hour, &newtime.minute,
       &newtime.second, &newtime.hsecond);

/* set year to 1993, month to January, day to 01,              */
/* adjust new day.                                             */

today.tm_year = 93;
today.tm_mon  = 0;
today.tm_mday = 1;
today.tm_mday = ( 0 + jday.tm_yday);
if( ( clock_time = mktime( &today )) != (time_t)-1 ) {
  printf( "\nConverting goes_clock time format...\n");
  }
else
    perror( "mktime failed ");
newdate.day       = today.tm_mday;
newdate.month = today.tm_mon + 1;
newdate.year      = today.tm_year + 1900;
newdate.dayofweek = today.tm_wday;
_dos_setdate( &newdate );
_dos_settime( &newtime );
printf( "%s\n", _strtime( time_buffer ));
printf( "%s\n", _strdate( date_buffer ));

   /* Fill up time_stamp buffer to be sent to all station */

sprintf( time_stamp, "%d %d %d %d %d %d %d %d",newdate.dayofweek,
         newdate.day, newdate.month, newdate.year,
         newtime.hour, newtime.minute, newtime.second,
         newtime.hsecond);

}
```

```
/*=============================================================*/
/*                          sendtime                           */
/*                                                             */
/* Description: Sends time & date stamp to stations who have   */
/*              requested it                                   */
/*                                                             */
/* Returns: -                                                  */
/* In:      -                                                  */
/* Out:     -                                                  */
/*                                                             */
/*-------------------------------------------------------------*/

/*
Include Files
-------------  */

#include "com.h"
#include "define.h"

extern char time_stamp[50];
int txdest;
int p;              /* Integer index variable for periodic update  */
int begin[10];
int period[10];
int none[10];
struct tm jday, today;
struct _dostime_t newtime;
struct _dosdate_t newdate;
time_t now, clock_time;


void sendtime(void)
{

    int ndest;          /* Port number for desired destination     */
    char string[220];  /* String buffer used to hold station name */
    int t, clock_setting, clock_return;
    char time_buffer[100];
    char date_buffer[40];
    char control[4];    /* To store the Control char "SOH"         */
    time_t start, finish;  /* Periodic time update variables       */

    /* Select the link only if station time stamp flag are set */
    /* and if look_com does not return BAD_STATION              */

    /* Select the link to 'station[1] = beacon_mon" (COM 1)       */

    if((( p == 1 ) && ( begin[1] == 1 )) || (( p >= 2 ) &&
         ( period[1] == 1 ))) {
       if ((ndest=look_com("beacon_mon"))==BAD_STATION) {
           printf("Not able to update the Beacon Monitor system
                  time\n");
           }
```

```
    else {
        send_com(ndest, TIME_STAMP, time_stamp);
        printf("Time_stamp sent to Beacon Monitor\n");
        }
    checkmsg();
    }

  /* Select the link to 'station[2] = burst_demod" (COM 2)     */

if((( p == 1 ) && ( begin[2] == 1 )) || (( p >= 2 ) &&
      ( period[2] == 1 ))) {
    if ((ndest=look_com("burst_demod"))==BAD_STATION) {
        printf("Not able to update the Burst Demodulator system
               time\n");
        }
    else {
        send_com(ndest, TIME_STAMP, time_stamp);
        printf("Time_stamp sent to Burst Demodulator\n");
        }
    checkmsg();
    }

  /* Select the link to 'station[3] = tx_proc"  (COM 3)        */

if((( p == 1 ) && ( begin[3] == 1 )) || (( p >= 2 ) &&
     ( period[3] == 1 ))) {
    if((ndest=look_com("tx_proc"))==BAD_STATION) {
        printf("Not able to update the CRC transmit Processor
               system time\n");
        }
    else {
        send_com(ndest, TIME_STAMP, time_stamp);
        printf("Time_stamp sent to Transmit Processor\n");
        }
    checkmsg();
    }

  /* Select the link to 'station[4] = ephem_proc"  (COM 4)     */

if((( p == 1 ) && ( begin[4] == 1 )) || (( p >= 2 ) &&
     ( period[4] == 1 ))) {
    if ((ndest=look_com("ephem_proc"))==BAD_STATION) {
        printf("Not able to update the Ephemeris Processor system
               time\n");
        }
    else {
        send_com(ndest, TIME_STAMP, time_stamp);
        printf("Time_stamp sent to Ephemeris Processor\n");
        }
    checkmsg();
    }
```

```
  /* Select the link to 'station[5] = sync_proc"   (COM 5)    */

if((( p == 1 ) && ( begin[5] == 1 )) || (( p >= 2 ) &&
     ( period[5] == 1 ))) {
   if ((ndest=look_com("sync_proc"))==BAD_STATION) {
       printf("Not able to update the Synchronization Processor
              system time\n");
       }
   else {
       send_com(ndest, TIME_STAMP, time_stamp);
       printf("Time_stamp sent to Synchronization Processor\n");
       }
   checkmsg();
   }

  /* Select the link to 'station[6] = crc_antenna"  (COM 6)   */

if((( p == 1 ) && ( begin[6] == 1 )) || (( p >= 2 ) &&
     ( period[6] == 1 ))) {
   if ((ndest=look_com("crc_antenna"))==BAD_STATION) {
       printf("Not able to update the CRC Antenna system time\n");
       }
   else {
       send_com(ndest, TIME_STAMP, time_stamp);
       printf("Time_stamp sent to CRC Antenna\n");
       }
   checkmsg();
   }

  /* Select the link to 'station[7] = T85_antenna"   (COM 7) */

if((( p == 1 ) && ( begin[7] == 1 )) || (( p >= 2 ) &&
     ( period[7] == 1 ))) {
   if ((ndest=look_com("t85_antenna"))==BAD_STATION) {
       printf("Not able to update the T-85 Antenna system
              time\n");
       }
   else {
       send_com(ndest, TIME_STAMP, time_stamp);
       printf("Time_stamp sent to T-85 Antenna\n");
       }
   checkmsg();
   }

}
```

```
/*=================================================================*/
/*                              readlog                            */
/*                                                                 */
/* Description:  Read LOG.CFG and determine the time stamp flags   */
/*               for each station.  Station flags are stored in    */
/*               a bi-dimensional array [station[i], begin[i],     */
/*               period[i], none[i]] where 1 <= i <= NSTATION.     */
/*                                                                 */
/* Returns:      0  :  LOG.CFG was sucessufully read               */
/*               1  :  Error occured.                              */
/*                                                                 */
/* In:           -                                                 */
/* Out:          -                                                 */
/*-----------------------------------------------------------------*/

/*
Include Files
-------------  */

#include "define.h"
#include "com.h"

#define  ENDFILE  -2

char user[200];
int begin[10];
int period[10];
int none[10];

int readlog()
{
    FILE *logcfg;          /* File stream for LOG.CFG                */
    char line[220];        /* Line being processed                   */
    char lstline[220];
    int ln;                /* Line number of Line being processed    */
    int i;                 /* Integer index variable                 */

    /* Open LOG.CFG and get a non-blank line.                         */
    /* Comment lines are ignored                                      */

    /* open LOG.CFG */

    if(( logcfg = fopen( "log.cfg", "r" )) == NULL ) {
       printf("Cannot open log.cfg\n");
       return(1);
    }

    /* Get time flags for each station & store them in cfg table  */

    ln = 1;
    while( fgets( line, 220, logcfg ) != NULL) {
```

```
/* Ignore comment and blank lines */

 if( line[0] == ';') line[0] = '\0';
 else if( isspace( line[0] )) line[0] = '\0';
       else {
         sscanf( line, "%s %d %d %d", &user, &begin[ln],
                 &period[ln], &none[ln] );
        ln++;
         }
   }
printf("\nnumber of lines read in log_cfg: %d", ln);
return(0);
}
```

This is the modified **checkkey** routine as used during the testing procedures.

```
/*=========================================================*/
/*                    checkkey                             */
/*                                                         */
/* Description: Checks to see if a key has been pressed    */
/*              and performs the necessary action such as  */
/*              sending various messages or exiting        */
/*           Uses 'bios_keybrd' to see if a key has        */
/*              been pressed.                              */
/*           Control-C/break is not done here, but         */
/*              reported by checkmsg                       */
/*                                                         */
/* Returns:     (int)       0 for normal return            */
/*                          1 for exit from main           */
/*                          program due to keypress        */
/* In:          (int ndest) destination port number for    */
/*                          messages                       */
/* Out:         -                                          */
/*---------------------------------------------------------*/

int checkkey(int ndest)
{
    int c;          /* Character from the keyboard */

       /* Is a key pressed ? */
       /* Get the character */

    if (_bios_keybrd(_KEYBRD_READY) != 0) {
      c = _bios_keybrd(_KEYBRD_READ) & 0x0FF;
      switch (c) {
          case '1': /* 1 = Send message 1 */
            printf("Sending message 1\n");
            send_com(ndest,COMMAND,"Check buffer");
            break;
          case '2': /* 2 = Send message 2 */
            printf("Sending message 2\n");
            send_com(ndest,STATUS,"Buffer OK too");
            break;
          case '3': /* 3 = Send message 3 */
            printf("Sending message 3\n");
            send_com(ndest,STATUS,"Do a third");
            break;
          case '4': /* 4 = Send message 4 */
            printf("Sending message 4\n");
            send_com(ndest,STATUS,"Quarter");
            break;
```

```
            case 'e':
            case 'E':
            case 'q':
            case 'Q': /* e,E,q,Q,x,X = quit */
            case 'x':
            case 'X':
              return(1);
            default:  /* Otherwise ignore */
              break;
        }
      }
      return(0);
}
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Ottawa
Ottawa, Ontario
K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Real-Time Data Storing Package for Skynet Trials (U)

4. AUTHORS (Last name, first name, middle initial)

Capt E.R. Boudriau

5. DATE OF PUBLICATION (month and year of publication of document)

April 1994

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

58

6b. NO. OF REFS (total cited in document)

5

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.

DREO Technical Note

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.

EHF SATCOM, Radar and Space Division, Defence Research Establishment Ottawa
Ottawa, Ontario, K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

Project D6470

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO Technical Note 94-3

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). however, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

Unlimited Announcement

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

A series of week-long experiments on an EHF downlink were conducted by the MILSATCOM groups at CRC and DREO. The software developed for these experiments included a reliable and robust data logging package to record the setup and measurements data required for post experiment analysis. The software provided the Skynet EHF Trials with a data logging package written in "C" that would interface with the existing serial communication software. The package was used as a real time storing device for recording experimental data such as hop power, noise power spectral density and pointing angles. In addition to data logging, the software had to provide several date and time references using the zulu time provided by a GOES satellite clock as the Trials' time reference. The software was loaded in a computer called Data Logger and was responsible for opening and closing all high and low level communications with the processors linked to it.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Date Logging
Skynet Trials