DSN Automation

R. B. Crow
Radio Frequency and Microwave Subsystems Section

Automation of the DSN has been under consideration for the past several years and has been justified on anticipated reduction in life cycle cost, resulting from increased productivity and reduced operations cost.

This article summarizes an overall hierarchical automation philosophy along with the results of the RF automation effort undertaken 2 years ago. A brief description of each subassembly controller's salient features, the software development process, and the common software used by these controllers will be presented. Comments will be made with respect to the relative advantages of PL/M high-level language and assembly language, the operational effectiveness of operator "Macro commands," and the program development, and a list of suggested future effort will be given.

This article provides for technology transfer and offers new ideas for consideration in future automation efforts.

I. Introduction

A philosophy of an hierarchical automation plan has been developed that emphasizes distributed control and the utilization of DSN hardware development engineers to develop the automation software.

Recent efforts in developing an automation capability for the RF subsystem are reported on. Comments are made on the facilities available to develop software for microcomputers and the common software developed for the RF automation demonstration.

II. Automation Philosophy

Several years ago, the DSN began investigating automation as a means of reducing DSN cost while furnishing the projects with more and better service.

A demonstration was held at the Goldstone Mars Station, DSS 14, in May of 1975, and more recently a second-generation demonstration was held at JPL in order to gain a data base. These experiences have contributed to the following automation philosophy.

A convenient approach in automating the DSN is to divide the task into four major areas: RF subsystems control, digital subsystems control, antenna subsystems control, and station control (see Fig. 1). This division of responsibility provides good development management posture since the technology base and administration control for each of the proposed major areas are contained in a single technical section.

A. RF Subsystems Control

The RF subsystems automation (see Fig. 2) has been accomplished by employing distributed control. Each major class of equipment (microwave, receiver-exciter, subcarrier demodulators, transmitters, RF subsystem) has its own controller. In this way, one or two engineers can be held responsible for assuring that the software they provide is capable of configuring, calibrating, and operating their equipment.

This is a slightly different approach from that implemented by the station manager control (SMC 2A) project, where the operator controls each of the RF subassemblies directly. The extra level of control (i.e., the subsystem controller) offers better monitoring, control and failure backup, and diagnostic capability than that offered by the SMC 2A approach, since the particular RF subsystem controller can devote most of its resources to monitoring and controlling the subsystem, and the required software to complement failure backup and diagnostics can be developed by the cognizant development engineer (CDE) of the equipment.

The RF subsystems controller serves the same function as the SMC in that it is responsible for subsystems coordination and is a focal point for all operator/machine interfaces. However, it differs from the SMC design in that it is intended that all RF subsystems at a DSS can be controlled from one control channel, instead of requiring three as the SMC has been implemented.

An important observation is that the further the controller is from the hardware, the more the level of detail is reduced (i.e., a CALIBRATE command to a receiver controller involves a large number of highly complicated communications, while a CALIBRATE command to a RF subsystem controller has little detail but more concern with subsystem coordination. A CALIBRATE command to the station controller would be a very high-level command with only simple intrasubsystem coordination required).

B. Digital Subsystems Control

The metric data subsystem (MDS) currently contains the same control structure used by the RF subsystems. A host task via the data system terminal is capable of controlling the other subsystems controllers (i.e., metric data assembly, command processor assembly, telemetry processor assembly, and communication monitor formatter). Further, each of the previously mentioned assemblies is also capable of configuration control and monitoring the digital hardware assemblies under its control (i.e., the telemetry processor controls the symbol synchronizer and the decoders).

C. Antenna Subsystems Control

The automation of the antenna mechanical subsystem will consist of antenna pointing as well as increased monitoring and control of the antenna servo and drive subsystems, so that unattended station operation is possible.

D. Station Controller

The station controller should be the focal point of the station and communicate with the operator in a high-level language. It should be adaptable and offer innovative solutions to new operational problems (i.e., each subsystem should have the ability to "read and write" all the functions in its area, but the station controller should be responsible for the "composition" of new operational scenarios). In this way, the antenna, RF subsystems, and digital subsystems areas can remain relatively unaffected, thereby localizing the changes to the station controller to meet the unexpected new problems. This division of control allows a viable fallback position for semiautomatic operation of the station should the station controller fail.

Some attention should be given to allowing the generation in near-real time of new operational scenarios. The new tools that will emerge from the integration of an automated station, such as improved acquisition procedures, real-time evaluation of downlink performance, rapid reconfiguration, etc., offer a considerable reward in increased productivity and should be pursued vigorously.

III. RF Automation Summary

The RF automation demonstration conducted in May 1975 (ref. DSN Progress Report 42-29, p. 66) identified two major goals for future activity:

(1) Improve hardware/software reliability and maintainability.

(2) Improve the operational effectiveness of the automation programs by providing *all* existing operator controls and offer new, more effective high-level operational techniques.

In answer to the problem of reliability and maintainability, an automation microcontroller was developed (ref. DSN Progress Report 42-30, p. 144) which appears to have satisfied the first goal.

Heavy Viking schedule pressure on DSS 14 caused the second RF demonstration, scheduled for November 1976, to be canceled. The demonstration was then rescheduled for the Compatibility Test Area at JPL (CTA 21); however, this has been subsequently delayed. A sequence of demonstrations is planned to be run at the Goldstone Venus Station, DSS-13, in FY'77 and '78 to demonstrate unattended station operation.

A JPL "lab" test has been run, in which the system capabilities have been exercised by simulating the proposed CTA 21 RF subsystem (the microwave subsystem, the Block IV receiver-exciter and the Block III subcarrier demodulator assemblies), that appears to have the capability of measurably expanding the operational tracking time in the DSN.

A brief description is presented for each subassembly controller in an attempt to illustrate any special features it may have and to note any change in its functional capability that was not present in the first RF automation demonstration.

A. RF Subsystems Controller

The RF subsystems controller links the operator (see Fig. 2) to the RF subsystems (i.e., microwave, Block IV receiver-exciter, and subcarrier demodulator assembly [SDA]) by accepting the operator's high-level input commands and distributing these commands to the appropriate subsystem controller. The RF subsystems controller also controls all subsystem coordination required to calibrate and acquire, maintains system status, and forms and stores mission configurations.

The RF subsystem controller has four phases (initiate, configure, calibrate, and operate). An improvement in operational effectiveness was achieved by allowing the operator to select any operational phase, instead of having a set sequence. The calibrate phase was modified to allow zero-delay range calibration and bit error test for the telemetry channel.

An important fundamental change was made to make the RF subsystem controllers essentially "station-independent." A subroutine was designed that allows the implementation of any set of logic tables which will direct its question to a particular input source. If the source message satisfies two test conditions (as programmed in the logic tables), the routine will set those status vectors identified in the logic table (for internal program control), send out control messages to other controllers, and output operator messages as defined in the logic table (i.e., if receiver-exciter subsystem [RCV] 3 sends a calibration complete message and if SDA 1 is connected to RCV 3, then set F23 = 1, send SDA 1 a calibrate command, and notify the operator via the cathode-ray tube [CRT] and teletype [TTY] that "RCV 3 has calibrated").

Because of the station independence resulting from this subroutine, the conversion from a DSS 14 design to a CTA 21 design was completed in less than a week. In fact, such versatility was designed into the program that it could readily be applied as an RF subsystem controller, or a station controller, at any DSS.

Table 1 documents the details of the software development.

B. Subcarrier Demodulator Assembly Controller

The SDA controller can control and monitor up to six SDAs. These SDAs can be either automated Block III SDAs or a standard Block IV SDA. The SDA controller determines how many and what type of SDAs are connected to it (by analyzing the monitor words returned by the SDA). Knowledge of the type of SDA (Block III or Block IV) is required to properly configure and diagnose failure in the SDAs. A continuous monitor is kept to assure that all required references are present, that all relays are operating correctly, and that all control switches that are not under computer control have been set correctly.

Diagnostics are restricted to those that can be deduced directly from the SDA monitor words, which in general will not isolate the problem to a particular replaceable module. Future expansion of these diagnostics is planned for later development (see Table 1 for details of software development).

C. Receiver-Exciter Controller

The receiver-exciter (R-E) controller can control one Block IV receiver-exciter subsystem (consisting of two S-X receivers, one S-X exciter, three programmed oscillators, an instrumentation control assembly, and a time code distribution assembly). The subsystem elements can be operated independently or in any combination to allow one-way, two-way, or three-way operations.

The R-E controller is controlled by the RF subsystem controller; however, in a backup mode it can be controlled by a local TTY. A second "standard" mode allows the station monitor and control assembly (SMC) 2A to control the R-E controller either from the SMC console or the SMC keyboard.

A continuous monitor is maintained on all necessary conditions for operation (references are present, all relays confirmed, the actual configuration agrees with the last

operator configuration input, etc.). All programmed oscillator performance is measured against a locally generated model to confirm its operation.

The R-E controller is capable of effecting automatic carrier acquisition and automatic gain control (AGC) calibration (selectable from single point to 50 points in integer steps).

The R-E controller has a preliminary set of diagnostics built in and is capable of outputting current status of VCO frequencies, exciter frequency, configuration, dynamic and status phase error, AGC, etc. (see Table 1 for details of software development).

D. Microwave Controller

The antenna microwave subsystem (UWV) controller can control up to 5 bays of microwave equipment (approximately 65 microwave switches).

The program has been structured to be station-independent by incorporating a "station configuration overlay."

Since it is possible to attempt impossible (or catastrophic) configurations, an internal editor resident in the UWV controller reviews all input configuration messages to determine if they are acceptable. The UWV controller will configure all those switches that are allowable and issue a diagnostic warning to the operator for those switches that could not be configured.

A continuous monitor is kept on all relays, and an operator diagnostic message is sent in case of failure (see Table 1 for details of software development).

IV. Software Development

Stand-alone microcomputer high-level compilers did not exist at the time the RF automation project was started, since industry had adopted the use of larger, general-purpose computers to serve as the program development facility.

Presently there are two development computers at JPL equipped with functioning assemblers and PL/M compilers. (PL/M is an offshoot of IBM's PL/1. PL/M is a block structured language, well suited to structured programing, and has been adopted by both Intel [8080] and Motorola [6800], which are the current industry leaders.)

The Univac 1108 system (see Fig. 3) allows program development via a time-shared terminal through a high-speed modem. Once the program has been compiled, a paper tape is made through the 8080 microcontroller/high-speed punch. This paper tape contains the object code that is read into the target microcontroller.

The Sigma 5 computer (see Fig. 4) has the same assembler and PL/M compiler installed in it. The operation of this equipment is similar to that of the Univac 1108 except that program development is via the batch mode and the object code is punched on cards. It is currently necessary to use a MAC 16 minicomputer to convert from the Sigma 5 cards to a paper tape suitable for input to the 8080 microcomputer.

A third development system (see Fig. 5) appears to be available in the near future. Intel has offered a "stand-alone" PL/M compiler that requires a 8080 microcomputer system with 64K bytes of memory and a floppy disk. The software for this compiler has been ordered and is due in the near future.

V. Common Software

A set of common software was developed to improve software maintainability and to reduce the overall RF subsystem software development activity. The common software consists of a variety of utility routines to make program development easier and to assist in debugging applications. The common software can be categorized as follows:

(1) Operator control via the console keyboard.

(2) Input/output (I/O) interface procedure.

(3) String procedures.

(4) Arithmetic procedures.

A. Operator Control Via the Console Keyboard

It is assumed that all controllers have an executive routine that monitors the console input. This software routine is used to handle all operator console keyboard inputs. The KEY routine has an editing capability so that the configuration data under consideration can be loaded, displayed, and changed, a tape of the desired configuration can be punched, and it can catalog or delete the entry and set the required global control variable used by the main program. (See Appendix I for examples of the operator display for the RF subsystems.)

B. I/O Procedures

I/O procedures are high-level PL/M routines to control the standard 15-line interface, the star switch controller, the console device, the TTY, paper punch, and reader. (See Appendix II for a description of the I/O procedures.)

C. String Procedures for PL/M

Various string procedures have been written to aid PL/M programmers in formatting, analyzing, and performing other string manipulations. The string procedures perform the following manipulations:

(1) Concatenation

(2) Segmentation

(3) Comparison

(4) Nulling

(5) Binary to ASCII decimal conversion

(6) Binary to ASCII hexadecimal conversion

(See Appendix III for a description of the string procedures.)

D. Arithmetic Procedures

Fixed-length, variable point arithmetic has been implemented because of the receiver-exciter control assembly's need for high precision. (See Appendix IV for a description of the arithmetic procedure.)

E. General Comments

The software development for the second RF demonstration was started in August 1975. The original estimate called for in an "in lab" demonstration by April 1976. This demonstration actually took place 5 months later than planned. Analyzing the reasons for the delay leads to the following observations.

The programs were more complex and detailed than first conceived in August 1975. The primary cause of the unforeseen complexity was a desire to be "station-independent" and produce "better operational effectiveness."

Several weeks' delay was caused by the installation of the PL/M compiler in the Univac 1108 computer with sufficient working core to handle programs up to 36K bytes. An interesting side note is the considerable unseen expense of Univac 1108 computer time. A 30K byte PL/M program costs approximately $50 to compile!

It is interesting to note that the PL/M programs require approximately 11 bytes/record, while the assembly program (microwave controller) required approximately 4.5 bytes/record (i.e., PL/M takes two and a half times more storage than the assembly language). However, since PL/M is a higher-level language, it is not surprising that each PL/M record would take the place of several assembly records. What small premium is paid for in memory efficiency is more than offset by improved program readability and maintainability.

The communication between computers was through the standard star switch and was found to work well (the interface software was written in assembly language and was under interrupt control) when the RF subsystem was in a "stand-alone" configuration; however, if later plans call for an integration of the RF subsystem into a station complex, a high-speed interface must be designed in order for the RF subsystem to communicate at the required 250K SPS (instead of the current 1K SPS).

VI. Summary

The RF automation effort has developed a capability to use the new microcomputer technology to make automation practical from a cost and reliability viewpoint.

An interesting and valuable side benefit is the capability that now exists of the software engineer being the same person as the hardware engineer. This capability greatly enhanced the "operational effectiveness" of the finished software.

Table 1. Second RF demonstration software development data

Program Name	No. of Records	Storage Requirement, bytes	Software Development Time, man-weeks			Computer Language
			Design	Code	Debug	
Common software						
Star	1125	2K	2	2	2	Assembly
15-line interface	600	1K	2	2	2	Assembly
Arithmetic string procedures	675	5.4K	3	3	2	PL/M
Key	3000	11K	3	3	6	Assembly
RF Demo	1750	18K	26	8	10	PL/M
SDA	600	6.5K	8	4	4	PL/M
R-E	3000	34K	24	12	10	PL/M
UWV	4000	18K	26	6	8	Assembly

Note: 1. Time to install and check out PL/M compiler in Univac 1108 and the Sigma 5 was not counted against this development.

2. Most programmers were learning the language (except for the SDA programmer).

3. Main problem during development has been access to PL/M compiler and assembler.

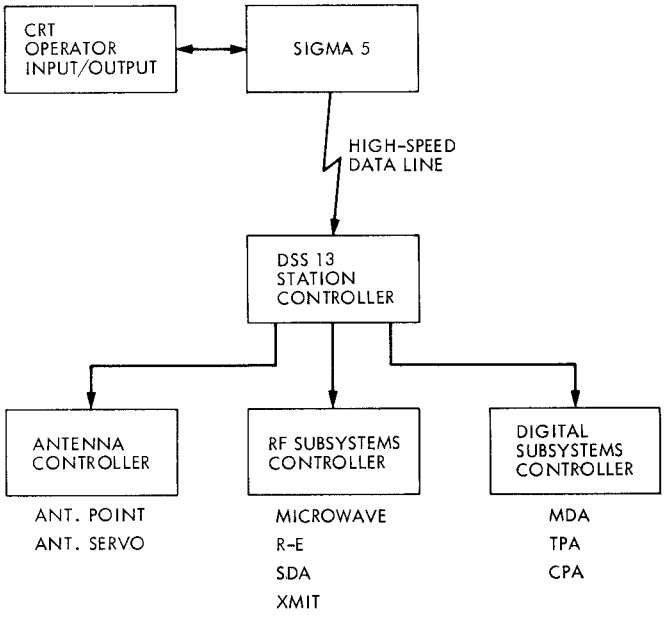

Fig. 1. Block diagram of DSS 13 unattended station control configuration

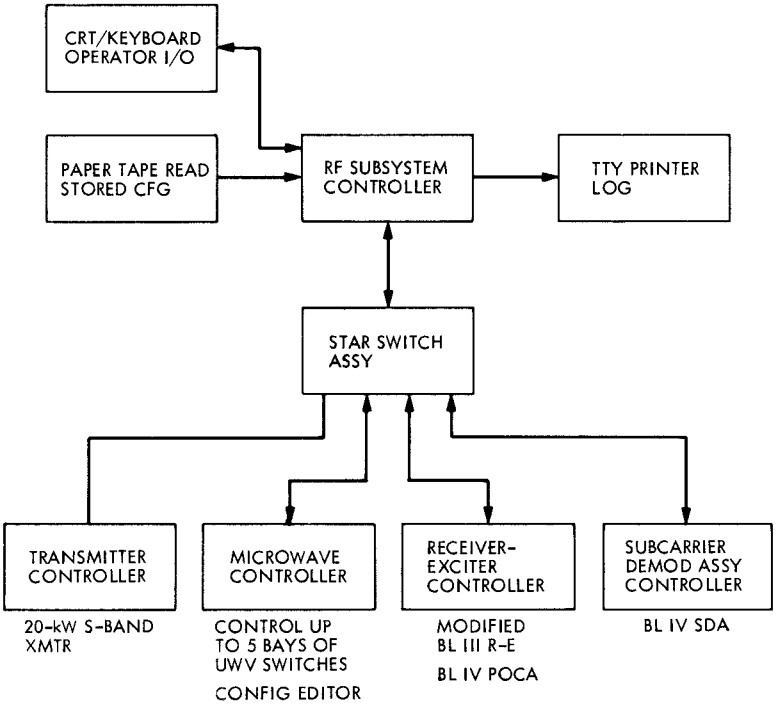

Fig. 2. Block diagram of DSS 13 RF subsystem automation configuration

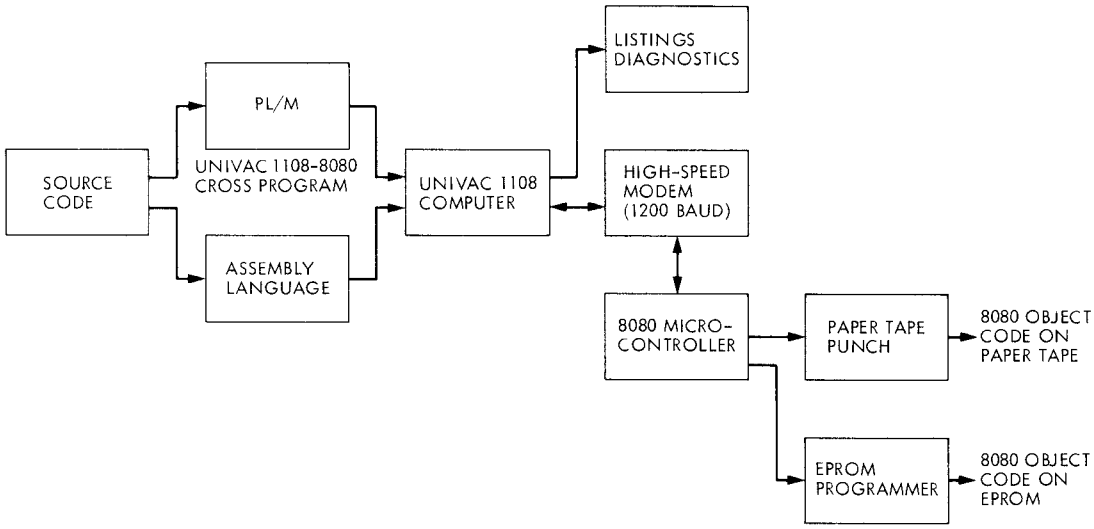

Fig. 3. 8080 software development using the UNIVAC 1108/8080 development system

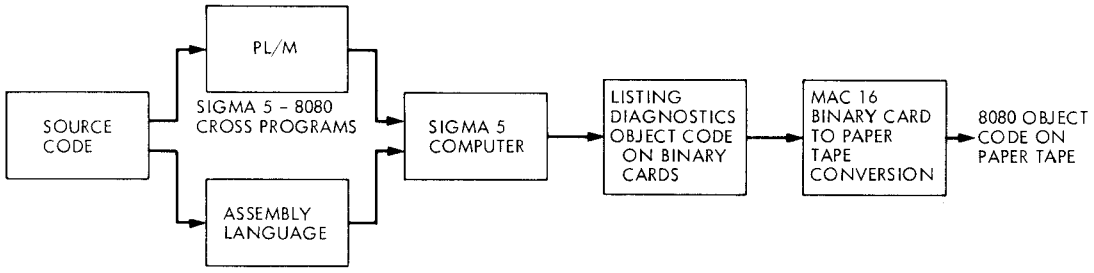

Fig. 4. 8080 software development using the Sigma 5/MAC 16 system

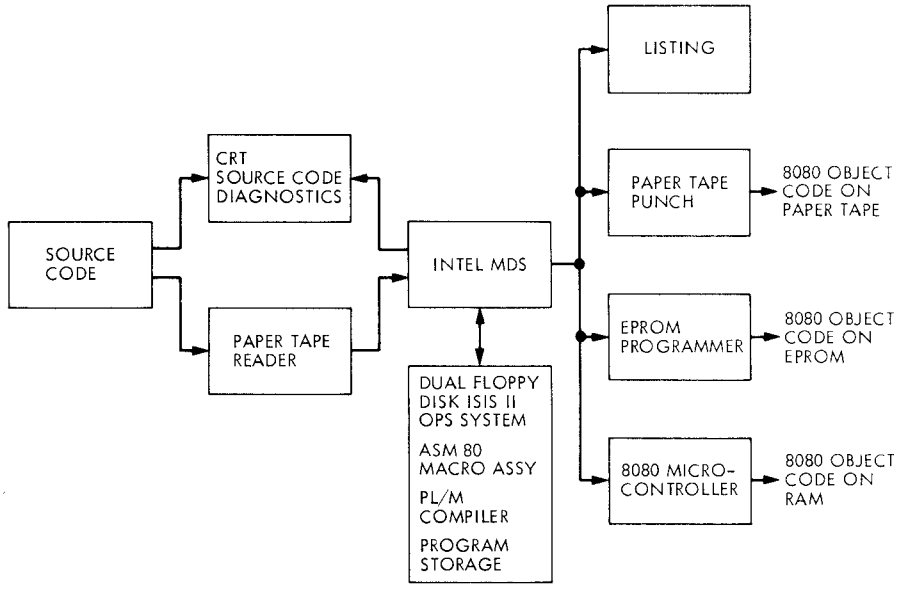

Fig. 5. Stand-alone 8080 software development system

Appendix A

"Key" Operator Display

I. Receiver 3 Configuration

A	BACKUP	=	R3 (R3, R4, NONE)
B	LOOP MODE	=	WIDE (WIDE, NARROW)
C	LOOP BW	=	4 (4, 3, 2, 1)
D	BAND SELECT	=	X (X, S)
E	AGC BW	=	WIDE (WIDE, MED, NARROW)
F	TELEMETRY BW	=	3 (4, 3, 2, 1)
G	ATZ ENABLE	=	ON (ON, OFF)
H	LOOP FILTER	=	OPERATE (SHORT, OPERATE)
I	LO AGC INPUT	=	NORM (EXT, NORM)
J	TRANSFER FCN	=	2ND (3RD, 2ND)
K	RECEIVER LOOP	=	OPEN (OPEN, CLOSED)
L	LO REF	=	ON (ON, OFF)
M	LO REF TO	=	EX4 (EX4, EX3, EX2, EX1)
N	LO OUTPUT	=	ON (ON, OFF)
O	PREDIX SET	=	11 (III, II, I)

SAMPLE, CFG/R3/F=2/D=S$

Note: This is a typical Block IV receiver frame. *All* configuration switches are listed so that no oversight is possible. The KEY program allows these frames to be updated to reflect any mission configuration. Once all subassembly configurations are updated to the desired state, a paper tape can be punched so that this operational configuration can be recalled at will. This feature allows "offline" generation of operational configurations and fast retrieval so that quick turnaround is possible.

II. Exciter 1 Configuration

A	RANG MODS ON	=	1 2 3 (MISS G ARE OFF)
B	CMD MODS ON	=	1 2 3 4 (MISS G ARE OFF)
C	BAND SELECT	=	X (X, S)
D	PH CTL LOOP	=	OPERATE (SHORT, OPERATE)
E	MOD DLY LOOP	=	SHORT (SHORT, OPERATE)
F	DOPPLER	=	NORM (SIMULATE, NORM)
G	EXC FREQ	=	NORM (SIMULATE, NORM)
H	CMD PH MOD	=	NORM (BYPASS, NORM)
I	DOPPLER BIAS	=	+ (+, -)
J	EXC BAND	=	X (X, S)
K	EXC DRIVE	=	ON (ON, OFF)
L	EXC TEST SIG	=	ON (ON, OFF)
M	PH CTL LOOP	=	EXC (EXC, KLYSTRON)
N	STEP ATTN	=	X (X, S)
O	PULS ATTN	=	X (X, S)
P	XB STEP ATTN	=	ZRO DLY (ZRO DLY, TEST)

SAMPLE, CFG/EI/A=13/C-S$

III. Exciter 2 Configuration

A	X-BAND EXC	=	DSN (RADAR, DSN)
B	TRAVSL (S/S)	=	ON (ON, OFF)
C	TRAVSL (S/X)	=	ON (ON, OFF)
D	TRAVSL (X/X)	=	ON (ON, OFF)
E	ZRO DLY (S/S)	=	ON (ON, OFF)
F	ZRO DLY (S/X)	=	ON (ON, OFF)
G	ZRO DLY (X/X)	=	ON (ON, OFF)
H	SB REFS ON	=	1 2 3 4 (MISS G ARE OFF)
I	XB REFS ON	=	1 2 3 4 (MISS G ARE OFF)
J	SB PULS ATTN	=	NORM (BYPASS, NORM)
K	XB PULS ATTN	=	NORM (BYPASS, NORM)
L	SB PULS ATTN	=	NORM (BYPASS, NORM)
M	XB PULS ATTN	=	NORM (BYPASS, NORM)
N	STEP ATTN	=	19 (2-DIGIT INTGR)
O	PULS ATTEN	=	199 (3-DIGIT INTGR)
P	PREDIX SET	=	II (III, II, I)

SAMPLE, CFG/EII/N=99/B=OFF $

IV. SDA 1 Configuration (Block III)

A	BACKUP	=	S1 (S1, S2, NONE)
B	INPUT	=	R1 (TEST, TAPE, R2, R1)

C VCO SHORT = OFF (ON, OFF)

D LOOP BW = WIDE (WIDE, MED, NARROW)

E MOD INDEX = 30 (0-30 DB)

F SUBCARRIER = 24000.00 (HZ)

G SYMBOL RATE = 99.00 (5.6 - 270000.0)

H OUTPUT = DEMOD (TEST, TAPE, DEMOD)

SAMPLE, CFG/SDA1/A=S2/D-MED/E=6$

V. SDA 5 Configuration (Block IV)

A BACKUP = NONE (S5, S6, NONE)

B LOCAL INPUT = TEST (TEST, TAPE)

C REMOTE INPUT = R2 (R2, R1)

D INPUT = REMOTE (LOCAL, REMOTE)

E VCO SHORT = OFF (ON, OFF)

F LOOP BW = WIDE (WIDE, NARROW)

G LOOP GAIN = HIGH (HIGH, LOW)

H AUTO ACQ = ON (ON, OFF)

I MODE SELECT = INTRPLX (INTRPLX, NORM)

J MOD INDEX = 30 (0-31DB)

K SUBCARRIER = 32012.10 (HZ)

L SYMBOL RATE = 199.00 (5.6 - 500000.0)

M DEMOD OUTPUT = ON (ON, OFF)

N LOCAL OUTPUT = TEST (TEST, TAPE)

SAMPLE, CFG/S5/B = TAPE/M = ON $

VI. Uplink Configuration and Status Data

A PREDIX = I (III, II, I)

B EXCITER = EXCI (EXC1 & 2, EXC2, EXC1)

C ALT EXCITER = EXC2 (EXC1 & 2, EXC2, EXC1)

D XMITTER = 400 R&D (400 R&D, 400 KW, 20 KW)

E ALT. XMITTER = 20 KW (400 R&D, 400 KW, 20 KW)

F POWER = 390 (KWS)

G ALT POWER = 20 (KWS)

H CONE = SPD (SPD, WTRLD)

I POLARIZATION = LINEAR (LINEAR, RCP, LCP)

J ANGLE = 180 (DEGREES)

K CONE MODE = 1 (SEE MODE PROMPT)

SAMPLE, CFG/UL/A = II/G = 39°/H = WTRLD$

MODES: 1. XMIT, 2. R&D, 3. LN RCV, 4. LN BYPASS, 5. DUAL RCV, 6. DUAL BYPASS, 7. DIPLX, 8. CAL SPD TWM, 9. CAL MOD TWM, 10. SAFE.

Note: This frame also serves as an uplink status since power (item F) will be the "actual" uplink power.

VII. Downlink 1 Configuration and Status Data

A PREDIX = II (III, II, I)

B CODE = SPD (SPD, XKRA, XK, RB, XRO)

C POLARIZATION = RCP (LINEAR, RCP, LCP)

D ANGLE = 0 (DEGREES)

E CONE MODE = 4 (SEE MODE PROMPT)

F

G ALT CONE MODE = 5 (SEE MODE PROMPT)

H RECEIVE = R3 (R1, R2, R3, R4)

I ALT RECEIVER = R4 (R1, R2, R3, R4)

J BL 3 SDA = 1 2 (MISSING ARE OFF)

K BL 4 SDA = 5 6 (MISSING ARE OFF)

L ALT 3 SDA = 2 1 (MISSING ARE OFF)

M ALT 4 SDA = 6 5 (MISSING ARE OFF)

N RECEIVER LOCK = ON

O BLK 3 LOCK = 1, 2

P BLK 4 LOCK = NONE

SAMPLE, CFG/PL1/A = III/B = XRO/K = 56$

MODES: 1. XMIT, 2. R & D, 3. LN RCV, 4. LN BYPASS, 5. DUAL RCV, 6. DUAL BYPASS, 7. DIPLEX, 8. CAL SPD TWM, 9. CAL MOD TWM, 10. SAFE.

Note: This frame also serves as a downlink 1 status, since items N, O, P reflect the in-lock status of the receiver and associated SDAs.

VIII. Predix 1 Configuration

A S/C CHANNEL = 0.000000000000 (MHz)
B ACQ BW = 0.000000 (Hz)
C ACQ DOPPLER = 0.000000 (Hz)
D ACQ RATE = 0.000000 (Hz/SEC)
E DOPPLER RATE = 0.000000 (Hz/SEC)
F RATE 2 = 0.000000 (Hz/SEC)
G RATE 3 = 0.000000 (Hz/SEC)
H TIME Ø = 000$000000 (DDD$HHMMSS)
I TIME 1 = 000$000000 (DDD$HHMMSS)
J TIME 2 = 000$000000 (DDD$HHMMSS)
K TIME 3 = 000$000000 (DDD$HHMMSS)
L EXC DRIVE ON = 000$000000 (DDD$HHMMSS)
M RANG MODS ON = 000$000000 (DDD$HHMMSS)
N CMD MODS ON = 000$000000 (DDD$HHMMSS)

SAMPLE, CFG/P1/K=100/65959/G-45$

Note: There are three Predix displays.

Appendix B

I/O Routines

I. Standard Interface Input and Output

The standard 15-line interface I/O routines are called L15IN and L15OUT for the input and output routines, respectively. The call for input is

```
CALL 15 IN(PORT,TIME$OUT,.DATA$ARRAY,MAX$LENGTH);
```

where $1 \leqslant$ PORT $\leqslant 12$ defines which of the standard interfaces is to be assessed. TIME$OUT is a positive integer indicating the time-out period, .DATA$ARRAY provides the address of the data array to receive the input data, and MAX$LENGTH is an integer defining the maximum number of bytes of input to accept. Note that the data array is a byte array. The calibration of one count in a time-out period will be established empirically and published later.

The routine L15IN sets the following items prior to exiting:

- FAIL to indicate successful receipt of data or failure
- FUNCTION to the received function code if data were received
- IO$LENGTH to the number of bytes received
- DATA$ARRAY receives the actual data bytes

The routine L15IN also utilizes the following global data items to determine its functional behavior:

- LOCKOUT to decide whether the routine should manipulate the enable/disable status of the interrupt system

Note that if the function code changes during input, the input is terminated. The user can pick up the data using the new function code by calling L15IN again.

The call to output via a standard 15-line interface is as follows:

```
CALL 15OUT(PORT,FCN,TIME$OUT,.DATA$ARRAY,LNGTH);
```

where $1 \leqslant$ PORT $\leqslant 12$ indicates the 15-line interface to use, $0 \leqslant$ FCN $\leqslant 3$ indicates the function code to pass with the data, TIME$OUT is a positive integer indicating the time-out period, DATA$ARRAY contains the byte(s) to output over the interface, and the positive integer LNGTH indicates the number of bytes to output with the given function code over the interface.

The routine L15OUT modifies the following items prior to exiting:

- FAIL to indicate success or failure of the transfer
- 10$LENGTH to the actual number of bytes transferred (= LNGTH if a successful transfer)
- FUNCTION to the received function code if the computer was outprioritied by the device during or prior to the transfer

The routine L15OUT utilizes the following global data items to determine its proper functional behavior:

- LOCK$OUT to decide whether the routine should modify the interrupt system enable/disable status
- STCOP to decide on the disposition of $\overline{\text{STC}}$ at the end of the transfer

II. Star Switch Input and Output

The star switch input and output routines, STARIN and STAROUT, respectively, are described separately from the 15-line interface routines because of routing parameters and other differences in the calling sequences.

For the RF automation demonstration, the following STAR$ID will be used for routing:

STAR$ID	Subassembly	
0	RF Auto. Demo Controller	
1	R-E Controller	
2	SDA Controller	
3	UWV Controller	
4	XMT Controller	
5	CCA	future
6	NOCC	
7	SMC	

Note that when a message is received from STARIN, the global variable SOURCE$ID contains the process code (STAR$ID) of the source of the message.

The call for star switch input is

```
CALL STARIN (.DATA$ARRAY,MAX$LENGTH);
```

where DATA$ARRAY is a byte array of length MAX$LENGTH, which will receive the first waiting message received over the star switch. Note that since star switch input is triggered by interrupts, the star switch input logic may have 0, 1, or multiple messages already within the computer memory.

The routine STARIN sets the following items prior to returning to the caller:

- FAIL to indicate successful/unsuccessful receipt of the data message (successful, always, for the user)
- STAR to indicate whether data were passed back to the caller
- 10$LENGTH to indicate the number of bytes in the message
- SOURCE$ID to furnish the process code of the system sending the message to this assembly

The call to output via the star switch controller is as follows:

```
CALL STAROUT (DESTINATION,.DATA$ARRAY,LNGTH);
```

where 0 DESTINATION 15 is the process code to which to send the message, which is of length, LNGTH, in bytes, and is stored in the byte array, DATA$ARRAY. The user must have previously set the global variable, STAR$ID, to the process code of this computer.

The routine STAROUT sets the following global data items:

- FAIL to indicate the success or failure of the operation
- 10$LENGTH to the actual number of bytes output

The routine STAROUT utilizes the following global data items for control:

- STAR$ID, the process code being sent from

The function code used over the star is $\overline{FO}$-$\overline{FT}$ = 0. The common software automatically transmits and verifies a checksum byte and also sends an acknowledge block for a successfully received message. These functions are performed automatically by the common software, so they are of no concern to the user.

III. Console Input and Output

The standard routines for console input and output are called CONIN and CONOUT, respectively. The call for console input is

```
CALL CONIN(.DATA$ARRAY,MAX$LENGTH);
```

where the byte array DATA$ARRAY is to receive the input bytes and MAX$LENGTH is the maximum number of bytes to accept into the array.

The routine CONIN sets the following global items prior to returning to the caller:

- FAIL to indicate the success or failure of the operation
- 10$LENGTH to indicate the actual number of bytes received

The routine CONIN looks for a carriage return to mark the end of an input operation.

The call for console output is

```
CALL CONOUT(.DATA$ARRAY,LENGTH);
```

where the data to output are in the byte array DATA$ARRAY and precisely LNGTH bytes are to be output.

The routine CONOUT sets the following global data items prior to returning to the caller:

- FAIL to indicate the success or failure of the I/O operation
- 10$LENGTH to indicate the number of bytes actually output

IV. Paper Tape Input

The routine PTR is used to get paper tape input. The calling sequence is

```
CALL PTR(.DATA$ARRAY,MAX$LENGTH);
```

where the byte array DATA$ARRAY is to receive the input bytes and MAX$LENGTH is the maximum number of bytes to accept into the array. The routine PTR sets the following global data items prior to returning to the caller:

- FAIL to indicate the success or failure of the operation

– 10$LENGTH to indicate the actual number of bytes received

V. List Output and Protocol

The routine to output to the TTY printer is called by the statement

```
CALL LISTOUT (.DATA$ARRAY, LNGTH);
```

where the data to output are in the byte array DATA$ARRAY and precisely LNGTH bytes are to be output.

The routine LISTOUT sets the following global data items prior to returning to the caller:

– FAIL to indicate the success or failure of the operation

– 10$LENGTH to indicate the number of bytes actually output

The subassembly controller will use the TTY printer of the RF demo controller using the following protocol:

(1) The subassembly controller will send a message over the star switch with a TASK = list device request.

(2) When a message is received from the RF demo with a TASK = list device available, the subassembly controller will send a message with TASK-list MSG. and the parameter string equal to or less than 72 ASCII characters.

(3) This process is repeated, at a rate determined by the RF demo controller until the message has been sent. After the last line has been sent, the subassembly controller sends a message with TASK = list device released, which terminates this list process.

(4) Should a subassembly controller request the list device when it is being used, the RF demo controller will place the request on a F1F0 que and send a message to the subassembly controller with a TASK = list device busy. When the list device is available, the RF demo controller will send a message with TASK = list device available.

VI. Punch Output

The call for high-speed punch output is

```
CALL HISPEED (.DATA$ARRAY, LNGTH);
```

where the data to output are in the byte array DATA$ARRAY and precisely LNGTH bytes are to be output.

The routine TTYP sets the following global data items prior to returning to the caller:

– FAIL to indicate the success or failure of the I/O operation

– 10$LENGTH to indicate the number of bytes actually output

Appendix C
String Procedures for PL/M

I. Editing Procedures

The editing procedures are

ADDSEG for appending a segment of one string to another

APPEND for appending an entire string onto the end of another

EQUATE for setting one string equal to another

SEGMENT for setting one string equal to a segment of another

REPLACE for copying one string into a fixed portion of another

A. The ADDSEG Procedure

This procedure, called by the statement

CALL ADDSEG (.M1,.M2, FIRST, LAST);

adds the segment from M2 (FIRST) to M2 (LAST) onto the end of M1, thereby increasing the length of M1. Here M1 and M2 are two strings and FIRST and LAST are variables, constants, or expressions indicating the segment of M2 to add onto M1.

Special cases:

(1) If FIRST = 0, a value of 1 is used for FIRST.

(2) If the (LAST - FIRST + 1: = LENGTH TO ADD TO M1) > (255 - M1(0): = SPACE IN M1), the segment to add is reduced via LAST = 255 - M1(0) + FIRST - 1.

(3) If LAST > M2(0), M2(0) is used for LAST.

(4) If FIRST > LAST, no action is performed.

Normal case:

(1) M1(0) is increased by LAST - FIRST + 1.

(2) M1(I) = M2(J) for J = FIRST, . . . , LAST, where I = M1(0) + 1 + J - FIRST.

B. The APPEND Procedure

This procedure, called by the statement

CALL APPEND(.M1,.M2);

adds the entire string M2 to the end of the string M1.

Special cases:

(1) M2 is void, no action is performed.

(2) M1 is full, no action is performed.

(3) Other special cases — see ADDSEG, as this procedure is equivalent to ADDSEG (.M1,.M2,1,255).

Normal case:

M1 is increased in length by the largest segment of M2 that will add onto M1.

C. The SEGMENT Procedure

This procedure, called by the statement

CALL SEGMENT (.M1,.M2,FIRST, LAST);

sets the string M1 equal to the selected substring from the string M2.

Special and normal cases:

The string M1 is first nulled, then the procedure ADDSEG is called via ADDSEG (.M1,.M2, FIRST, LAST).

D. The EQUATE Procedure

This procedure, called by the statement

CALL EQUATE (.M1,.M2);

sets the string M1 equal to the string M2.

Special and normal cases:

(1) This procedure is equivalent to CALL NULL (.M1); then CALL APPEND (.M1,.M2);

(2) See those procedures for special and normal cases.

E. The REPLACE Procedure

This procedure, called by the statement

CALL REPLACE (.M1,.M2, FIRST, LAST);

sets the substring of M1 defined by the values of FIRST and LAST to be equal to the characters M2(1) to M2(LAST - FIRST + 1). The string M1 ought to be of length at least LAST.

II. Length Manipulations

The length manipulation procedures are

NULL for making a string empty

TRUNCATE for truncating a string

A. The NULL Procedure

The procedure, called by the statement

CALL NULL (.M1);

sets the length of the string M1 to zero.

B. The TRUNCATE Procedure

This procedure, called by the statement

CALL TRUNCATE (.M1, LAST)

truncates the string M1 to length LAST, provided M1 is of length greater than LAST prior to the call.

III. Comparison

The procedure, COMPARE, is used to compare two strings. The procedure returns a value according to

0, string 1 less than string 2

1, string 1 = string 2

2, string 1 greater than string 2

The returned value can therefore be tested as a logical expression having value TRUE if the strings are equal and FALSE otherwise, or the value can be used in a relational expression or a DO-case construction. The expression to obtain the comparison value is

.COMPARE (.M1,.M2)

where M1 and M2 are the strings to be compared. If one string is shorter than the other, the shorter is considered to be rounded out with blanks for comparison purposes.

IV. Conversions

The procedures

BINTOHEX

BINTODEC

HEXTOBIN

DECTOBIN

convert binary values to their ASCII representations in either hexadecimal or decimal to binary.

The calling sequences are similar:

```
CALL BIN$TO$HEX (BIN$NO,.M);
CALL BIN$TO$DEC (BINSNO,.M);
Y = HEX$TO$BIN(.M);
Z = DEC$TO$BIN(.M);
```

For the latter two cases, the procedure actually produces an address value as its result. Note that the global variable, SYNTAX, is set according to

SYNTAX = 0 if no syntax errors

SYNTAX = 1 if a syntax error was discovered

The conversions that produce strings generate left-justified strings containing one space on the right-hand side.

Appendix D
Arithmetic Procedures

Because of the receiver/exciter control assembly's need for high precision, the software for fixed-length, variable-point operations has been implemented.

I. Fixed-Point Operations

These operations require definition of a global data item called FXPT$LENGTH, which describes the length of all the fixed point numbers in *bytes*. The internal structure of a fixed point number is a packed decimal digit string with two digits per data byte. The sign of a fixed point number is determined by the leftmost byte of the number. A zero here indicates a positive number, while any nonzero value indicates a negative number. Hence, the precision used by a program is 2*FXPT$LENGTH - 2 digits, because of the presence of the sign.

The calling sequence

```
CALL FXPTADD (.A,.B,.C);
```

calculates A = B + C,

```
CALL FXPTSUB (.A,.B,.C);
```

calculates A = B - C,

```
CALL FXPTMPY (.A,.B,.C);
```

calculates A = B*C. In all these circumstances, the array A must be distinguished from the arrays B and C. That is,

```
CALL FXPTADD (.A,.B,.A);
```

is illegal.