HP 4954A Protocol Analyzer

HP 18320A DataCommC Programming Language

Demonstration Guide

HEWLETT PACKARD

Manual Part Number: 18320-99703
Microfiche Part Number: 18320-98810

Printed in U.S.A. July, 1989
E0789

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

If your software application or hardware should fail, contact your local Hewlett-Packard Sales Office listed in the protocol analyzer operating manual.

Colorado Telecommunications Division
5070 Centennial Boulevard
Colorado Springs, CO, 80919-2497

Contents

1

Introduction

The programs presented in this guide are intended to demonstrate some of the many capabilities of the DataCommC application. Each of the programs are described in detail in Chapter 2 of this manual. The demo disc that is provided with this guide contains an executable file for each of the demo programs. In addition, the source code for the programs has been included to provide you with examples and ideas of how to write data communication programs. A hard copy of the source code for four of the demonstration programs is included in Appendix A because we feel that they are excellent examples of how to perform some rudimentary data communications tests with DataCommC.

Note Keys on the HP 4954A keyboard will be referred to as follows:
<KEY> used for softkeys
[KEY] used for keys other than softkeys

Here is a summary of each manual section.

Chapter 2 of this manual provides a brief explanation of how to run any of the demonstration programs and how to put any of the programs into the DataCommC program manager. This section also contains some general information about the demonstration programs.

Chapter 3 goes through each of the demonstration programs in detail. The only program that you will actually be asked to modify is the first program, START_C.

Appendix A contains the source code for four of the demonstration programs. These programs are well documented and have been provided as working examples of how programs are written with the DataCommC application.

2 Introduction

2

Getting Started

The DataCommC application needs to be loaded on the HP 4954A in order to run any of the following demo programs. If the application has not already been loaded, Appendix D of the "DataCommC User's Guide" contains a complete description of how to install DataCommC.

Loading the Demonstration Programs

To load the demonstration programs, the DataCommC programming language must first be installed. For details on how to install DataCommC, consult the DataCommC User's Guide. Once DataCommC has been loaded, return to the HP 4954A top level menu by pressing [Reset].

Currently there should be two applications named "Drivers" and "C_manager" in the application menu which need to be deleted. To do this, press <Other Choices>, <Applic Menu>, and then press <Delete Applic> twice for each application. When you install the demo programs, the "Drivers" and "C_manager" applications are automatically re-installed in the application menu. Press <Exit> to return to the top level menu.

To load the demonstration programs onto your hard disc, use the following procedure.

1. Place the Demonstration disc in the floppy disc drive and press <Other Choices>, <Mass Store>, <Change Directory>, <Select Unit>, <Flexible Disc>, <Execute>, and <Exit Operation> to view the files on the floppy disc.

2. Press <Load File>, <Select File>, and use the <Next File> key to highlight the "Install.Applicmodule" file. Press <Execute> to start the installation process.

3. It will take a few minutes to load all the demonstration programs and to reinstall DataCommC. Once the installation process is complete, remove the floppy disc from the drive and press [Return] to return to the top level menu.

Once the demonstration programs are installed, you can access the programs by using either method described below.

Running a Program

There are two basic ways to run the demonstration programs: through the DataCommC program manager or through the DataCommC Program Run Environment. These two methods are described below in detail.

Method 1

Once the demonstration programs have been loaded and the HP 4954A has returned to the top level menu, press <Other Choices>, <C Program Manager>. In the program manager, there will be an entry titled "DEMO PROGRAMS". To execute this program, use the arrow keys to highlight this entry and press [Return]. When you run this program, a screen will come up and ask you to press the softkey corresponding to the program you wish to run. From this screen, you can run any of the demonstration programs. If you wish to exit the demo shell, simply press [Reset].

Method 2

All of the files included on the demonstration disc are put on the hard disc in the "c:C/Demo/" directory when the demonstration disc is loaded. Therefore, you can also run any of the demonstration programs through the DataCommC Program Run Environment. Most of the files in the c:C/Demo/ directory will have either a .program or a .csource extension (e.g., START_C.csource, START_C.program). A file with the .csource extension contains the source code for a program and is the file the we can read and edit. A file with the .program extension contains executable code and is the code that can be executed from the DataCommC Program Run Environment.

To execute one of the demonstration programs from the DataCommC Program Run Environment, press <Other Choices>, <C Program Manager> from the HP 4954A top level menu. Next, press <Run Other Programs>. Enter the path, filename, and extension of the program to be run on the program file line (e.g., c:C/Demo/START_C.program). The file name that you enter on the program file line must have a .program extension. Now, simply press <Run Program> and the program will execute.

To stop any program while it is running, simply press [RESET] and the program will be halted.

Modifying a Program

All of the files contained on the demonstration disc, as well as those that are put in the c:C/Demos/ directory when the DataCommC application is installed, are write-protected so that they may not be directly modified. This has been done for your benefit. If you would like to modify the source code for any of the programs, you simply need to copy the source code into another file and modify the newly created file. See the HP 4954A Operating Manual for details on how to use mass store to move files. The original source code can either be copied from the floppy disc or from the hard disc. Therefore, if you make changes to the source code and you have problems running the program with the changes, you will still have a copy of the original program that will always run.

Inserting a Program Into the DataCommC Program Manager

To load any of the demonstration programs into the DataCommC program manager so that you can run it without first having to go through the demo program shell, press <Manage Programs> from the top level menu of the DataCommC program manager. The top of the screen should say "DataCommC Program Run Environment". Enter the path, filename, and extension of the file you wish to add to the program manager on the Program File line. (for example, c:C/Demo/START_C.program). Press <Add To Menu>. You will then be asked, "Is this the program to be added?" Press <Yes> to continue, or press <No> if you need to change the filename. After pressing <Yes>, enter the name of the program as you wish to have it appear in the program manager and press <Exit field>. Now, enter a description of the program (this field is optional) and press <Exit field>. If you have entered all of the information correctly, press <Yes> and the file will now be added to the DataCommC program manager. If you need to make changes to your entry, press <No>. If you no longer wish to add a file to the manager and wish to quit this function, press <Abort>.

Once a program has been added to the DataCommC program manager, you can run the program by using the four arrow keys to highlight the program and then pressing [Return].

Moving Demonstration Files to the Hard Disc

To move any of the files from the demonstration disc to the hard disc of the HP 4954A, see the HP 4954A Operating Manual section on how to use the mass store menu.

About the Demonstration Programs

The only program that you will be instructed to modify is START_C. You will first be asked to run the program so that you can see what it does. Then, you will be instructed on how to modify the code. When modifying the code, you will be purposely instructed to insert an error in the program. When the program is compiled, you will then be told how to edit the error-log, fix the error, and then to compile, link, and run the program again. The purpose of this exercise is to provide step-by-step instructions on how to edit, compile, and link a program as well as how to edit the error-log.

For demonstration programs other than START_C, you will simply be told how to run the program and be given some details on what the program is doing.

Read_me File

In addition to all of the program and source code files contained on the demonstration disc, there is also a file called Read_me.text. This file contains information about any changes that have been made to the demo programs between the time this document was published and the time the demonstration disc was produced. Read_me.text will also contain a description of any demonstration programs that are added to the demonstration disc but were not created in time to be documented in this guide. To read this text file, you simply need to edit it. To do this, press <Develop Programs> from the DataCommC program manager menu, enter "c:C/Demo/Read_me.text" (or a:Read_me.text if you are reading the file off of the demonstration disc) on the work file line, and press <Edit>. You can then scroll through and read the Read_me.text file.

3

Demonstration Programs

START_C

Demo Disc File: **START_C**

Purpose: The purpose of this demo is to show the user how to edit, compile, link and run a program. Also, this demo will give details on how to edit the error-log and will show some of the different character attributes available with DataCommC.

Configuration: Since this program does not perform any data communications tests, there is no need to set the protocol, datacode, etc., in the setup menu.

Demo Procedure: It is assumed that the DataCommC application has already been loaded before starting this demo and that you are in the DataCommC program manager. If DataCommC has not been loaded, Appendix D of the DataCommC User's Guide gives a detailed explanation on how to load the application and enter the program manager.

Run the START_C program. Chapter 2 gives a detailed description of how to run any of the demonstration programs. In a couple of seconds, the screen shown in Figure 3-1 will appear and instruct you to enter a number depending on whether you want to view one of the printing attributes of DataCommC or if you want to exit the demo.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

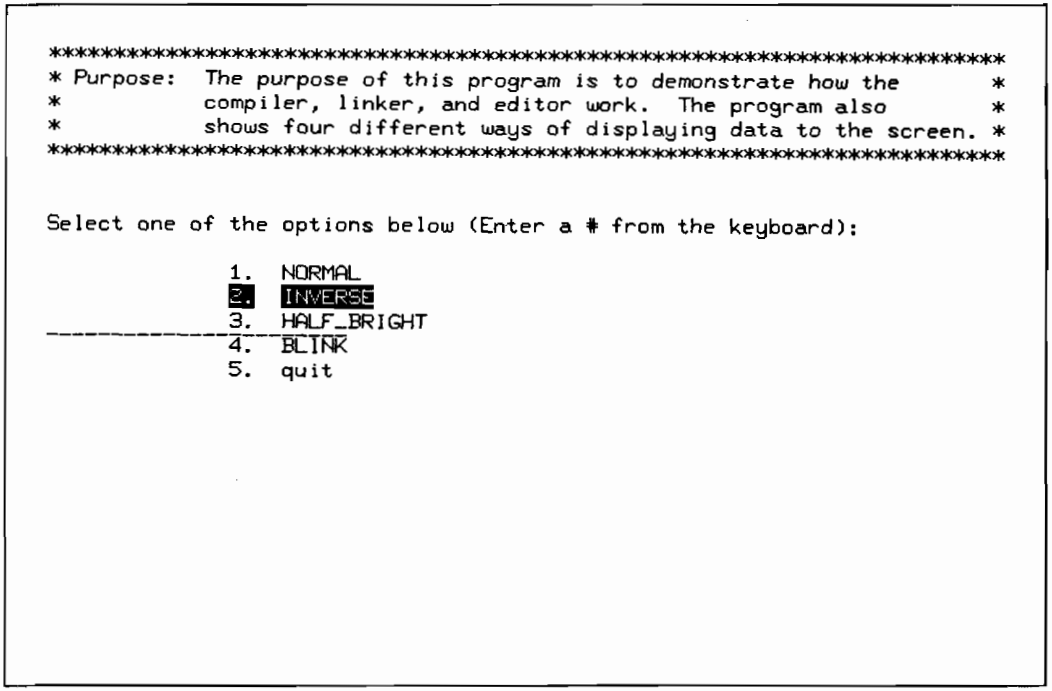

Figure 3-1.

The purpose of the rest of this demo is to give a detailed description of how a program progresses from an idea on a piece of paper to a file that can be run by the protocol analyzer.

Exit the program currently running (if you haven't already) by entering a "5" and any other character from the keyboard.

Editing and Saving a Program

First, we are going to edit the program we just ran. The source code for a program (the source code is the readable code that we create) should be stored in a file with an extension of .csource (for example: test.csource). When we wish to change a program, this file must be edited and then recompiled and relinked before it can be run with the changes we make.

From the DataCommC Program Manager press <Develop Programs> (you may have to press this key twice) to load the development environment. Type in "C:C/Demo/START_C.csource" and press <Edit>. The file now on your screen is the

source code to the program we ran earlier. You can use the up and down arrow keys or the [Roll Up], [Roll Down], [Prev Page], and [Next Page] keys to scroll through the code.

Note To create a new program, you would develop it in the editor in the same manner stated above. Simply type in the path name (e.g., C:C/User/) and the file name with a .csource extention (e.g., new_program.csource) and press <Edit>. You will be placed in an empty file buffer where you can then type in your program.

Use the up and down arrow keys to place the cursor at line 43 (the control line at the top of the screen tells you the line that the cursor is presently on). Use the right arrow key to move the cursor into the postion just to the right of the semicolon on line 43, and press the [Back Space] key. We are purposely putting an error in the source code so that when the code is compiled an error will be found. This will give you the opportunity to learn how to interpret and edit the error log.

If you now try to save the file START_C.csource with the changes you have made, an error will appear at the top of the screen indicating that the HP 4954A could not create a backup file. On page 4 there is a note explaining that all of the demos have been write protected and cannot be directly modified. To save the changes you have made, you have to write the source code to a new file. To do this, press <Escape> "F" (for file) "W" (for write), then type in "c:C/Demo/START_C1.csource" and press <Execute>. The source code in the editor is placed in the file START_C1.csource.

The editor will still appear on the screen and you have to press <Escape> "F" "Q" (for quit) "Y" (for yes) to return to the development environment.

Compiling a Program

The work file line in the development environment should currently read "c:C/Demo/START_C.csource". When we finished editing the START_C file we placed the edited code in a file called START_C1, thus we now have to compile the START_C1 file. Enter "c:C/Demo/START_C1.csource" on the work file line and press <Compile>. After a few seconds, the screen shown in figure 3-2 appears.

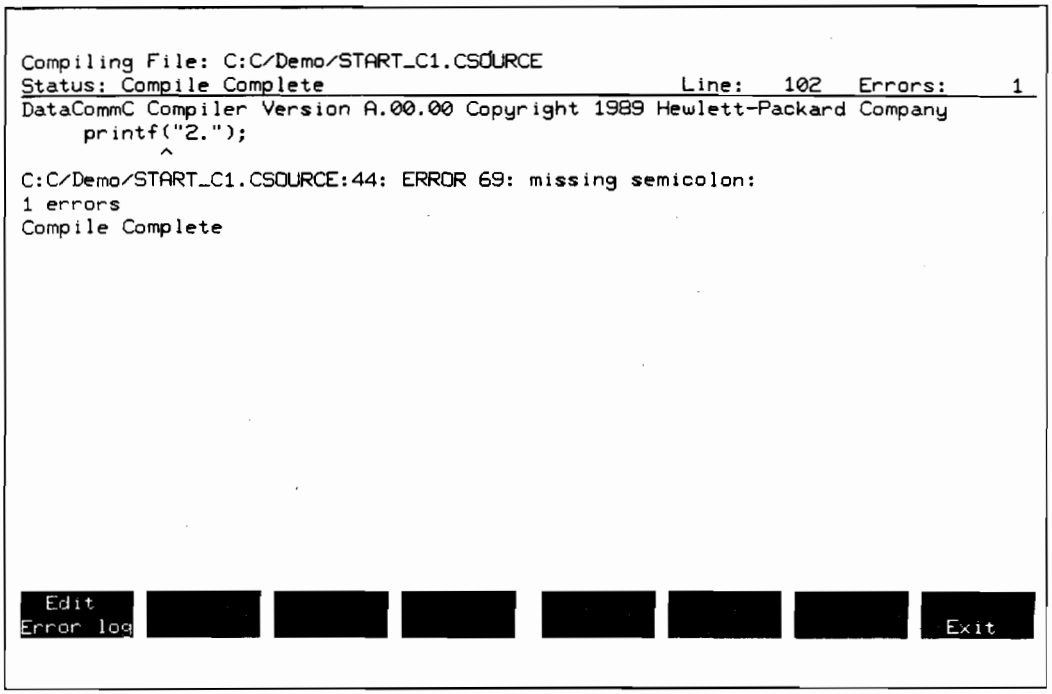

Figure 3-2.

In the upper right hand corner you can see that there is one error in the program. Below this, there is an error message that tells you the line of the error (whenever there is a semicolon missing, a C compiler will flag the next line containing text after the actual line with the error) and what type of error has occured.

Editing the Error Log

In order to view the error-log, which currently contains a list of all the errors found by the compiler and the source code that contains the error, press <Edit Error log> and the screen shown in Figure 3-3 will appear.

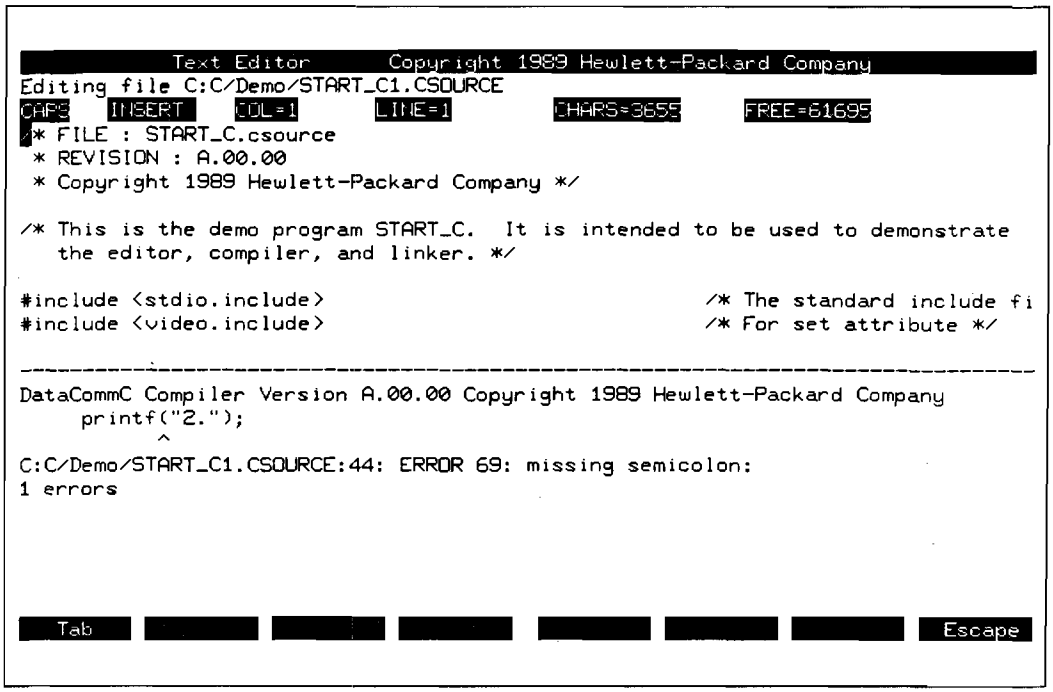

Figure 3-3.

The error log will appear in the bottom half of the screen and the source code will appear in the top half of the screen. Scroll the cursor down to line 43 of the program and reinsert the semicolon at the end of the line.

Currently when you move the cursor, you can scroll through the START_C1 program but you cannot access the error log. Although the editor lets you view two files, only one file can be active at any given time. In order to make the error-log the active file, press <Escape>, "W" (for window), "O" (for other) and the cursor will now be in the error-log. Since the file we compiled only had one error, we can already view all the errors. If there had been several errors, however, you could now scroll through the error-log to view any errors not currently displayed. Now, return to the START_C1 source code by pressing <Escape>, "W", "O".

Press <Escape>, "F", "S" to save the corrected source code, then press <Escape>, "F", "Q" (for quit) to exit the error log.

Now try compiling START_C1 again by pressing <Compile>. This time you should not get any errors. Press <Exit> to return to the development environment.

Linking the Program

After compiling the program, you will then need to link the program. The compiler creates a file that has the same name as the source code file, only the extension is .object - this is the file you need to link. Move the cursor over the first c in "csource" and type "object" in as the new extension. Now press <Link>. No errors should be found by the linker.

Note Instead of compiling and linking a file in two steps, we could have simply used the Make function. When in the development environment, the name of the file with the .csource extension should be on the work file line. You can then press <Make> and the file will be both compiled and linked plus a .program file will be created (if the file is successfully compiled and linked) that can be executed through the runtime environment. For more details on the make utility, see Chapter 8 of the DataCommC User's Guide.

Running the Program

The linker creates another file with the same name as the source code, but this time the extension is .program. This file contains executable code that can be run from the run time environment. To run START_C1, press <Exit> twice to return to the DataCommC Program Manager then press <Run Other Programs>. Type in "C:C/Demo/START_C1.program" and press <Run Program>. You will now see the same program that you saw running earlier.

Appendix A contains a listing of the program START_C1 to give you a view of a DataCommC program.

CUSTOM_COUNT

Demo Disc File: **CUSTOM_COU**

Purpose: This program is an example of how a custom display can be created, how to access the real-time clock, and how the computational power of the C language can assist you in terms of your data communications needs. This demo counts good, bad, and abort frame check sequences (FCSs) and creates a bar graph display of each of the different types of FCSs. CUSTOM_COUNT can be run from an active line, or can be run on a file that contains previously captured data.

Configuration: The parameters used by CUSTOM_COUNT are not actuallly specified in the program through the use of the set-protocol() command; thus the parameters set in the setup menu will be used. Only bit oriented protocols (BOPs) can be used with this program, as only BOPs use frame check sequences.

Demo Procedure: Run the CUSTOM_COU program. Chapter 2 gives a detailed description of how to run any of the demonstration programs. The first screen that appears will ask you if you want to use the line or a buffer data file as the data source. Follow one of the two options below depending on what you would like your data source to be.

Option #1

If you wish to run from line, press <On Line> and the program will begin counting the frame check sequences of the frames it receives from the line.

Option #2

If you wish to run from a buffer data file press <Buffer Data>. You will then be prompted for the device (hard or floppy disc), path, filename, and file type of the file you wish to run from. A default file name of "c:C/Demo/HDLC_data.menus&data" will already be filled in for you. This file contains frames with all three different types of frame check sequences: good, bad and abort. To select the default file, simply press [Return]. To enter a different file name, hit any key (except return) and the default file name will disappear; you can then enter the new file.

If this is the first time you are running this program, you may want to run from the default file that appears when you select <Buffer Data> so that you can see how the program should run.

As CUSTOM_COU runs, several things will happen on the display:

- A counter for each type of FCS will be updated as the FCSs are counted.
- The percentage of each type of frame check sequence, relative to the total FCS count, will be updated and displayed. Note that due to rounding errors, this number is accurate to plus or minus .5%.
- A bar chart will be created to represent each of the three types of frame check sequences. The bar chart will expand by one unit for every 25 frame check sequences.
- Three timing measurements will be displayed: the time the program begins counting FCSs, the time stamp of the last frame received, and the time stamp of the last bad FCS occurence.

When CUSTOM_COU is running from buffer data, the program will stop execution when it gets to the end of the buffer data file and will display "The run is complete" at the bottom of the screen. If you wish to halt execution of the program before the entire buffer data file has been read, press <Stop Run>. When the program finishes counting FCSs, the statistics gathered will still be shown on the screen so that they can be analyzed. When running from line, the only way to stop execution is to press <Stop Run>. The program will not stop on its own since it has no idea if there will still be more data coming across the line or not.

If you specify buffer data as the data source, and use the default file to run from, the screen shown in Figure 3-4 will appear when the end of the file is encountered.

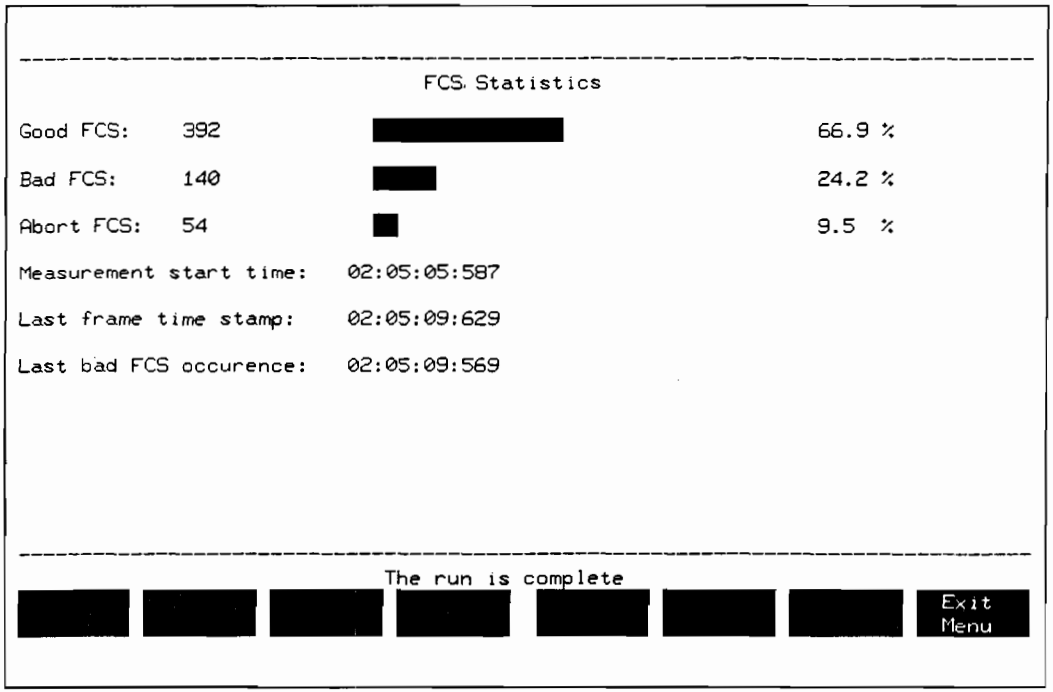

Figure 3-4.

After the program has stopped collecting data and you are done examining the statistics display, press <Exit Menu> to return to the start of the demo program. At this point, you can either run the program again or press <Exit> which will return you to either the DataCommC program manager or the DataCommC program run environment (depending on where you initially started the program).

MONITOR

Demo Disc File: **MONITOR**

Purpose: The purpose of this program is to simply monitor data on the line.

Configuration: The parameters (protocol, data code, parity, etc.) used by MONITOR are not actually specified in the program through the use of the set_protocol( ) command; thus the parameters set in the setup menu will be used.

Demo Procedure: Since this program simply monitors line data, you need to have an active line to monitor data on before running this demonstration. Otherwise, you will see a blank display.

Run the MONITOR program. Chapter 2 gives a detailed description of how to run any of the demonstration programs. All of the data flowing across the line will be displayed on the screen. DTE data will appear on the left half of the screen, and DCE data will appear on the right half of the screen. The display format being used is the standard run-time display of DataCommC. While the display is running, you can press <Freeze Display> to freeze the display. This allows you to look at a portion of the data that has already been captured, while the analyzer continues to monitor the data line. While the screen is frozen, you can use the up and down arrow keys to scroll through the most recently captured events. To continue displaying the data as it is captured, press <Continue>.

While the program is running, you can change the way the data is displayed by pressing <Display Format>. (This softkey is available while the screen is frozen, but changing the display format will not affect the data being displayed until you press <Continue>.) This feature allows you to display as much or as little information as you desire, and also lets you adjust the protocol and datacode while the display is running. For more details on the run-time display, see Chapter 9 of the DataCommC User's Guide.

To exit this program, either press [Reset], or press <Stop Run>.

The source code for this program can be found in Appendix A. This program has been provided so that the user would have a "canned" program to monitor the line and would not have to write the program himself. The source code for this program can be used as an example of how to invoke the run-time display.

UNIX_SHELL

Demo Disc File: **UNIX_SHELL**

Purpose: The purpose of this demo is to demonstrate the power and flexibility of the C language that has been implemented on HP 4954A. This program performs several standard UNIX-like commands, however it is by no means a complete UNIX (R) operating system. This program does not perform any data communications functions.

Configuration: Since this program does not perform any data communication tests, there is no need to set the protocol, datacode, etc.

Demo Procedure: Run the UNIX_SHELL program. Chapter 2 gives a detailed description of how to run any of the demonstration programs. The first screen of the UNIX shell is shown on Figure 3-5.

UNIX is a U.S. registered trademark of AT&T in the U.S.A. and in other countries.

```
UNIX_SHELL                                          Version 1.0
The following commands are supported by this shell:
    ls - lists the files in the current directory
    cp - copy files from one directory to another; filename remains unchanged
           ex. cp filename1.extension c:C/User/*.*   NOTE:  *.* is mandatory
   cat - displays the contents of a file
           ex. cat filename.extension
    cd - change to the indicated directory
           ex. cd c:C/User/
           ex. cd ..  (goes back one level in the directory tree)
   cls - clears the display
     ? - displays this help screen

   set - sets variables in the environment
           ex.  set a c:C/User/ ;sets a to the given directory
                ls *a           ;lists the files in the c:C/User/directory
                set             ;displays environment variables currently set
                set a           ;removes a from the environment
WILDCARD MATCHING - applies to ls and cp funtions only
       ex.  ls b*.* lists all files in the current directory beginning with b
CONTROL CHARACTERS:  [CNTL]-d exits the shell
                     [CNTL]-l retreives the last command
                     [CNTL]-s freezes the display (used with ls and cat)
                     [CNTL]-q resumes the display after a [CNTL]-s is used
c:C/Demo/>>
```

Figure 3-5.

The following functions are supported by the UNIX shell:

Commands:

ls - Lists out the files in the current directory.

cp - Used to copy the contents of a file from one directory to another directory.
example: cp filename1.csource c:C/User/*.* (the *.* is mandatory)

cat - Displays the contents of a file to the screen.
example: cat filename1.csource

cd - Used to move into a different directory.
example: cd c:C/User/
example: cd .. (moves you back one level in the directory tree)

cls - Clears the current display.

? - Displays the help screen shown in Figure 3-5.

Control Sequences:

[CNTL]-[D] - Leaves the UNIX shell.

[CNTL]-[L] - Retrieves the last command.

[CNTL]-[S] - Used with the ls and cat commands to freeze the display.

[CNTL]-[Q] - Used with the ls and cat commands after cntl-S is used to resume scrolling a listing.

Wildcard Matching:

Wildcard matching can be used with the ls and the cp commands.

example: ls b*.* lists out all files beginning with b in the current directory.

example: cp *.csource c:C/User/ copies all csource files from the current directory to the c:C/User/ directory.

Environment Variables:

The command "set" is used to set variables in the environment.

example: set a c:C/User/ This command sets a to be equivalent to c:C/User/.

ls $a Lists out all of the files in the c:C/User/ directory.

set Displays all environment variables that have currently been set.

set a Removes "a" from the environment.

To exit the UNIX_SHELL program at any point, either press [CNTL]-[D] or press [Reset].

GRAPHICS

Demo Disc File: **GRAPHICS**

Purpose: The purpose of this program is to demostrate some of the extraordinary abilities of DataCommC: character cell graphics, access to a real-time clock, multiple processes, etc.

Configuration: Since this program does not perform any data communication tests, there is no need to set the protocol, datacode, etc.

Demo Procedure: Run the GRAPHICS program. Chapter 2 gives a detailed description of how to run any of the demonstration programs. This program will cycle through several different displays. Some of the displays are simply text that describes the DataCommC features and applications. For the most part, however, the screens show examples of the features of DataCommC through the use of intricate character cell graphics.

The program will cycle through all of its displays without any user input. If, however, at any point in the program you wish to pass by the current screen and proceed to the next display, simply press the space bar. Pressing the space bar will clear the current display, and will bring up the next screen of the program. In this way, you can speed up the rate at which screens are shown. To exit this program, press [Reset].

Note There are two files on the solutions disc, screenio.csource and figures.csource, which provide access to the figures generated in the GRAPHICS demo. The file screenio.csource contains routines which actually generate the figures, whereas figures.csource provides examples of how to call the routines.

TRANSMIT

Demo Disc File: **TRANSMIT**

Purpose: The purpose of this program is simply to send data to the line.

Configuration: This program sends out DCE data that utilizes the HDLC protocol, coded in ASCII8, with CRC_CCITT error checking. These parameters are set through the set_protocol command.

Demo Procedure: Run the TRANSMIT program. Chapter 2 gives a detailed description of how to run any of the demonstration programs. This program sends 100 frames to the line. All of the frames are INFO type HDLC frames. Each time a frame is sent to the line, a message is also printed to the screen to tell you the number of the frame that was sent out. The address on each frame is incremented by one each time a frame is sent. Thus the address of the first frame is one, and the address on the last frame is 100.

Once all of the frames have been sent and the messages stop scrolling up the screen, you can press any key to leave the program. While the program was running, we saw the messages being sent to the screen, but we did not see the data being sent to the line because we did not activate the run-time display. The data that was sent to the line is being stored in the event buffer. To view the event buffer, press <Buffer Manager> from the DataCommC program manager. Use the arrow keys to highlight the decode you wish to use, and press <Examine Buffer>. You can now look at the data that was sent to the line. The softkeys allow you to display the data in several different formats. To see the real-time stamp and event number of each event (for the case of bit oriented protocols, an event is a frame or a lead change), press <Diplay Format>, <Event/Timestamp>. To view the current status of the leads, press <Lead Status>. Leads shown in half-bright are off, whereas those shown in full-bright are on. Currently, the data is being decoded as if it was LAPB data. To change the protocol to HDLC, press <BOPS Layer 2>, <LAPB Decode>, <LAP Decode>. Your display should look somewhat like figure 3-6.

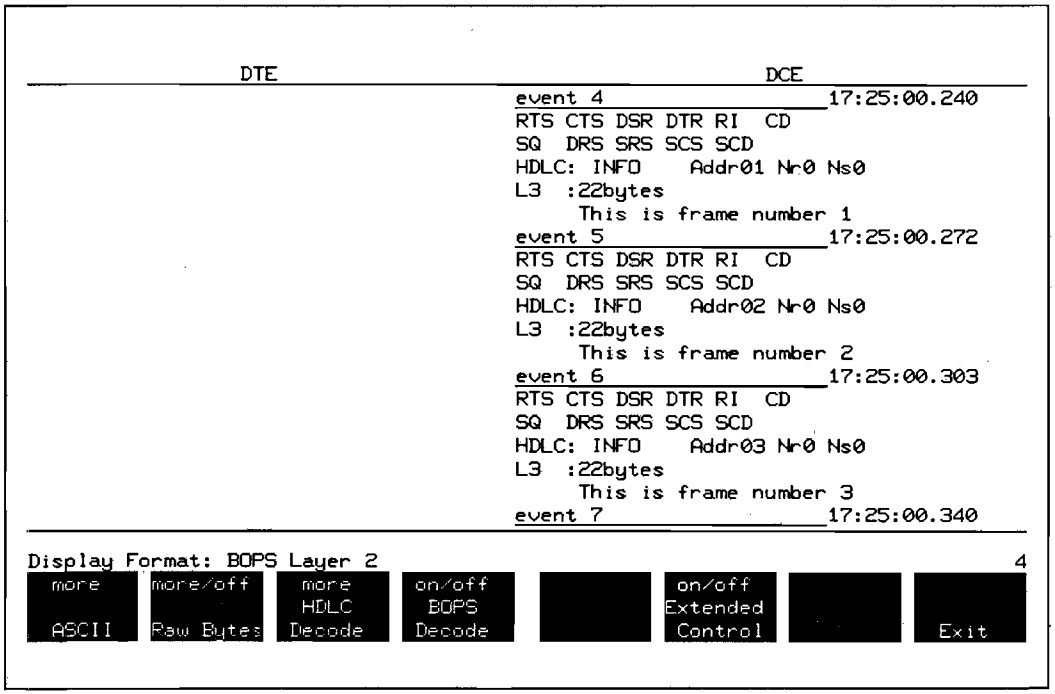

Figure 3-6.

For more details on the function of the examine events decode, see Chapter 10 of the DataCommC User's Guide. To exit the examine events decode, press <Exit> until you return to the DataCommC program manager.

A

Source Code

This appendix contains a listing of the source code to four of the demonstration programs: START_C, CUSTOM_COUNT, MONITOR, and TRANSMIT.

START_C

```
/* This is the demo program START_C.  It is intended to be used to demonstrate
   the editor, compiler, and linker. */

#include <stdio.include>                              /* The standard include file */
#include <video.include>                              /* For set attribute */

#define  ROW                16                       /* Where to put main code */
#define  END_ROW            25                       /* The last row to wipe */
#define  COLUMN             1                        /* Where to put main code */

/* What gets displayed as the output */

#define  JINGLE             "Hewlett-Packard is YOUR network solution.\n"

void     set_attribute();
void     set_cursor();
void     wipe();

main()

{

     char character;                                  /* Holds the selection */

     /* A description of this program is put to the screen */

     printf("*****************************************************************************\n");
     printf("* Purpose:  The purpose of this program is to demonstrate how the       *\n");
```

```
printf("*          compiler, linker, and editor work.  The program also        *\n");
printf("*          shows four different ways of displaying data to the screen. *\n");
printf("*****************************************************************************\n");
printf("\n");


printf("\nSelect one of the options below (Enter a # from the keyboard):\n");
printf("\n");
printf("              1.  NORMAL\n");
printf("              ");
set_attribute(INVERSE);
printf("2.");
set_attribute(NORMAL);
printf("  ");
set_attribute(INVERSE);
printf("INVERSE\n");
set_attribute(HALF_BRIGHT);
printf("              3.  HALF_BRIGHT\n");
set_attribute(BLINK);
printf("              4.  BLINK\n");
set_attribute(NORMAL);
printf("              5.  quit\n");

/* The main loop.  It loops forever until option five is selected. */

set_rows(ROW,END_ROW);

while (TRUE) {                                  /* Always loop */

     character = getch();                       /* Get the input */

     if (character == '5')                      /* Handle quit */
          break;                                /* Break from while loop */

     if (character == '4')
          set_attribute(BLINK);

     else
          if (character == '3')
               set_attribute(HALF_BRIGHT);

     else
```

```
            if (character == '2')
                set_attribute(INVERSE);

        else
            if (character == '1')
                set_attribute(NORMAL);

        else {
            printf("I am defaulting to NORMAL.\n");
            set_attribute(NORMAL);
        }

        printf(JINGLE);                          /* Print out the saying */
    }

    /* This is code to quit the program gracefully.  It also demonstrates the
       usage of getch() to keep the screen around until the user is ready
       to leave */

    wipe(ROW,END_ROW);
    set_cursor((ROW + 1),COLUMN);
    set_attribute(NORMAL);                    /* Make sure the attribute is normal */
    printf("Bye, Bye!\n");                      /* Words to the screen */
    printf("Please hit any character to return to the C manager menu.\n");
    getch();                                    /* Get any character */
}
```

CUSTOM_COUNT

```
/* This is a demo program and it is intended to show how to do some level 2
   statistics.  It works either on-line or from disk.  This program
   demonstrates how to read events and release events.  It requires that the
   datacomm configurations be set properly in the setup menu, simulate menu,
   and run menu before execution of this program. */

#include <video.h>                              /* The video attributes */
#include <stdio.h>                              /* The standard i/o */
#include <conio.h>                              /* The softkey definitions */
#include <retval.h>                             /* The return values */
#include <dlib.h>                               /* Datacomm definitions */
#include <message.h>                            /* All the messaging things */
#include <setjmp.h>                             /* Used for setjmp and longjmp */

#define  INPUT_STRING          0x32             /* Length of input string */
#define  FIRST_ROW             0x01             /* The first row of the screen */
#define  FIRST_COLUMN          0x01             /* The first screen column */
#define  LAST_ROW              0x19             /* The last row of the screen */
#define  LONGJMP_VALUE         0x01             /* The val parameter for longjmp */
#define  WAIT_TIME             1l               /* How long to wait for a message */
#define  PERCENT               100              /* Make things into a percent */
#define  X_NUMBER              0x19             /* How often to display x's */

#ifdef   BACKSPACE
#undef   BACKSPACE
#endif

#define  BACKSPACE             0x15             /* The backspace character */

#ifdef   LINEFEED
#undef   LINEFEED
#endif

#define  LINEFEED              0x0a             /* The linefeed character */

#ifdef   WAIT
#undef   WAIT                                   /* Undefine NO_WAIT */
#endif
```

```
#define  WAIT                1                    /* NO_WAIT is zero for read message */

void     put_all_sks();
void     center();
void     set_attribute();
void     wipe();
void     set_cursor();
void     putch();
void     decode();
void     get_cursor();
void     cursor_on();
void     cursor_off();

jmp_buf  envbuf;                                  /* Used by setjmp to save registers */

main()

{

     int  soft_key_input;                           /* Which softkey has been input */
     int  check;                                    /* Used to check return values */
     char filename[INPUT_STRING + 1];               /* Used to hold the filename */
     int  counter;                                  /* A loop counter */
     int  row, column;                 /* The rows and columns of a get_cursor */

     setjmp(envbuf);                                /* Store the registers */

     while (TRUE) {
          cursor_off();
          wipe(FIRST_ROW,(LAST_ROW - 0x3));
          set_attribute(NORMAL);
          center("Please strike a softkey for the data source",FIRST_ROW);
          put_all_sks("On 'Line","Buffer 'Data","","","","","","Exit",INVERSE);
          soft_key_input = getch();
          wipe((LAST_ROW - 0x2),(LAST_ROW - 0x2));    /* Wipe yelp line when there is a
                                                                        character */

          switch (soft_key_input) {

               case SK1 : check = set_data_source(LINE_RUN,"Bogus string");
                          if (check != SUCCESSFUL) {
```

```
                    center("Problem setting data source to be on line",/
                                                    (LAST_ROW - 0x2));
                    longjmp(envbuf,LONGJMP_VALUE);
               }
               decode();
               break;

case SK2 : put_all_sks("","","","","","","","",INVERSE);
     set_cursor((LAST_ROW - 0x0a),FIRST_COLUMN);
     cursor_on();
     printf("To use the default file specified below, press [Return].\n");
     printf("Or, hit any key then type in the device, path, filename,\n");
     printf("and file type of the data file you wish to use:\n\n");
     strcpy(filename,"c:C/Demo/HDLC_data.menus&data");
     printf("%s",filename);
     if ((getch()) != LINEFEED) {
          set_cursor((LAST_ROW - 0x09),FIRST_COLUMN);
          wipe((LAST_ROW - 0x0a),(LAST_ROW - 0x06));
          printf("Please enter the device, path, filename, and file type\n");
          printf("of the file you wish to use:\n");
          for (counter = NULL; counter < (INPUT_STRING + 1); counter++)
               filename[counter] = NULL;
          for (counter = NULL; counter < INPUT_STRING; counter++) {
               filename[counter] = (char) getch();
               while (filename[counter] == BACKSPACE) {
                    if (counter == NULL) {
                         filename[NULL] = NULL;
                         counter--;
                    }
                    else {
                         get_cursor(&row,&column);
                         set_cursor(row,(column - 1));
                         putch(' ');
                         set_cursor(row,(column - 1));
                         filename[counter] = NULL;
                         counter--;
                         filename[counter] = (char) getch();
                    }
               }
               if (filename[counter] == LINEFEED) {
                    filename[counter] = NULL;
                    break;
```

```
                    }
                    if (counter >= NULL)
                        putch(filename[counter]);
                }
            }
            check = set_data_source(DISC_RUN,filename);
            if (check != SUCCESSFUL) {
              center("Prob. setting data source to be buf data",(LAST_ROW - 0x2));
              longjmp(envbuf,LONGJMP_VALUE);
            }
            decode();
            break;

        case SK8 : goto QUIT;
                   break;

        default : break;
      }                                          /* End of switch */
    }                                            /* End of while */
QUIT:
;
}                                                /* End of main */

void decode()

{

    int         key_input;                       /* Hold softkey inputs */
    MESSAGE     message;                         /* Storage for a message */
    FRAME_EVENT *frame_event;              /* Used to easily access a frame event */
    long        good_fcs = 0;                    /* Holds the good fcs's */
    long        abort_fcs = 0;                   /* Holds the abort fcs's */
    long        bad_fcs = 0;                     /* Holds the bad fcs's */
    int         check;                           /* Used to check return values */
    long        total_frames = 0;               /* Stores the total number of frames */
    int         start_flag = 0;                  /* Used to flag the first frame */
    int         good_column = 0x1d;              /* Used for putting up X's */
    int         bad_column = 0x1d;               /* Used for putting up X's */
    int         abort_column = 0x1d;             /* Used for putting up X's */

    cursor_off();
    wipe(FIRST_ROW,LAST_ROW);
```

```
put_all_sks("","","","","","","","Stop 'Run",INVERSE);
center("-------------------------------------------------------",FIRST_ROW);
center("-------------------------------------------------------",(LAST_ROW - 0x3));
center("FCS Statistics",(FIRST_ROW + 0x1));
set_cursor((FIRST_ROW + 0x3),FIRST_COLUMN);
printf("Good FCS:\n");
printf("\n");
printf("Bad FCS:\n");
printf("\n");
printf("Abort FCS:\n");
printf("\n");
printf("Measurement start time:\n");
printf("\n");
printf("Last frame time stamp:\n");
printf("\n");
printf("Last bad FCS occurence:\n");

check = start_data();
if (check != SUCCESSFUL)
   switch (check) {
      case ERROR_1 : center("Error opening the data file",(LAST_ROW - 0x2));
                     goto QUITWHILE1;
      case ERROR_2 : center("Unrecognized front end data source",(LAST_ROW - 0x2));
                     goto QUITWHILE1;
      case ERROR_3 : center("Front end is already running",(LAST_ROW - 0x2));
                     goto QUITWHILE1;
      case ERROR_9 : center("There is no pod attached",(LAST_ROW - 0x2));
                     goto QUITWHILE1;
      case ERROR_10 : center("No comm. with front end hardware",(LAST_ROW - 0x2));
                     goto QUITWHILE1;
      default : center("Start_data() returned something unexpected",(LAST_ROW - 0x2));
                     goto QUITWHILE1;
    }

while (TRUE) {

    if (is_key_avail()) {
          key_input = getch();
          if (key_input == SK8)
               goto QUITWHILE;
    }
```

```
check = read_message(&message,WAIT,WAIT_TIME);
switch (check) {

  case SUCCESSFUL : if (message.subtype == END_OF_DISC_RUN)
          goto QUITWHILE;

       else
          if ((message.subtype == DTE_FRAME) || (message.subtype == DCE_FRAME))
{

            frame_event = message.body.event.event_ptr;

            if (!start_flag) {
                start_flag++;
                set_cursor((FIRST_ROW + 0x9),(FIRST_COLUMN + 0x1a));
                printf("%02d:%02d:%02d:%03u",frame_event->hour,
     frame_event->minute,frame_event->second,frame_event->msec);
            }

            set_cursor((FIRST_ROW + 0xb),(FIRST_COLUMN + 0x1a));
            printf("%02d:%02d:%02d:%03u",frame_event->hour,frame_event->minute,
                    frame_event->second,frame_event->msec);

            total_frames++;

            if (frame_event->fcs == GOOD_FCS) {
               good_fcs++;

               if (((good_fcs % X_NUMBER) == NULL) && (good_fcs != NULL)) {

                  if (good_column < 0x33) {
                      set_cursor((FIRST_ROW + 0x3),good_column++);
                      set_attribute(SPECIAL);
                      printf("%c",0xf1);
                  }
                  else
                      if (good_column == 0x33)
                        center("Bar chart no longer valid - screen too small/
                                          - percentages still valid",23);
               }
```

```
            set_cursor((FIRST_ROW + 0x3),(FIRST_COLUMN + 0xd));
            printf("%ld",good_fcs);
            set_cursor((FIRST_ROW + 0x3),(FIRST_COLUMN + 0x3f));
            printf("%-5.1f%%",((((double) good_fcs) / (double) total_frames)
                                        * (double) PERCENT));
        }
        else
            if (frame_event->fcs == BAD_FCS) {
               bad_fcs++;
               if (((bad_fcs % X_NUMBER) == NULL) && (bad_fcs != NULL)) {
                  if (bad_column < 0x33) {
                      set_cursor((FIRST_ROW + 0x5),bad_column++);
                      set_attribute(SPECIAL);
                      printf("%c",0xf1);
               }
               else
                     if (bad_column == 0x33)
                       center("Bar chart no longer valid - screen too small/
                                    - percentages still valid",23);
        }

        set_cursor((FIRST_ROW + 0x5),(FIRST_COLUMN + 0xd));
        printf("%ld",bad_fcs);
        set_cursor((FIRST_ROW + 0xd),(FIRST_COLUMN + 0x1a));
        printf("%02d:%02d:%02d:%03u",frame_event->hour,frame_event->minute,
                        frame_event->second,frame_event->msec);
        set_cursor((FIRST_ROW + 0x5),(FIRST_COLUMN + 0x3f));
        printf("%-5.1f%%",((((double) bad_fcs) / (double) total_frames) *
                                              (double) PERCENT));
    }
    else
        if (frame_event->fcs == ABORT_FCS) {
           abort_fcs++;
           if (((abort_fcs % X_NUMBER) == NULL) && (abort_fcs != NULL)) {
              if (abort_column < 0x33) {
                 set_cursor((FIRST_ROW + 0x7),abort_column++);
                 set_attribute(SPECIAL);
                 printf("%c",0xf1);
              }
              else
                 if (abort_column == 0x33)
                    center("Bar chart no longer valid - screen too small/
```

```
                                        - percentages still valid",23);
                    }

                    set_cursor((FIRST_ROW + 0x7),(FIRST_COLUMN + 0xd));
                    printf("%ld",abort_fcs);
                    set_cursor((FIRST_ROW + 0x7),(FIRST_COLUMN + 0x3f));
                    printf("%-5.1f%%",((((double) abort_fcs) / (double) total_frames)
                                                    * (double) PERCENT));
               }
          }

          if ((message.subtype == DTE_FRAME) || (message.subtype == DCE_FRAME) ||
                                        (message.subtype == LEAD_CHANGE)) {
               check = release_event(message.body.event.event_ptr);
               if (check != SUCCESSFUL) {
                    center("Release_event() could not find event",(LAST_ROW - 0x2));
                    goto QUITWHILE1;
               }
          }

          break;

     case ERROR_1 : break;

     default : center("Read_message() returned something unexpected",/
                                                    (LAST_ROW - 0x2));
          goto QUITWHILE1;
     }
}

QUITWHILE:
     wipe((LAST_ROW - 0x2),(LAST_ROW - 0x2));
     center("The run is complete",(LAST_ROW - 0x2));

QUITWHILE1:

     stop_data();

     wipe((LAST_ROW - 0x1),LAST_ROW);
     put_all_sks("","","","","","","","","Exit 'Menu",INVERSE);

     while(TRUE)
```

```
        if (is_key_avail()) {
             wipe((LAST_ROW - 0x2),(LAST_ROW - 0x2));
             key_input = getch();
             if (key_input == SK8)
                  return();
        }
}
```

MONITOR

```
/* This program is intended to be a simple monitor program that demonstrates
   the run-time display.  The 4954 configurations must be set using the
   setup menus, the simulate menu, and the run menu.  This program is intended
   to run on-line. */

#include    <message.h>                          /* The messaging definitions */
#include    <dlib.h>                             /* Datacomm definitions */
#include    <stdio.h>


/*****************************************************************************
 * This is the main body of code.                                            *
 *****************************************************************************/

main()

{

     long    disp_QID;                        /* This is used as a dummy parameter */
     MESSAGE message;                           /* Declares storage for a message */
     int     check;                             /* Checks return values */


  set_data_source(LINE_RUN,NULL); /* set the data source to the line /*

  set_channelconfig(DCE,RX); /* set the DCE channel to receive */
  set_channelconfig(DTE,RX); /* set the DTE channel to receive */



/* Start the run-time display */

     check = start_display("c:C/Decode/BOPS.program",22,0,&disp_QID);
     if (check) {
          printf("start_display() returned an error.\n");
          getch();
          exit(0);
```

```
    }

/* Start the front end */

    check = start_data();
    if (check) {
        printf("start_data() returned an error.\n");
        getch();
        exit(0);
    }

/* Loop forever looking for the message to say that the run-time display
   has finished */

    while(1) {

        if (!(read_message(&message,WAIT,WAIT_FOREVER))) {

            if (message.type == DISPLAY) {
                stop_data();
                return();
            }
            else {

                if ((message.body.event.event_ptr))
                    release_event(message.body.event.event_ptr);
            }
        }
    }
}
```

TRANSMIT

```
/* This is the demo program TRANSMIT.  This program sends 100 DCE frames, each
   with a different address, to the line. */

#include "C:C/Include/dlib.include"
#include "C:C/Include/leads.include"

main()                          /* all programs must start with main() */
{                               /* opening brace for program */

  int frame_count;
  char address, control;
  char frame_no[10];

  frame_count = 1;
  address = 0x01;
  control = 0x0;

  /* first lets do some initialization */

  set_protocol(HDLC,SYNC_DCE,ASCII8,NONE,CRC_CCITT); /* explicitly set
                                                              the protocol */

  set_channelconfig(DTE,RX);
  set_channelconfig(DCE,TX);  /* configure for simulate DCE */

  /* initialization complete */

  start_data();               /* start acquisition of data   */

  set_lead(DSR,ON);
  set_lead(CTS,ON);

  while (frame_count <= 100)  /* go through this loop 100 times */
  {                           /* begin while */
```

```
    itoa(frame_count,frame_no,10); /* Convert frame_count to an ascii string so it can be
                                      sent by sendf() */
    sendf("%2nThis is frame number %s",address++,control,frame_no); /* transmit frame  */

    printf("Sending frame %d\n", frame_count++);

  }  /* end while */

  set_lead(DSR,OFF);
  set_lead(CTS,OFF);
  getch();  /* wait here until any character is pushed */
}    /* closing brace for program */
```