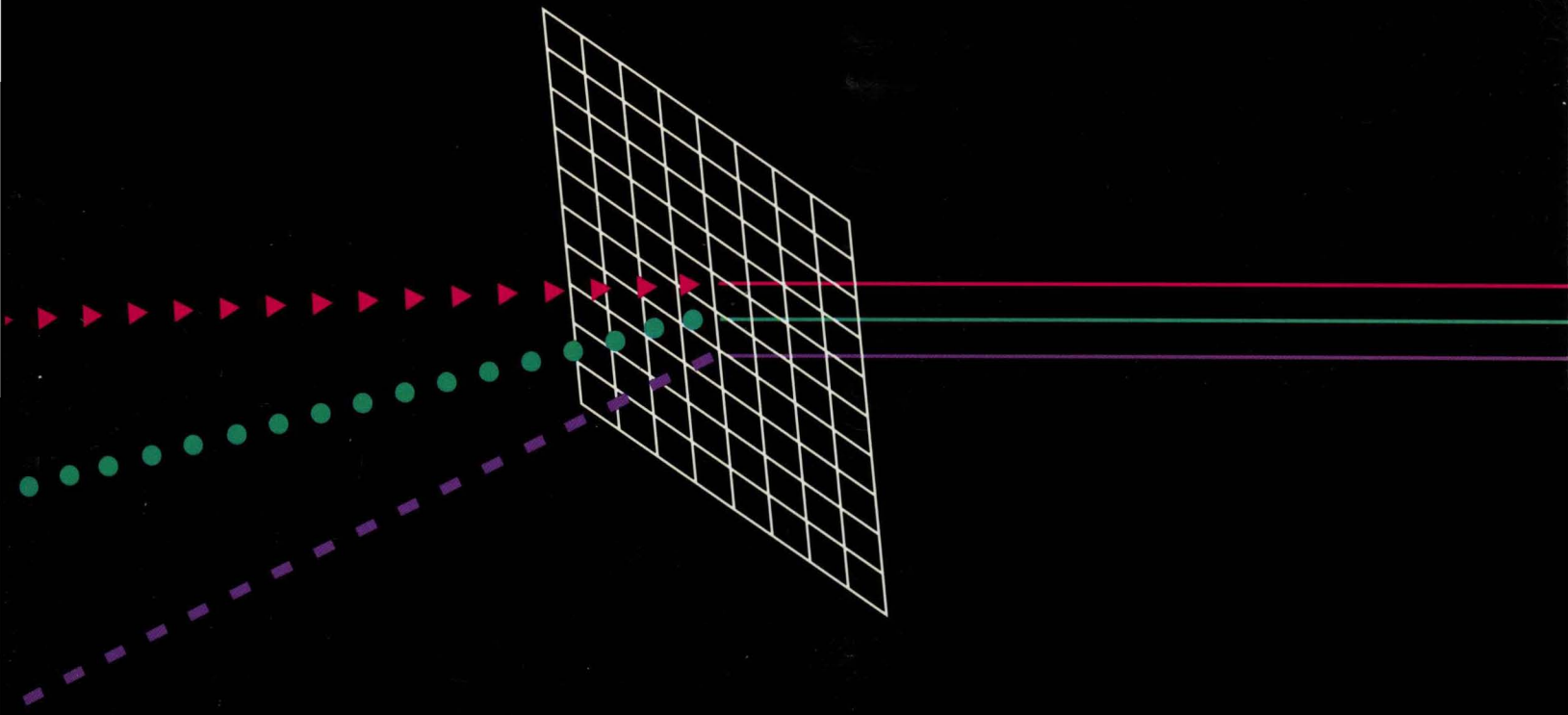
TEKELEC



TEKELEC

Chameleon 32

C Development System

902-9041

CHAMELEON 32
C DEVELOPMENT SYSTEM

Version 2.7

TEKELEC
26580 Agoura Road
Calabasas, California
91302

Part Number 909-3384

November, 1992

TABLE OF CONTENTS

CHAPTER DESCRIPTION PAGE

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1.1 C PACKAGE DESCRIPTION

The Chameleon 32 C Compiler System provides a complete development environment for C programming. The C Compiler is a complete implementation of Kernighan and Ritchie C, and includes the following features:

- C shell
- vi style editor
- Linker
- Assembler
- Disassembler
- Librarian

The program editing functions are also accessible via softkeys, for easy program development. The Chameleon 32 C Shell controls all C activities using a command line interface.

The Chameleon 32 provides a multi-tasking operating system, page display system, custom keyboard, and color display for a powerful development environment which is familiar to UNIX and PC C language programmers. With the multi-tasking system, you can edit a program, compile a second program, and run a third program simultaneously.

Note

If you are unfamiliar with the use of the Chameleon 32 keyboard or the concept and use of *pages*, refer to the *Chameleon 32 User's Guide,Chapter 3: Using the Chameleon 32.*

Libraries are provided for UNIX compatible standard I/O (file I/O, memory access). There are also protocol-specific libraries, which include:

- BOP
- HDLC
- SDLC
- LAPD
- Analysis
- Async
- BSC
- Basic Rate Interface
- Primary Rate Interface

Libraries are also included for floating point math and window interface functions.

Compatibility with other Systems

The Chameleon 32 C package is compatible with MS-DOS version 2.x.

C programs developed on other hosts (such as VAXes or PCs) can be compiled and run without change on the Chameleon 32 if they use UNIX style calls to access console and file I/O devices. Programs developed on the Chameleon 32 that use the same facilities can also be compiled and run on other hosts.

Programs developed on a Chameleon 32 that access the Protocol Specific I/O libraries will not run on another computer, unless an equivalent protocol specific library is developed for that computer's hardware. Similarly, C programs which rely on special system facilities of other computers may not be portable to the Chameleon 32 without modification.

C File Upload and Download To/From a Host Computer

Chameleon 32 file upload and download functions may be used to transfer C source files between the Chameleon 32 and other computers. Program code must be linked with Chameleon 32 system interface and library code to make executable tasks. This means that executable and object code files cannot be transferred from other computers to the Chameleon 32.

In the past, test instruments did not offer convenient high level language development environments. As a result, present users would like to be able to perform program development on host computers, and then download programs to the test instrument for execution. Due to various portability issues described above, functional testing of programs can only be done in the test instrument. The Chameleon 32 allows download of program source code from a host, and, in addition, offers a fast, powerful, complete and easy to use stand alone development environment that is familiar to UNIX and PC C programmers.

Run Time Environment

There are two types of user programs (analysis and simulation). Since the run time environment is different for the two types, a user program can only be executed in the same environment in which it was compiled and linked.

Each user program is a separate task with its own default *page*. The shell is responsible for starting the task, assigning its default *page* and killing it upon termination.

Run files can be loaded and executed from the development system shell. An additional selection of User Task can be made (in the Main Menu) to select all files that can be executed (ON/OFF). Turning off a task (or using the KILL command) will close the task's *page*.

User programs interact with the Chameleon 32 software through libraries. The Libraries include:

- I/O library (for standard console I/O, device access, memory and file manipulation),
- Protocol specific libraries
- Library to support acquisition buffer activities

These libraries are described in Appendix B.

1.2 LOADING THE C PACKAGE

Introduction

This section gives you brief instructions for loading the C Development System. If you need additional information about booting and configuring the Chameleon 32, refer to the *Chameleon 32 User's Guide, Chapter 3*.

To load the C Development System, do the following:

1. Power up and boot the Chameleon 32.
2. The main configuration page should appear as shown in Figure 1.2-1. If this menu is not displayed, move the arrow cursor to Setup Mode and press ***F1 Menu***.

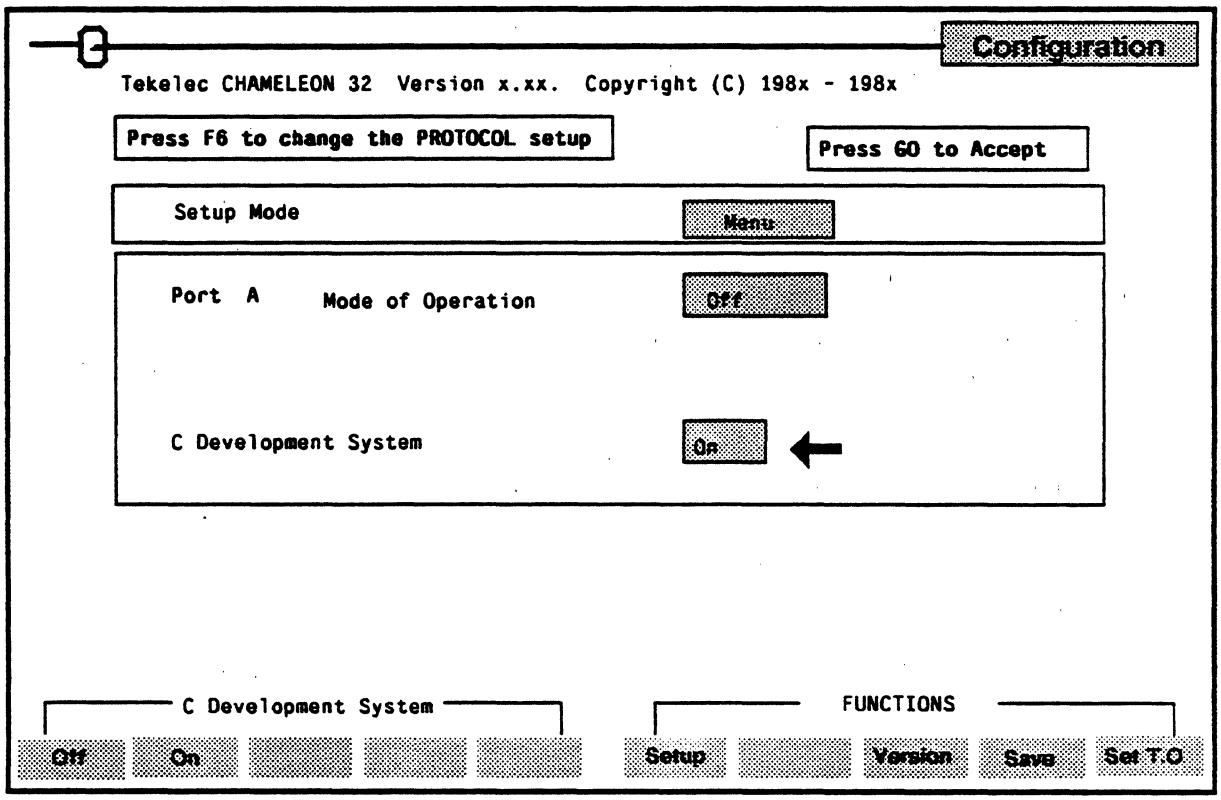


Figure 1.2-1: Configuration Page (Menu Setup Mode)

3. Move the red arrow cursor to the **C Development System** parameter at the bottom of the screen. (If this parameter does not appear, your C package is not installed.)
4. Press ***F2 On***.

5. Press **Go**. The message **Loading C-Shell** appears in the upper right corner of the Configuration page.

   The **C Shell** banner appears at the bottom of the screen and the Applications Selection page is displayed.

6. To use the C Shell page, press ***Select*** until the C Shell banner is highlighted (active). You can then use the keys listed in the table below to change the size of the C Shell page.

KEY	FUNCTION
Move ↑	Moves the page banner upward one line at a time (increases the size of the page).
Move ↓	Moves the page banner downward one line at a time (decreases the size of the page).
Scroll ↑	Scrolls the data displayed in the page upward one line at a time.
Scroll ↓	Scrolls the data displayed in the page downward one line at a time.
Shift Scroll ↑	Scrolls the data displayed in the page upward the number of lines displayed in the page.
Shift Scroll ↓	Scrolls the data displayed in the page downward the number of lines displayed in the page.
Shift Hide Page	Hides the active page so that the banner is no longer visible on the screen (the application continues to run).
Show Page	Displays a page that has been hidden with Shift Hide Page.
Replace	Replace the active page with one that has been hidden using Shift Hide Page.
Shift Move ↑	Displays the page in a special full-screen mode referred to as Blow Mode (indicated by the letter B on the top left side of the banner). Other pages cannot be accessed when the active page is in Blow Mode. Shift Move ↑ again disables Blow Mode, and returns the screen to its previous state.

The hardware and protocol are configured from your C program. Refer to the appropriate section for a description of the available functions:

Section 1.3 contains a short tutorial to acquaint you with the Chameleon 32 C compiler and editor.

Chapter 2 contains a description of C Shell usage and shell commands.

7. To turn the C Development System off, use one of the following methods:

   a. At the C Shell prompt %, enter the command:

      %exit <RETURN>

   b. Select the Configuration page and press ***F10 Exit***.

Executing C Applications

There are two ways to execute a C program that has been compiled on the Chameleon 32:

- You can run it from the C Shell. To use this method, the C Development System must be installed on the Chameleon 32. Refer to Section 2.1 for more information.

- You can run it from the Applications Selection menu. You **must** use this method to execute a C application in the following cases:

  - To run a C program on a Chameleon 32 that does not have the C Development system installed

  - To run a C program on a Chameleon 20. The Chameleon 20 C Run-Time module must be installed on the Chameleon 20 in order to do this.

  In addition, this method enables you to include your C application in your configuration file, so that the program can be executed automatically when the configuration file is loaded. This procedure is described below.

To execute a C application from the Chameleon 32 or Chameleon 20 Applications Selection menu, do the following:

1. Compile the C program on a Chameleon 32. See Chapter 2.2 for compiler syntax.

2. The application file name must have the extension **.exe.**

3. Copy the file to the hard disk of the Chameleon on which you want to run the application. The directory determines when the application will be displayed in the Applications Selection menu, using the conventions described below.

   To have the program appear in the Monitoring window of the Applications Selection menu, copy the program to:

   a:\tekelec\analysis*xxxx*

   xxxx is one of the sub-directories of *analysis*. If copied to *a:\tekelec\analysis\appl*, the application is displayed in the Monitoring window for all protocols.

If the application is copied to a protocol sub-directory of *a:\tekelec\analysis*, the application is displayed in the Monitoring window only when the Chameleon is configured for that protocol. For example, if the application resides in *a:\tekelec\analysis\x25*, it appears in the Monitoring window only when X.25 is the selected protocol.

To have the program appear in the Simulation window of the Applications Selection menu, copy the program to:

A:\tekelec\simul*xxxx*

xxxx is one of the sub-directories of *simul*. If copied to *a:\tekelec\simul*, the application is displayed in the Simulation window for all protocols.

If the application is copied to a protocol sub-directory of *a:\tekelec\simul*, the application is displayed in the Simulation window only when the Chameleon is configured for that protocol. For example, if the application resides in *a:\tekelec\simul\x25*, it appears in the Simulation window only when X.25 is the selected protocol.

Notes

Only applications copied to *a:\tekelec\analysis\appl* can be started on Ports A+B on Chameleon 32 Dual Port machines. Applications in all other directories must be started on each port independently.

Applications developed for the Chameleon 20 use Port A only, since Dual Port is not available.

4. In the main configuration menu, set up the Chameleon port for the mode of operation (Monitor or Simulate) and protocol appropriate for the C application.

5. Press ***Go.*** This displays the Applications Selection menu, with the C application name displayed in the window according to the conventions described in step 3.

6. Move the red arrow cursor to the application name and press the function key that starts it on the appropriate port (***F1 Load A***, ***F2 Load B***, ***F3 Load AB***).

7. Press ***Go*** to start the application.

Programming Notes:

When running a C application from the Applications Selection menu, the Chameleon automatically opens a window for the application when it is started and closes the window when the application is stopped. This is similar to running the program from the C Shell in background mode, using the syntax:

%progname&

See page 2.1-5 for more information about using & (background mode) in the C Shell.

When using the Applications Selection menu, a pointer to the application file name is passed to argv[0] and a pointer to the port selected by the user is passed to argv[1]. This is equivalent to executing a program from the C Shell using the convention:

%progname A
%progname B
%progname AB

This information can then be used in the C application to initialize and access the appropriate port(s) using the protocol library functions.

In order to exit from the application properly, use the function *onexit*. This function enables you to finish your program when the user stops the application from the Applications Selection menu. See page 5.2-37 for more information.

1.3 C PROGRAMMING TUTORIAL

Introduction

This section contains a brief tutorial that introduces you to the process of developing programs in the Chameleon 32 C environment. In the tutorial, you write a short program that causes the Chameleon 32 to transmit a short message.

C Program Development

Figure 1.3-1 illustrates the steps required to develop programs using the Chameleon 32 C package. The chapter that contains more information about each facility is also indicated.

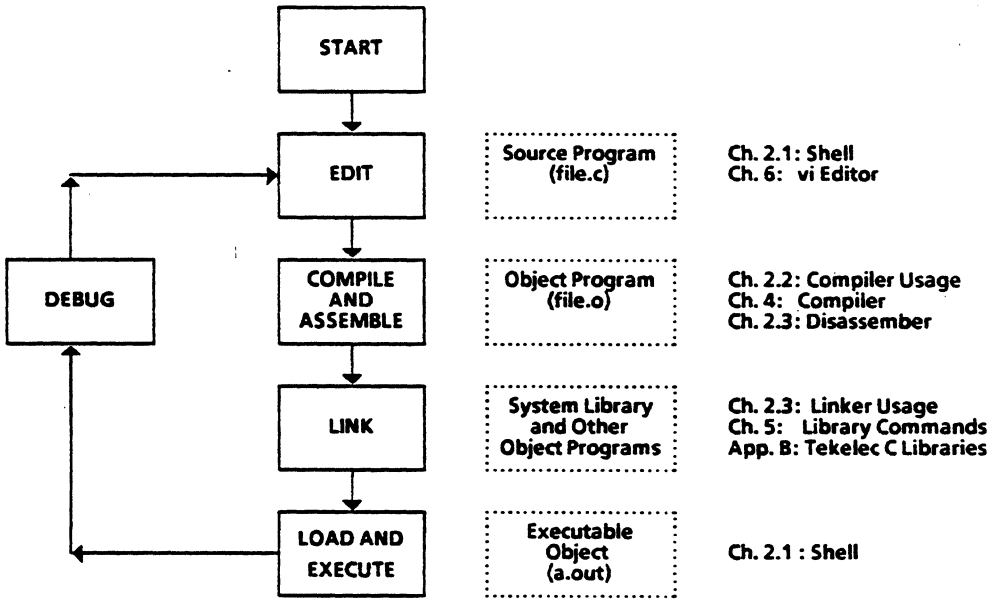


Figure 1.3-1: C Development Cycle

Tutorial

The following is a hands-on introduction to the process of writing, compiling, and executing a C program. This program transmits a short message.

1. Boot the Chameleon 32.

2. Use the port Configuration page to turn the C Development System *ON*, and then press ***Go***. After a few seconds, the **C Shell** page banner should appear at the bottom of the screen.

3. Display **C Shell** page so that the **%** prompt is visible. The login file was executed and put you in the home directory (as defined in the login file).

4. List the files and directories on the hard disk by entering:

   ls *<Return>*

5. The list should include a directory called **USR**. This directory has been provided for your user application files (although you do not *have* to store your files in it.)

   Change to the **USR** directory so that the program you write will be saved to this directory. To change directories, enter:

   cd usr *<Return>*

6. You will use the vi editor to write a program called **test1.c**. In the command, you will use an ampersand (**&**) to open a separate vi page to enter and edit the program. This is referred to as background mode. To call up the vi editor in background mode, enter:

   \vi test1.c & *<Return>*

   The PATH environment variable will look for vi in the \bin directory. If you don't have a login file you should set the path (see SETENV command in Section 2.1).

   The **VI** banner appears at the bottom of the screen. (If this is the first time the vi editor has been used this session, you may have to wait a few seconds for the banner to appear.)

7. Display the **VI** page.

   The cursor is positioned at the top of the screen and tildes (~) appear on the other lines. The message **"test1.c" [New File]** appears at bottom of page.

Note

If there is a window banner at the bottom of the screen, you will not be able to see the messages on the bottom.

8. **vi** will be in command mode. To enter a program, use the insert mode. To change to insert mode, press:

   i (Do not press ***Return***.)

The screen does *not* change appearance or display a message to distinguish between command and insert mode.

9. Enter the program that is listed in **bold type** below. Programming remarks have been added for your benefit; These appear in non-bold type and do not have to be entered.

   *Press **Return** at the end of each program line. Use the same spacing and capitalization as shown below.* Press ***Tab*** one or more times to indent.

```
#include    <cham.h>   /*Defines constants that will be used*/
#include    <stdio.h>
#define     GOOD_CRC   0
main ()
{
      initp1 (DCE, NRZ, 9600L, FILLFF);   /*Initializes simulation of
                                           bop library*/
      printf ("Hit RETURN to transmit frame\n");
      getchar ();
      transmit (GOOD_CRC, "hello",5);      /*Transmits the frame 'hello'
                                           with a good CRC and 5 as the
                                           frame length in bytes*/
      puts ("Transmission completed");
}
```

10. When the program is completely entered, press ***Esc*** to return to command mode.

11. To save the file, enter:

    :w <***Return***>

    A message at the bottom of the screen verifies the number of lines and characters entered.

12. To quit vi, enter:

    :q <***Return***>

    The **VI** page disappears, but the **C Shell** page banner is still displayed.

13. Display the **C Shell** page and the **%** prompt.

14. Now you are back in the shell and ready to compile and link the program, using the cc command, which is in the BIN\ directory. To do this, enter:

    cc test1.c -lbop *<Return>*

    The **-l** option calls the library **libbop.a** . If you entered the program correctly, the % prompt is redisplayed. If you typed something incorrectly, the errors are described on the screen. If there are errors, repeat step 6 to edit the file using vi, and then repeat step 12 to compile and link the corrected program.

15. To execute the compiled program, at the **%** prompt, enter:

    a.out *<Return>*

16. The following message appears:

    Hit RETURN to transmit frame

17. Press a key, the frame will be transmitted, and this message displayed:

    Transmission completed

2.1 SHELL

Introduction

The Tekelec development shell provides a command line user interface for C development. The shell enables you to run programs and contains programs for maintaining files and sub–directories.

There are a number of built–in shell commands that are loaded into memory as part of the shell. Typing a command other than one of these built–in shell commands is treated as an attempt to load and execute a program.

Some shell commands are built into the shell and also exist as programs. These commands include: *cc*, *cp*, *rm*, and *mv*. If you execute one of these commands from a batch file or make file, the program is used.

Each shell and program has the notion of current directory, so you can view and work from different directories simultaneously.

Login File

The **login** batch file is executed automatically when the shell starts. It contains a list of shell commands and can be modified using the vi editor. A default **login** file is provided. Use the **more** command to view the contents of the **login** file. If you create a new **login** file and save it, the new **login** file overwrites the existing **login** file.

The loader uses an environment variable called PATH to locate programs to execute. The PATH variable is set in the **login** file on the distribution disk. Command line argument passage is supported.

Configuration File

The **CONFIG.SYS** file is used to configure certain functions of the Chameleon 32. This file is located in the **/TEKELEC/UTIL** directory. As the Chameleon is booting, **CONFIG.SYS** is read by the operating system and acted upon based on commands within the file. Should this file not exist, default values are assumed and acted on.

CONFIG.SYS supports the following commands:

BOOT – For the P6 board of the Chameleon 32-plus only, this command tells the Chameleon which application is to be booted at start-up.

The valid options are:

A:\\tekelec\\util\\stmenu.sys
(stmenu = standard menu)

A:\\tekelec\\util\\stshell.sys
(stshell = C shell)

REM – Remark or Comment. Used primarily for documentation of the **CONFIG.SYS** file itself. All other commands on this line are ignored.

DBGPORT – Location of the Debugger Port. Used to place the Debugger Port on the AUX 1 serial port or on the Chameleon CRT and keyboard. Also used to turn the debugger off. When the Debugger is not placed on AUX 1, this port is available for other applications uses (See Section 5.7).

This command takes one operand having the following syntax:

AUX1 – places the debugger on the serial port. The default location.

VT – places the debugger on the CRT/keyboard.

OFF – turns the debugger off.

For Chameleons operating over an Ethernet:

GW – allows you to specify the Internet address of the gateway connecting your Chameleon 32 to other subnets, using standard decimal dot notation. For example:

192.9.200.104

IMASK – allows you to specify the Internet mask of the Internet if subnet masking is used. For example:

255.255.255.0

INET – allows you to specify the Internet address of the Ethernet board, using standard decimal dot notation. For example:

192.9.200.102

Example.

The following is an example of what a **CONFIG.SYS** file might look like:

```
REM         This file is used to place the debugger on
                the
REM         Chameleon 32 CRT and keyboard.
REM         By doing this I may now access AUX 1 via a
                C program.
DBGPORT     VT
```

Device Files

Devices are referred to by file names, using the following conventions:

- .CON = Console
- .PRT = Printer
- .AUX = Serial port 2 (unformatted data)
- .TTY = Serial port 2 (formatted data)

These names must be in upper case letters. For example:

```
shell <.TTY> .TTY
```

redirects shell input and output through Serial port 2 and will work from a remote terminal.

Filename Substitution

The * and [ ] characters can be used for filename substitution, as described below.

- *.c — Matches all filenames with .c extension.
- test.* — Matches all files named *test.* with any file extension.
- [x] — Matches filenames containing the letter enclosed in the brackets, in this example, *x.*
- [bgy] or [b,g,y] — Matches filenames containing the indicated letters (logically ORed). In this example, filenames with *b* or *g* or *y.*
- [(ch),(te)] — Matches filenames containing *ch* or *te.*
- [b–g] — Matches filenames with the letters in the indicated range. In this example, filenames with b, c, d, e, f, or g would be matched

For example, if a directory contains the following files:

hello.c test1.c test2.c test2.o test1.o a.out

If you use the **ls** (list files) command as shown below, the resulting filenames are listed.

```
ls *        helloc.c test1.c  test2.c  test2.o test1.o a.out
ls *.c      hello.c  test1.c  test2.c
ls t*       test1.c  test2.c  test2.o  test1.o
ls *[o]*    hello.c  a.out
```

Any word enclosed in single or double quotation marks prevents filename expansion, or creates a single argument to pass to a program. For example:

```
% a.out  *.c   receives:  argv[0]  :  "a.out"
                          argv[1]  :  "HELLO.C"
                          argv[2]  :  "TEST1.C"
                          argv[3]  :  0L (NULL)

% a.out  '*.c' receives:  argv[0]  :  "a.out"
                          argv[1]  :  "*.c"
                          argv[2]  :  0L (NULL)
```

I/O Redirection

The shell supports redirection of standard output and input from any command line, as follows:

<	name	Read the file 'name' in place of stdin
>	name	Write stdout to the file 'name'
>>	name	Append stdout to the file 'name'
>&	name	Write stderr to the file 'name'
>>&	name	Append stderr to the file 'name'

The default for stdin is the keyboard. The default, for stdout and stderr is the window for the calling program. I/O redirection operators and names are not passed to the calling program.

Built–in shell commands also use I/O redirection operators, for example:

dis filename.o ¦ more	Uses shell command *more*
ls > filename	Redirects file listing to *filename*

I/O redirection is available through the *system* function. The following example illustrates this.

```
main( )
{       char  line[80];
        system (line,"ls");
        system (line,"ls > filename");
}
```

Environmental Variables

The shell supports environment variable setting (setenv) and printing (getenv). These variables are stored internally as a string of the form: name='value'.
The variable PATH (note upper case) is used by the loader to find programs to execute. A typical PATH is:

```
'. \bin \user'
```

This command string means first search the current directory (.), the \bin directory, and then the \usr directory. *If you set a path, you must include the period (.) or the current directory will not be searched.* If no path variable is set, only the current directory is searched.

The variable VIINIT is loaded by the editor as it is executed. The value of VIINIT may be any valid vi commands to be processed as the editor is started. The variables *FC* and *BC* (foreground color and background color) are used as colors for window creation when a program is run as a separate task.

See the description of **setenv** and **getenv** for additional information.

Commands

The shell supports the commands listed below. In the syntax descriptions, angle brackets (<>) indicate that the field is user supplied. Square brackets ([ ]) indicate optional items.
The following methods can be used to execute more than one command in a single command line.

% a; b; c Executes commands, a, b, and c as if they were entered as follows:

```
% a
% b
% c
```

% a & b & c Executes commands a, b, and c as if they were entered as follows:

```
% a &
% b &
% c
```

% a | b | c Executes commands a, b, and c as if they were entered as follows:

```
% a > pipe1
% b < pipe1 > pipe2
% c < pipe2
```

The shell commands are listed in alphabtical order, one command per page, on the following pages.

& (Background Mode)

Description

The ampersand symbol (&) is the command that runs an executable file in background mode. When the ampersand is used, a window is opened for the task and it is used as stdin and stdout.

Syntax

<name> &

where: **name** is the filename of an executable file.

To modify the window color, use the **setenv** command. The window will be closed when the task is killed.

See Also

run (page 2.1–30)

Example

To run the file named PROG1 in background mode, enter:

```
% PROG1 &
```

(Remark)

Description

The # sign enables you to enter a remark that is ignored by the shell. The remark is not echoed to the screen. It is useful for inserting programmer's remarks in batch files.

Syntax

#[text]

where: **text** is the remark you want in the batch file.

Example

To enter a remark, enter:

```
#Setting environment.
```

' (Echo Text)

Description

The ' sign enables you to echo text to the screen. It can be used in batch files to echo messages or instructions to the screen.

Syntax

'[text]

where: **text** is the text to echo to the screen.

Example

To display the message 'Hello out there', enter:

```
%'Hello out there!  <RETURN>
Hello out there!
```

batch

Description

The **batch** command executes a batch file. A batch file is a file which contains a sequence of shell commands.
If a batch file named **login** exists on the root directory of either the hard or floppy disk, it is automatically executed when the shell is launched.

Syntax

batch <filename>

where: **filename** is the name of the batch file.

Example

To execute a batch file named **startup**, enter:

```
batch startup
```

cat

Description

The **cat** command prints files to standard output. It is used primarily to display the contents of one or more files.

Syntax

cat <file...>

where: **file** is the name display on the screen. If a single file name is specified, the contents of that file is displayed on the screen.

If two or more files are specified, **cat** concatenates them in the order given and writes the output to the indicated file.

If the **file** parameter is not specified **cat** reads from standard input.

Example

To display the contents of the file prog1, enter:

```
cat prog1
```

To concatenate the contents of **filea** and **fileb** and write the result into **filec**, enter:

```
cat filea fileb >filec
```

Errors

File not found: name The file or directory name does not exist.

cd

Description

The **cd** command changes the current sub–directory.

To determine the names of the sub–directories of the current directory, use the **ls** command. This lists the entries of the current directory with sub–directory names followed by a slash (\).

The root directory is referred to by a back slash (\).

Syntax

cd <path>

where: **path** is the path name of the sub–directory that you want to use.

Example

To change from the root directory to the USR/ sub–directory, enter:

```
cd usr  <RETURN>
```

To change back to the root directory, enter:

```
cd \
```

cp

Description

The **cp** command copies one or more files into a specified directory. The cp command can be used to make copies of files, rename files, or transfer files to a different directory.

Syntax

cp <oldfile> <newfile>

where: **oldfile** is the name of the source file and **newfile** is the name of the new copy of the file. The **cp** command will not copy a file onto itself, therefore a **newfile** name must be supplied.

If **newfile** already exists, its contents are overwritten. If newfile is overwritten by the **cp** command, the original mode and owner are preserved. If **newfile** is a new file, the mode and owner of the source file is used.

The **cp** command copies one or more files into a different directory, using the following syntax:

cp <file...> <dir/>

where: **file** is the name of the file to copy and **dir** is the name of the directory to copy the file into. Note that the directory name is followed by a slash (/) to indicate that it is a directory and not a new filename. The filenames remain unchanged.

Example

To make a copy of a file named PROG1 and give the copy the name PROG2, enter:

```
cp prog1 prog2
```

To copy a file named PROG1 into USR/ sub–directory, enter:

```
cp prog1 usr/
```

Errors

Can't copy file to itself: name	CP was given the same file name to copy to as the source file name.
name: not a directory	The name of the directory to copy to was invalid.
Can't open: name	The source file does not exist, or there is something wrong with the disk.

ctags

Description

ctags is a utility that helps you locate definitions in multiple C program files. It searches specified files and creates a file named *tags* which contains a list of the functions that were found in the program files. The *tags* file can then be used in conjunction with the vi editor to quickly locate specific functions in the files you are editing.

Syntax

ctags *files*

where: **files** are the C program filenames that you want to search through. You can use wildcards to specify the files to search. For example, this searches all C source files beginning with the name test:

ctags test*.c

This creates a file named *tags* which contains the following information about the definitions (tags) in the files that were searched.

tag filename String to search for

You can use the *more* command to view the contents of the *tags* file.

There are two ways to use the *tags* file with the vi editor:

1. Use the –t option when you invoke the vi editor. For example:

   vi –t tagname

   This edits the first file listed in the *tags* file which contains the tag specified by *tagname*. The cursor is positioned at the first occurrence of the tag in the file.

2. Invoke the vi editor and then use the **:tag** command. For example:

   vi *.c
   :tag tagname

   This first invokes the vi editor to edit all .c files. The **:tag** command then positions the cursor at the first occurrence of the specified tag within the files being edited.

dump

Description: The **dump** command prints one or more files in hex to standard output.

Syntax:

dump file

Example: To print the contents of the file named test1 in hex, enter:

```
dump test1
```

Errors:

File not found: name	The file name does not exist.

exit

Description

The **exit** command exits from the C shell and returns to the Chameleon 32 port configuration page. It causes the C shell to no longer be active.

Syntax

exit

Example

To return to the Chameleon 32 port configuration page from the C shell prompt (%), enter:

```
exit
```

format

Description — The **format** command formats a floppy diskette.

Warning — Formatting erases all the data on the disk. Make back up copies of files you want to keep before you format.

Syntax — **format b**

Example — To format the floppy disk, enter:

format b

The following message is displayed:

Do you wish to format the floppy disk B:? (Yes or No) Y *<cr>*

getenv

Description

The **getenv** command prints the string value of all environmental variables or of a specified environment variable. Use the setenv command to set the value of an environment variable.

Syntax

getenv	Displays values of all environmental variables.
getenv <name>	Displays value of a specified environmental variable.

where: **name** is the name of the environment variable to print. The following variable names are used by the Shell, but up to 20 variables can be defined by the user. The following variable names must be entered in UPPER CASE letters.

PATH	Displays the default search path for locating files.
FC	Displays the foreground color for new windows.
BC	Displays the background color for new windows.
YEAR	Displays the global _curr_year in the libraries
HOME	Displays a path that is changed to when the cd command (with no argument) is used.

Example

To display the current default search path, enter:

```
% getenv PATH  <RETURN>
.b:\bin
```

To display the foreground color for new windows, enter:

```
% getenv FC <RETURN>
white
```

help

Description

The **help** command displays a list of shell commands and their usage.

Syntax

help

Example

```
% help
```

jobs

Description

The **jobs** command prints job control status, including process id (pid), program name, and whether resident or running.
If running, a program is active and can be killed using the **kill** command.

If resident, a program is in memory, and when started, is loaded from memory and not from disk.

Syntax

jobs

Example

To display the current jobs and their process id numbers, enter:

```
% jobs
[ 0]  Running   B:\SHELL
[ 1]  Resident  B:\BIN\CP
```

kill

Description

The **kill** command kills a process that is running. It does not remove the process from residency.

Syntax

kill <pid>

where: **pid** is the process id of the program to kill. Refer to the **jobs** command to print **pids** and other information about programs.

Example

If you want to view the current jobs, enter:

```
jobs
```

A display such as the following appears:

```
[ 0] Running    B:\SHELL
[ 1] Resident   B:\BIN\CP
[ 2] Running   B:\USR\PROG1
```

This display indicates that PROG1 is running and is assigned process id 2. To stop the process but leave PROG1 resident, enter:

```
kill 2
```

ls

Description

The ls command prints a list of files and directories, and information about them. If a directory (or drive specifier) is given, the directory name, number of files in the directory, and a list of all files in the directory is displayed. If a file name is given, matching file names are listed. If no file/directory names are given, the contents of the current directory is listed. In the absence of a sorting option, names are sorted alphabetically.

In the resulting display, sub–directory names are followed by a slash (/).

Syntax

ls [–L] [–K] [–S] [–D] [spec]

ls options are:

–L	Long listing format. Provides the name, size (excluding header information) and date of the file, with each file displayed on a separate line.
–K	Listing is sorted by file extension (kind)
–S	Listing is sorted by size
–D	Listing is sorted by date of last modification.

If one of the sorting options is note used, the list is sorted by filename. Filename substitution is performed.

Example

To list the files in the current directory including the date and size, enter:

```
ls -L
```

To list all entries in the current directory that begin with the letter S, enter:

```
ls s*
```

Error Messages

Unknown option: option	An option was given that LS does not recognize.
File not found: name	The file or directory names does not exist.
Drive *DRIVE*: not available	The drive is not available.

man

Description

The **man** command displays a named help file for C commands, programs, and library functions.

Syntax

man <filename>

where: **filename** is the name of the help file. The following help files are available:

Filename	Topic
anal	Analysis library
ar	Librarian
asynclib	Async library
aux2lib	Aux Port 2 library
bop	BOP library
bri	BRI library
bsc	BSC library
cc	compiler
dis	Disassembler
egrep	egrep
filefunc	Low level MS–DOS file functions
hdlc	HDLC library
lapd	LAPD library
ld	Linker
libc	Standard C library functions
make	make command
mathlib	Math library
mlink	Multi–Link LAPD library
pri	PRI library
sdlc	SDLC library
shell	Shell commands
v120	V.120 library
vi	vi commands
window	Window functions

Example

To display the help file for the BOP library, enter:

```
man bop
```

mkdir

Description

The **mkdir** command creates a sub–directory. If a partial pathname is given, mkdir creates the sub–directory in the current directory.

Syntax

mkdir <name>

where: **name** is the name of the sub–directory you are creating.

Example

To make a new sub–directory named PROGS, enter:

```
mkdir progs
```

Errors

Can't create directory: name The directory name already exists, or the disk is write protected.

mkres

Description

The **mkres** command makes a program RAM resident. Normally, any program you execute becomes RAM resident. The mkres command enables you to make a program RAM resident without running it. It also prints the process id (**pid**) of the program.

Syntax

mkres [–p] <prog>

where: **prog** is the filename of the program.

The **–p** option indicates that the program cannot be removed from memory to satisfy a request for a block of free memory. If the memory manager receives a request for a block of free memory which it cannot satisfy, it removes the least recently used programs until it can satisfy the request. The **–p** option indicates that the program cannot be removed by the memory manager.

This option is useful for large programs, such as the compiler, which take some time to load.

Example

To make a program named PROG1 resident in RAM, enter:

```
mkres prog1
```

more

Description

The **more** command displays the contents of a specified file or pipe, one screenful at a time. If the display is longer than a single screen, it pauses when the screen is full, and prints the following prompt at the bottom of the screen:

—more— (n%)

The **n%** is an integer that indicates the percentage of the file (in characters) that has already been read. The percentage is not displayed if **more** is reading from a pipe.

You have three options for displaying additional text on the screen when the **—more—** prompt appears. These options are:

- Press *Return* to display the next line from the file
- Press the *Space bar* once to display the next screen
- Type a number and then press *Space bar* to display that number of lines. For example, to display the next 10 lines, enter:

  10 <Space bar>

Syntax

more <file>

where: **file** is the name of the file to display.

If **more** is redirected to a device other than a terminal, it transmits the file.

Example

To list the contents of the file PROG1, one screenful at a time, enter:

```
more prog1
```

mv

Description

The **mv** command moves a file, removing the original copy of the file. You can use the command to move the replace an existing file with another file or to move files from one directory into ancther directory.

Syntax

mv file1file2	Replace file2 with the contents of file1. Both files must already exist.
mv files dir	Move the specified files from the current directory into the specified directory.

Example

To replace the contents of test1 with test2, enter:

```
mv test2 test1
```

To move all files named test to the parent of the current directory, enter:

```
mv test* ..
```

Where .. specifies the parent of the current directory.

Errors

Can't copy file to itself: name	MV was given the same file name to move to as the source file name.
name: not a directory	The name of the directory to move to was invalid.
Can't open: name	The file that is to be moved does not exist, or there is a disk error.

pwd

Description

The **pwd** command displays the name of the current directory on the screen.

Syntax

pwd

Example

to display the current directory, enter:

```
% pwd
B:\USR\
```

rm

Description

The **rm** command deletes one or more files from the disk.

Syntax

rm <file> [file...]

where: **file** is the name of the file to delete. Note that you can delete more than one file with a single **rm** command by listing the filenames separated by blank spaces. Filename substitution is performed.

Example

To delete a file named PROG1, enter:

```
rm prog1
```

To delete all files that have PROG as the first four letters of the filename, enter:

```
rm prog*
```

Errors

Usage: rm file... There are no options to rm.

rmdir

Description

The **rmdir** command deletes a sub–directory. You cannot delete a sub–directory if it is the current directory. You must be in the parent of the sub–directory in order to delete it. The directory must be empty before you can delete it.

Syntax

rmdir <name>

where: **name** is the name of the sub–directory you are deleting.

Example

To remove the directory named PROGS, enter:

```
rmdir progs
```

Errors

No such directory: name	The directory name does not exist or the disk is write protected.

rmres

Description

The **rmres** command makes a RAM resident program non–resident.

Syntax

rmres <pid>

where: **pid** is the process **id** of the program. Refer to the **jobs** command to print **pids** and other information about programs.

Example

If you want to view the current jobs, enter:

```
jobs
```

A display such as the following appears:

```
[ 0] Running    B:\SHELL
[ 1] Resident   B:\BIN\MORE
[ 2] Resident   B:\BIN\CP
```

This display indicates that the more program is resident and is assigned process id 1. To remove the more program from RAM, enter:

```
rmres 1
```

The screen then displays:

```
[ 1] Removed  B:\BIN\MORE
```

run

Description

The **run** command runs a program as a separate process. It loads the program, creates a new window for the program's standard I/O, and executes the program.

Syntax

run [–xxx] <prog>

The **–xxx** option sets the process priority within the range 1 – 230, with 230 having the highest priority. The process priority is actually the priority given to the MTOS–UX operating system when a task is created. The default priority is 100 unless otherwise specified.

The **&** character may be used as the last character of a command line to indicate 'run'.

By setting your task's priority, you can force your task to run by taking CPU time from other tasks. For example, the following are the process priorities assigned to Chameleon Monitoring applications:

Real Time display	100
History	200
Statistics	200

Note:

If your task does only CPU processing (no I/O), it may not allow processing time for other applications. If this occurs, you can use the MTOS–UX **pause** function to allow time for other tasks to run.

See Also

& (page 2.1–5)

Example

To run the program named PROG1, enter:

```
run prog1
```

A **prog1** banner appears at the bottom of the screen for the I/O of the program.

To run the program named PROG1 with a process priority of 200, enter:

run –200 prog1

setenv

Description

The **setenv** command sets an environment variable.

Syntax

setenv <name> <'value'>

where: **name** is the name of the environment variable and **'value'** is the string value of **name**. Note that the value is enclosed in quotation marks to protect the string.

The following variable names are used by the Shell, but up to 20 variables can be set and used by a user program. The following variable names must be entered in UPPER CASE letters.

BC	Sets the background color for new windows.
FC	Sets the foreground color for new windows.
HOME	Contains a path to change to if the **cd** command is used without an argument. If no HOME variable is found, the path is set to the root.
PATH	Sets the default search path for locating files.
YEAR	Sets the global _curr_year in the libraries.

The available foreground and background colors are:

- black
- red
- green
- yellow
- blue
- magenta
- cyan
- white

Example

To set a path from the current directory (.) to the root directory to the BIN sub–directory on the A drive, enter:

```
setenv PATH a:.'\bin b:\'
```

To set the foreground color to blue, enter:

```
setenv FC 'blue'
```

shell

Description

The **shell** command starts another shell that will do all that the C shell does. The other shell can be run in background mode, if desired.

Note: The cd command will not work on all shells.

Syntax

shell <name> &

where: **name** is the shell name and the & optionally runs the shell in background mode.

Example

To run a shell called **newshell** in background mode, enter:

```
shell newshell &
```

size

Description

The size command prints size information for the different segments of object or executable files to standard output.

Syntax

size files

Example

To display the size of an object file names scripts.o, enter:

```
size scripts.o
```

The resulting display will be:

```
text    data    bss    dec        hex
17590   4384    0      21974      55d6    scripts.o
```

time

Description

The **time** command displays the current time as maintained by the Chameleon 32 clock.

Syntax

time

Example

To display the current time, enter:

```
% time
23 SEP 1987 02:06:34
```

SHELL ERROR MESSAGES

Table 2.1–1 lists the Shell error messages and their meanings.

ERROR MESSAGE	MEANING
L_FNFERR	File not found.
L_OPNERR	Command not found.
L_MEMERR	Memory allocation error.
L_REDERR	File read error.
L_RELERR	File contains external references.
L_FMTERR	File not executable.
L_RUNERR	Program is currently running.
L_FULERR	Too many programs currently loaded.
L_PNRERR	Tried to restore a non-resident program.
L_BSYERR	Could not get semaphore to use loader.
S_COMERR	Command line error.
S_CRTERR	Unable to create a task.
S_KEYERR	Unable to get a new key.
S_MEMERR	Out of memory.
S_TOKERR	Bad command line syntax.
S_PTYERR	Priority must be from 1 - 230

Table 2.1-1: Shell Error Messages

2.2 COMPILER COMMANDS

Introduction

This section describes the compiler commands *cc* and *mcc*.

cc

The **cc** command compiles and links files. The **cc** command runs the compiler **mcc** and/or the linker **ld** on the specified files depending on the file extension.

Compiler flags and files with the **.c** extension (C source files) are passed to **mcc**. Linker flags and files with the **.o** extension (object files) are passed to **ld**. This includes **.o** files produced from **.c** files by the compiler. When the **cc** command calls the linker, the proper init code and C system library are included automatically.

The cc command functions as shown in this example:

cc [flags] [file.c] mcc [flags] [file.c] -I\include
ld [flags] \lib\init.o file.o \lib\libc.a \lib\libm.a

cc [flags] [file.o] ld [flags] \lib\init.o file.o \lib\libc.a \lib\libm.a

The compiler commands cc and mcc are described in this section. ld is described in Section 2.3. Square brackets [ ] indicate an optional field; angle brackets < > indicate a user specified field.

The **cc** command uses the following syntax:

cc [-c] [flags] [file.c/file.o...]

The cc fields are:

-c Compiles only; does not link

flags Optionally specified flags for *ld* and *mcc* as summarized below.

ld flags: -d Debug option. Causes the linker to include the names of functions in the executable program.

-lxxx Library search path. The linker automatically searches the path **\lib\libxxxx.a** where libxxxx.a is the name of the library.

	-m	Prints names and addresses of globals which are included in the executable program. Creates MAPFILE of globals in program. This is a set of symbols and their offset from the beginning of the file.
	-o	Writes output to specified output file (default is a.out).
	-txxx	Linker adjusts references wihtin program as if program were at hex memory location xxx.
mcc flags:	--Ipath	Causes the compiler to search the specified path for include files. Default path is **\include**.
	-dname	Equivalent to inserting **#define name** in the source.
	-dname = value	Equivalent to inserting **#define name value** in the source.
	-x	Trace Mode. Adds calls to the debugging routines **_debugin** as each function is entered and **_debugout** as each function terminates.

file.o If an object file is specified, cc calls ld only (links), but does not call mcc (compiler).

file.c If a C sourcefile is specified, cc calls ld (linker) and mcc (compiler), unless the -c option is included..

mcc

mcc is the name of the compiler and compiles one C source file at a time. It uses the following syntax:

mcc [-dname[= value]] [-lpath] [-x] [-I\ifile] < file.c >

The mcc command fields are:

-dname Equivalent to inserting **#define name** in the source.

-dname = value Equivalent to inserting **#define name value** in the source.

Example: **Dtest = 1** is the same as **#define test 1** in the source file.

-lpath Causes the compiler to search the specified path for include files. Default path is **\include.**

-x Trace Mode. Adds calls to the debugging routines **debugin** as each function is entered and **debugout** as each function terminates. The routines shown below are the default routines that will print the procedure names and the passed parameters. These routines can be overwritten by user routines.

```
debugin(args, format)
char **args;
char *format;
```

args is a pointer to the parameters on the stack

format is a pointer to a printf type format string containing the name of the function entered and a % conversion for each *arg*.

```
debugout(name)
char *name;
```

name is a pointer to a string containing the name of the fucntion that is terminating.

file.c The C source file name. C source files have the file extension **.c**. The compiler produces an object file and assigns the same file name as the corresponding C file, but changes the extension to **.o**.

Compiler Errors

Refer to section 2.8 for a list of global error codes which can be returned by the compiler.

2.3 LINKER COMMAND (ld)

Introduction

The Tekelec Linker is similar to the UNIX linker *ld*. The linker takes as input multiple object and library files and creates an executable file from them by resolving all external references (references to symbols not defined in the file making the reference). The user specifies which object files he wants loaded and which library files he wants searched.

The utility program named **cc** includes the correct libraries and initialization code. You will normally use program **cc** to compile and link your user programs.

The linker must be used even if a program doesn't contain any external references because an object file created by the compiler is not executable.

Symbols defined in the user specified object files will override definitions of the same symbol in the libraries because user object files are loaded first. Likewise, symbols defined in the first libraries in the list override definitions from latter libraries (the system library is always read last). A programmer may make use of this feature by writing his own versions of system library functions (such as `malloc` for instance) while still using other procedures from the library.

Usage

The linker is run from the shell. Linker command syntax is shown below. Square brackets **[ ]** indicate optional fields; angle brackets **< >** indicate user specified fields.

ld [-V] [-Llib] [-M] [-X] [-Txxx] [-o output] <objects> [libraries]

The fields for the **ld** command are:

-V Verbose option. Displays the names of the functions in each of the object or library files specified in the command line.

-Llib Library search path. The linker automatically searches the path **\lib\libxxxx.a** where libxxxx.a is the name of the library.

For example, to compile prog.c using the libsdlc (SDLC library), you can use the command:

ld prog.o -lsdlc

This causes the same result as entering:

ld prog.o lib\libsdlc.a

-M Prints names and addresses of globals which are included in the executable program. Writes this information to *MAPFILE*. This is a set of symbols and their offset from the beginning of the file.

-X Debug option. This option causes the linker to include the names of functions in the executable program. If the program terminates abnormally (dure to a processor exception), a stack dump can be printed. Static functions begin with a tilde (~) while global functions being with an underscore (_).

-Txxx Causes the linker to adjust references within the program as if the program was at hex memory location **xxx**. Normally, the program is linked as if it were based at location zero, and relocation information is included so that when a program is run, the references may be adjusted for the actual memory location. Setting this option also prevents this relocation information from being included.

-o output Writes output to a file, where **output** it the name of the output file. The default output file name is **a. out**

objects One or more input object files. This must always include:

/lib/init.o

libraries One or more input library files, if not already specified with the **-Llib** option.

Linker Errors

If an error occurs during the link, the link is aborted and no output program is written. Error messages are listed below. Also refer to section 2.8 which describes the global error codes which can be returned by the linker.

Usage: ld [-d] [-bxxx] [-m] <infile> [-lxxx] [-o outfile]

Either an invalid link option was specified, or no object or library files were given.

File open error: name
The object or library file name was not found. Check to see that the file name and path name are given correctly and that the file actually exists.

File read error: name
Likely a disk problem. Try a newly formatted disk.

File write error: name
Either the disk onto which the linker output is being written is full, or there is a physical problem with the disk. Check to see that adequate space for the output is avialable.

Unable to open output file: name
Check to see that the disk is not write protected, and that the path given for the output, if any, is correct. May also be the result of a problem with the disk.

File format error: name
The named input file is not in the correct format or has been corrupted. Assure that only object files and library files are specified.

Undefined symbol(s):
The linker found references to function name(s) or global variable name(s) for which there is no definition. Make sure that the listed globals are actually defined, and that references to library functions are spelled correctly. Note that a leading underscore (_) is added to each global by the compiler and should be ignored by the user.

Duplicate name definition: name
The global name has been defined in more than one place. Eliminate or rename one of the functions/variables.

No name list: file
File is missing symbol table information. Object files must have at least one global name to be linked.

No string table: file
File is missing its string table, a list of the actual names referred to by the symbol table.

The Linking Process

The linker examines each argument in the order given. Object is always included, while libraries are searched by the linker and only those object code modules which are needed are actually included in the final executable program. Since libraries frequently contain many object code modules, the archiver may be used to add an index of global function and variable names to the beginning of a library. Using this index, the linker can quickly resolve external references, greatly speeding the linking process.

The index, if it exists, is loaded into memory and searched repeatedly until either no more undefined names need resolving, or a complete pass of the index is made and no additional object code modules are extracted. If the library does not contain this index, the linker will make only one sequential pass through the library, including code modules only if they are needed. Therefore, without the index, references in the library must refer to object modules which appear further into the file.

Object File Format

A.out is the format of the object file that is created by the compiler. This object file format is the same one that is used by Unix systems. The file has five sections: a header, the program text, the program data, relocation information, a symbol table, and a string table (in that order). The text segment contains the actual machine code for the program, while the data segment contails initialized variables. A segment for uninitialized variables, called the bss segment, is set up at the time the program is run.

Format using the C structure definitions are shown on the next page.

```
/* Header prepended to each object file.
*/
typedef struct {
        long            a__magic;       /* magic number 0x0107          */
        long            a__text;        /* size of text segment         */
        long            a__data;        /* size of initialized data     */
        long            a__bss;         /* size of unitialized data     */
        long            a__syms;        /* size of symbol table         */
        long            a__entry;       /* entry point                  */
        long            a__trsize;      /* size of text relocation      */
        long            a__drsize;      /* size of data relocation      */

/* Format of a relocation datum
*/
typedef struct {
        long            r__address;     /* address which is relocated   */
8/
        unsigned long   r__info;        /* r__symbolnum, r__pcrel, r__length,
*/
                                        /* r__extern                    */
        } relocation__info;

/* Macros to access the r__info field
*/
#define         r__symbolnum(x)         ((x>>8) & 0xffffffL)
#define         r__pcrel(x)             ((x>>7) & 0x1L)
#define         r__length(x)            ((x>>5) & 0x3L)
#define         r__extern(x)            ((x>>4) & 0x1L)
```

If **r__extern** is zero, then **r__symbolnum** is actually the **N__TYPE** (see below) for the relocation rather than an index into the symbol table.

```
/* Format of a symbol table entry.
/*
typedef struct {
        char            *n__name;       /* string table index           */
        char            n__type;        /* type flag, i.e. N__TEXT etc  */
        char            n__other;       /* unused                       */
        char            n__desc;        /* currently not used           */
        long            n__value;       /* value of this symbol         */
        }nlist;

/* Simple values for n__type
*/
#define N__UNDF         0x0             /* undefined                    */
#define N__ABS          0x2             /* absolute                     */
#define N__TEXT         0x4             /* text                         */
#define N__DATA         0x6             /* data                         */
#define N__BSS          0x8             /* bss                          */
#define N__COMM         0x12            /* common (internal to ld)      */
#define N__FN           0x1f            /* file name symbol             */

#define N__EXT          01              /* external bit, ORed in        */
#define N__N-YTPE       0x1e            /* mask for all the type bits   */
```

Object File Format

Object files are composed of up to four sections: a header, the text and data segments, an optional symbol table, and optional relocation information. The header, the first component in the file, specifies the size and starting address of the other components in the object file whichar are listed below.

```
/* header
*/
typedef struct {
        int             c_magic;        /* magic number (0x601A)                */
        long            c_text;         /* size of text segment                 */
        long            c_data;         /* size of initialized data             */
        long            c_bss;          /* size of uninitialized data           */
        long            c_syms;         /* sizeof symbol table                  */
        long            c_entry;        /* entry point                          */
        long            c_res;          /* reserved, always zero                */
        } header;

/* Symbol table entry
*/
typedef struct {
        char            name[8];
        int             type;
        long            value;
        } symbol;

/* values for symbol types
*/
#define         DEFINED         0x8000  /* The symbol is defined                */
#define         EQUATED         0x4000  /* The symbol is an equate              */
*/
#define         GLOBAL          0x2000  /* The symbol is global                 */
#define         EQU_REG         0x1000  /* The symbol is a register             */
*/
#define         EXTERNAL        0x0800  /* The reference is external            */
*/

#define         DAT_REL         0x0400  /* Data segment reference               */
#define         TEX_REL         0x0200  /* Test segment reference               */
#define         BSS_REL         0x0100  /* Bss segment reference                */
*/
```

The above values may be ORed together to indicate symbol type.

One word (16-bit) of relocation information exists for each word of text and data. The type of relocation is indicated in bits 0-2 of the word. If the relocation is an external reference, the remianing bits (15-3) form an index into the symbol table, indicating the name of the external reference.

```
/* relocation word values (bits 0-2)
*/
#define     NO_RELOC        0
#define     DATA_BASED      1
#define     TEXT_BASED      2
#define     BSS_BASED       3
#define     UNDEF_SYMBOL    4
#define     LONG_REF        5
#define     PR_RELATIVE     6
#define     INSTRUCTION     7
```

2.4 LIBRARIAN

Introduction

The Tekelec librarian, **ar**, maintains a group of files combined into a single archive. Its main purpose is to create object file libraries to be used by the linker.

The librarian is compatible with the UNIX program **ar** (file archiver). It also provides the function of the UNIX utility **ranlib**, which creates a dictionary of symbols that the linker uses to speed the process of searching through libraries.

Random Library

The **ar** command includes an option (I) which converts an archive of object files into a random library. This enables the linker to search the archive more efficiently.

The librarian performs this randomization by examining the entire library, collecting global function and variable names, and information about the object modules in which they are defined, and writing a special component into the library. This component, named __SYMDEF, is always the first component of the library.

Always randomize a newly created library. Once randomized, the librarian automatically re-randomizes any library which is changed.

If a library has a __.SYMDEF and it is changed, the librarian automatically recreates the __SYMDEF.

The usage of **ar** is on the following page.

Usage

The **ar** command uses the following syntax:

ar <key> [v] [pos] <afile> <file> [file...]

The **ar** fields are:

key One of the commands listed below.

- **t** List a table of contents of the archive..
- **r** Replace (add) file to the archive. If the archive does not exist, it is created. If an archive component name matches <file>, it is replaced. Otherwise <file> is appended to the end of the archive in the order specified.
- **ra** Same as option **r**, except the replace/add begins after the component in the archive named in [pos]. The file **pos** is first located, then the replace command is executed.
- **d** Delete file from archive.
- **x** Extract copy of file from archive.
- **w** Write the file to the standard output.
- **l** Convert archive into random library. *Always randomize a newly created library.*
- **v** The letter **v** (verbose) can be appended to any of the commands, causing the librarian to print information about the action performed.

pos Used with option **ra**. Indicates where the file is to be archived. **pos** is the file that the new file should follow in the archive.

afile Archive file name.

file One or more file names, used according to **key**.

ERROR MESSAGES

Table 2.4-1 lists the librarian error messages and their meanings. Also refer to section 2.8 which describes the global error codes that can be returned by the librarian.

ERROR MESSAGE	MEANING
Usage: ar...	An invalid key was specified, or not object or library files were specified.
File open error: name	The file name was not found. Check to see that the file name and path are correct, and that the file actually exists.
File read error: name	Try a newly formatted disk.
File write error: name	Disk is full or there is a physical program with the disk. The librarian writes a temporary file, called AR..TMP to the disk. Make sure there is adequate space available on both the librarian disk and the disk on which the library exists.
File create error: name	Unable to create a new library. Make sure that the disk is not write-protected, and that the path for the output is correct. Could also indicate a disk problem.
Temporary file open error	Unable to create the temporary file. There is either a problem witht he disk or the disk from which the librarian is being run is full. Make sure there is adequate space available on both the librarian disk and the disk on which the library exists.
File format error: name	File given is not a library file.
Memory allocation error	Memory is exhausted. Remove RAM disk or cache.
Malformed archive (0xXXX)	The library file is internally corrupt. Create or copy a new file. The hex number given is the address where the librarian expected to find the beginning of a component file, but did not.

Table 2.4-1: Librarian Error Messages

2.5 DISASSEMBLER

Description

The Disassembler, called **dis**, prints the assembly language equivalent of an object code file. This allows you to check the compiler's code generation for errors or determine if it can be improved.

If the file contains symbol information, it is used where possible; otherwise, actual reference values are printed. Since references internal to an object file are resolved by the compiler, there will be instances where no name is associated with a reference. In these instances, the Disassembler makes an educated guess as to the name of a reference, and prints it , rather than a value. All numeric values are printed in hex.

The output of the Disassembler is not directly compatible with the in-line assembly of the compiler, because the compiler inserts extra characters that are not part of the assembler syntax. This additional information can be removed using the vi editor.

The Disassembler uses Motorola mnemonics to print the assembly language equivalent of an object file. The disassembler prints labels as they would appear in an assembly language program by examining the symbol table and the relocation information.

Usage

The disassembler command uses the following syntax:

dis [-n] [-r] [-a] [-i] ofile [ofile . . .]

-n Suppress reference names and addresses. Print actual reference values. Default is for symbol names to be printed.

-r Relative branches. Normally branch instructions, which specify addresses relative to the program counter, are converted to absolute addresses. This option suppresses the conversion. Default is the absolute address of the destination of the branch.

-a Assembly format. Print as an assembly file, suitable for compiling. Often, slight modifications will be necessary before it will compile correctly.

-i Instruction print. The hex value of each instruction is printed before the instruction is disassembled.

ofile An object or executable file

Disassembler Errors

Table 2.5-1 lists the Disassembler error messages and their meanings. Also refer to section 2.8 which describes the global error codes that can be returned by the disassembler.

ERROR MESSAGE	MEANING
Usage : dis ...	An invalid option was specified or no object or program files were given.
File open error: name	The input file name was not found. Check to see that the file name and the path name are correct, and that the file exists.
Memory full while processing	Memory exhausted. Remove RAM disk or cache.
File format error: name	The file name is not an object or program file, or it is corrupt.

Table 2.5-1: Disassembler Error Messages

2.6 EGREP

Description

Egrep searches files for patterns that the user specifies. The patterns are in the form of regular expressions. Normally, each line that matches the user-defined pattern is copied to the standard output. Egrep patterns are extended regular expressions using a fast deterministic algorithm that sometimes needs exponential space. Lines are limited to 1024 characters; longer lines are truncated.

Egrep prints the file name if there is more than one input file.

Usage

Egrep uses the following syntax:

egrep.ttp [-C] [-L] [-V] [-N] [-S] pattern [files]

-C Matching line count.
This option prints the number of lines that matched the pattern.

-L File listing.
This option prints the file names containing matching lines.

-V Print all non-matching lines.
This option prints the lines that do *not* match the pattern.

-N Print line number.
This option prints the line number of the matching line.

-S Silent option.
This option prints only error messages.

[files]

Pattern

Egrep accepts extended regular expressions. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression.

Care should be taken when using the characters $ * [^ | ( ) and \ in the expression, as they may also be meaningful to the shell. It is safest to enclose the entire expression arguments in single quotes ('). In the following description, the term *character* excludes newline:

- \ (back slash) followed by a single character (other than newline) matches that character
- ^ (caret)matches the beginning of a line
- . (period) matches any character
- Any other character matches that character
- A string enclosed in brackets [ ] matches any single character from the string. Ranges of ASCII character codes can be abbreviated, for example, a-z0-9. A right bracket (]) can occur only as the first character of the string. A literal - must be placed where it cannot be mistaken as a range indicator.
- A regular expression followed by an asterisk (*) matches a sequence of *zero* or *more* matches of the regular expression.

  A regular expression followed by plus (+) matches a sequence of *one* or *more* matches of the regular expression.

  A regular expression followed by a question mark (?) matches a sequence of *zero* or *one* matches of the regular expression.
- Two regular expressions concatenated match a match of the first followed by a match of the second.
- Two regular expressions separated by | or newline match either a match for the first or a match for the second.
- A regular expression enclosed in parentheses matches a match for the regular expression. The order of precedence of operators at the same parenthesis level is as follows: [ ] then * + ?, then concatenation, then | and newline.

Examples

If a file named test1 contains the line:

The lazy dog jumped over the the cow

To search for the word **dog** in the file, use egrep as follows:

```
egrep 'dog' test1
```

To list all functions in a file, use egrep as follows:

```
egrep '^([a-zA-Z]|[_]) (a-zA-Z0-9]|[_])*[ ]*\(' test2.c
```

If test2.c is the following program:

```
main()
{
        int i;
        foo();
}
foo()
{
        int i;

main();
}
```

Egrep would print:

```
main()
foo()
```

Egrep Errors

Table 2.6-1 lists egrep error messages and their meanings. Also refer to section 2.8 for a description of global error codes which can be returned by egrep.

ERROR MESSAGE	MEANING
Usage : egrep.ttp...	No pattern or files were given.
Unable to open: name	Egrep cannot open the file name.
Unknown flag: flag	Flag is not used by egrep.
Invalid regular expression	Something is wrong with the regular expression.
Unmatched (	A right parenthesis has been omitted from the expression.
Unmatched)	A left parenthesis has been omitted from the expression.
Premature end of regular expression	The expression finished before it should have.
Nesting too deep	The nesting of parentheses was too great.
Regular expression too big	The expression was too big for egrep to compute.
Memory Exhausted	Egrep ran out of memory.

Table 2.6-1: Egrep Error Messages

2.7 SYMBOL NAMER

Introduction

Object files and application files can contain symbolic information in their symbol tables. This symbolic information can be printed using the Symbol Namer utility.

Each symbol is preceded by its value (in hexadecimal) and one of the following letters:

- A Absolute
- B Bss segment
- C Common symbol
- D Data segment
- T Text segment
- U Undefined symbol

If the letter is lower case, the symbol is local. If upper case, the symbol is global.

Usage

The Symbol Namer syntax is:

nm [file]

file — An executable file which has been linked so that it still has its symbol table.

Error Messages

Table 2.7-1 lists the Symbol Namer error messages and their meanings.

ERROR MESSAGE	MEANING
Usage: nn...	An invalid option was specified or no object or application file was specified.
File open error:	The input file was not found. Make sure that the file name and path name are correct, and that the file actually exists.
File format error: name	The file is not a valid object or application file or program file, or is corrupt.
No name list	The file has no symbol table.

Table 2.7-1: Symbol Namer Error Messages

2.8 GLOBAL ERROR CODES

Description

This table below lists the global error codes which may be returned when you are using C programs, such as the compiler, make, librarian, disassembler, or egrep. Additional error codes are listed in the appropriate section for each program.

ERROR CODE	MEANING
0	Successful (no error)
-100	No command given
-101	Error creating task
-102	Unable to get key
-103	Out of memory
-104	Invalid command line token
-105	Invalid priority
-106	No match on file name expression
-107	openvt error
-108	Ambiguous redirection
-109	Unable to open redirection

Table 2.8-1: Global Error Codes

2.9 BASIC/SITREX/TEXT FILE CONVERSION

Introduction

The Chameleon 32 BASIC/SITREX/TEXT File Conversion utility provides the following file conversion capability:

- Converts a Chameleon BASIC (FRAMEM or SIMP/L) or SITREX program file to a text file
- Converts a text file to a Chameleon BASIC/SITREX file

This gives you the ability to write and edit your BASIC/SITREX programs using the C vi Editor, or a text editor on a PC or other computer. The text file can then be converted to a BASIC or SITREX program file and run on the Chameleon 32.

The Chameleon 32 BASIC/SITREX/TEXT File Conversion utility requires that you have the optional C Package installed on your Chameleon 32. It includes the following programs:

- **totext** Converts a Chameleon BASIC or SITREX file to a text file
- **tobas** Converts a text file to a Chameleon BASIC or SITREX file

General Guidelines

General guidelines for using the conversion utility are listed below. Steps for performing a specific type of file conversion begin on page 2.9-3.

1. Verify that the two conversion programs are in the **\BIN** directory of the Chameleon 32 hard disk drive. They are automatically copied to the correct directory when the system software is installed.

   Your environment (setenv) should contain a search path to the **\BIN** directory so that the conversion programs can be executed from any directory of the hard disk.

2. If you are using vi or another text editor to write your BASIC or SITREX programs, be sure that your text files conform to the necessary syntax rules before you convert them. In the *Chameleon 32 Simulation Manual*, BASIC is described in Chapter 3 and SITREX is described in Chapter 8.

3. BASIC and SITREX files have unique 2-character filename extensions and must be located in specific directories in order to be used. Figure 2.9.1 on the next page lists the extensions and directories for each file type.

FILE TYPE	DIRECTORY	FILE EXTENSION
SITREX	\TEKELEC\SIMULATE\SITREX	.BA
FRAMEM SDLC/HDLC	\TEKELEC\SIMULATE\FBOP	.CB
SIMP/L SDLC	\TEKELEC\SIMULATE\SSDLC	.DB
SIMP/L HDLC	\TEKELEC\SIMULATE\SHDLC	.EB
SIMP/L V.120	\TEKELEC\SIMULATE\V120	.GB
BISYNC	\TEKELEC\SIMULATE\BISYNC	.HB
ASYNC	\TEKELEC\SIMULATE\ASYNC	.IB
FRAMEM DMI	\TEKELEC\SIMULATE\FDMI	.JB
FRAMEM LAPD	\TEKELEC\SIMULATE\FLAPD	.LB
SIMP/L LAPD SIMP/L MLAPD	\TEKELEC\SIMULATE\SLAPD	.MB

Figure 2.9.1: Simulation Directories and File Extensions

3. When executing the conversion programs, you can specify path names so that the BASIC/SITREX files do not have to be copied to a specific directory before converting. However, the amount of typing you do will be minimized if you change to the directory that contains the files you want to convert.

4. Converted files are copied to the same directory containing the BASIC/SITREX file you are converting.

5. Text files can be exchanged between the Chameleon 32 and other computers using the following methods:

   - Files on 3 1/2" MS-DOS floppy diskette can be accessed directly from the Chameleon 32 floppy disk drive or copied to the hard disk.

   - Files stored on other media can be transferred with the Chameleon 32 Kermit File Transfer utility. In order to use Kermit, the other computer must have a Kermit-compatible file transfer program.

     Text files should be transferred using the Kermit ***text mode*** option.

     Refer to the *Chameleon 32 User's Guide, Chapter 10* for more information about Kermit file transfer.

Converting BASIC/SITREX Files to Text Files

The ***totext*** program converts a Chameleon 32 BASIC or SITREX file to a text file so that it can be edited with the C vi Editor or a text editor on another computer.

To convert BASIC/SITREX files to text files, do the following:

1. Copy the files you want to convert to the directory ***a:\usr***.

   BASIC and SITREX files are located in specific directories of the hard disk and have unique 2-character file extensions. This information is listed in Figure 2.9.1 on page 2.9-2.

2. Access the C Shell prompt and change to the directory ***a:\usr***.

3. Execute the ***totext*** program using the following syntax

   totext *file1.ext file2.ext*

   file1.ext **file1** is the BASIC/SITREX program filename of 1 - 8 characters.

   ext is the 2-character file extension specific to each type of file as listed in the Figure 2.9.1 on page 2.9-2.

   file2.ext You can convert more than one file at a time, by delimiting each filename with a space.

   You can also use wildcards to convert more than one file at a time.

4. When converted, the text file will be located in the a:\usr directory. The new file will have the same filename as the BASIC/SITREX file, with the file extension .ED.

   You can edit the text file with the C vi Editor, or use the Kermit File Transfer utility to transfer the file to another computer.

5. After editing the text file, you must convert it back to a BASIC or SITREX file and copy it to the appropriate directory before you can use the program on the Chameleon 32.

Converting Text Files to BASIC/SITREX Files

The ***tobas*** program convert a text file to a Chameleon 32 BASIC or SITREX file. After converting a text file to a BASIC/SITREX file, it must be copied to the appropriate hard disk directory before it can be used. This information is listed in Figure 2.9.1 on page 2.9-2. To convert a text file to a BASIC/SITREX file, do the following:

1. Access the C Shell prompt (!).
2. Copy the text files to the directory ***a:\usr***.

   If they are located on another device, use the Chameleon 32 Kermit File Transfer Utility to transfer them from the other device to the Chameleon 32 hard disk drive. The text file must be transferred using the **Kermit text mode** option.
3. Execute the ***tobas*** program using the following syntax:

   tobas -type file1.ext -type file2.ext

   -type **-type** is the 2-character file extension for the type of BASIC/SITREX file being created.

   For example, a FRAMEM HDLC file must have the extension .cb, therefore if you were converting a text file to FRAMEM HDLC, the type would be **-cb**. The **type** must be one of the following in lower case letters:

IF CONVERTING TO:	USE THE TYPE:
SITREX	-ba
FRAMEM SDLC/HDLC	-cb
SIMP/L SDLC	-db
SIMP/L HDLC	-eb
SIMP/L V.120	-gb
BISYNC	-hb
ASYNC	-ib
FRAMEM DMI	-jb
FRAMEM LAPD	-lb
SIMP/L LAPD SIMP/L MLAPD	-mb

file1.ext **file1** is any valid MS-DOS filename of 1 - 8 characters.

ext is a 1-3 character file extension.

file2.ext You can convert more than one file at a time, by delimiting each filename with a space.

You can also use wildcards to convert more than one file at a time.

4. When converted, the file will be located in the same directory as the text file. The new file will have the same filename as the text file, with the file extension specified in the **type** parameter.

   Before you can run the converted BASIC/SITREX file it must be copied to the appropriate directory and given the correct file extension as indicated in Figure 2.9.1 on page 2.9-2.

3.1 MAKE UTILITY

Introduction

Programmers often divide large programs into smaller pieces. These smaller units are easier to work with on an individual basis, but tracking the relationships and dependencies among the pieces becomes a time-consuming task. As you modify your program, it is difficult to remember which files depend on which others, which files have been modified, and the exact sequence of operations needed to make or test a new version of a program.

make automates a number of program development activities so that you can maintain up-to-date versions of your programs with a minimum of effort.

- Find the name of a specified target file(s)
- Ensure that the files that the target depends on (dependencies) exist and are up-to-date
- Update or create the target to incorporate modifications that have been made to the dependencies since the target was last modified

Makefile

To use make, you create a description file, referred to as a *makefile*, that identifies the target files, the dependencies of the targets, and commands. The information in the makefile enables make to identify the operations necessary to update and compile your program after you make modifications.

Built-In Rules

In addition to the information in the makefile, make maintains a table of built-in rules in a special makefile called SUFFIXES. It uses the information in SUFFIXES to determine which file name suffixes are interesting, and how to transform files with specific suffixes into files with other suffixes.

For example, a rule in the SUFFIXES table could specify that files with a **.c** suffix (C source files) are to be transformed into **.o** (object files). This rule causes C source code files to be compiled.

You can add or modify suffixes and rules in the SUFFIXES table, thus enabling you to define global rules that make will apply to any makefile. Additionally, you can inhibit the use of built-in rules in the SUFFIXES table by entering the **make** command **-r** option, described later in this section.

Macro Feature

make includes a macro substitution facility that enables you to perform string substitution in dependency lines and command strings.

MAKE COMMAND

The **make** command executes commands in a makefile, causing specified target files to be updated or created to reflect changes made to files they depend on.

The **make** command executes the file with the default name **MAKEFILE**, unless a different name is specified.

Syntax

The syntax of the **make** command is shown below. Items enclosed in square brackets [ ] are optional items.

make [opt] target] [macro = value] [Fname . . .]

The **make** command fields are described below.

opt The following options are available:

-I Ignore error codes returned by invoked programs. Alternately, you can ignore error codes, using two other methods:

- Enter **.IGNORE** as a false target in the makefile
- Press **TAB --** (tab followed by a hypen) preceding a command in the makefile

-N No execute mode. Print commands, but do not execute them.

-R Do not use Make Utility built-in rules specified in SUFFIXES. Alternately, you can inhibit the use of the SUFFIXES table by entering **.SUFFIXES**, without a dependency list, as a false target name in the makefile.

-S Silent mode. Do not print command lines before executing. Alternately, you can specify silent mode, using two other methods:

- Enter **.SILENT** as a false target in the makefile
- Enter @ as the first character of a command in the makefile

-P Print all macros and targets

-Q Question up-to-dateness of a target

-X Print a list of all targets in the makefile

target The names of one or more target file names separated by a blank space. If target files are not specified in the **make** command, the target(s) specified in the first line of the makefile are updated/created.

macro = value Define a macro (see page3.1-6)

-Fname The name of the makefile to use. In the absence of this option, **make** looks for the default name of **MAKEFILE**. More than one **-fmakefile** parameter can occur in a **make** ommand.

MAKEFILE STRUCTURE

To use the **make** command, you create a makefile that specifies the target files and the files that depend on them. A makefile contains the following information:

- Entries (targets + dependencies + commands)
- Comments
- Macros

Entries

The entry is the most important part of a makefile. It consists of the target file names, their dependencies, and command lines.

There are two types of entries:

- Dependency lines
- Command lines

The general form of an entry is described below. Note that items in square brackets [ ] are optional items; items in parentheses are mandatory. An ellipsis (...) indicates that more than one like item can be entered.

A dependency line defines the target files and their dependencies (the files that the target depends on). Optionally, a dependency line can contain one or more commands. The form of a dependency line is:

target...:[:] [dependent...] [;command...]

A command line contains a progra name followed by program parameters. *Command lines must begin with a TAB.* The form of a command line is:

(tab)[command...]

The items in a makefile entry are described below.

target The **target** is the name of one or more target files. These are the files that you want updated or created. Target names are strings of letters, digits, periods, and slashes. Multiple target names are separted by blank spaces. Shell metacharacters * and ? are expanded.

dependent The **dependent** is the name of one or more files that the target files depend on. Dependent names are strings of letters, digits, periods, and slashes. Multiple dependent names are separated by blank spaces. Shell metacharacters * and ? are expanded.

: You can use a single colon (:) or double colon (::) to separate the targets from the dependencies. A target name can appear on more than one dependecy line, but all lines that it appears on must be of the same (single or double colon) type.

If a target appears on more than one dependency line, and a single colon is used, only one of the dependency lines can have a command sequence associated with it. If the target requires updating, and a command sequence is specified, it is executed.

If a target appears on more than one dependency line, and a double colon is used, each dependency line can have a command sequence associated with it. If the target requires updating, the associated commands are executed, including built-in rules. The double-colon form is valuable for updating archive-type files.

command A **command** is a program name followed by optional program parameters (any string of characters, excluding a # or carriage return).

Commands can appear on a dependency line or on the line immediately following a dependency line. If a command appears on the dependency line, it is preceded by a semicolon. If a command appears on the line following a dependency line, *the command line must begin with a tab*.

Command lines are executed one at a time, each by its own shell. This is important to remember when using commands that have meaning only within a single Shell process; the results are forgotten before the next line is executed. These types of commands include **cd** and Shell control commands.

A line is printed when it is executed unless the **make** command **-s** option is used or **.SILENT** is entered as a false target name in the makefile.

Commands returning nonzero status cause the make command to terminate unless the **make** command **-i** option is used or **.IGNORE** is entered as a false target name in the makefile.

Some commands return nonzero status inappropriately. Use the make command -i option or begin the particular command with **<TAB>** **<HYPHEN>** in the makefile.

Make remembers embedded newlines and tabs in Shell command sequences. If you write a **for** loop in the makefile with tabs, **make** retains the tabs and backslashes when the commands are displayed. Output can be piped to the Shell and is readable.

Command lines can appear on a dependency line or on the line immediately following a dependency line. If a command appears on the dependency line, it is preceded by a semicolon. If a command appears on the line following the dependency line, thecommand must begin with a tab.

A line is printed when it is executed unless the -S options is used or .SILENT is entered as a false target name in the makefile.

Commands returning non-zero status cause make to terminate, unless the -I option is used or .GNORE is entered as a false target name in the makefile. Some commands return non-zero status inappropriately. For these cases, use the -I option, or begin the particular command with (tab) (hyphen) in the makefile.

Entry logic

The order of your entries in a makefile is significant. Lower level dependencies must be defined before higher level dependencies. For example, If target A depends on B, and target B depends on C, the entries must appear in the following order in the makefile:

```
B:C
A:B
```

This logic causes **make** to update B based on C, before it updates A from B. In order for **make** to update your files correctly, you must use this logic when creating your makefiles.

Comments

The pound sign (#) indicates a comment. All characters, from a pound sign to the end of the line, are ignored. Blank lines and lines beginning with # are ignored totally. Comments can appear on dependency lines or command lines.

Macro Definition

Make also provides a simple macro substitution facility for substituting strings in dependency lines and commands.

A macro line contains an equal sign (=) *which is not preceded by a colon or a tab*. The macro name is the string to the left of the equal sign (trailing blank and tabs are stripped). The macro is assigned the string of characters to the right of the equal sign (leading blanks and tabs stripped).

For example, to define a macro named **PROGRAM** as the three object files, **1.o**, **2.o** and **3.o**, you enter:

```
PROGRAM = 1.o 2.o 3.o
```

You can assign a null string as a macro value by leaving the right of the equal sign blank. For example, to assign a null value to the macro named **ZIP**, enter

```
ZIP =
```

You can also define macros in the **make** command itself.

A macro is invoked using a dollar sign ($) as shown below:

$(macro name) *or* **${macro name}**

If the macro name is a single character, the parentheses or braces are optional. Macro names exceeding one character in length, must be enclosed in parentheses ( ) or braces { }, as shown.

For example, to invoke a macro named **Y**, a single-character name, enter either:

$Y or **$(Y)** or **${Y}**

To invoke a macro named PROGRAM, enter either:

$(PROGRAM) or **${PROGRAM}**

There is also a facility to perform translations when a macro is referenced and evaluated. The general syntax for a macro reference is:

$(macro : string1 = string2)

This causes each occurrence of string1 to be substituted with string2 in the macro being evaluated, where **macro** is the name of the macro being evaluated. All environment variables which are defined as make is executed, become macro definitions in make.

Implicit Macros

If a file is generated using one of the built-in transformation rules, the following macros can be used:

- **$*** Name of the file to be made (excluding the suffix)
- **$@** Full name of the file to be made
- **$<** List of the dependencies
- **$?** List of dependencies that are out of date

Dynamic Dependency

To use these implicit macros, there is a dynamic dependency parameter referenced by the notation:

$$@

It has meaning only when it appears on a dependency line. The **$$@** refers to the item(s) to the left of the colon, which is referenced by the **$@** implicit macro.

The following is an example using implicit macros and the dynamic dependency parameter.

```
PROGS= s1 s2 s3 s4
```

Defines macro PROGS as files s1 - s4.

```
$(PROGS) : e.c
```

Invokes the PROGS macro, defining the target file names as s1, s2, s3 and s4. Defines their dependencies as C source files (.c) with the same filenames: s1.c, s2.c, s3.c, and s4.c.

There is also a second form of the dynamic dependency parameter which refers to the file part of **$@**. This form is referenced using the notation **$$(@F)**.

SUFFIXES TABLE

As mentioned previously, make maintains a table of suffixes and built-in transformation rules in a suffixes table. You can change the table with the .SUFFIXES directive. For example:

```
# Add the suffixes .o and .c to the suffixes table
.SUFFIXES : .o .c
```

When attempting to determine a transformation for a file which has no explicit target mentioned in the makefile, *make* uses the suffixes table. Make looks for a file with the desired suffix, and uses the associated transformation rule to create or update the target file.

Table 7-1 lists the default suffixes in the SUFFIXES file.

SUFFIX	FILE TYPE
.o	Object file
.c	C source file
.r	Ratfor source file
.s	Assembler source file
.y	Yacc-C source grammar
.p	Pascal source
.l	Lex source grammar
.h	Include file

Table 7-1: Default Suffix List

Transformation Rules

A transformation rule name is the concatenation of the two suffixes. For example, the name of the rule that tranforms **.c** files to **.o** files is **.c.o**. For example:

```
# Compile (with CC) a .c file to produce a .o file
.c.o
    cc -c $*.c
```

A transformation rule is used only if the user's makefile does not contain an explicit command sequence for these suffixes.

The order of the SUFFIXES list is significant. **Make** scans the list from left to right, and uses the first name that has both a file and a rule associated with it. To append new names to the suffix list, enter **SUFFIXES** as a special target in your own makefile, listing the new suffixes as dependencies. The dependencies will be added to the suffix list.

For example, to transform a source file into an object (.o) file, make calls up the appropriate compiler. There are also transformation rules to create library (.a) files from source files.

To delete the built-in suffix table, enter .SUFFIXES as a target, without listing any dependents in the makefile. It is necessary to do this to clear the current list, if changes in the order of the suffixes is desired.

EXAMPLES

Some example makefiles are described below.

Example 1: For this example, the SUFFIXES files contains a built-in rule that enables make to compile three source files, x.s, y.s and z.s to generate the needed object files x.o, y.o and z.o. (tab) indicates that you enter a tab character.

```
#Example 1
#
prog: x.o y.o z.o
```
States that the target file **prog** depends on three object files: x.o, y.o, and z.o.

```
(tab) cc x.o y.o z.o -o prog
```
Describes how to load the three object files to create prog. Note that command line begins with TAB.

```
x.o y.o: defs
```
The target files x.o and y.o depend on the header file defs.

Example 2: This example illustrates the use of macros.

```
OBJECTS = x.o y.o z.o
```
Defines the macro OBJECTS to be the three object files x.o, y.o and z.o.

```
LIBES = -lm
```
Defines the macro LIBES as **-lm**.

```
prog: $(OBJECTS)
```
Defines the dependencies of the prog target file as x.o, y.o and z.o by invoking the macro OBJECTS.

```
(tab) cc $(OBJECTS) $(LIBES) -o prog
```
Builds the target prog by loading the three object files with the lm library.

4.1 MACHINE DEPENDENCIES

Data Elements

The C compiler supports all of the standard scalar types of the C language: char, int, short, long, unsigned, float, and double, as well as pointers to all types. Also unsigned char and unsigned long are supported. The amount of space allocated for each data type (in terms of 8-bit bytes) is as follows:

char:	1	unsigned char:	1
unsigned:	2	unsigned long:	4
short:	2	float:	4
int:	2	double:	8
long:	4	*anything:	4

Floating point types are stored in IEEE standard format.

The maximum size of an identifier or string constant is 255 bytes.

Space for variables of type char and short are allocated on the next available byte boundary in memory if the variable is within a `struct` or `union` or is of storage class `auto`, or on the next available word boundary if the variable is `extern` or `static`. Space for all other variables, including those of any other storage class as well as arrays, struct's and union's, is always allocated on the next available word boundary, regardless of storage class. Bit fields within struct's are allocated in unsigned units, starting from the least significant bit.

External Names

Identifiers (names of variables and functions) may contain up to 255 characters each. Only the first ten characters are used to distinguish one identifier from another, however. As per the standard for the C language, both upper and lower case letters are allowed in identifiers, and are distinct from each other. In other words, the names `myvar` and `MyVar` are different. The underscore character (_) is also legitimate within identifiers, as are digits. The only restriction is that an identifier may not begin with a digit. It should be noted that various internal functions, such as floating point routines and support for long integers, have names beginning with an underscore. Programmers should therefore avoid extern identifiers beginning with an underscore if possible.

Include File Processing

The `#include` feature of the standard C preprocessor allows file names to be given within either double quotes or angle brackets. Angle brackets will cause the compiler to look in "include". Double quotes cause the compiler to look first in the directory containing the `".c"` file being compiled and then to look inthe predefined places.

Include files may be nested to a depth of 6 levels, including the main module level. An attempt to nest beyond this maximum (such as would be the case if an include file inadvertently `#included` itself) results in an error message.

Floating Point

All floating point operations in Tekelec C can be carried out in either single (32 bit) or double (64 bit) modes. The single precision mode is the default and is about three times faster than double mode.

Register Variable Support

Each function in a C program can expect up to three registers available for register storage class variables. One data registers is available for integral types (char, short, int, long, and unsigned), and two address registers are available for pointer variables. Judicious use of register variables can substantially increase execution speed and decrease code size.

4.2 COMPILER PROCESSING

Error Processing

Error messages generated during compilation are reported to the screen, accompanied by the line of source code containing the error. Error messages are of the form

```
"file-name". line line-number: error message text
```

To simplify correction of errors in a program, error messages may be redirected to a file (see Shell: I/O redirection). This file may be used while editing the source to correct mistakes.

Code Generation

The C compiler, including preprocessor, syntax check, and code generation, is one-pass. In other words, all work which needs to be done by the compiler is finished after looking at the contents of the source file once. The compilation process is thus quite fast.

Linkable object code is generated directly by the compiler; there is no assembly post-pass. C performs many processor specific "strength reduction" optimizations, such as using MC68000 "quick" instructions, replacing multiplies and divides by powers of two with shifts, and avoiding intermediate register loads when possible. Simple statements, such as increments and assignment operations involving constants, frequently generate only one machine instruction.

For example, the statement

```
i++;
```

compiles into a single instruction to increment the variable i. The statement

```
i = 50;
```

will compile to a single MOVE instruction.

Certain expressions involving constants will be evaluated at compile time.

Therefore, the statement

```
i += 5 * ARRAYSIZ;
```

will generate one ADD instruction, assuming ARRAYSIZE is a constant which was #defined.

4.3 RUN-TIME PROGRAM STRUCTURE

Each program is executed under MTOS-UX as a task. As a task is initiated, it creates a "virtual terminal" on the Chameleon 32 screen through which standard (terminal) input/output is done. Calls to C memory allocation routines (`malloc` and `calloc`) allocate memory from an MTOS-UX memory pool which is created when the Chameleon 32 is booted. All tasks allocate and deallocate from this pool (pid is "POOL"). As a task terminates, it is killed and all memory allocated by the program (through`malloc` and `calloc`) will be returned to the pool. Programs may call MTOS-UX memory pool management routines directly, but must assume responsibility for resource disposal.

C stores all string constants with a terminating null byte, as per the standard for the C language.

System Library

All code for functions from the system library is included in each executable program by the linker.

Program Entry/Exit

When a program is linked after compilation, an object module containing startup code is automatically included by cc. The following declaration will allow program parameters:

```
main (argc, argv)
int      argc;
char     *argv[];
```

where `argc` is the number of strings in the `argv` array. `Argv[0]` is always the program name. If you do not need program parameters, just declare `main()` without any parameters and the linker will not include the code to handle them.

Function Call Conventions

Parameter expressions encountered in function calls are evaluated and then passed to the function on the stack. The parameters are pushed in the reverse of the order given in the parameter list. Reversal of the parameter list is necessary for functions with variable numbers of parameters. Such functions may access lists of parameters as follows:

```
max(n, p);
/* Return max of list of into; n gives list length */
int n, p, {
     int *pp, max = -32768;
     for (pp=&p, n, pp++, n--)
          if (max < *pp)
               max = *pp,
     return max;
}
```

The above function `max()` returns the maximum of an arbitrary number of integers. The number of integers is passed as the first parameter, followed by the list of values:

```
m = max(5, i, j, k * 2, 87, f(abc));
```

Note that the pointer variable `pp` is incremented in the `for` loop of the above function. The pointer will move down through the stack towards higher memory locations retrieving each parameter in turn. Any functions which use this method of obtaining parameters are not necessarily portable to other implementations of C.

Values are returned from functions in processor register DO. It is the responsibility of the calling environment to remove parameters from the stack after return from a function call. Each function must ensure that any registers used to hold register variable values are saved and then restored when the function terminates.

Structs may be passed by value.

4.4 LIBRARY IMPLEMENTATION

Line Separators

Because of the heritage of the C language, the ASCII line feed character (numerically, 10 decimal) is usually considered to be the line separator character.

In text files, a line feed separates lines, however, upon output to the screen, line feeds are automatically converted to carriage return/line feed pairs.

Memory Allocation

The memory allocation routines `malloc()` and `calloc()` are available to the C programmer. To avoid excessive fragmentation of the common memory pool, memory is allocated in 8 KB blocks, breaking up the blocks as necessary to satisfy the requests made from the C program. The `free()` routine will coalesce space which is returned and the allocation system will reuse deallocated space.

Note that because pointers are 32 bits long, a C program can use as much memory as is available on the machine through dynamic allocation.

IMPORTANT NOTE: you must make the declaration:

```
extern char *malloc();
```

in your program before you use `malloc` (the same is true for calloc). If you don't do this the compiler will assume `malloc` returns an `int` (which is only 16 bits wide).

Also note that malloc requires an unsigned int as its parameter. If more than 64 Kbytes of memory is needed, MTOS-UX memory allocation routines may be used (see malloc (), alloc ()).

4.5 LANGUAGE EXTENSIONS

ASSEMBLER

Introduction

The Tekelec C compiler allows the addition of assembly language code to a C program directly in-line with the C code. The C language has been extended to include the construct:

```
asm {
    . . .
    MC68000 Assembler Instructions
    . . .
}
```

The code within the braces after the keyword `asm` is assembled and included in-line with code generated from surrounding C statements.

The in-line assembler obviates the need for a separate assembler. General control structure, input/output, and complex data structures can be implemented in C, while certain low-level routines can be coded in assembly language within the same module. The problem of interfacing C functions to assembly language functions and vice-versa is eliminated, because calling sequences can be written in C for functions coded in assembler. Programs can first be developed in C to debug algorithms and to generate quickly a working prototype. Functions which comprise the most time consuming sections of the program (generally less than 10% of the code) can then be re-coded in assembly language. Because of the efficiency of the C code generator, such a hybrid approach yields execution speeds favorably comparable with pure assembly language code while retaining the ease of modification and maintenance of a pure high-level language approach.

Use of assembly language decreases readability, exacerbates debugging headaches, and drastically reduces portability. Discretion must be used when considering functions for hand translation. There are some situations where speed is critical, most notably graphics. Such applications frequently involve system or machine dependencies anyway, so portability is not an issue. In such cases, the availability of in-line assembly language is a great benefit.

Syntax

The general syntax for in-line assembly language follows. `{stuff}` means stuff is repeated one or more times. `(choice1|choice2|. . .)` means one of the choices must appear. `[stuff]` means stuff is optional.

```
<asm-statement> --> asm {{<asm-line>}}
<asm-line> --> {<label>:|<op-code>[.(B|W|L|S)][<operand>
     [,<operand>]]}[(;<comment>|/*<comment>*/)]
<operand> --> (D<n>|A<n>|(A<n>)+|-(A<n>)|<disp>(A<n>)|
     <disp> (A<n>,<ix>)|<constrxpr|[.W|L]|<disp.(PC,<ix>|
     #<constexpr>|<reglist>|CCR|SR|USP)
<disp> --> (<identifier>[(+|-)<constexpr>]|<constexpr>)
<ix> -->(A<n>|D<n>)[.(W|L)]
<n> -->(0|1|2|3|4|5|6|7)
<reglist> --> (<reggroup>)/<reggroup>
<reggroup> --> (A<n>|D<n>)[-A<n>|D<n>)]
```

The syntax of the in-line assembler is almost identical to that described in the Motorola 68000 manual. Exceptions are noted below.

In-line assembly may appear anywhere in your program; it is not necessary to place it inside a function. Please note that `<identifier>` is the same as a C identifier, and `<constexpr>` is the same as the C constant expression. Opcodes are the same as in the Motorola literature and may be given in upper or lower case. The size modifiers `B`, `W`, `L`, and `S` can also be given in upper or lower case. The register names are defined only in uppercase. Expansion of `#defined` macros is performed within sections of assembly language, so the programmer is free to rename instructions or registers.

Each line of assembly language may consist of one or more instructions, optionally followed by a semicolon and comment text. Comments may also be given as C comments. Note that `#defines` can be used to create simple macros, using the multiple statement per line feature. Within macros, C style comments must be used instead of the normal semicolon-to-end-of-line assembly language comments.

Expressions which give displacement values are restricted in that only one identifier may be involved. A constant expression may be added to or subtracted from this identifier. In such expressions, the identifier must be placed first in the expression; in other words, the statement

```
MOVE D0, x+2(A4)
```

is legal, but the instruction

```
MOVE D0, 2+x(A4)
```

is not.

The application of addressing modes to instructions is not completely orthogonal in the MC68000 instruction set. For

complete information on addressing modes and instruction forms, consult a Motorola databook.

Defaults

If no size specifier is given for an instruction which can operate on more than one size, the assembler defaults to word. If a size specifier is not applicable to a particular instruction, no specifier may be given. All labels given default to local code labels unless declared as something else previously. This means that all functions called, for example, must be declared or defined previously in C.

Branches default to word-sized displacements. The code improver will change the word branches to short branches where possible. A short branch can be forced by using a `.s`, but no warning message will be given if the necessary displacement is too large for a short branch.

Accessing C Objects

External and static variables from the C environment are accessed using the name of the variable. Auto variables are accessed using the name of the variable as the displacement from the A6 register (the Address Register Indirect with Displacement mode). Register variables may also be accessed by name. The first four non-pointer register variables are placed in data registers; the first two pointer register variables are placed in address registers. Any excess register variables must be accessed relative to A6. The assembler will not report misuse of any variable names.

Functions in the C program can be referred to by name. Arguments are passed to functions on the stack in reverse of the order they are written in C. Values are returned from functions in data register DO, or in `global_fpreg0` if the value is double.

Available Registers

Registers D0-D3 and A0 and A1 may be used without saving them. Registers D4-D5, A2, and A3 are used for register variables, and are allocated in reverse numeric order. Each of these registers not used for a register variable within a function containing in-line assembly language must be saved by the assembly code if modified therein. Register A6 is used to access auto variables.

Creating Global Symbols

This section is not for the novice user of the in-line assembly and discusses the use of a construct that is very dangerous. It is almost never needed and should be avoided if at all possible.

The normal functions in C start with a link instruction to make room for local variables and then end with a corresponding unlink instruction. These instructions can be avoided by making a label inside assembly to be called instead of the C function name. A `rts` instruction must also be placed at the end of the routine to avoid the unlink instruction. To indicate that this is an extern or static symbol it must be so declared before it is used as a label. This is done by declaring it as an extern or static function in C. Remember, by overriding the normal entry point a lot of nice things that C does about parameter passing and setting up local variables is lost.

Assembly Language Example

```
/*
     Function to do a block move from the first pointer to the. The routine moves
     one char at a time to allow odd addresses.
*/

block_move (source, dest, count)
register char *source, *dest; /* uses address registers */
register int count; /* placed in a data register */
{
     asm {
          subq    #1, count ;because dbf counts to -1
     lp : move.b  (source)+, (dest)+
          dbf     count, lp
     }
}

/*
     An example of a macro to use in assembly language
*/

#define INC(x) addq #1, x
```

Structure Assignment

C supports structure and union assignment and passing. If x and y are structures of type `stype` then the following statements are legal:

```
x = y ;              /* contents of y are copied to x */
foo(x);              /* x is passed by value to foo() */
struct stype bar(); /* function returning struct */1
```

Character Constants

The definition of character constants has been extended in C to allow int and long size as well as char. The syntax is a single quote followed by 1, 2 or 4 characters and a closing single quote. The resultant type will be a char, int or long respectively.

Scope of Identifiers

In general, name scoping within the C compiler is as per standard C. One exception to this standard is the treatment of identifiers of structure members. In Tekelec C, structure member names need not be unique across struct boundaries. Therefore it is valid for two different structures to contain members at different relative offsets with identical names.

A restriction imposed by the one-pass nature of the Tekelec C compiler is that static functions must be declared before the first reference in a program. The declaration need not be the definition of the code of the function. A simple declaration such as

```
static my-func();
```

will do.

Forward Pointer References

A problem arises when two structures must refer to each other: the reference in the first structure causes an undefined type error because the second structure hasn't been defined yet. This mutual referencing almost invariable arises with some kind of linked data structure. The Tekelec C compiler has been extended to allow pointer references to structs or unions that have not yet been defined. Note that this only works with pointers to structs or unions with a tag name (typedefs will not work). Additional errors will be generated later in the compile if the struct or union is never defined.

5.1 LIBRARY INDEX

Introduction

The functions described in this chapter are compatible with functions by the same names which are available to C programmers using the UNIX operating system. Most of these routines are available in all C implementation; even those on microcomputers without UNIX. Therefore, use of these functions simplifies the task of porting a C program to another computer.

File I/O

The system library contains routines for buffered and unbuffered input/output to disk files. Buffered routines, for the stream file interface, begin with the the letter *f*. The unbuffered routines are the low-level read() and write() routines. Both levels of I/O allow random access to disk files. Along with these routines you can use the BIOS routines for input/output.

Stream I/O

A stream file is a pointer to a FILE data structure declared in the head file STDIO.H. Each stream is associated with a regular file via a file descriptor returned by open or creat. Streams buffer data through the file descriptor so that single character I/O is efficient. To increase speed, you can change the default buffer size (512 bytes) using the setbuffer call. Streams provide a larger number of functions than the Basic I/O level.

Three streams are open when a program start:

- stdin — Open for reading only, and is connected to the keyboard (file descriptor 0).
- stdout — Open for writing only, and is connected to the screen (file descriptor 1).
- stderr — Open for writing only, and is connected to the screen (file descriptor 1).

I/O Redirection

I/O redirection is a mechanism where stdin and stdout are changed from using the keyboard and screen, to using files, as follows:

- stdin — Changed by passing *<INFILE* on the command line.
- stdout can be changed two ways:
  - *>OUTFILE* opens and erases outfile
  - >>OUTFILE appends to an existing outfile

You do not have to change the program for I/O redirection to work, although you must declare the parameters argc and argv for main().

Device I/O

All system devices are available to you through the C input/output system. For most device input/output, it is wise to use setbuf() to prevent buffering on the stream connected to the device.

When using the unbuffered input/output services, the only significant flag in the mode word is the binary (O_BINARY) flag. If this flag is set, there will be no special treatment for line separator characters. Note that you cannot creat() a device.

You can use BIOS routines to manipulate devices, but these routines require the file descriptor number. This number is the fileno(), defined in <stdio.h>, of the stream or the file number returned by open().

Memory Allocation

The memory allocation routines are malloc() and calloc(). The free() routine coalesces space which is returned, and the allocation system will reuse deallocated space.

Program begin execution with 8 Kbytes of stack space available. This is sufficient for more applications. The C compiler, for example, uses less than 5 Kbytes. The size of the stack may be changed by declaring global variable _stksize and initializing that variable to the size of the stack required. For example:

```
long _stksize = 16384L
```

Since pointers are 32 bits long, a C program can use as much memory as is available on the machine through dynamic allocation.

Note

Before you use malloc or calloc, you must make the declaration:

```
extern char *malloc()
extern char *calloc()
```

If you fail to do this, the compiler will assume that malloc() or calloc() returns an int, which is only 16 bits wide. The declaration is included in <stdio.h>.

Program Parameters

Program parameters passed from the shell are available through the *argc* and *argv* program parameters to main(). For example:

```
main(argc, argv, envp)
int    argc;
char *argv[];
char *envp[];
```

argc is the number of strings in the *argv* array. *argv[0]* is not defined. If you do not need program parameters, declare main() without parameters, and the link will not load the code to retrieve them.

envp is a pointer to a NULL terminated list of environment variables from the previous program, and is optional.

Library Index

The system library functions are listed alphabetically on the next page. On page 5.1-5 they are listed in functional groups. Detailed descriptions of each function are provided in section 5.2.

The C system library functions are listed alphabetically below. Refer to the page number indicated for a detailed description. The functions are listed by function on the next page.

This section lists the functions in the C Library by function. Refer to the page number indicated for a detailed description.

	Command	Page	Description
Basic I/O	close	5.2-9	Close a file
	creat	5.2-10	Create a file (old method--use *open*)
	lseek	5.2-34	Reposition file
	open	5.2-38	Open a file
	read	5.2-49	Read data from file
	unlink	5.2-62	Delete a file
	write	5.2-63	Write data to file
Stream I/O	clearerr	5.2-8	Remove error state
	fclose	5.2-14	Close a stream
	feof	5.2-16	Test end of file
	ferror	5.2-15	Test for error
	fflush	5.2-17	Write buffer to disk
	fgetc	5.2-18	Fast read byte
	fgets	5.2-19	Read string
	fileno	5.2-20	File associated with stream
	fopen	5.2-21	Open a stream
	fprintf	5.2-39	Formatted write
	fputc	5.2-22	Write byte
	fputs	5.2-23	Write string
	fread	5.2-24	Read data from stream
	freopen	5.2-21	Use different file with stream
	fscanf	5.2-53	Formatted read
	fseek	5.2-26	Reposition stream
	ftell	5.2-27	Report position
	fwrite	5.2-28	Write data to stream
	getc	5.2-29	Read byte
	getchar	5.2-30	Read byte from stdin
	gets	5.2-31	Read string from stdin
	getw	5.2-32	Read word
	printf	5.2-39	Formatted write to stdout
	putc	5.2-43	Fast write byte
	putchar	5.2-44	Write byte to stdout
	puts	5.2-45	Write word (integer) to the output stream
	putw	5.2-46	Write string to stdout
	rewind	5.2-52	Reposition stream to front
	scanf	5.2-53	Formatted read from stdin
	setbuf	5.2-56	Set buffer (standard size)
	setbuffer	5.2-56	Set buffer (any size)
	setlinebuf	5.2-56	Set buffer mode
	sprintf	5.2-39	Formatted write to array
	sscanf	5.2-53	Formatted read from array
	ungetc	5.2-61	Put byte back on stdin
Conversion and Classification	atof	5.2-4	ASCII to float
	atoi	5.2-5	ASCII to int
	atol	5.2-5	ASCII to long
	isalnum	5.2-33	Test for alphanumeric

	isalpha	5.2-33	Test for letter
	isascii	5.2-33	Test for ASCII
	iscntrl	5.2-33	Test for control character
	isdigit	5.2-33	Test for digit
	islower	5.2-33	Test for lower case
	isprint	5.2-33	Test for printable character
	ispunct	5.2-33	Test for punctuation
	isspace	5.2-33	Test for white space
	isupper	5.2-33	Test for upper case
	isxdigit	5.2-33	Test for hex digit
	strtol	5.2-5	ASCII (any base) to long
	toascii	5.2-60	Int to ASCII
	tolower	5.2-60	Byte to lower case
	_tolower	5.2-60	Fast tolower
	toupper	5.2-60	Byte to upper case
String Functions	index	5.2-58	Find byte in string
	rindex	5.2-58	Find byte from end
	strcat	5.2-58	Append strings
	strcmp	5.2-58	Compare strings
	strcpy	5.2-58	Copy string
	strlen	5.2-58	Lenght of string
	strncat	5.2-58	Append *n* bytes
	strncmp	5.2-58	Compare *n* bytes
	strncpy	5.2-58	Copy *n* bytes
	xtrcat	5.2-58	Append, but return end
	xtrcpy	5.2-58	Copy, but return end
	xtrncpy	5.2-58	Copy *n* bytes, return end
Memory Allocation	alloca	5.2-3	Allocate on stack
	bcmp	5.2-6	Compare two blocks of memory
	bcopy	5.2-6	Copy ablock of memory to another block
	bzero	5.2-6	Zeroes a block of memory
	calloc	5.2-7	Allocate and clear
	free	5.2-25	Release memory
	lcalloc	5.2-7	Allocate a lot and clear
	lmalloc	5.2-36	Allocate lots of memor
	lrealloc	5.2-50	Resize a lot of memoryy
	malloc	5.2-36	Allocate memory
	realloc	5.2-50	Resize allocated memory
Miscellaneous	abs	5.2-2	Absolute value of int
	longjmp	5.2-35	Non-local goto
	execl	5.2-11	Executes a file
	execv	5.2-12	Execute a file
	exit	5.2-13	Terminate program
	onexit	5.2-37	Adds logic to exit function
	perror	5.2-42	Displays system error message
	qsort	5.2-47	Quick sort
	rand	5.2-48	Random number
	rename	5.2-51	Rename a file on disk
	setjmp	5.2-57	Non-local label
	srand	5.2-48	Start random sequence

5.2 C LIBRARY DESCRIPTION

This section contains detailed descriptions of the standard C functions supported by the Chameleon 32 C Development System compiler. These functions are defined in the file **libc.a** and are listed in alphabetical order. Refer to Section 5.1 for a list of the functions by page number.

abs

Declaration

```
#include <stdio.h>

int abs (i)
int i;
```

Description

abs returns the absolute value of the number that is the parameter.

alloca

Declaration

```
char *alloca (size)
unsigned int size;
```

Description

This function allocates *size* bytes of space in the stack frame of the calling function. This space is temporary and is automatically released upon the return of the calling function.

alloca does not check for stack overflow. The size of the stack is set to the value in extern long __stksize when the program starts (default is 8 kbytes). __stksize should be redefined if more space is needed,

See Also

malloc, free, calloc

atof

Declaration

```
double atof (nptr)
char *nptr;
```

Description

This function converts a character string pointed to by *nptr* to a double-precision floating-point number. The first unrecognized character ends the conversion. *atof* recognizes an optional string of white-spaced characters, then an optional sign, then a string of digits optionally containing a decimal point, then an optional E or e followed by an optionally signed integer.

Returns

If the string begins with an unrecognized character, then a zero is returned

atoi, atol, strtol

Declaration

```
int atoi (str)
char  *str;

long atol (str)
char  *str;

long strtol (str, ptr, base)
char  *str;
char  **ptr;
int   base;
```

Description

These functions convert strings to integers.

strtol returns as a long integer the value represented by the character string *str*. The string is scanned up to the first character inconsistent with the *base*. Leading white-space characters are ignored.

If the value of *ptr* is not (char **) NULL, a pointer to the character terminating the scan is returned in **ptr*. If no integer can be formed, **ptr* is set to *str*, and zero is returned.

If *base* is positive and not greater than 36, it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "0X" is ignored if base is 16.

Truncation from long to int can take place upon assignment or by an explicit cast.

atol takes the ASCII representation of a number and converts it into a long integer.

atoi takes the ASCII representation of a number and converts it into an integer.

bcmp, bcopy, bzero

Declaration

```
int bcmp(block1, block2, len)
char  *block1, *block2;
int len;

int bcopy(source, destin, len)
char  *source, *destin;
int len;

int bzero(block1, len)
char  *block1;
int len;
```

Description

These functions perform operations on blocks of memory.

bcmp compares two blocks of memory *block1* and *block2*. The size of the blocks is *len*. A value of 1 is returned if they are identical.

bcopy copies the *source* block of memory to the block of memory pointed to by *destin*. Both blocks are of size *len*.

bzero zeroes the memory pointed to by *block1*. The block is of size *len*.

calloc, lcalloc

Declaration

```
char *calloc (nelem, elsize)
unsigned int  nelem, elsize;

char *lcalloc (nelem, elsize)
unsigned long  nelem, elsize;
```

Description

calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

lcalloc is like *calloc* but accepts long parameters.

See Also

malloc, free, alloca

Returns

Return a null pointer if there is no available memory.

clearerr

Declaration

```
#include <stdio.h>

clearerr (stream)
FILE *stream;
```

Description

This function resets the error indicator and EOF indicator to zero on the named *stream*. This function is implemented as a macro and therefore cannot be declared or redeclared.

See Also

feof, ferror, fileno

close

Declaration

```
int close (fildes)
int fildes;
```

Description

This function closes a file. *fildes* is a file descriptor obtained from *creat* or *open*. *close* will fail if *fildes* is not a valid, open file descriptor.

See Also

creat, open

Returns

0 = Successful
-1 = Error

creat

Declaration

```
int creat (fname, oflag)
char *fname;
int oflag;
```

Description

This function creates a new file or writes to an existing one. If the file exists, the length of the file is reduced to 0.

If successful, the file descriptor is returned and the file is opened for writing. The file pointer is set to the beginning of the file.

oflag may be set to O_BINARY to indicate the untranslated mode. No other flag values are allowed here (see *open*).

creat will fail if an OS error occurs.

No process may have more than 12 files open simultaneously.

This function has been superceded by *open* with the O_CREAT flag.

See Also

close

Returns

File descriptor (non-negative integer) = successful

-1 = Error

execl

Declaration

```
execl (name, arg0, arg1, . . . , argn, 0L)
char *name;
char *arg0, arg1, . . . argn;
```

Description

This function executes a file. PATH is not evaluated from execl. For example, from the shell, you execute the *cp* (copy) program, as follows:

%cp x y

where: **cp** is arg0, **x** is arg1 and **y** is arg2.

In a program, you use *execl* to execute the *cp* program, by entering the following:

execl("\\bin\\cp" ,"cp" ,"x" , "y" , 0L)

where: **\\bin\\cp** is the path, **cp** is arg0, **x** is arg1, and **y** is arg2.

See Also

execv

execv

Declaration

```
int execv (pathname, argv)
char *pathname, *argv[ ];
```

Description

execv executes a program from the disk. The parameter *pathname* is a pointer to a string which contains the path and the name of the program to be executed.

The *argv* parameter is necessary only if the program being started has arguments to main(). *argv* is an array of character pointers to strings, creating an argument list that is made available to the new program. When used, at least one argument must be present in this array, with the first element of the array being the name of the executed program. For more information, refer to the Program Parameters section at the beginning of Chapter 5.

The parameter *envp* is also an array of character pointers to strings which are not command line arguments, but system environment variables.

When the executed program begins, it is called as follows:

```
main(argc, argv, envp)
        int argc;
        char *argv[];
        char *envp[];
```

where *argc*, the arg count, is the number of elements in *argv*, and *argv* is the array of character pointers to the arguments themselves.

The parameter *envp* is a pointer to an array of strings which are the environment variables from the calling program. Note that a pointer to this array is also stored in the global variable *extern char **environ*. Each string consists of a name, an = sign, and a null-terminated value. The array of pointers is terminated by a null pointer. The result from *execv* is the exit code or status of the program.

See Also

execl

exit, _exit

Declaration

```
exit (status)
int  status;

_exit (status)
int status;
```

Description

These functions terminate a process.

exit performs the following cleanup operations before terminating the program:

- The *onexit* functions are called in the reverse order in which they were added
- All open streams are flushed and closed
- All remaining file descriptors (opened with *open* or *creat)* are closed
- _exit is called

_exit terminates the program immediately without performing any cleanup operations.

fclose

Declaration

```
#include <stdio.h>

int fclose (stream)
FILE *stream;
```

Description

This function writes any buffered data to disk and closes a stream. It is called for each open stream by *exit*.

See Also

fflush

Returns

0 = Successful
EOF = Unsuccessful

ferror

Declaration

```
#include <stdio.h>
int ferror (stream)
FILE *stream;
```

Description

This function returns a non-zero when an I/O error has previously occurred reading from or writing to the named *stream*. Otherwise a zero is returned. This function is implemented as a macro and therefore cannot be declared or redeclared.

See Also

clearerr, feof, fileno

feof

Declaration

```
#include <stdio.h>
int feof (stream)
FILE *stream;
```

Description

This function returns a non-zero when EOF has previously been detected reading the named input *stream*. Otherwise zero is returned. This function is implemented as a macro, and therefore cannot be declared or redeclared.

See Also

clearerr, ferror, fileno

fflush

Declaration

```
#include <stdio.h>

int fflush (stream)
FILE *stream;
```

Description

This function writes any buffered data to disk and clears the input buffer, but does not close the stream.

See Also

fclose

Returns

0 = Successful
EOF = Unsuccessful

fgetc

Declaration

```
int fgetc (stream)
FILE *stream;
```

Description

This function returns the next byte from the named input stram and positions the pointer ahead one byte in the stream. *fgetc* performs the same function as *getc*, however it is a true function. It is slower, but takes less space per invocation.

EOF is returned when end-of-file or error is encountered.

See Also

getc, getchar, getw

fgets

Declaration

```
#include <stdio.h>
char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

Description

This function reads characters from the stream into an array pointed to by *s*, until *n*-1 characters are read, or a new-line character is read and transferred to s, or an EOF is encountered. The string is terminated with a null character.

See Also

gets

Returns

s = Successful

If EOF is encountered and no characters have been read, then no characters are transferred to s and a null pointer is returned.

If an error occurs, a null pointer is returned. Attempting to use this function on a file that has not been opened for reading, causes an error.

fileno

Declaration

```
#include <stdio.h>
int fileno (stream)
FILE *stream;
```

Description

This function *fileno* returns the integer file descriptor for the named *stream*. This function is implemented as a macro and therefore cannot be declared or redeclared.

See Also

clearerr, feof, ferror

fopen, freopen

Declaration

```
#include <stdio.h>
FILE *fopen (file_name, type)
char *file-name, *type;

FILE *freopen (file_name, type, stream)
char *file_name, *type;
FILE *stream;
```

Description

fopen opens the file named by *file_name* and associates a stream with it. It returns a pointer to the *FILE* structure associated with the stream. *file_name* points to a character string that contains the name of the file to be opened. *type* is one of the following:

r	Open for reading
w	Truncate or create for writing
a	Append; open or create for writing at end of file
r+	Open for update (reading and writing)
w+	Truncate or create for update
a+	Random open for read or write; pointer will be repositioned to end of file for writing

freopen substitutes the named file in place of the open stream. The original stream is closed whether the open succeeds or not. *freopen* returns a pointer to the FILE structure associated with stream. It is typically used to attach the pre-opened streams associated with *stdin*, *stdout*, and *stderr* to other files.

If a file is open for update, both input or output may be attempted on the stream. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek*, *rewind*, or an input operation which encounters end-of-file.

Files open for append cannot have information overwritten. All output is appended to the end of file regardless of the current pointer position. After output is completed, the pointer is positioned at the end of the file.

fopen can be used to direct output to Chameleon 32 devices: .AUX (Serial Port 2 unformatted data), .TTY (Serial Port 2 formatted data), and .PRT (printer). For example:

```
FILE *fp;
fp = fopen (".AUX", "w");
fprintf (fp, "This is unformatted output to Serial Port 2");
```

Referencing the Chameleon hard disk directories requires the use of double back slashes as shown in the following example:

```
fopen("a:\\usr\\hisfile","r");
```

Returns

If unsuccessful, these routines return a NULL pointer.

fputc

Declaration

```
#include <stdio.h>
int fputc (c , stream)
char c;
FILE *stream;
```

Description

fputc writes the character *c* to the output stream at the current pointer position. It is similar to *putc* but it is a true function, it is slower, and takes less space per invocation.

See Also

putc, putchar, putw

Returns

If successful, the value written is returned.

If unsuccessful, EOF is returned. This can occur if the file is not open for writing, or if the output file cannot be grown.

fputs

Declaration

```
include <stdio.h>
int fputs (s, stream)
char *s;
FILE *stream;
```

Description

This function writes the null-terminated string, pointed to by s, to stream. The string is not followed by a new-line character. It does not write out the terminating null character.

See Also

puts

Returns

EOF is returned if an error occurs. This will happen if output is attempted to a file not open for writing.

fread

Declaration

```
#include <stdio.h>
int fread (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;
```

Description

This function is for binary input. It places into an array *nitems* of data read from the input stream beginning at *ptr*. The data items are a sequence of bytes of length *size*.

Reading is stopped when an error occurs, end-of-file is encountered, or *nitems* of data have been read. *fread* places the pointer, if any, at the byte following the last byte read, if one exists. The contents of the stream are not changed.

Note

fseek or *rewind* must be called before switching between reading and writing on a stream that allows both.

See Also

fwrite

Returns

Returns the number of items read. If a non-positive number is given for *nitems*, then a 0 is returned and nothing is read.

free

Declaration

```
free (ptr)
char *ptr;
```

Description

This function makes space, pointed to by *ptr* (and formerly allocated by *malloc*, *lmalloc, calloc or lcalloc*,) available for further allocation. *free* does not affect the contents of the space.

See Also

malloc, lmalloc, calloc, lcalloc, alloca

fseek

Declaration

```
#include <stdio.h>
int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;
```

Description

This function sets the position of the next input or output operation on the stream. The new position is at the signed distance *offset* bytes from the beginning, from the current position, or from the end of the file, depending on the value of *ptrname*. *ptrname* has the following values:

0	Offset from beginning of file
1	Offset from current position in file
2	Offset from end of file

fseek undoes the effects of *ungetc*. After *fseek*, the next operation to the file may be either input or output.

See Also

rewind, ftell

Returns

0 = Successful
Non-zero = Unsuccessful. This can occur if *fseek* is attempted on a file not open via *fopen*, or if it is used on something other than a file.

ftell

Declaration

```
#include <stdio.h>
long ftell (stream)
FILE *stream;
```

Description

This function returns the offset of the current byte relative to the beginning of the file associated with the named *stream*.

See Also

fseek, rewind

fwrite

Declaration

```
#include <stdio.h>
int fwrite (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;
```

Description

This function is for binary output. It attempts to append *nitems* of data from the array pointed to by *ptr* to the named output *stream*.

fseek or *rewind* must be called before switching between reading and writing on a stream that allows both.

See Also

fread

Returns

Returns the number of items written. If a non-positive number is given for *nitems*, then a 0 is returned and nothing is written.

getc

Declaration

```
#include <stdio.h>
int getc (stream)
FILE *stream;
```

Description

This function returns the next byte from the named input *stream* and positions the pointer ahead one byte in *stream*. *getc* is a macro and cannot be used where a function is required. For example, a function pointer cannot point to it.

See Also

getchar, fgetc, getw

Returns

EOF is returned when end-of-file or error is encountered.

getchar

Declaration

```
#include <stdio.h>
int getchar()
```

Description

getchar is a macro that returns the next character from the standard input stream, *stdin*.

The character is returned to the program only after pressing ***Return***. To get the character immediately, refer to the window interface functions in Section 5.4.

See Also

getc, fgetc, getw

Returns

EOF is returned when end-of-file or error is encountered.

gets

Declaration

```
#include <stdio.h>
char *gets (s)
char *s;
```

Description

This function reads characters from the standard input stream, *stdin*, into the array pointed to by *s*, until an end-of-file or new-line character is encountered. The new-line character is discarded and the string is terminated with a null character.

See Also

fgets

Returns

s = Successful

If EOF is encountered and no characters have been read, then no characters are transferred to s and a null pointer is returned.

If an error occurs, a null pointer is returned. Attempting to use one of these functions on a file that has not been open for reading will cause an appropriate error.

getw

Declaration

```
int getw (stream)
FILE *stream;
```

Description

getw returns the next word (integer) from the named input *stream*. The file pointer is positioned at the next word. No special alignment is assumed.

EOF is returned when end-of-file or error is encountered.

See Also

getc, getchar, fgetc

Returns

EOF is returned if end-of-file or an error is encountered. Since EOF is a valid integer, use *feof* or *ferror* to check the success of *getw*.

isalnum, isalpha, isascii, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper, isxdigit

Declaration

#include <ctype.h>

int isalnum(c)	*c* is alphanumeric
int isalpha(c)	*c* is a letter
int isascii(c)	*c* is an ASCII character, code less than 0200
int iscntrl(c)	*c* is a delete character (0177) or an ordinary control character (less than 040)
int isdigit(c)	*c* is a digit
int islower(c)	*c* is a lower case letter
int isprint(c)	*c* is a printing character, 040 (space) through 0176 (tilde)
int ispunct(c)	c is a punctuation character (neither control nor alphanumeric)
int isspace(c)	*c* is a space, tab, carriage return, new-line, or formfeed
int isupper(c)	*c* is an upper case letter
int isxdigit	c is a hexadecimal digit

int c;

Description

These macros classify character-coded integer values. *isascii* is defined on all integer values; the other functions are defined where *isascii* is true and for EOF (-1). If the argument of any of these macros lies outside its domain, the result is undefined.

Returns

0 = True
Non-zero = False

lseek

Declaration

```
long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;
```

Description

This function moves the read/write file pointer. It sets the file pointer associated with *fildes*, by *offset* from the position specified by *whence*. *whence* has the following values:

0	Pointer set to *offset* bytes
1	Pointer set to current position plus *offset* bytes
2	Pointer set to file size plus *offset* bytes

lseek will fail and the pointer will remain unchanged if:

- *fildes* is not an open file descriptor
- *whence* is an invalid value.
- the resulting pointer position would be negative

Returns

-1 Unsuccessful

If successful, it returns the pointer position in bytes from the beginning of the file.

longjmp

Declaration

```
#include <stdio.h>

longjmp (env, val)
jmp-buf  env;
int val;
```

Description

This function is a non-local goto. It is useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

longjmp restores the environment saved by the last call of *setjmp* with the same *env* argument. After *longjmp* is called, program execution continues as if the corresponding call of *setjmp* had just returned the value *val*.

longjmp cannot cause *setjmp* to return the value 0. If *longjmp* is invoked with a second argument of 0, *setjmp* will return 1. All accessible data have values as of the time *longjmp* was called.

If *longjmp* is called when *env* was never primed by a call to *setjmp*, or when the last such call is in a function which has since returned, the result will be unpredictable.

See Also

setjmp

malloc, lmalloc

Declaration

```
char *malloc (size)
unsigned int size;

char *lmalloc (size)
unsigned long size;
```

Description

This function returns a pointer to a block of at least *size* bytes aligned for any use. The *size* parameter limits the size of the block to 64K.

lmalloc is like *malloc* but accepts a long parameter, allowing more than 64K bytes per allocation.

See Also

free, calloc, alloca

Returns

Returns a null pointer if there is no available memory.

onexit

Declaration

```
onexit(f)
int  (*f) ( );
```

Description

onexit allows the user to add logic to the *exit*() function. When a program is terminated normally or abnormally (using ^C, kill, or the Applications Selection menu), the *exit*() function is called, which calls up to 10 functions defined by the user. These functions can be defined by giving the function pointer to *onexit*(). This is shown in the example below.

See Also

exit

Example

```
myexit ()
{
        puts("exiting");
}
main ()
{
        onexit(myexit);
}
```

Result: This will display the message ***exiting*** on the screen when the program is terminated.

open

Declaration

```
#include <fcntl.h>

int open (fname, oflag)
char *fname;
int oflag;
```

Description

This function opens a file for reading or writing as specified by *oflag*. *fname* points to a string containing the name of the file file. *oflag* values are constructed by ORing flags from the following list (only one of the first three may be used):

0-RDONLY	Open for reading only.
0-WRONLY	Open for writing only.
0-RDWR	Open for reading and writing.
0-BINARY	Open in binary (untranslated) mode.

The ASCII line feed character (10 decimal) is usually considered to be the line separator character. Tekelec C considers a carriage return/line feed combination to be the line separator. In order to easily overcome this difference, the run time library automatically converts carriage return/line feed to line feed on input, and converts line feed to carriage return/line feed on output to files.

This conversion occurs at a very low level within the library routines. Files can be opened in untranslated or binary mode by setting a flag when the open procedure is called.

Upon completion, the file pointer is set to the beginning of the file. No process may have more than 12 file descriptors open simultaneously.

Returns

If successful, the file descriptor is returned.
If unsuccessful, -1 is returned and errno is set appropriately.

printf, fprintf, sprintf, _fprintf, _sprintf

Declaration

```
#include <stdio.h>
int printf (format [ , arg] . . . )
char *format;

int fprintf (stream, format [ , arg] . . . )
FILE * stream;
char *format;

int sprintf (s, format [ , arg] . . . )
char *s, format;

int _fprintf(stream, format, args)
FILE *stream;
char *format, *args;

int _sprintf(s, format, args)
char *s, *format, *args;
```

Description

These functions print formatted output, as described below. All buffers passed to printf() are limited to 256 characters.

printf places output on the standard output stream *stdout*.

fprintf places output on the named output stream.

_fprintf is like *fprintf* except the arguments are retrieved from the pointer *args*.

sprintf places "output", followed by a null character (\O) in consecutive bytes starting at *s. It is your responsibility to ensure that enough storage is available.

_sprintf works like *sprintf* except the arguments are retrieved from the pointer *args*, which normally points into the stack.

Each function returns the number of characters transmitted (not including \0 for *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its args under control of the format. The format is a character string that contains two types of objects:

- Plain characters are copied into the output stream
- Conversion specifications results in fetching zero or more args

The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- On optional flag which modifies the meaning of the conversion specification.
- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag has been given), with spaces, to the field width. A leading zero indicates zeros should be used instead of spaces.
- A precision which gives the maximum number of characters to be printed from a string, or the number of digits to be printed to the right of the decimal point for float or double.
- An optional 1 specifying that a following d, o, u, or x conversion character applies to a long integer arg.
- A character indicating the type of conversion to apply.

The only flag character is the minus sign (-). When used, the result of the conversion will be left justified within the field. A field width or precision may be * instead of a digit string. In this case, an extra integer argument provides the field width or precision.

The conversion characters and their meanings are:

d,o,u,x The integer arg is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation respectively. The letters *abcdef* are used for *x* conversion.

f The float or double arg is converted to decimal notation in the style:

[-]<digits>.<digits>

where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output. If the precision is zero, no decimal point appears.

For example, the float or double arg is converted to the style:

[-]<digit>.<digits>E(+|-)<digits>

where there is one digit before the decimal point and the number of digits after it is equal to the precision. When the precision is missing, six digits are output. If the precision is zero, no decimal point appears.

c The character arg is printed.

s The arg is taken to be a string (character pointer) and characters from the string are printed until a null character (\O) is encountered, or the number of characters indicated by the precision specification is reached. If the precision is missing, it will be taken to be infinite, so all characters up to the first null character are printed. A null arg will yield undefined results.

% Print a %. No argument is converted.

In no case does a non-existent or small field width cause truncation of a field. If the result of the conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by printf and fprintf are printed as if putc had been called.

perror

Declaration

```
perror(s)
char  *s;

extern int sys_nerr;
extern char *sys_errlist[];
```

Description

perror writes a short description of the last error that set *errno* onto the standard stream *stderr*. The string *s* is printed first, then a colon, then the message and a newline. The string *s* is usually the name of the program which called *perror*.

perror should only be called when a function which sets *errno* indicates an error has occurred since *errno* is not cleared upon successful execution.

The messages printed are stored in the array *sys_errlist* and may be indexed by *-errno* (this is not compatible with UNIX where *errno* is always positive). The number of entries in *sys_errlist* is stored in *sys_nerr*.

putc

Declaration

```
#include <stdio.h>
int putc (c , stream)
char c;
FILE *stream;
```

Description

putc is a macro that writes the character *c* to the output stream at the current pointer position.

See Also

putchar, fputc, putw

Returns

If successful, the value written is returned.

If unsuccessful, EOF is returned. This can occur if the file is not open for writing or if the output file cannot be grown.

putchar

Declaration

```
#include <stdio.h>
int putchar (c)
char c;
```

Description

putchar is a macro that is defined as putc(c, stdout). (*putc* is a macro that writes the character *c* to the output stream at the current pointer position. See previous page.)

See Also

putc, fputc, putw

Returns

If successful, the value written is returned.

If unsuccessful, EOF is returned. This can occur if the file is not open for writing or if the output file cannot be grown.

puts

Declaration

```
include <stdio.h>
int puts (s)
char *s;
```

Description

puts writes the null-terminated string, pointed to by *s*, to the standard output stream *stdout*. The string is followed by a new-line character. It does not write out the terminating null character.

See Also

fputs

Returns

EOF is returned if an error occurs. This will happen if output is attempted to a file not open for writing.

putw

Declaration

```
#include <stdio.h>
int putw (w, stream)
int w;
FILE *stream;
```

Description

putw writes the word (integer) *w* to the output stream at the current pointer position. *putw* does not force even alignment on the file.

See Also

putc, putchar, fputc

Returns

If successful, the value written is returned.

If unsuccessful, EOF is returned. This can occur if the file is not open for writing or if the output file cannot be grown.

Because EOF is a valid integer, *ferror* should be used to check for error when using *putw*.

qsort

Declaration

```
qsort(base, nelem, width, compare)
char *base;
int nelem, width;
int (*compare) ();
```

Description

qsort is an implementation of the quicksort algorithm. The parameter *base* is a pointer to the base of the data. The parameter *nelem* is the number of elements in the array. The parameter *width* is the width of each element in bytes. The parameter *compare* is a pointer to the comparison routine to be called.

This user-defined function will be passed two arguments which are pointers to the elements being compared. This routine must return an integer less than, equal to, or greater than zero, since the first argument is to be considered less than, equal to, or greater than the second.

The quicksort algorithm used is recursive.

Example

```
#include <stdio.h>
int test(a, b)
int *a, *b;
{
      return *a - *b;
}

main()
{
      int x[100], i;
      for (i=0; i<100; i++)/* Create some random data */
            x[i] = rand();

      qsort(x, 100, sizeof(int),test);

      for (i=0, i<100; i++)/* Display sorted result */
            printf("%d ", x[i]);

      puts (Press RETURN to continue"); getchar () ;
}
```

rand
srand

Declaration

```
#include <stdio.h>

int rand()

srand(seed)
long seed;
```

Description

rand and *srand* are macros that function as simple random-number generators.

rand uses a multiplicative congruential random-number generator.

srand can be called at any time to reset the random-number generator to a new starting point. The generator is initially seeded with a value of 1.

read

Declaration

```
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

Description

read attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*.

fildes is a file descriptor obtained by using an *open* or *creat*.

read will fail if *fildes* is not a valid file descriptor open for reading, or if an operating system error occurs.

If the O_BINARY flag is not set, linefeed/carriage return combinations are translated to linefeeds, except from the keyboard.

Returns

0 = EOF is reached.

If successful, a non-negative integer is returned indicating the number of bytes actually read.

If unsuccessful, a -1 is returned and *errno* is set appropriately.

realloc, lrealloc

Declaration

```
char *realloc(ptr, size)
char *ptr;
unsigned size;

char *lrealloc(ptr, size)
char *ptr;
unsigned long  size;
```

Description

These are RAM allocator functions. *realloc* changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (potentially moved) block. Note that the data will remain unchanged, and any data defined beyond *size* will be lost.

lrealloc is like *realloc* but accepts a long parameter.

Returns

A null pointer if the memory requested is not available.

rename

Declaration

```
int rename (from, to)
char *from, *to;
```

Description

rename changes the existing name of a file on a disk to another name. The parameter *from* is a pointer to the name of the current file on disk. The parameter *to* is a pointer to the new name for the file.

Returns

-1 Unsuccessful

rewind

Declaration

```
#include <stdio.h>
rewind (stream)
FILE *stream;
```

Description

rewind sets the position of the next input or output operation on the stream. The new position is at the signed distance offset bytes from the beginning, from the current position, or from the end of the file. *rewind* is equivalent to *fseek*(stream, OL, 0), except no value is returned.

rewind undoes the effects of *ungetc*. After *rewind* the next operation to the file may be either input or output.

See Also

fseek, ftell

scanf, fscanf, sscanf

Declaration

```
#include <stdio.h>
int scanf (format [ , pointer] . . . )
char *format;

int fscanf (stream, format [ , pointer] . . . )
FILE *stream;
char *format;

int sscanf (s, format [ , pointer] . . . )
char *s, *format;
```

Description

Each function reads characters, converts them according to a format, and stores the results in its arguments. The arguments consist of a control string format and a set of pointer arguments indicating where the converted input should be stored.

scanf reads from the standard input stream *stdin*.

fscanf reads from the named input stream.

sscanf reads from the character string *s*.

The control string may contain:

- White-space characters (blanks, tabs, and new-lines) which cause input to be read up to the next non white-space character.
- An ordinary character (not %), which must match the next character of the input stream.
- Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, an optional l indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-white-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates the interpretation of the input field. For a suppressed field, no pointer argument should be given. The following conversion codes are legal:

%	A single % is expected in the input at this point; no assignment is done.
d	A decimal integer is expected; the corresponding argument should be an integer pointer.
h	A short decimal integer is expected, the corresponding argument should be a short pointer.
o	An octal integer is expected; the corresponding argument should be an integer pointer.
x	A hexadecimal integer is expected; the corresponding argument should be an integer pointer.
e,f,g	A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits, possibly with a decimal point, followed by an optional exponent field consisting of an e, or an E, followed by an optionally signed integer.
s	A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \O, which will be added automatically. The input field is terminated by a white-space character.

c A character is expected. The corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case. To read the next non-space character, use 1s. If a field width is given, the corresponding argument should refer to a character array. The indicated number of characters is read.

The conversion characters d, o, and x may be preceded by 1 to indicate that a pointer to long rather than int is in the argument list. Also, the conversion characters e, f, and g may be preceded by 1 to indicate that a pointer to double rather than to float is in the argument list.

scanf conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

scanf returns the number of successfully matched and assigned input items. This number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

Note

Trailing white space (including a new-line) is left unread unless matched in the control string.

Returns

These functions return EOF on end of input and a short count for missing or illegal data items.

setbuf, setbuffer, setlinebuf

Declaration

```
#include <stdio.h>

setbuf (stream, buf)
FILE *stream;
char *buf;
char buf[BUFSIZE];

setbuffer (stream, buf, bufsize)
FILE *stream;
char *buf;

setlinebuf(stream)
FILE *stream
```

Description

Three types of buffering are available:

- Unbuffered — Information appears on the destination file or terminal as soon as written
- Block buffered — Many characters are saved up and written as a block. Normally, all files are block buffered.
- Line buffered — Characters are saved up until a newline is encountered.

setbuf is used after a stream has been opened, but before it is read or written. It causes the character array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is a NULL character pointer input/output will be completely unbuffered. A constant *BUFSIZ*, defined in the <stdio.h> header file, tells how big an array is needed.

setbuffer sets up a user-defined I/O buffer whose size is determined by the parameter *bufsize*. If *buf* is NULL, the I/O buffer will be completely unbuffered. This function should only be used after a stream has been opened, but before it has been read or written.

setlinebuf changes *stdout* or *stderr* from block buffered or unbuffered to line buffered. Unlike *setbuf* and *setbuffer*, it can be used at any time that the file descriptor is active.

If the space passed as *buf* cannot be freed (it was not allocated by malloc, for example), then the stream must be set to unbuffered before closing.

setjmp

Declaration

```
#include <stdio.h>

int setjmp (env)
jmp_buf env;
```

Description

This is a non-local goto which is useful for dealing with errors and interrupts encountered in a low-level subroutine of a program. *setjmp* saves its stack environment in *env* (whose type, jmp_buf, is defined in the <stdio.h> header file), for later use by *longjmp*. It returns the value 0.

See Also

longjmp

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex, xtrncat, xtrcpy, xtrncpy

Declaration

```
#include <string.h>

char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

int index (s, c)
char *s, c;

int rindex (s, c)
char *s, c;

char *strcpy (s1, s2)
char *s1, *s2;

char strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *xtrcat(s1, s2)
char *s1, *s2;

char *xtrcpy(s1, s2)
char *s1, *s2;

char *xtrncpy(s1, s2)
char *s1, *s2;
```

Description

These functions perform string operations as described below. The arguments *s1*, *s2*, and *c* point to strings (arrays of characters terminated by a null character). The functions *strcat*, *strncat*, *strcpy*, *strncpy*, *xtrcat*, *xtrcpy*, and *xtrncpy* all alter s1. They do not check for overflow of the array pointed to by *s1*.

strcat appends a copy of string *s2* to the end of string *s1*, and returns *s1*.

xtrcat appends but returns a pointer to the end of *s1*, pointing at the null byte.

strncat appends at most *n* characters.

strcmp compares its arguments and returns an integer less than, equal to, or greater than 0, depending on whether *s1* is lexicographically less than, equal to, or greater than *s2*.

strncmp makes the same comparison as strcmp, but looks at a maximum of *n* characters.

strcpy copies string *s2* to *s1*, stopping after the null character has been copied. The result is *s1*.

xtrcpy copies but returns a pointer to the end of s1.

strncpy copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result will not be null-terminated if the length of *s2* is *n* or more.

xtrncpy copies like *strncpy*, but returns a pointer to the end of *s1*.

strlen returns the number of characters in *s*, not including the terminating null character.

index returns a pointer to the first occurrence of *c* in string *s*. NULL is returned if *c* is not in *s*.

rindex returns a pointer to the *last* occurrence of *c* in string *s*. NULL is returned if *c* is not in *s*.

toupper, tolower, -tolower, toascii

Declaration

```
#include <ctype.h>

int toupper (c)
int c:

int tolower (c)
int c;

int __tolower (c)
int c;

int toascii (c)
int c;
```

Range

The range for toupper and tolower is -1 to 255.

Description

These functions convert characters as described below.

If the argument for *toupper* is a lower case letter, the result is a corresponding upper case letter. It does not check for already upper case.

If the argument for *tolower* is an upper case letter, the result is a corresponding lower case letter. Arguments other than the ones mentioned are returned unchanged.

__tolower is similar to *tolower* but has a smaller domain and is faster. It requires an upper case letter as its argument. Undefined results occur if arguments are other than required.

toascii returns the argument with all but the low order 7 bits set to zero.

ungetc

Declaration

```
#include <stdio.h>

int ungetc (c, stream)
char c;
FILE *stream;
```

Description

This function pushes the character *c* into the buffer associated with an input *stream*. *c* will be returned by the next read from that stream. *c* is returned and the stream is left unchanged.

A *read* must be performed prior to the ungetc. c can be read by *getc*, *getchar*, fread, *gets*, *fgets*, *fgetc*, *fscanf*, and *scanf*.

One character pushback is guaranteed, provided that something has been read from the stream.

fseek erases all memory of inserted characters.

Returns

If *c* equals EOF, *ungetc* does nothing to the buffer and returns EOF.

EOF is returned if *ungetc* cannot insert the character.

unlink

Declaration

```
int unlink (fname)
char *path;
```

Description

This function removes the directory entry pointed to by *fname*.

The named file is unlinked unless the operating system returns an error (see errno).

Returns

0 = Successful
-1 = Error (errno is set appropriately)

write

Declaration

```
int write(fildes, buf, nbyte)
int fildes;
char *buf,
unsigned nbyte;
```

Description

This function writes on a file. It writes *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*.

fildes is a file descriptor obtained from a *creat* or *open*.

Writing begins at the current pointer position and is incremented by the number of bytes actually written after returning from *write*.

write will fail if an operating system error occurs. The pointer position will remain unchanged in this event.

If the O_BINARY flag is not set, linefeeds and returns are translated to carriage returns (except to the screen).

Returns

If successful, the number of bytes actually written is returned.

If unsuccessful, -1 is returned and errno is set appropriately.

5.3 SYSTEM LIBRARY GLOBALS

Introduction

Table 5.3-1 below lists C global variables that are defined in libc.a. You can read these variables to determine information about the current virtual terminal (window), cursor, time, and other information.

You must declare these variables as external, using the C **extern** command, before you can use them. These variables may be used; however, changing their values is only for experienced users.

GLOBAL	FUNCTION
long __stdvt	Returns the current virtual terminal number (vtnum)
int__tab__poss	Returns the cursor position in the line
int__echo__mode	Echoes on getchar
int__cr__mode	Maps CR to CR LF
int__ctl__c__mode	Allows Ctrl C to exit
int curr__year	Returns current year, which is set in the shell and is used by other programs.
long tab__width	Returns the number of characters per TAB
long tkey	Returns the task key of the current program
int close__vt__ok	Indicates if it is OK to close the VT o exit()
char **environ	Environment for environmental variables
__init()	Returns base address of current program
char **argv	Program args
int argc	Programs args count

Table 5.3-1: System Library Globals

5.4 WINDOW INTERFACE FUNCTIONS

Introduction

The extended C library has additional Window Interface functions that enable you to control windows for your applications. The library includes:

- Functions
- Escape sequences

Both of these features are described in this section. Definitions of escape sequences, key values and function declarations are in the file *video.h*.

Standard Input/ Standard Output

When your program starts, a window is assigned to you for standard input and output (*stdin, stdout, stderr*). The window number (*vtnum*) is in the external variable *__stdvt*.

If you want to use any of the window interface functions without opening another window, you should use this variable as the value returned from *openvt()*.

All the regular I/O functions (printf, getchar, etc.) will operate on the standard window. If you open your own window, you should overwrite *__stdvt* with your window number, and then restore it. For regular usage without another window, you do not need to be concerned with these functions.

VT100 Format

You can use the Window Interface functions to address the screen, as follows:

- Rows: 1 - 22
- Columns: 1 - 80
- Row 23 is reserved for the LED display, which can be set by calling the assignleds() function.
- Row 24 is reserved for the window banner and cannot be accessed by the application or the library.

Form Mode

You can change a window to form mode, using the *openform()* function. In form mode a full screen is available; there are no banner and LED lines.

Default Window Attributes

When an *openvt()* call is made, a default attribute is set with the following characteris ics:

- Banner line with the name given in *openvt()*
- Cursor is in (1,1) position and is disabled
- All characters will be written at the current cursor position
- Any Escape sequences remain in effect until changed to a different one or to the default

The window interface functions are described beginning on the following page.

assignleds

Declaration

```
assignleds(vtnum, pleds, ledword)
long vtnum;
char *pleds;
long ledword;
```

Range

vtnum Virtual terminal number which is the value returned by the fopen function.

pleds Pointer to 80 characters which will appear on the LEDs line.

ledword Bits 10-31 are reserved.
Bits 0 - 9: If Bit i = 1, LED i + 1 is on.
If Bit i = 0 LED i + 1 is off

Description

This function creates or changes the LEDs for an application.

closeform

Declaration

```
closeform(vtnum)
long vtnum;
```

Description

This function releases the screen from form mode and returns to window mode. In other words, it restores the screen to it previous status.

See Also

openform

closevt

Declaration

```
closevt(vtnum)
long vtnum;
```

Description

This function releases the virtual terminal. The virtual terminal is determined by the virtual terminal number (**vtnum**) which is a value returned by the openvt function.

See Also

openvt, putvt

disablecur

Declaration

```
disablecur(vtnum)
long vtnum;
```

Description

This function causes the cursor to be invisible on the screen (default setting). **vtnum** is the virtual terminal number of the window returned by the fopen function.

See Also

enablecur

enablecur

Declaration

```
enablecur(vtnum)
long vtnum;
```

Description

This function causes the cursor to be visible on the screen. **vtnum** is the virtual terminal number of the window returned by the openvt function.

See Also

disablecur

getch

Declaration

```
unsigned char getch(vtnum)
long vtnum;
```

Description

This function gets a character from standard input without waiting. If no character is available, it returns a 0xFF. When standard input is used (not opened using the *openvt()* function), the *vtnum* value is in *__stdvt*. The characters are not echoed.

See Also

getcwt

getcwt

Declaration

```
unsigned char getcwt(vtnum)
long vtnum;
```

Description

As with the getch function, this function gets a character from standard input. It does not return a value until a character is available.

Note that if you use this function, you will have to force a key when killing the task from the shell.

When *stdio* is used, the *vtnum* value is in *__stdvt*.

See Also

getch

openform

Declaration long openform()

Description This function puts the screen in non-window (form) mode of 24 lines and returns the virtual terminal number (vtnum). It clears the screen and works in form mode using putvt() calls. Only one open form is allowed in the system, and therefore this function should be in response to a request from the user.

See Also closeform

Returns 0 (the *vtnum* of the form window)

openvt

Declaration

```
long openvt(pname)
char *pname;
```

Description

This function assigns a virtual terminal to an application. **pname** is a 25-character null-terminated string which will appear on the window banner. The function returns the virtual terminal number (vtnum) which is referenced in other functions.

The first 10 characters of the string constitute an escape sequence that defines the default foreground and background colors of the window.

See Also

putvt, closevt

Example

```
long myvt;
myvt = openvt("\033[36m\033[44mMyWindow");
```

where:

\033[36m	defines the foreground color as cyan
\033[44m	defines the background color as blue
MyWindow	defines the text for the window banner

prndata

Declaration

```
prndata(data)
char *data;
```

Description This function sends data to the printer.

See Also endprint, selprn

putvt

Declaration

```
putvt(vtnum,string)
long vtnum;
char *string;
```

Description

This function displays a string on a virtual terminal (window). The virtual terminal is determined by the virtual terminal number (**vtnum**) which is a value returned by the *openvt* function.

The **string** is a maximum of 80 ASCII characters in VT100 format. Esc sequences are defined.

See Also

openvt, closevt

selprn

Declaration

selprn (device, br, bits, sb, par)
long device, br, bits, sb, par;

Description

This function selects the parameters for outputing to a printer. Use the numbers indicated below to select the setting for each parameter.

The default printer settings are:

Parallel, 9600, 8 bits, 2 Stop bits, Even.

device	Printer type	**1**	Parallel
		0	Serial
br	Baud rate	**3**	300
		6	600
		12	1200
		24	2400
		48	4800
		96	9600
		192	19200
bits	Data Bits	**0**	5 bits
		2	6 bits
		1	7 bits
		3	8 bits
sb	Stop bits.	**1**	1 stop bits
		2	1.5 stop bits
		3	2 stop bits
par	Parity	**0**	None
		1	Odd
		3	Even

See Also

endprint, prndata

WINDOW INTERFACE ESCAPE SEQUENCES

Introduction

The Chameleon 32 C Escape sequences are a subset of the VT100 Escape sequences and are listed in the table below.

Note

Where a value is required for the Escape sequence (indicated by Pn in the syntax) enter the ASCII value of the value. For example, to move the cursor up 7 seven lines, you would print the string Esc [7 a. In C you use use the enter the following:

```
printf("\033[7a")
```

where \033 represents the Esc key in octal.

Esc Sequence	Function
Esc[Pna	Move cursor up Pn lines
Esc[Pnb	Move cursor down Pn lines
Esc[Pn c	Move cursor right Pn columns
Esc[Pnd	Move cursor left Pn columns
Esc[Pi;Pnf	Move cursor to line Pi column Pn
Esc[f	Move cursor home
Esc[PnL	Insert Pn lines (lines below the cursor move down)
Esc[PnM	Delete Pn lines (lines below the cursor move up)
Esc[4h	Insert mode
Esc[4i	Replace mode (default)
Esc[1P	Delete 1 character
Esc[0K	Erase to the end of the line
Esc[0J	Erase to the end of the screen
Esc[2J	Clear the screen

Table 5.4-1: Window Interface Escape Sequences

Screen Attributes

Use the following command to set the screen attributes (color, highlight, blink, reverse video, etc):

Esc [*nn* m

where: nn is one of the attribute numbers listed in the table below.

Attribute Number	Attribute
0	Reset attributes (Underline, Reverse, Blink, Highlight)
1	Highlight
4	Underline
5	Blink
7	Reverse
30	Foreground Black
31	Foreground Red
32	Foreground Green
33	Foreground Yellow
34	Foreground Blue
35	Foreground Magenta
36	Foreground Cyan
37	Foreground White
40	Background Black
41	Background Red
42	Background Green
43	Background Yellow
44	Background Blue
45	Background Magenta
46	Background Cyan
47	Background White

Table 5.4-2: Window Interface Attribute Options

5.5 MATH LIBRARY

Introduction

The *libm.a* library includes the math functions described in this section. The *libm.a* library is in the \lib directory. When compiling a program, *\lib\lib* is automatically searched for library files, so that you can compile a program using *libm.a* by entering either:

```
cc  prog.c  \lib\libm.a
```

or

```
cc  progc.  -lm
```

The format of a double precision floating point number is:

- The leftmost bit (63) is the sign for the mantissa
- The next bit (62) is the sign for the exponent
- The next 10 bits (61 - 52) contain the binary exponent, which has a bias of 0x3ff (1023)
- The mantissa (in bits 51 - 0) is preceded by an implied 1-bit (left of the binary point). Therefore, the theoretical precision is $53 \times \log_{10}(2) = 15.95$ decimal digits.

All intermediate floating point operations are done in double precision. The transcendental functions use radians.

Zero

A zero is represented by all zeros in the floating point variable.

Largest Value

The largest possible value for a float variable is contained in the math library variable *double dcsu*. The value of this variable is 0×7fffffffffffffff.

Infinity

The value of infinity is represented by the math library variable *double dcin*. The value of this variable is $0 \times$ ffffffffffffffff. This value is returned in the instances where a floating point operation exceeded the maximum value of a double floating point number.

The smallest number $x > 0$ is:

Smallest Value

$$x = 0 \times 0000000000000001$$
$$= ((1 + (2^{-52}))(2^{1025})$$
$$= 1.1125369292536009 \times 10^{-308}$$

If the absolute value of a result is smaller than this number (called underflow), a zero is returned.

The routines are described on the following page.

Declaration

#include <math.h>

double log(x)	Base *e* logarithm function
double log10(x)	Base 10 logarithm function
double log2(x)	Base 2 logarithm function
double exp(x)	Base *e* exponential function
double exp10(x)	Base 10 exponential function
double exp2(x)	Base 2 exponential function
double sin(x), cos(x), tan(x)	Transcendental functions
double asin(x), acos(x), atan(x)	Inverse transcendental
double sqr(x)	x^2
double sqrt(x)	$\sqrt{x}$
double powerd(x, y)	x^y (equivalent to exp2(x*log2(y))
double poweri(x,a)	x^a (equivalent to exp2(x*log2(a)) where a is an integer)
double dabs(x)	|x|
int dint (x)	Integer part of the double that is the parameter. The fractional part is truncated. This is equivalent to: $\text{sgn}(x) \times \lfloor \lvert x \rvert \rfloor$ where: sgn(x) = -1, if x<0; = 0, if x=0; = 1, if x>0
double mulpower2(x, k)	Performs a fast floating point multiplication by 2^k.
double lngamma(x)	Natural logarithm of the gamma function if $0 < x < 5.1 \times 10^{305}$. Outside of this range *dcin* (infinity) is returned.
double fac(*k*)	*k*!, where $0 \leq k \leq 170$

```
double x,y;
int a, k;
```

```
double matinv(a, c, n)
      double *a;
      long *c;
      long n;
```

matinv is the matrix inverse of the n×n array *a*. The data in *a* may be stored in either row or column major order (C double dimension arrays are row major). *c* is a vector (one dimensional array) of *longs* used during the computation. *matinv* returns the determinant of *a* as the function result, and the inverse of *a* in *a*. c has no meaning after *matinv* finishes. A determinant value of zero indicates failure (*a* is destroyed).

For example:

```
#include <math.h>

double e[2] [2] = {1,0,0,1};          /*Identity Matrix*/

main()
{
      double det;
      long   C[2]

      det=matinv(e, C, 2L);
      printf("The determinant of e is %f\n", det);
}
```

5.6 CONTROL CHARACTERS

Introduction The control characters are listed in the table below.

Control Character	Control Key	Function
BEL (7)	Ctrl G	Bell
BS (8)	Ctrl H	Back Space
LF (10)	Ctrl J	Line feed
VT (11)	Ctrl K	Move cursor down 1 line
FF (12)	Ctrl L	Move cursor forward 1 character
CR (13)	Ctrl M	Carriage return

Table 5.5-1: Control Characters

5.7 USING AUX SERIAL PORTS 1 & 2

Introduction

The C Development System includes functions for accessing the Aux Serial Ports 1 and 2. There are four functions for each port. Those for Port 1 are:

- initportb — Initializes the Aux Serial Port 1. You must use this function to initialize the port before you can transmit or receive.
- sndpb — Transmits data using Aux Serial Port 1 to another device
- recpb — Receives data using Aux Serial Port 1 from another device
- rstdrvb — Flushes the driver reception buffer.

Port 1 is only available if the debugger is not attached to it. For details on the debugger, see *Chapter 2.1, Configuration File.*

Those for Port 2 are:

- initporta — Initializes the Aux Serial Port 2. You must use this function to initialize the port before you can transmit or receive.
- sndpa — Transmits data using Aux Serial Port 2 to another device
- recpa — Receives data using Aux Serial Port 2 from another device
- rstdrv — Flushes the driver reception buffer.

These functions are described on the following pages. The Port 1 functions are given on pages 5.7–2 through 5.7–5. The Port 2 functions are given on pages 5.7–7 through 5.7–10. Sample programs are provided on pages 5.7–6 and 5.7–11.

PORT 1 FUNCTIONS

initportb

Declaration

```
#include "paval.h"
int initportb (stopbit, bitchar, bitrate, parity)
long stopbit;
long  bitchar;
long  bitrate;
long  parity;
```

Ranges

stopbit	ST1	(1 stop bit)
	ST15	(1.5 stop bits)
	ST2	(2 stop bits)
bitchar	DB5	(5 data bits)
	DB6	(6 data bits)
	DB7	(7 data bits)
	DB8	(8 data bits)
bitrate	F110	(110 bits per second)
	F300	(300 bits per second)
	F120	(1200 bits per second)
	F240	(2400 bits per second)
	F480	(4800 bits per second)
	F960	(9600 bits per second)
	F192	(19200 bits per second)
parity	PANO	(No parity)
	PAEV	(Even parity)
	PAOD	(Odd parity)

Description

This function initializes the Chameleon 32 Aux Serial Port 1 to transmit and receive data. When you use *initporta*, the driver reception buffer is automatically flushed.

Returns

0	Successful
–1	Parameter error
–2	Port 1 not available

sndpb

Declaration

```
#include "paval.h"
int sndpb (ptr, nb, timeout)
char   ptr;          /*user buffer pointer*/
long   nb;           /*number of bytes*/
long   timeout;      /*timout value*/
```

Description

This function transmits data using Aux Serial Port 1. You must first initialize the port using the *initportb* function, before you can transmit or receive data using this port.

ptr is a pointer to a buffer containing the data to transmit. *nb* is the number of bytes of data to transmit. *timeout* is the amount of time to wait for the other device to receive the data. It is in millisecond units.

Returns

nb	Number of bytes transmitted
0	Time out
–1	Parameter error
–2	Port 1 not available

recpb

Declaration

```
#include "paval.h"
int recpb (ptr, timeout)
char  *ptr;
long  timeout;
```

Description

This function receives data using Aux Serial Port 1. You must first initialize the port using the *initportb* function, before you can transmit or receive data using this port.

ptr is a pointer to a buffer to put the received data.

timeout determines the amount of time to wait to receive data data in millisecond units.

If timeout > 0, the function immediately returns the number of characters currently in the reception buffer. If the reception buffer is empty, it waits the timeout period before returning the number of bytes or a timeout.

If timeout = 0 the function immediately returns the number of characters currently in the reception buffer. If the buffer is empty, the Chameleon waits until a character is received before returning the number of bytes.

Returns

nb	Number of bytes received
0	Time out (no characters in the reception buffer)
–1	Parameter error
–2	Port 1 not available

rstdrvb

Declaration

#include "paval.h"
rstdrvb()

Description

This function flushes the driver reception buffer. Note that the *initportb* function automatically flushes the driver reception buffer when the port is initialized.

Returns

–2	Port 1 not available

SAMPLE PROGRAM

A sample program using the Aux Serial Port 1 functions is provided below. This program initializes the port for terminal emulation.

```
#include "paval.h"
extern  long    _stdvt ;
/*-------------------------------------------------
            Terminal emulation program
-------------------------------------------------*/
main()
{
char        rbuf[300] ;      /* reception buffer */
char        dbuf[300] ;      /* display buffer */
char        c ;                    /* typed character */
int         len ;            /* number of data received */
int         i,j ;            /* local variables */
puts("Terminal Emulation program, Type ESC to exit") ;
/*---- Initialization of the port
     1 stop bit
     8 bits per characters
     bit rate = 9600
     No parity
-----------------------------------*/
initportb(ST1,DB8,F960,PANO) ;
for(;;)
     {
     /*---- poll local keyboard, echo char and send char
     */
     if ( (c=getch(_stdvt)) != -1 )
             {
             if ( c == 0x1b )       /* exit with ESCAPE */
                     break ;
             else
                     {
                     putvtsd(_stdvt,&c,1L) ;
                     sndpb(&c,1L,10L) ;
                     }
             }
     /*---- poll reception on AUX 1 port and display */
     else if ( (len = recpb(rbuf,100L)) != 0 )
             {
             for(i=0,j=0;i!=len;i++)
                     {
                     if ( rbuf[i] >= ' ' )
                                     dbuf[j++] = rbuf[i] ;
                     }
             putvtsd(_stdvt,dbuf,(long)j) ;
             }
     }
}puts("\nDisconnected") ;
}
```

PORT 2 FUNCTIONS

initporta

Declaration

```
#include "paval.h"
int initporta (stopbit, bitchar, bitrate, parity)
long stopbit;
long bitchar;
long bitrate;
long parity;
```

Ranges

stopbit	ST1	(1 stop bit)
	ST15	(1.5 stop bits)
	ST2	(2 stop bits)
bitchar	DB5	(5 data bits)
	DB6	(6 data bits)
	DB7	(7 data bits)
	DB8	(8 data bits)
bitrate	F110	(110 bits per second)
	F300	(300 bits per second)
	F120	(1200 bits per second)
	F240	(2400 bits per second)
	F480	(4800 bits per second)
	F960	(9600 bits per second)
	F192	(19200 bits per second)
parity	PANO	(No parity)
	PAEV	(Even parity)
	PAOD	(Odd parity)

Description

This function initializes Aux Serial Port 2 to transmit and receive data. When you use *initportb*, the driver reception buffer is automatically flushed.

Returns

0	Successful
–1	Parameter error

sndpa

Declaration

```
#include "paval.h"
int sndpa (ptr, nb, timeout)
char        *ptr;       /*user buffer pointer*/
long        nb;         /*number of bytes*/
long        timeout;    /*timout value*/
```

Description

This function transmits data using Aux Serial Port 2. You must first initialize the port using the *initporta* function, before you can transmit or receive data using this port.

ptr is a pointer to a buffer containing the data to transmit. *nb* is the number of bytes of data to transmit. *timeout* is the amount of time to wait for the other device to receive the data. It is in millisecond units.

Returns

nb	Number of bytes transmitted
0	Time out
–1	Parameter error

recpa

Declaration

```
#include "paval.h"
int recpa (ptr, timeout)
char        *ptr;
long        timeout;
```

Description

This function receives data using Aux Serial Port 2. You must first initialize the port using the *initporta* function, before you can transmit or receive data using this port.

ptr is a pointer to a buffer to put the received data.

timeout determines the amount of time to wait to receive data data in millisecond units.

If timeout > 0, the function immediately returns the number of characters currently in the reception buffer. If the reception buffer is empty, it waits the timeout period before returning the number of bytes or a timeout.

If timeout = 0 the function immediately returns the number of characters currently in the reception buffer. If the buffer is empty, the Chameleon waits until a character is received before returning the number of bytes.

Returns

nb	Number of bytes received
0	Time out (no characters in the reception buffer)
–1	Parameter error

rstdrv

Declaration

```
#include "paval.h"
rstdrv()
```

Description

This function flushes the driver reception buffer. Note that the *initporta* function automatically flushes the driver reception buffer when the port is initialized.

SAMPLE PROGRAM

A sample program using the Aux Serial Port 2 functions is provided below. This program initializes the port for terminal emulation.

```
#include "paval.h"
extern  long    _stdvt ;
/*-------------------------------------------------
            Terminal emulation program
-------------------------------------------------*/
main()
{
char        rbuf[300] ;      /* reception buffer */
char        dbuf[300] ;      /* display buffer */
char        c ;              /* typed character */
int         len ;            /* number of data received */
int         i,j ;            /* local variables */
puts("Terminal Emulation program, Type ESC to exit") ;
     /*---- Initialization of the port
     1 stop bit
     8 bits per characters
     bit rate = 9600
     No parity
-----------------------------------*/
initporta(ST1,DB8,F960,PANO) ;
for(;;)
     {
     /*---- poll local keyboard, echo char and send char
     */
     if ( (c=getch(_stdvt)) != -1 )
             {
             if ( c == 0x1b )       /* exit with ESCAPE */
                     break ;
             else
                     {
                     putvtsd(_stdvt,&c,1L) ;
                     sndpa(&c,1L,10L) ;
                     }
             }
     /*---- poll reception on AUX 2 port and display */
     else if ( (len = recpa(rbuf,100L)) != 0 )
             {
             for(i=0,j=0;i!=len;i++)
                     {
                     if ( rbuf[i] >= ' ' )
                                     dbuf[j++] = rbuf[i] ;
                     }
             putvtsd(_stdvt,dbuf,(long)j) ;
             }
     }
}puts("\nDisconnected") ;
}
```

5.8 MS-DOS COMPATIBLE FILE FUNCTIONS

Introduction

The Chameleon MTOS-UX file system is designed to be compatible with MS-DOS 2.x and 3.x. The functions described in this section have been added to provide low-level access to the Chameleon file system.

File Names

The format of a filename is set by the system to be eight bytes in length, plus a three-byte extension. The set of characters allowed in a filename and extension are:

A-Z, 0-9, - _ ! @ # $ % ^ & ( ) " ~ { }

All filenames must be terminated with a NULL (0x00) character.

Wild Card Characters

The question mark can be used as a wild card to describe multiple files having similar names. A wild card character can be used only in the search function. When used with other functions, it may produce incorrect results.

The question mark is a single character wild card. For example, if you have three files named test1.txt, test2.txt, and test3.txt, they could be identified simultaneously by using the name test?.txt.

Directories

The C library contains functions for creating and removing file directories and subdirectories. This enables you to have a hierarchical directory structure to provide a higher order of organization of files on the drive.

Pathnames

The use of pathnames enables you to access a drive or directory other than the current one. A filename is composed of the drive name and the directory specifiers as described below.

The format of a pathname is as follows:

drive:\directory\directory\...filename

drive is Chameleon disk drive you want to access, and is one of the following:

a: hard disk drive
b: floppy disk drive

You do not need to specify the drive if you want to access the current drive.

Following the drive, you specify the hierarchy of directories necessary to access the desired directory. Each directory name is followed by the back slash (\) character. If a directory is not specified, the current directory is assumed.

Following the directory path list is the file name of the files you want to access. This must include the file extension. The wild card ? can be used to specify more than one file name.

File Functions

The standard C library (lib.c) contains the following MS-DOS compatible file functions:

Fmkdir Makes a new directory
Frmdir Removes a directory
Fsearch Searches for a specified file or directory

These functions are described on the following pages.

Sample Usage

Sample programs are provided beginning on page 5.8-7 to illustrate the usage of the file functions.

Error Codes

When a file function is completed, it returns either 0 for success, or a negative number if an error has occurred during the execution of the function.

The table below lists the possible codes:

ErrorCode	Description
0	Successful
-1	Configuration error
-2	Opcode error
-3	Parameter error
-4	Error in volume access
-5	File not found error
-6	File already exists error
-7	Queue empty
-8	Physical read error
-9	Physical write error
-10	Directory full error
-11	Files open on directory
-12	File aready open
-13	Error in filename
-14	File locked
-15	Function option error
-16	Attribute error
-17	End of file unexpected error
-18	EOF with partial record
-19	Fatal error
-20	Disk full - Temporary

Fmkdir

Declaration

```
#include <msfsuse.h>
Fmkdir(dirname)
char * dirname;
```

Description

This function creates a directory with the name ***dirname***. ***dirname*** is a pointer to the directory name and can include the path for the new directory.

Returns

See error codes on page 5.8-3.

Example

```
...
if (Fmkdir (".\\abc\\def") ! = 0)  {
    puts ("make dir error");
    ...
```

Also see sample programs beginning on page 5.8-7.

Frmdir

Declaration

```
#include <msfsuse.h>
Frmdir(dirname)
char *dirname;
```

Description

This function removes a directory with the name ***dirname***. ***dirname*** is a pointer to the directory name and can include the path to the directory.

The directory must be empty before it can be removed from the disk.

Returns

See error codes on page 5.8-3.

Example

```
...
if (Frmdir (".\\abc\\def") ! = 0)  {
    puts ("remove dir error");
    ...
```

Also see sample programs beginning on page 5.8-7.

Fsearch

Declaration

```
#include <msfsuse.h>
Fsearch(name, option, rec)
char * name;
int option;
struct DREC *rec
```

Description

This function searches for a file or directory specified by name. ***name*** is a pointer to the file/directory name and can also include the path.

If located, the file information is copied to the structure of type DREC, which is defined in msfsuse.h, as shown below:

```
struct DREC
   {
   char              dc_fn[ ];   /*File name*/
   char              dc_ex[ ];   /*File extension*/
   char              dc_at;      /*File attributes*/
   char              dc_rs[ ];   /*Reserved bytes*/
   unsigned short    dc_tim;     /*Time of file creation*/
   unsigned short    dc_dat;     /*Date of file creation*/
   unsigned short    dc_str;     /*Starting cluster number*/
   unsigned long     dc_fsz;     /*File size in bytes*/
   };
```

File attributes are defined in msfsuse.h as follows:

FA_RDF	0x01	Read Only File
FA_HDF	0x02	Hidden File
FA_SYS	0x04	System File
FA_VOL	0x08	Volume
FA_SDR	0x10	Sub-directory
FA_ARF	0x20	Archive

Returns

See error codes on page 5.8-3.

Example

See sample programs beginning on page 5.8-7.

Sample Usage

The following sample programs demonstrate the use of the MS-DOS compatible file functions. Error messages printed by the programs are defined in a program named **msfsmsg.c** and declared in the file **msfsmsg.h**, which are not included in these samples.

The first sample program creates a directory named **test_dir**, searches for it, and then removes it.

```
#include   <stdio.h>
#include   <video.h>
#include   <msfsuse.h>
#include   "msfsmsg.h"

main ()
{
        char    *name;
        int a;

        name= "test_dir";
        if ((a=Fmkdir(name)) == SUCCESS)   {
                puts ("Fmkdir completed successfully");
                ls_search (name);
        }
        else    {
                printf("Fmkdir Error: %s", msfsmsg [-a] );
                exit (0);
        }
        printf ("Hit a key to remove test directory");
        fflush (stdout);
        getcwt(_stdvt);
        if ((a=Frmdir(name)) == SUCCESS    {
                puts ("Frmdir completed successfully");
                ls_search(name);
        }
        else    {
                printf("Frmdir Error: %s", msfsmsg [-a] );
        }
        printf ("\nGoodbye\n");
}
ls_search(name)
char    *name;
{
        struct DREC myrec;
        int a;
        if ((a=Fsearch(name, 0, &myrec)) == SUCCESS) {
            puts ("Fsearch completed successfully");
            ls_print (&myrec);
        }
        else    {
            printf ("Fsearch Error: %s", msfsmsg [-a] );
        }
}
```

The second sample program illustrates how to search for a specified file/directory. This program locates all items of the specified name on the drive.

```
#include   <stdio.h>
#include   <video.h>
#include   <msfsuse.h>
#include   "msfsmsg.h"

main (argc, argv)
char     **argv;
{
    struct DREC myrec;
    char buf [80];
    int a, b,;
    if (argc ! = 2)
        strcpy (buf, ".");
    else
        strcpy (buf, argv [1]);
    for (a = 0; ; a++) {
        printf ("Pass # %d\n", a+1);
        if (a > 0) {
            printf( "\033[1m-- MORE -- \033[0m");
            fflush (stdout);
            getcwt(_stdvt);
            puts("");
        }
        if ((b = Fsearch (buf, (a==0?0:1),&myrec))==SUCCESS)  {
            puts ("Fsearch completed successfully");
            ls_print (&myrec);
        }
        else   {
            printf ("Fsearch Error: %s\n", msfsmsg [-b]);
            break;
        }
    }
}
```

This third program illustrates how the information resulting from the file search function can be displayed for the user.

```
#include    <msfsuse.h>
ls_print (rec)
structDREC  *rec;
{
      int   a;
      printf ("file name == %s\n", rec-> dc_fn);
      printf ("file extension == %s\n", rec-> dc_ex);
      printf ("file attributes == %02x\n", rec-> dc_at);
      if ((rec-> dc_at & FA_RDF) == FA_RDF)
            puts ("Read Only file");
      else
            puts("Read Write");
      if ((rec-> dc_at & FA_HDF) == FA_HDF)
            puts ("Hidden file");
      else
            puts ("Not Hidden file");
      if ((rec-> dc_at & FA_SYS) == FA_SYS)
            puts("System file");
      else
            puts ("Not System file");
      if ((rec-> dc_at & FA_VOL) == FA_VOL)
            puts ("Volume bit == 1");
      else
            puts ("Volume bit == 0");
      if ((rec-> dc_at & FA_SDR) == FA_SDR)
            puts ("Sub-directory file");
      else
            puts ("Not a Sub-directory file");
      if ((rec-> dc_at & FA_ARF) == FA_ARF)
            puts ("Archive file");
      else
            puts ("Not Archive file");
      printf ("reserved bytes == ");
      for (a=0; a < RS_LEN; a++)
            printf ("%02x ", rec-> dc_rs [a]);
      puts ("");
      printf ("time file was created== ");
      ls_time (rec-> dc_tim);
      puts("");
      printf ("date file was created== ");
      ls_time (rec-> dc_dat);
      puts("");
      printf ("starting cluster number == %04x\n", rec-> dc_str);
      printf ("file size (bytes) == %-10ld\n", rec-> dc_fsz);
}
ls_time (time)
int   time;
{
      int   hrs = (time>>11)   &0x1f;
      int   min = (time>>5)    &0x3f;
      int   sec   = ((time)    &0x1f) *2;
      if (hrs == 0)
            hrs = 12;
      printf("%02d:%02d:%02d", hrs, min, sec);
}
ls_date (date)
int   date;
{
      int   mth = (date>>5)    &0xf;
      int   day = (date)       &0x1f;
      int   yr = ((date>>9)    &0x7f) + 80;
      printf("%02d-%02d-%02d", mth, day, yr);
}
```

5.9 NON-PRINTING ASCII CHARACTERS

The following ASCII characters can be displayed on the screen, but will not appear in output to a printer:

Hex	Octal	ASCII	Hex	Octal	ASCII	Hex	Octal	ASCII
80	200	$^{N}_{U}$	a2	242	□	c2	302	
81	201	$^{S}_{H}$	a3	243		c3	303	
82	202	$^{S}_{X}$	a4	244		c4	304	
83	203	$^{E}_{X}$	a5	245		c5	305	
84	204	$^{E}_{T}$	a6	246		c6	306	
85	205	$^{E}_{Q}$	a7	247		c7	307	
86	206	$^{A}_{K}$	a8	250		c8	310	
87	207	$^{B}_{L}$	a9	251		c9	311	
88	210	$^{B}_{S}$	aa	252		ca	312	
89	211	$^{H}_{T}$	ab	253		cb	313	
8a	212	$^{L}_{F}$	ac	254		cc	314	
8b	213	$^{V}_{T}$	ad	255	→	cd	315	
8c	214	$^{F}_{F}$	ae	256	←	ce	316	
8d	215	$^{C}_{R}$	af	257		cf	317	
8e	216	$^{S}_{O}$	b0	260		d0	320	
8f	217	$^{S}_{I}$	b1	261		d1	321	
90	220	$^{D}_{L}$	b2	262		d2	322	
91	221	$^{D}_{1}$	b3	263		d3	323	
92	222	$^{D}_{2}$	b4	264		d4	324	
93	223	$^{D}_{3}$	b5	265		d5	325	
94	224	$^{D}_{4}$	b6	266		d6	326	
95	225	$^{M}_{L}$	b7	267		d7	327	
96	226	$^{S}_{Y}$	b8	270		d8	330	
97	227	$^{E}_{B}$	b9	271		d9	331	
98	230	$^{C}_{H}$	ba	272		da	332	
99	231	$^{E}_{M}$	bb	273		db	333	
9a	232	$^{S}_{B}$	bc	274		dc	334	
9b	233	$^{E}_{C}$	bd	275		dd	335	
9c	234	$^{F}_{S}$	be	276	•	de	336	
9d	235	$^{G}_{S}$	bf	277		df	337	
9e	236	$^{R}_{S}$	c0	300		e0	340	
9f	237	$^{U}_{S}$	c1	301		e1	341	
a0	240	↓						
a1	241	↑						

5.10 BERT LIBRARY FUNCTIONS

This section presents the functions for Bit Error-Rate Testing supported by the Chameleon 32 C Development System compiler. These functions are defined in the file **bertlib** and are given here in alphabeitcal order within three, major functional categories (see the Section Table of Contents on the next page). These are:

STARTUP AND IDLE MODE FUNCTIONS

FUNCTIONS USED WHILE FEP IS RUNNING A TEST

FUNCTIONS RELATED TO COLLECTING TEST DATA

The section INTERVAL TESTING gives a sample program using some of these functions.

GENERAL NOTES AND REQUIREMENTS

As in other simulation libraries, before calling start_sync() or start_async(), the user program should call setport(). If this is not done, port selection will default to port A.

User programs must be linked with the math library (libm.a) provided with system release 4.51 or later. Older versions may cause incorrect exponents to be displayed when the value is quite small.

The *include* file (bertlib.h) must be included in the user program. Otherwise, the returns to user calls to library functions that return long or double values will be misinterpreted.

This library uses a sub–task started by start_sync() or start_async. This sub-task manages the collection of test data from the FEP, counts run-time seconds, errored seconds and timed tests. Because of this, there are three rules that must be observed when writing a user program:

- Program loops must contain a 'pause()' call that will stop the program for a short time to allow the sub–task and any other tasks access to processor time. This means that 'mtosux.h' should be included in the user program.
- While this library may be run on either port, testing on both ports requires running two separate programs. That is, you cannot start both ports from a single task, as the same sub task will be started twice and cause unpredictable results.
- Terminate with an 'exit(0)' call to assure termination of the sub–task.

SECTION TABLE OF CONTENTS

The 29 functions of the BERT Library fall into three categories:

The functions in each category are listed one the first of the pages devoted to that category, as listed above.

STARTUP AND IDLE MODE FUNCTIONS

BLOCK_LEN

Declaration block_len(bllen)

Description Sets block length to be used in determining block errors.

bllen = unsigned int, 64k max.

CLR_PREAM

Declaration clr_pream()

Description Stops using preamble.

See SET_PREAM.

CONT_RUN

Declaration cont_run()

Description Sets FEP to active test mode. The test will continue to run until STOP_TEST( ) is called or the test program is terminated by a call to exit (0).

ONE_BLOCK

Declaration one_block()

Description Runs test for one block length, then returns to idle mode.

SET_ERR_RATE(SEL)

Declaration set_err_rate(sel)

Description Selects automatic error insertion rate. Only effective in SYNC testing.

sel = 0, none
1, 1.00E–5
2, 1.00E–4
3, 9,84E–4
4, 1.00E–3
5, 1.02E–3
6, 1.04E–2

Returns 0 – no problems
–1 – parameter error

See ERROR_ON ( ), ERROR_OFF ( ), ONE_ERROR ( )

SET_MODE

Declaration set_mode(mode)

Description Selects test mode, remote/local loopback, rx only.

mode = 1, Remote loopback
2, local loopback
4, receive only

Returns 0 – no problems
–1 – parameter error

SET_PREAM

Declaration set_pream(ch1,ch2)

Description Selects 2 byte user preamble to be transmitted at beginning of test run.

ch1 = first byte to send
ch2 = second byte to send

See CLR_PREAM ( )

SET_PTRN

Declaration set_ptrn(patsel)

Description Selects the test pattern. For user defined pattern, pattern location and length MUST be set using user_ptrn().

parsel = 0, bert 63
1, bert 511
2, bert 2047
3, bert 32767
4, 10101010
5, foxmes
6, user defined

Returns 0 – no problems
–1 – parameter error

START_ASYNC

Declaration start_async(interface,dbits,sbits,parsel,rate)

interface	=	0, DCE 1, DTE	
dbits	=	5, 5 6, 6 7, 7 8, 8	(Data bits)
sbits	=	0, 1 1, 1.5 2, 2	(Stop bits)
parsel	=	2, even 1, odd 0, none	(Parity)
rate	=	1, 50 2, 75 3, 110 4, 150 5, 300 6, 600 7, 1200 8, 2400 9, 4800 10, 9600 11, 19200	(Baud rate)

Returns

0 – no problems
–1 – parameter error
(see also global error table in the C library manual.)

START_SYNC

Declaration start_sync(interface,rate)

interface = 0, DCE
1, DTE
2, ISDN

rate = 50 – 64000 bps if DCE is selected, any if DTE is selected.

Returns

0 – no problems
–1 – parameter error
(see also global error table in the C library manual.)

TIMED_TEST

Declaration timed_test(length)

Description Run test for length seconds.

length = long int

For the convenience of the user, multipliers have been added to bertlib.h to convert hours and minutes to seconds.

EXAMPLE

timed_test(2*RN_HOURS + 10*RN_MINUTES + 30); will run a test for 2 hours 10 minutes and 30 seconds.

NOTE

This function sets up and starts the timed test, then returns. It will NOT tie up the user program for the duration of the test. All other library and user program functions will operate normally during a timed test.

A call to stop_test() may be used to terminate a timed test before the selected length of time.

USER_PTRN

Declaration user_ptrn(loc,len)

Description Passes address and length of user defined pattern. *loc = pointer to user defined pattern len = length of pattern to send. Max length 4000 bytes.

*loc = pointer to user defined pattern

len = length of pattern to send. Max length 4000 bytes.

Returns 0 – no problems
–1 – parameter error

FUNCTIONS USED WHILE FEP IS RUNNING A TEST

ERROR_OFF

Declaration error_off()

Description Stops automatic error insertion.

See ERROR_ON ( ), ONE_ERROR ( ), SET_ERR_RATE

ERROR_ON

Declaration error_on()

Description Starts automatic error insertion. Requires previous call to'set_err_rate()' to select error insertion interval.

See ERROR_OFF ( ), ONE_ERROR ( ), SET_ERR_RATE

ONE_ERROR

Declaration one_error()

Description Inserts one error in transmitted pattern independently of automatic error insertion. Ineffective if test is not running.

See ERROR_ON ( ), ERROR_OFF ( ), SET_ERR_RATE ( )

RESYNC

Declaration resync()

Description Resynchronizes on data pattern.

In the case of an excessively high error rate (25% or more) in one second, resynchronization will occur automatically. The errors for that second will be discarded and the sync loss counter will be incremented. This is because such a high error rate normally means a loss of data, a data slip, or a change in received pattern has occurred.

This function allows you to program a lower resynch threshold than that provided by the automatic resynch function.

STATUS

Declaration status()

Description Returns sync status.

0 = idle (test is not being run)
1 = running, not in sync
2 = running in sync

STOP_TEST

Declaration stop_test()

Description Stops test and return to idle mode.

See CONT_RUN ( ), ONE_BLOCK ( ), TIMED_TEST ( )

FUNCTIONS RELATED TO COLLECTING TEST DATA

DOUBLE GET_ERR_RATE

Declaration double get_err_rate()

Description Returns cumulative bit error rate in floating point format.

To display the result, use the format code 2.2le.

EXAMPLE:

```
printf("The error rate is %2.2le\n",get_err_rate());
```

LONG GET_BLKERRS

Declaration long get_blkerrs()

Description Returns the number of received block errors. (See block_len())

LONG GET_ERRSEC

Declaration long get_errsec()

Description Returns the number of seconds during which one or more bit errors were received.

LONG GET_RBITS

Declaration long get_rbits()

Description Returns the number of received bits.

LONG GET_TBITS

Declaration long get_tbits()

Description Returns the number of transmitted bits.

LONG GET_RBITERRS

Declaration long get_rbiterrs()

Description Returns the nunber of received bit errors.

LONG GET_RUNTIME

Declaration long get_runtime()

Description Returns the cumulative number of seconds during which the library is in active test mode. For example, this function will return '7' when:

- *cont_run()* is called
- 5 seconds later *stop_test()* is called
- then, the test is restarted 10 seconds later and again stopped after 2 seconds

LONG GET_SERRSEC

Declaration long get_serrsec()

Description Returns the number of seconds during which the received bit error rate was 10E–3 or greater.

LONG GET_SYNCLOSS

Declaration long get_syncloss()

Description Returns the number of times that the received error rate was so high that an automatic resync occured.

The returned value does not include calls to RESYNC( ) by the user program.

LONG GET_TBITERRS

Declaration long get_tbiterrs()

Description Returns the number of transmitted bit errors when auto error insertion is on.

RESET_DATA

Declaration reset_data()

Description Resets the counters for all the above to zero.

TIMED TEST EXAMPLE

The following program fragment is suggested as a means of running a BERT test for a selected time interval. At this point, the FEP should have been started using *start_sync*() or *start_async*() and the parameters (e.g., *mode*, *pattern*, *block length*, etc.) set.

```
printf("\033[2J"); /* Clear screen */

timed_test(5 * RN_MINUTES); /* Start 5 minute timed test */

while(status()){  /* as long as test is running */

    printf("\033[1;1f");  /* position curser at 1,1 */

    printf("Run time ..... %010.0lμseconds\n"
    get_runtime(); /* %010.0lu prints in fixed 10 place format */

    /* print all other test results of interest */

    pause(1 + HMS); /* wait 100 msec so other tasks can use
    processor */

    } /* end of while loop */

/* Using 'HMS' or other mnemonic in pause() requires inclusion of
MTOSUX.h */
```

ADVANCED PROGRAMMING TECHNIQUE: GETTING CORRECT RESULTS FROM VERY LONG TEST RUNS

A structure of long integers is used to store the collected test data. A long, as defined by the Tekelec C compiler, will overflow at a value slightly above two billion. To avoid problems during long test runs, bertlib.h provides the user with a pointer to the data structure so that individual values can be reset and an extension counter incremented. An example of this is in the following program fragment.

```
unsigned int tbit_oflo = 0;

    if(get_tbits() >= 1000000000L){

        tes_data->tx_bits -= 1000000000L;/*adjust the count*/

        tbit_oflo ++; /* and incr the extension counter */

        }
```

It is important that rx_bterrs should be reset any time rx_bits is cleared to avoid incorrect bit error rate calculation by the get_err_rate() function.

6.1 USING THE vi EDITOR

The Chameleon 32 C package provides an editor, called *vi*, which is similar in function to versions of the vi editor commonly in use.

If you are familiar with *vi*, you may not need to read this section in detail. The last few pages of the section contain a listing of all commands and their definitions. A vi quick reference chart is also included in the *Chameleon 32 Quick Reference Guide*.

If you are unfamiliar with *vi*, you may want to create a text file to practice using the *vi* commands as you read this chapter.

Special Keys

Esc, ***Return***, and ***Delete*** have the following functions in *vi*:

Esc The ***Esc*** key cancels incomplete commands. Press ***Esc*** if you have typed an incorrect command, or if you are not sure what commands you have already typed. ***Esc*** also exits from *vi* insert mode.

Return The ***Return*** key terminates commands.

Delete When inserting a command line, ***Delete*** moves the cursor one character to the left. When you then press ***Esc*** or ***Return***, the characters to the right of the cursor are erased.

Entering *vi*

To edit a file in the C Shell page, type the command:

vi *filename*

If the file exists, the file is displayed in the C Shell page. If the file does not exist, vi will assume you are creating a new file.

To edit a file in a separate page, use the syntax:

vi *filename* &

This displays the file for editing in a VI page, which is separate from the C Shell page. Likewise, if the file does not currently exist, vi assumes it is a new file you wish to create.

The tags (-t) option invokes the vi editor to edit one or more files which contain a specified function (tag). For example, the following command invokes the vi editor to edit the files that contain the function specified by *tagname*, and positions the cursor at the first occurrence of the function:

vi -t tagname

In order to use the -t option, you must have used the shell ***ctags*** command, which creates a *tags* file containing information about the functions in the target files. Refer to the ***ctags*** command in Section 2.1 for more information.

Also refer to the ***:tag*** command on page 6.1-3 which enables you to search for tags once you are in vi.

Exiting *vi*

To exit vi, use the following commands:

zz Exits vi and saves the changes made to the file.

:q Exits vi without saving the changes made to the file. *Only use this command if you are sure you want to discard the changes made to the file.*

Text Display

The commands below move the text within the page.

CTRL B Moves backward one page.

CTRL D Moves half a page forward.

CTRL E Moves the page down one line, leaving the cursor at its current location.

CTRL F Moves forward one page.

CTRL U Moves half a page backward.

CTRL Y Moves the page up one line, leaving the cursor at its current location.

z. Causes current line to be displayed in the center of the page.

z-
z <Return> Causes current line to be displayed at the top of the page.

Search Commands

You can also move through the file by searching for a string, using the following command:

/xxxxxxx <Return>

where *xxxxxxx* is any character string. If the string is not present in the file, the editor indicates that this is the case, and returns the cursor to its original position. Additional search commands are:

n Searches forward for the next occurrence of the string

?xxxxxxx <Return> Searches backward through the file for the character string *xxxxxxx*.

/^xxxxxxx A caret at the beginning of the character string searches for strings at the beginning of a line.

/xxxxxxx$ A dollar sign at the end of the character string searches for strings at the end of a line.

:tag tagname Positions the cursor at the first occurrence of the function specified by *tagname* in the files being edited.

In order to use :tag, you must have used the shell ***ctags*** command, which creates a *tags* file containing information about the functions in the target files. Refer to the ***ctags*** command in Section 2.1 for more information.

Positioning Commands

You can move through the file by specifying a line number using these commands:

xG Moves the cursor to line x.

G Moves the cursor to the end of the file.

Ctrl G Display information about the file, including:

-the line number the cursor is positioned on
-the name of the file you are editing
-the number of lines in the buffer
-the percentage of the way through the buffer

`` Two back quotes moves you back to a previous position

Screen Movement Commands

The four arrow keys (→ ↓ ← ↑) move the cursor to any position on the screen. Other cursor commands include:

+	Moves to the first non-white character on the next line
***n*+**	Moves to the first non-white character down *n* lines from the current line
- or ***k***	Moves up one line.
n*-** or ***nk	Moves to the first non-white character up *n* lines from the current line

Line Movement

Once you reach the line you want, you can move the cursor within the line using the following commands:

b	Moves to the beginning of the preceding word.
***n*b**	Moves back *n* words.
e	Moves to the end of the current word.
***n*e**	Moves forward *n* words to the end of the word
backspace or **h**	Moves left one character
***n*h**	Moves left *n* characters
spacebar or **l**	Moves right one character
***n*l**	Moves right *n* characters
w	Moves to the beginning of the next word.
***n*w**	Moves forward *n* words.

Note that the **w** and **b** commands treat punctuation as words, and therefore stop the cursor at punctuation marks. The corresponding upper case commands do *not* treat punctuation as words:

B	Moves to the beginning of the preceding word.
***n*B**	Moves back *n* words.
W	Moves to the beginning of the next word.
***n*W**	Moves forward *n* words.

Insert Mode

When you first enter the *vi* editor you are in command mode. The command mode enables you to move the cursor around in the file. If you want to add, change, and/or delete text, you must change to insert mode. To enter insert mode, type:

i

In insert mode, the keys that you press are interpreted as text input, and are entered to the left of the cursor.

There is also an append mode, which causes typed characters to be entered to the right of the cursor. To enter append mode (from command mode), type:

a

To exit either the insert mode or the append mode, and return to command mode, press ***Esc***.

If you want to insert a new line between existing lines of text, use the following commands:

o Creates a new line below the current line, and enters insert mode.

O Creates a new line above the current line and enters insert mode.

While in either the **O** or **o** mode, press ***Return*** to insert additional new lines of text. To exit **o** or **O** mode, press ***Esc***.

You can also enter position the cursor and enter insert mode using the following commands:

A Move to the end of the line and enter insert mode

I Move to the beginning of the line and enter insert mode

Deleting Characters

To make a correction when you are in command mode, use the positioning keys described previously to place the cursor over the character to be corrected, enter insert mode, and then use one of the following commands:

x Deletes the character directly under the cursor

X Deletes the character left of the cursor

***n*x** Deletes *n* characters beginning with the character directly under the cursor.

rx Deletes the current character and replaces it with character x.

sbbbb Esc Deletes the current character and replaces it with the string *bbbb*.

nsbbbb Esc Deletes *n* characters beginning with the character directly under the cursor, and replaces them with the character string *bbbb*.

Delete Command

Some commands are used with other commands to augment their functions. For example, the **d** (delete) command can be used in conjunction with other commands to delete different amounts of text. For example:

db Deletes the word preceding the cursor

dd Deletes the current line

***n*dd** Deletes *n* lines

dL Deletes all lines up to and including the last line on the screen

d*n*L Deletes all lines up to the *n*th line from the bottom of the screen.

dw Deletes a word

***n*dw** Deletes *n* words

Change Command

The **c** (change) command can be used as follows:

cw*xxxx***Esc** Changes the text of a word to text *xxxx*

cc*xxxx* ***Esc*** Changes the current line to text *xxxx*

Undoing Changes

The following ocmmand are available for undoing (reversing) the effect of the preceding commmand:

u Undoes the last command, as if the command were never performed

uu Undoes the undo command, replacing the change which was undone by the **u** command

U Undoes the last command (up to 9 commands)

Moving Around Quickly in a File

One way to move quickly to a certain place on a line is to use a search command to search for a punctuation mark, and a repeat command command to move to successive occurrences of the punctuation mark.

To do this use the following commands:

f*x* Move to the next occurrence of character *x* on the current line

; Moves to the next instance of the same character.

To work with characters up to but not including character *x*, you can use the **t** (up to) command. For example:

dt*x* Deletes characters up to but not including the character *x*.

To move to the first non-white position on the current line, use the command:

^

To move to the begining of the previous line, use the command:

-

To move to the end of a line, use the command:

S

For example, the command **Sa** allows you to add new text to the end of a line.

Tab Characters

Tab and non-printing characters are treated as if they are a single character, even though they take up move than one character space in a file. Tabs are represented by ***CTRL I***. When the cursor moves to a tab or non-printing character, it rests on the last space that represents the tab or character.

If you space or backspace over a non-printing character (represented by a CTRL character and another character), the cursor moves over it as if it were a single character.

The vi editor ignores the CTRL character if you insert it into a file. If you want to be sure the vi editor does not ignore the character, you must type a ***CTRL V*** before the CTRL character.

REARRANGING TEXT

The *vi* editor provides a number of text buffers. There is an unnamed buffer which the editor uses to save the last deleted or changed text, and a set of 26 buffers named **a** through **z** which you can use to save copies of text, to move text around in your file, and to move text between files.

You can use the command:

"xy

to put selected text into a buffer, where *x* specifies the buffer name (**a-z**). If there is no buffer specified, the text will be put into the unnamed buffer. To put the contents of a buffer back into a file, use these commands:

p Puts the text after or below the cursor

P Puts the text before or above the cursor

If the text you put in the buffer started with a line that was not an entire line, it will be put back directly next to the cursor. If the text you put in the buffer formed complete lines, it will be put back as complete lines.

YP Makes a copy of the current line and positions the cursor on the copy.

nYP Makes a copy of *n* lines and positions the cursor on the copy.

Yp Makes a copy of the current line, and positions the cursor after the current line.

nYp Makes a copy of *n* lines, and positions the cursor after the current line.

To move text from one place in a file to another place in a file, use a delete (d) command to move the text into a buffer and a put (p) command to move the text from the buffer into the new location in the file). For example, to move six lines of text, use the following command sequence:

"a6dd Deletes six lines and puts the deleted text into buffer *a*

Move the cursor to the new location for the text.

ap Puts the text in buffer a following the current position of the cursor.

Marking Your Place

You can return to the previous cursor position using the command ". You can also mark places in your file with single letter labels, then return to these labels later by specifying the label name. The mark commands are:

mx The cursor position is marked with the reference x, which is a letter from a - z.

'x Returns the cursor to the poitiosn marked with the reference x.

`x Moves to the beginning of the line in which the mark is located

Note that the labels disappear when you edit another file.

Adjusting the Screen

The following commands move lines and formfeeds:

Ctrl L This is the formfeed character, and causes the screen to be repainted.

z Moves the current line to the top of the screen

z. Moves the current line to the center of the screen

z- Moves the current line to the bottom of the screen.

Switching Between Files

You can edit a different file without leaving the *vi* editor by entering the command:

:e *filename* ***<Return>***

If you enter the command and you have not yet saved your editing changes, the editor will remind you to do this first. You can use the command **:w** ***<Return>*** to to save the file. Then, give the **:e** *filename* ***<Return>*** command again. Or, you can use the command:

:e! *filename* ***<Return>***

which causes all the changes you made to the existing file to be discarded, and opens up the specified new file for editing.

Editor Options Editor options are shown in the table below.

Name	Command	Default	Description
autoindent	ai	noai	Automatic indentation on/off
autowrite	aw	noaw	Automatic write before :n, :ta, CTRL^, ! on/off
ignorecase	ic	noic	Ignore case in searching on/off
list	list	nolist	Tabs print as CTRL I; end of lines marked with S on/off
magic	magic	magic	The characters .[and * are special in scans on/off
shiftwidth	sw = n	sw = 8	Shift n characters for <, > and input CTRL D and CTRL T
tabstop	ts = n	ts = 4	Sets n number of characters indented for tab
wrapscan	ws	ws	Wrapscan on/off

You can put these statements in your EXINIT, or give the commands while running *vi*. While in *vi*, you must precede the comments with a colon (:) and end them with ***Return***. There are two kinds of options:

- numeric options
- on/off toggle options

You can set numeric options by a statement of the form:

set *option* = *value*

For example, in *vi*, the following sets the tabstop to 5:

:set ts = 5

You can change toggle options on or off, using the form:

set *option*
set no*option*

For example, in *vi* to turn autoindent on, use the command:

:set ai

To turn autoindent off in *vi*, use the command:

:set noai

To display a list of set options and their current settings, use the command:

:set all ***< Return >***

set can be abbreviated **se**. Multiple options can be placed on one line, for example:

:se ai aw ***< Return >***

Options set by the **set** comand only last while you are in the vi editor. If you want to have certain options set whenever you use the vi editor, you can create a list of commands to set up those options. The command list will be run every time you open up the editor.

The following example sets the autoindent and autowrite options, makes the @ character function as a "delete line" key; and makes the # character function as a "delete a character" key.

set ai aw|map @ dd|map # x'

Redefining characters is described later in this chapter.

The command string should be put in the variable **EXINIT**. If you use **csh**, put this line in the file **.login** in your home directory.

Recovering Deleted Text

The vi editor saves the last nine deleted blocks of text in a set of registers numbered 1-9. To get the nth previous deleted text back into your file, use the command:

"*n*p

where *n* is the number of the buffer you need. If you want to see the buffer after the nth buffer, use the command:

.

The period command, in general, causes the last command typed to be repeated. For the **"*n*p** command, however, it increments the buffer number before it re-executes the command.

Automatic Word Wrap

If you are inputting large amounts of text, and you want to avoid pressing ***Return*** at the end of every line, use the command:

wm = 10 ***< Return >***

This command causes each line to be broken at a space of at least 10 columns from the right margin. To put a broken line back together, use the command:

J

If you need to put several lines together, you can precede the J with the number of lines. For example, **4J** joins 4 lines together.

High Level Editing Functions

The *vi* editor provides a number of high level editing functions such as automatic tab indention and parenthesis matching. These functions are described below.

Autoindent

The command:

:se ai ***< Return >***

sets autoindent mode. While in this mode, all lines typed after a tab will automatically line up under that tab. You will not be able to backspace over the automatically inserted tabs, but you can use the **Ctrl D** command to backtab over the tabs. For every the cursor will back up one tab position, usually 8 spaces. You can use the command:

:se sw = *x* ***< Return >***

to change the number of spaces represented by one tab position. *x* represents the number of spaces. **sw** is called the *shiftwidth* function.

Line Shifting

You can shift lines in your file to the right and to the left by using the command:

>>

to shift your line one *shiftwidth* to the right, and

<<

to shift your line one *shiftwidth* to the left.

Matching Parentheses

If you want to track how parentheses are matched in a complex expression, you can move the cursor to the right or left parenthesis and use the command:

%

This will show you the matching parenthesis. The command can also be used for { } and [ ].

Macros

The *vi* editor allows you to create macros. Macros are user-specified combinations of keystrokes set up to execute at a single keystroke command. There are two types of macros:

1. You can put the macro in a buffer register, for example, Register *x*. You will then be able to type @*x* to run the macro. You can also type @@ to repeat the last macro.

2. You can use the map command from *vi* as follows:

   :map *x y*

   where *x* is the name of the macro, and *y* is the listing of commands. Note that:

   - *x* should be one keystroke because, it must be entered within one second (unless *notimeout* is set).
   - *x* can be no more than 10 characters
   - *y* can be no longer than 100 characters
   - To put a space, tab, or newline into *x* or *y*, you need to escape them with a **Ctrl V**.
   - Spaces and tabs inside *y* need not be escaped.

To delete a macro, use the command:

unmap *x*

If the macro *x* is **#0** through **#9**, this maps the particular function key instead of the 2 character sequence of the pound sign and the number.

You can place a **!** after the word **map** to cause the mapping to apply to input mode, instead of command mode.

Word Abbreviation

There is a word abbreviation function similar to the macro function which allows you to type an abbreviation that the editor will automatically spell out. Use the command:

:ab *xxx yyyyyyyyyy yyyyyy yyyyy*

where *xxx* is the abbreviation and *yyyyyyyyyy yyyyyy yyyyy* is the expanded version of the abbreviation. Only whole words are affected. If *xxx* is part of a larger word, the vi editor will not change it. *xxx* does not need to be limited to a single keystroke.

Lines and the vi Editor

The editor folds long logical lines onto many physical lines in the display. Line commands affect logical lines. The vi editor puts only full lines on the display. If there is not enough room for an entire logical line, the vi editor will not put part of the line on the screen, but will leave it empty and use a @ as a place holder.

You can cause tabs to be represented as ***Ctrl I*** and the ends of lines represented by **$** by using the command **:se list** ***<Return>*** to enable, and **:senolist** ***<Return>*** to disable.

When the file ends above the last line on the screen, the missing lines are represented by the character ~.

Counts

As stated previously, many *vi* commands use a number preceding the command to alter the command. For example:

new window size	: / ? '
scroll amount	CTRL D CTRL U
line/column number	z G
repeat	most of the rest

The vi editor remembers the current default window size. This is the size used when the vi editor clears and refills the screen after a search or repositions the cursor far from the current window.

The scroll commands CTRL D and CTRL U remember the amount of scroll last specified, starting with one half the window size.

Except for a few commands, the rest of the editor commands use a count to indicate a simple repetition of their effect. For example, 6w advances six words on the current line.

File Manipulation Commands

The following table lists the file manipulation commands.

Command	Description
:w	Write back changes
:wq	Write and quit
:x	Write (if necessary) and quit (same as ZZ)
:e *name*	Edit file name
:e!	Re-edit, discarding changes
:e + *name*	Edit, starting at end
:e + n	Edit, starting at line n
:e#	Edit alternate file
:w *name*	Write file name
:w! *name*	Overwrite file name
:x,yw *name*	Write lines x through y to name
:r *name*	Read file name into buffer
:n	Edit next file in argument list
:n!	Edit next file, discarding changes to current
:n args	Specify new argument list

As the list indicates, if you make changes to the editor copy of the file, but do not want to write them back, then you must give an ! after the command you are using.

The **:e** command can be given a + argument to start at the end of the file, or a +n argument to start at line n.

The % character replaces the current filename, and the # character replaces the alternate filename.

You can write part of the buffer to a file by finding out the lines that bound the range to be written using the **Ctrl G** command, and giving these numbers after the **:** and before the **w**, separated by ,'s. You can read another file into the buffer after the current line by using the **:r** command. You can also read in the output from a command by using **!cmd** instead of a file name.

To edit a number of files in succession, you can give all the names on the command line, and then edit them in turn by using the command **:n**.

SEARCHING FOR STRINGS

When you search for strings using / and **?**, the editor normally places you at the next or previous occurrences of the string. If you want to affect lines up to the line before the line containing the string, you can use a command of the form:

/pat/-**n**

to refer to the nth line before the next line containing *pat*, or you can use + instead of - to refer to the lines after the one containing *pat*. If you do not give a line offset, the vi editor affects characters up to the *pat*, rather than whole lines; thus use **+0** to affect to the line which matches.

To have the vi editor ignore the case of the words in the search, use the command:

:se ic *<Return>* **.**

The command **:se noic** *<Return>* disables this.

Magic

Strings searched for may contain regular expressions. If you do not need this, you can give the command:

set nomagic

in your EXINIT. Then, only the characters ^ and **S** are special. The character \ may be used for an extended pattern matching facility. It is necessary to use a \ before a / in a forward search, or a **?** in a backward search.

The table below shows extended forms when **magic** is set.

Command	Description
^	At beginning of pattern, matches beginning of line
$	At end of pattern, matches end of line
.	Matches any character
\<	Matches the beginning of a word
\>	Matches the end of a word
[*str*]	Matches any single character in *str*
[^*str*]	Matches any single character not in *str*
[*x*-*y*]	Matches any character between *x* and *y*
*	Matches any number of the preceding pattern

If you are using **nomagic**, the . [and * primitives are given with a preceding \.

Correction Characters

There are a number of characters which you can use to make corrections during input mode. These are shown in the following table. Your system kill character (@, **CTRL X** or **CTRL U**) will erase all the input you have given on the current line. If you want to type in your erase or kill character, you must precede it with a \.

Command	Description
CTRL H	Deletes the last input character
CTRL W	Deletes the last input character, defined as by b
erase	Your erase character, same as **CTRL H**
kill	Your kill character, deletes the input on this line
\	Escapes a following **CTRL H** and your erase and kill
ESC	Ends an insertion
DEL	Interrupts an insertion, terminating it abnormally
CR	Starts a new line
CTRL D	Backtabs over *autoindent*
0 CTRL D	Kills all the *autoindent*
^CTRL D	Same as **0 CTRL D**, but restores indent next line
CTRL V	Quotes the next non-printing character into the file

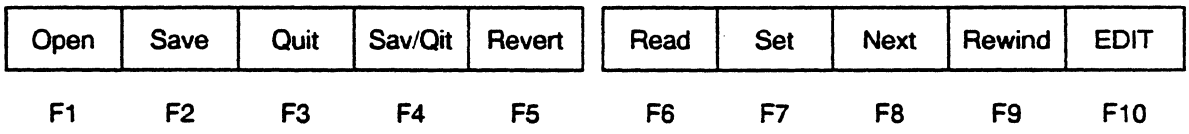
vi Softkeys

Introduction

There are several softkeys that correspond to vi commands that you can use to edit your programs. The softkeys act as macros. When you press the softkey, the command is typed onto the screen. You must then type any additional information (filename, for example), that is required by the syntax, and then press ***Return*** to execute it.

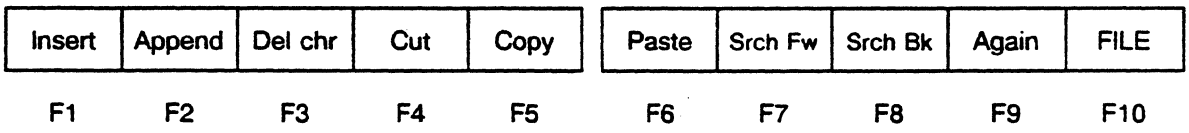
Softkey Options

There are two sets of vi softkeys. When you load the vi editor, the first softkey strip appears at the bottom of the Chameleon 32 screen, as follows:



These are the FILE softkeys. In general, the FILE softkeys enable you to select a file for editing, save files, set editor options, and exit from vi.

When you press ***F10 Edit*** the second set of vi softkeys, the EDIT softkeys, are active, as shown below:



In general, the EDIT sofkeys enable you to make changes to the text in your program file. If you select ***F1 Insert*** or ***F2 Append*** the softkey strip disappears from the screen because the other options are invalid when you are in either of these modes. To escape from the Insert or Append mode and redisplay the softkey strip, press the Esc key.

Use the F10 key to move between the EDIT and FILE softkey strips.

Table 6.1-1 lists the softkeys, a brief description of the commands they represent, and a page reference for more information about the command syntax and usage.

Softkey Strip	Command			
	Softkey	Key	Description	See Page
FILE SOFTKEYS	F1 - Open	:e	Edit file	6.1-16
	F2 - Save	:w	Save changes	6.1-16
	F3 - Quit	:q	Quit vi and return to %	6.1-2
	F4 - Sav/Qit	ZZ	Save changes and quit vi	6.1-2
	F5 - Revert	:e!	Discard changes made to current file and re-edit	6.1-16
	F6 - Read	:r	Read file into buffer	6.1-16
	F7 - Set	:set	Set editor options	6.1-11
	F8 - Next	:n	Edit next file in argument list	6.1-16
	F9 - Rewind	:rew	Edit previous file in argument list	6.1-16
EDIT SOFTKEYS	F1 - Insert	i	Insert before cursor	6.1-4
	F2 - Append	a	Append after cursor.	6.1-4
	F3 - Del chr	x	Delete a character	6.1-5
	F4 - Cut	Y	Put line into buffer	6.1-8
	F5 - Copy	Yp	Make a copy of the current line below the cursor	6.1-9
	F6 - Paste	p	Copy line from buffer below cursor	6.1-8
	F7 - Srch Fw	/	Search forward from cursor for specified string	6.2-5
	F8 - Srch Bk	?	Search backward from cursor for specified string	6.2-5
	F9 - Again	.	Redo last operation	6.2-4

Table 6.1-1: vi Softkey Assignments

vi On-Line Help

When you are in the vi editor, you can access on-line help by pressing the Help key on the Chameleon 32 keyboard. It is located to the right of the space bar.

To exit from the on-line help, and return to the vi editor, press any key.

6.2 COMMAND REFERENCE

The following tables list each *vi* command and give detailed definitions. Control characters are listed first, then special characters, then digits, and finally, upper and lower case.

A brief Quick Reference Guide is included in the Quick Reference section of this manual.

CONTROL CHARACTERS

Command	Definition
CTRL A	Density
CTRL B	Move a full window backward. A count specifies repetition. Two lines of continuity are kept if possible.
CTRL C	Not used
CTRL D	As a command, scrolls down a half-window of text. A count gives the number of logical lines to scroll, and is remembered for future CTRL D and CTRL U commands. During an insert, backtabs over autoindent white space at the beginning of a line; this white space cannot be backspaced over.
CTRL E	Move the window down one line.
CTRL F	Move a full window forward. A count specifies repetition. Two lines of continuity are kept if possible.
CTRL G	Equivalent to :f CR, printing the current file, whether it has been modified, the current line number and the number of lines in the file, and the percentage of the way through the file that you are.
CTRL H (BS)	Move left one character (same as left arrow or h). During an insert, eliminates the last input character, backing over it but not erasing it; it remains so you can see what you typed if you wish to type something only slightly different
CTRL I (TAB)	Not a command character. When inserted, it prints as some number of spaces. When the cursor is at a tab character, it rests at the last of the spaces which represent the tab. The spacing of tab stops is controlled by the tabstop option.
CTRL J (LF)	Same as down arrow.
CTRL K	Move up one line (sme as k).
CTRL L	Move right one character (same as l)
CTRL M	Move to the first non-white character on the next line.
CTRL O	Not used
CTRL Q	Not a command character. In input mode, CTRL Q quotes the next character
CTRL R	Redraws the current screen, eliminating logical lines not corresponding to physical lines (lines with only a single @ character on them).
CTRL S	Unused
CTRL U	Moves half a window forward.

CONTROL CHARACTERS

Command	Definition
CTRL V	Not a command character. In input mode, quotes the next character so that it is possible to insert non-printing and special characters into the file
CTRL W	In insert mode, deletes the last word typed.
CTRL X	Not used
CTRL Y	Moves the window up one line.
CTRL Z	Not used
**CTRL **	Not used
CTRL]	Searches for the word which is after the cursor as a tag. Equivalent to typing :ta, this word, and then a CR.
CTRL ^	Equivalent to :e # CR, returning to the previous position in the last edited file, or editing a file which you specified if you got a "No write since last change diagnostic" and do not want to have to type the file name again.
CTRL _	Not used

SPECIAL CHARACTERS

Command	Definition
space	Same as right arrow
"	Precedes a named buffer specification. There are named buffers 1-9 used for saving deleted text and named buffers a-z into which you can place text.
$	Moves to the end of the current line. If you :se list CR, then the end of each line will be shown by printing a $ after the end of the displayed text in the line. Given a count, advances to the count following the end of line.
%	Moves to the parenthesis or brace which balances the parenthesis or brace at the current cursor position.
'	When followed by a ' returns to the previous context at the beginning of a line. The previous context is set whenever the current line is moved in a non-relative way. When followed by a letter a-z, returns to the line which was marked with this letter with an m command, at the first non-white character in the line. When used with an operator such as d, the operation takes place over complete lines; if you use ', the operation takes place from the exact marked place to the current cursor position within the line.
*	Not used
+	Same as CR when used as a command
.	Repeats the last command which changed the buffer. Especially useful when deleting words or lines; you can delete some words/lines and then hit. to delete more and more words/lines. Given a count, it passes it on to the command being repeated.
@	A macro character. If this is your fill character, you must escape it with a \ to type it in during input mode, as it normally backs over the input you have given on the current line.
>	An operator which shifts lines right one shiftwidth, normally 8 spaces. Like all operators, affects lines when repeated, as in > >. Counts are passed through to the basic object.

SPECIAL CHARACTERS

Command	Definition
/	Reads a string from the last line on the screen, and scans forward for the next occurrence of this string. The normal input editing sequences may be used during the input on the bottom line. The search begins when you hit CR to terminate the pattern; the cursors moves to the beginning of the last line to indicate that the search is in progress; the search may then be terminated with a DEL or RUB, or by backspacing when at the beginning of the bottom line, returning the cursor to its initial position. Searches normally wrap end-around to find a string anywhere in the buffer. When used with an operator, the enclosed region is normally affected. By mentioning an offset from the line matched by the pattern you can force whole lines to be affected. To do this give a pattern with a closing / and then an offset +n or -n. To include the character / in the search string, you must escape it with a preceding \. A ^ at the beginning of the pattern forces the match to occur at the beginning of a line only; this speeds the search. A $ at the end of the pattern forces the match to occur at the end of a line only. More extended pattern matching is available.
1-9	Used for numeric arguments to commands
:	A prefix to a set of commands for file and option manipulation and escapes to the system. Input is given on the bottom line and terminated with an CR, and the command then executed. You can return to where you were by hitting the DEL or RUB if you hit : accidentally.
;	Repeats the last single character find which used f F t or T. A count iterates the basic scan.
<	An operator which shifts lines left one shiftwidth, normally 8 spaces. Like all operators, affects lines when repeated, as in << . Counts are passed through to the basic object.
?	Scans backwards, the opposite of /.

UPPER CASE COMMANDS

Command	Description
A	Append at the end of the line
B	Backs up a word, where words are composed of non-blank sequences, placing the cursor at the beginning of the word. A count repeats the effect.
C	Changes the rest of the text on the current line; same as c$
D	Deletes the rest of the text on the current line; same as d$
E	Moves forward to the end of a word, defined as blanks and non-blanks, like B and W. A count repeats the effect.
G	Goes to the line number given as the preceding argument, or the end of the file if no preceding count is given. The screen is redrawn with the new current line in the center if necessary.
H	Move to the first line in the window (home position).
I	Inserts at the beginning of a line; same as ^i.
J	Joins together lines, supplying appropriate white space; one space between words, two spaces after a ., and no spaces at all if the first character of the joined on line is a). A count causes n lines to be joined.
K	Not used
L	Move to the last line in the window or to the nth line above the last line in the window.
N	Scans for the next match of the last pattern given to / or ?, but in the reverse direction; this is the opposite of n.
O	Opens a new line above the current line and inputs text there up to an ESC.
P	Puts the last deleted text back before/above the cursor. The text goes back as whole lines above the cursor if it was deleted as whole lines. Otherwise, the text is inserted between the characters before and at the cursor. May be preceded by a named buffer specification "x to retrieve the contents of the buffer; buffers 1-9 contain deleted material, buffers a-z are available for general use.
Q	Not used
R	Enter replace mode. Terminates with an ESC.

UPPER CASE COMMANDS

Command	Description
S	Changes whole lines, same as cc. A count substitutes for that many lines. The lines are saved in the numeric buffers, and erased on the screen before the substitution begins.
U	Restores the current line to its state before you started changing it.
V	Not used
W	Moves forward to the beginning of a word in the current line, where words are defined as sequences of blank/non-blank characters. A count repeats the effect.
X	Deletes the character before the cursor. A count repeats the effect, but only characters on the current line are deleted.
Y	Yanks a copy of the current line into the unnamed buffer, to be put back by a later p or P. A count yanks that many lines. May be preceded by a buffer name to put lines in that buffer.
ZZ	Exits the editor. Same as :x CR. If any changes have been made, the buffer is written out to the current file. Then the editor quits.
\	Not used
^	Moves to the first non-white position on the current line
-	Not used
'	When followed by a ' returns to the previous context. The previous context is set whenever the current line is moved in a non-relative way. When followed by the letter a-z, returns to the position which was marked with this letter with an m command. When used with an operator such as d, the operation takes place from the exact marked place to the current position within the line; if you use ', the operation takes place over complete lines.

LOWER CASE COMMANDS

Command	Description
a	Appends arbitrary text after the current cursor position; the insert can continue onto multiple lines by using CR within the insert. A count causes the inserted text to be copies, but only if the inserted text is all on one line. The insertion terminates with an ESC.
b	Backs up to the beginning of a word in the current line. A word is a sequence of alphanumerics, or a sequence of special characters. A count repeats the effect.
c	Deletes the specified text and enters insert mode.
d	Deletes the specified text.
e	Advances to the end of the next word, defined as for b and w. A count repeats the effect.
f	Finds the first instance of the next character following the cursor on the current line. A count repeats the find.
g	Not used
h	Left arrow. Moves the cursor one character to the left. Like the other arrow keys, either h, the left arrow key, or one of the synonyms (^H) has the same effect. A count repeats the effect.
i	Inserts text before the cursor, otherwise like a
j	Down arrow. Moves the cursor one line down in the same column
k	Up arrow. Moves the cursor one line up. ^P is a synonym.
l	Right arrow. Moves the cursor one character to the right. SPACE is a synonym.
m	Marks the current position of the cursor in the mark register which is specified by the next character a-z. Return to this position or use with an operator using ' or `.
n	Repeats the last / or ? scanning commands.
o	Opens new lines below the current line; otherwise like O.
p	Puts text from the unnamed buffer following the cursor.
q	Not used.

LOWER CASE COMMANDS

Command	Description
r	Replaces the single character at the cursor with a single character you type. The new character may be a CR; this is the easiest way to split lines. A count replaces each of the following count characters with the single character given. R is usually more useful than r.
s	Changes the single character under the cursor to the text which follows up to an ESC; given a count, that many characters from the current line are changed. The last character to be changed is marked with $.
t	Advances the cursor up to the character before the next character typed. Most useful with operators such as d and c to delete the characters up to a following character. You can use . to delete more if this doesn't delete enough the first time.
u	Undoes the last change made to the current buffer. If repeated, will alternate between these two states, thus is its own inverse. When used after an insert which inserted text on more than one line, the lines are saved in the numeric named buffers.
v	Not used
w	Advances to the beginning of the next word.
x	Deletes the single character under the cursor. With a count deletes that many characters forward from the cursor position, but only on the current line.
y	An operator, yanks the following object into the unnamed temporary buffer. If preceded by a named buffer specification, "x, the text is placed in that buffer also. Text can be recovered later by a p or P.
z	Redraws the screen with the current line placed as specified by the following character: CR specifies the top of the screen, . the center of the screen, and - at the bottom of the screen. A count may be given after the z and before the following character to specify the new screen size for the redraw. A count before the z gives the number of the line to place in the center of the screen instead of the default current line.
-	Not used

APPENDIX A

LIMITS AND EXTENSIONS

COMPILER LIMITS

char, unsigned char	1 byte
int, unsigned, short	2 bytes
long, unsigned long	4 bytes
float	4 bytes
double	8 bytes
any type*	4 bytes
Variable names:	255 bytes (first 10 must be unique)
Floating point:	IEEE format: 32 bit float, 64 bit double, 80-bit intermediate
Register variables:	6 (2 pointer, 4 scalar)

EXTENSIONS

Structure passing, assignment and returning.

In-line M68000 assembly (parsed and assembled by compiler).

Character constants can be int and long size (ie. 'xx' and 'xxxx').

The same structure member name can be used in more than one structure.

Forward pointer references to Structures and Unions.

Addition of types unsigned long and unsigned char.

LINKER

Symbols:	256 character maximum Local and Global symbols are supported.
Segment types:	TEXT (code), DATA (initialized data), BSS (uninitialized data)

LIBRARIAN

Files per library:	unlimited
Operations:	Add file, Delete file, Extract file, create random library, list files

B.1 COMMON LIBRARY FEATURES

Introduction

This section contains information that is common to all the protocol libraries described in Appendix B. The return codes described in this section are defined in the file cham.h in the directory a:/include. **You will find it helpful to print a copy of the cham.h file for your reference.**

Sample Programs

The following sections describe each library separately. At the end of each library description, you will find sample programs that utilize that library. Some sample programs utilize more than one library; these sample programs are cross-referenced instead of being repeated.

All the sample programs can be found on the C Sample Program Diskette. The samples consist of three programs for each protocol. These samples are designed to be run alone or against each other as follows:

XXXa.c Runs on a single or dual port machine on Port A. The program begins by initiating a transmission. It can be run against program XXXb.c (with XXXb.c on a second machine, or with both programs on the same dual port machine--one on Port A and one on Port B).

XXXb.c Runs on a dual port machine on Port B. Can also run on a single port machine if the setport command is changed to call Port A instead of Port B. This program begins by waiting for a transmission.

XXXab.c Runs on a dual port machine by running Port A against Port B.

FEP State Codes

The **initp1** or **init_anal** function initializes the Chameleon Front End Processor (FEP) for a specific library. After using one of these functions, the state of the FEP is indicated by one of the following codes:

Number	Meaning
100	FEP is being used by another application and cannot be initialized by the simulation library. (The FEP is busy.)
101	The FEP has **not** been initialized by the simulation library (The FEP is currently free).
102	FEP is initialized by the current simulation library. (The FEP is running.)

Error Codes

The error codes shown below may be returned by any of the protocol library functions. Note that FEP refers to the Chameleon Front End Processor.

Number	Meaning
-200	Port is busy
-201	FEP parameter error
-202	FEP Parameter port
-203	Not available on an ISDN interface
-208	Code not found
-209	FEP cannot be started
-211	Transmission mode not valid
-212	Timeout
-213	No memory available
-214	FEP Code read
-215	FEP copier not found
-216	FEP Code not loaded
-217	Cannot halt FEP
-218	No Port B
-219	Internal error
-220	FEP Load error
-222	Undefined status
-224	FEP Data not set (initp1 not performed)
-225	Unknown FEP error

Functions

The functions listed in this section are included in all protocol libraries. The functions are described on the following pages:

FLUSH

Declaration int flush()

Description This function clears all outstanding frames in the reception buffer.

Returns

0 Successful
3 Receive buffer overflow

See global errors on page B.1-1.

Note To clear a receive buffer overflow condition, perform an **initp1**.

GETPHY

Declaration int getphy()

Description This function indicates the setting of the physical lines.

Returns 2-byte integer that can be interpreted using the figure below.

See global errors on page B.1-1.

BYTE 0 (LSB)

	BIT:	7	6	5	4	3	2	1	0
CCITT Circuit No.	PIN:	105	108	140	141	104	103	114	115
V.24 Reference	PIN:	4	20			3	2	15	17
RS232 Signal Name	SIG:	RTS	DTR			RD	TD	SCT	SCR

BYTE 1 (MSB)

	BIT:	7	6	5	4	3	2	1	0
CCITT Circuit No.	PIN:	106	107	109		125	142		
V.24 Reference	PIN:	5	6	8		22			
RS232 Signal Name	SIG:	CTS	DSR	CD		RI			

GETPORT

Declaration int getport()

Description This function returns which port is currently communicating with the library. Use the setport function to select the port.

Returns

0 Port A selected.
1 Port B selected

See global errors on page B.1-1.

GETIME

Declaration

```
#include <mtosux.h>
int getime(msbfr)
unsigned char *msbfr;
```

Description

This function gets the number of milliseconds since the system was started. *msbfr* is the address of a 6-byte buffer to receive the time value.

Returns

NOERR	Time value successfully copied
BADPRM	Unable to write into msbfr

Example

```
unsigned char mstime[6];
getime(mstime);
```

INITTIME

Declaration inittime()

Description This function initializes the .01 second and 1 second timers. The timers are set to their end time: **00** for the down counters, and **FFFE** for the up counters. You then can set the timers using the **settimer** command.

Returns See global errors on page B.1-1.

P1RESET

Declaration

```
int  p1reset(kind)
int  kind;
```

Range

kind	0	Restart simulation
	1	Stop simulation

Description

This function either restarts or resets (stops) P1 simulation. The restart function clears the reception buffer. The stop function is similar to a hardware reset.

Returns

0	Successful
-1	Parameter error

See global errors on page B.1-1.

SETLEDS

Declaration

```
int  setleds(port)
int  port;
```

Range

port	0	Port A LEDs are displayed
	1	Port B LEDs are displayed

Description

For Dual Port machines, this function controls which port's LEDs are displayed on the front panel of the Chameleon 32.

Returns

0	Successful
1	Invalid parameter
2	Dual Port board not installed

See global errors on page B.1-1.

SETPHY

Declaration

```
setphy(val)
int val;
```

Description

This function sets the physical lines using *val*, which is bit-mapped as follows:

SIMULATING DCE

	BIT:	7	6	5	4	3	2	1	0
CCITT Circuit No.	PIN:	106	107	109		125	142		
V.24 Reference	PIN:	5	6	8		22			
RS232 Signal Name	SIG:	CTS	DSR	CD		RI			

SIMULATING DTE

	BIT:	7	6	5	4	3	2	1	0
CCITT Circuit No.	PIN:	105	108	140	141				
V.24 Reference	PIN:	4	20						
RS232 Signal Name	SIG:	RTS	DTR						

Returns

See global errors on page B.1-1.

SETPORT

Declaration

```
int  setport(port)
int  port;
```

Range

port	0	Port A
	1	Port B

Description

This function sets the library to use either Port A or Port B. Use the getport function to determine which port is currently being used by the library.

Returns

0	Successful
-1	Parameter out of range
-2	Attempted to select Port B on Chameleon with a single port (Port A)

See global errors on page B.1-1.

SETTIMER

Declaration

```
int  settimer(number, value)
int  number;
unsigned int  value;
```

Range

number	0	.01 seconds (counting down)
	1	.01 seconds (counting up)
	2	seconds (counting down)
	3	seconds (counting up)

Description

This function sets the timer specified by *number* to the indicated *value*.

Returns

0	Successful
1	*number* outside of range
2	**inittime** not performed

See global errors on page B.1–1

TIMER

Declaration

int timer(number)
unsigned int(number)

Range

number	0	.01 seconds (counting down)
	1	.01 seconds (counting up)
	2	seconds (counting down)
	3	seconds (counting up)

Description

This function returns the value of a timer specified by **number**.

Returns

See global errors on page B.1–1

B.2 BIT ORIENTED PROTOCOL EMULATION C LIBRARY

Introduction

The Bit-Oriented Protocol Emulation **C** Library (libbop.a) is valid for any Bit Oriented Protocol. It is in the \lib directory.

There are two library functions which initialize the Chameleon Front End Processor:

- The initp1 function initializes the Chameleon to handle a maximum frame size of 2 kbytes. When initialized with this function, you can run Chameleon monitoring applications simultaneously to analyze the simulation traffic.
- The initp1_8k function initializes the Chameleon to handle frames up to 8 kbytes, but ***does not*** allow you to run Chameleon monitoring applications simultaneously.

The functions are described on the following pages:

Also refer to Appendix B.1 for a description of common library functions and error codes.

A sample program using the BOP library is provided at the end of this section.

DISCARD

Declaration int discard ( )

Description This function provides a means oif discarding a frame without first receiving it into a buffer.

Returns

0	Frame discarded or no frame in buffer
<0	standard error codes: *FEP not started*, etc.

See also the global error codes on page B.1–1.
See GET_NXLEN, GET_NXSTAT.

GET_NXLEN

Declaration int get_nxlen ( )

Description This function returns the length of the next frame to be passed from the FEP, providing you with information needed to optimize buffer usage and detect abnormally long frames that might otherwise exceed the allocated buffer space and overwrite part of the program.

Returns

0	No new frame
>0	Length of next frame to be received
<0	standard error codes: *FEP not started*, etc.

See also the global error codes on page B.1–1.

GET_NXSTAT

Declaration int get_nxstat ( )

Description This function lets you 'look ahead' at the status of the next frame to be received, allowing you to decide whether to receive the frame or discard it.

Returns

0	No new frame
1	Frame ok
2	Frame received has a parity error
3	Frame received contains an abort sequence
<0	standard error codes: *FEP not started*, etc.

See also the global error codes on page B.1–1.

INITP1

Declaration

```
int  initp1(type, encode, bitrate, flag)
int  type;
int  encode;
unsigned long bitrate;
int flag;
```

Range

type	0	DCE
	1	DTE
	2	ISDN
encode	0	NRZ
	1	NRZI
bitrate	50L - 64000L	
flag	0	FF
	non-0	7E

Description

This function initializes the Front End Processor (P1) and loads its simulation software. The maximum frame size that can be handled by the simulator when initialized with this code is 2 kbytes.

Note

The initp1_8k function enables you to initialize the Chameleon to handle frames up to 8 kbytes in size. However, when initialized for 8 kbyte frame size, you cannot run monitoring application simultaneously to analyze the simulation data.

Returns

0	Successful
-1	One or more parameter errors
-2	P1 program file could not be loaded

See also the global error codes on page B.1-1.

INITP1_8K

Declaration

```
int  initp1_8k(type, encode, bitrate, flag)
int  type;
int  encode;
unsigned long bitrate;
int flag;
```

Range

type	0	DCE
	1	DTE
	2	ISDN
encode	0	NRZ
	1	NRZI
bitrate	50L - 64000L	
flag	0	FF
	non-0	7E

Description

This function initializes the 8 kbyte frame size version of the Front End Processor (P1) and loads its simulation software.

The maximum frame size that can be handled by the simulator when initialized with this code is 8 kbytes. However, you do not have the ability to simultaneously monitor the line.

The initp1 function initializes the Chameleon for frames up to 2 kbytes in size and allows you to run monitoring applications simultaneously for analyzing the simulation data.

Returns

0	Successful
-1	One or more parameter errors
-2	P1 program file could not be loaded

See also the global error codes on page B.1-1.

RECEIVE

Declaration

```
int  receive(frame)
char  *frame;
```

Description

This function receives a frame from P1 and places the frame starting at the address pointed to by the passed variable *frame*.

The external global variable *rxlen* is set to the length of the received frame. If *rxlen* = 0, then no frame was received.

Returns

0	Good CRC or no frame waiting
1	Bad CRC
2	**initp1** not performed
3	Overflow
4	Abort frame received

See also the global error codes on page B.1-1.

When a RECEIVE buffer overflow error occurs (error code 3), use the **initp1** function to clear the condition. The flush function wil not clear this condition.

Example

```
...
do {
    receive(frame);
} while (rxlen == 0);
...
```

SETFLG

Declaration

```
int setflg (flag)
int flag;
```

Range

flag	0	Fill with FFs
	1	Fill with 7Es

Description This function changes the idle fill pattern.

Returns See also the global error codes on page B.1-1.

TRANSMIT

Declaration

```
int   transmit(mode, frame, length)
int   mode;
char  *frame;
int   length;
```

Range

mode	0	Good CRC
	1	Bad CRC
	2	Abort sequence

Description

This function transmits the number of bytes specified by the passed variable *length,* starting at the address pointed to by the passed pointer ***frame**.

Returns

0	Successful
1	P1 busy (transmitting previous frame)
2	**initp1** not performed
3	Parameter error
4	Buffer overflow (to clear condition use **initp1**)

See also the global error codes on page B.1-1.

TREADY

Declaration int tready()

Description This function returns the status of the transmitter.

Returns

0 Transmitter is ready for next frame
1 Transmitter is sending previous frame
2 **initp1** not performed
3 Overflow

See also the global error codes on page B.1-1.

SAMPLE PROGRAMS

There are three BOP Library sample programs provided on the C Sample Programs Disk:

- bopab.c
- bopa.c
- bopb.c

BOPAB.C

This sample program runs on a Dual Port machine and demonstrates transmitting and receiving over a V.24 interface using the libbop.a library.

```
#include    <stdio.h>
#include    <video.h>
#include    <cham.h>

/*
 * INITIALIZE PORTS A AND B (LOAD FRONT END PROCESSORS)
 */
init_ports()
{
int err;
  /*********** SETUP PORT "A" ***********/
  if ( (err=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", err);
    exit(0);
  }
  if ( (err=initp1(DCE,NRZ,96001,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", err);
    exit(0);
  }

  /*********** SETUP PORT "B" ***********/
  if ( (err=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", err);
    exit(0);
  }
  if ( (err=initp1(DTE,NRZ,96001,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", err);
    exit(0);
  }
}

/*
 * RECEIVE DATA FROM THE PORT SPECIFIED
 */
receive_data(port)
unsigned char port;
```

```
{
extern unsigned int rxlen;   /* global variable for receive data length */
unsigned char rxbuf[128];
int i;
  setport(port); /* activate appropriate port */
  rxlen = 0;
  /* WAIT FOR DATA */
  while (rxlen==0)  /* if rxlen=0 then no data was received */
    receive(rxbuf);

  for(i=0;i<rxlen;i++)
    printf("%x   ",rxbuf[i]); /* print results of data transfer */
}


/*
 * TRANSMIT DATA OUT THE PORT SPECIFIED
 */
send_data(port,buffer)
unsigned char port,*buffer;
{
int err;
  setport(port); /* activate appropriate port */
  if((err=transmit(GOOD_CRC,buffer,100)) != 0) {
    printf(FRED); /* change text color */
    switch (err) {
      case 1:
        printf("\ntransmit=%d; P1 busy sending previous frame\n",err);
        break;
      case 2:
        printf("\ntransmit=%d; Initp1 not performed\n",err);
        break;
      case 3:
        printf("\ntransmit=%d; Parameter error\n",err);
        break;
      case 4:
        printf("\ntransmit=%d; Buffer overflow, do a P1reset\n",err);
        break;
    }
    printf(FWHITE); /* change text color back to normal */
  }
}


/*
 * PROMPT USER FOR STATUS OF TRANSMITTER
 */
query_transmit_status()
{
extern long _stdvt;
unsigned char ans;
```

```
  while (1) {
    printf("\n%sPress RETURN to Transmit or any other key for Status%s\n",FYELLOW,FWHITE);
    if((ans=getcwt(_stdvt)) != '\r')
      switch (tready()) {
        case 0:
          printf("%sTransmitter is ready to send the next frame%s\n",FGREEN,FWHITE);
          break;
        case 1:
          printf("%sP1 busy sending previous frame%s\n",FRED,FWHITE);
          break;
        case 2:
          printf("%sInitp1 not performed%s\n",FRED,FWHITE);
          break;
        case 3:
          printf("%sBuffer overflow%s\n",FRED,FWHITE);
          break;
      }
    else
      break; /* exit routine */
  }
}


main()
{
  unsigned char atrans[100]; /* transmit array */
  int i;

  printf(CLEARS); /* clear the screen */

  /*****************************************/
  /* INITIALIZE BOTH FRONT END PROCESSORS */
  /*****************************************/
  init_ports();

  /*****************************************/
  /*  TRANSFER DATA FROM PORT A TO PORT B */
  /*****************************************/
  for (i=0;i<=99;i++)
    atrans[i]=0x66; /* store hex 66 into transmit array */

  query_transmit_status(); /* check status before transmitting */

  send_data(PORTA,atrans); /* transmit data out port A */

  receive_data(PORTB);  /* receive data on port B */

  /*****************************************/
  /*  TRANSFER DATA FROM PORT B TO PORT A */
  /*****************************************/
  for (i=0;i<=99;i++)
```

```
    atrans[i]=0x22; /* store hex 22 into transmit array */
  query_transmit_status(); /* check status before transmitting */
  send_data(PORTB,atrans); /* transmit data out port B */
  receive_data(PORTA);  /* receive data on port A */
  printf("\n\n");
}
```

BOPA.C

This sample program demonstrates transmitting and receiving on Port A over a V.24 interface using the libbop.a library. If this program is run against bopb.c, they must both be run in background mode.

```
#include <stdio.h>
#include <cham.h>
main()
{
  extern    long        _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  unsigned char atrans[100],rxbuf[128]; /* transmit & revieve arrays */
  int result,err,i;
  unsigned char c;

  /* SET THE ACTIVE PORT TO "A" */
  if( (err=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", err);
    exit(0);
  }

  if ( (err=initp1(DCE,NRZ,96001,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", err);
    exit(0);
  }

  for (i=0;i<=99;i++)
    atrans[i]=0x66; /* store hex 66 into transmit array */

  printf("\n hit RETURN to send the data out of port 'A'\n");
  getchar();

  result=transmit(GOOD_CRC,atrans,100); /* transmit 100 hex 66 & get result */
  printf("\nresult of transmit=%d\n",result);

  printf("\nWaiting to receive data on port 'A'\n");
  RxLEN=0;

  /* WAIT FOR DATA */
  while (rxlen==0) {
    receive(rxbuf); /* if rxlen=0 then no data was received */
    if( (c=getch(_stdvt)) == 'q' ||  c == 'Q' ) exit(0); /* fail safe */
  }

  for(i=0;i<rxlen;i++)
    printf("%x   ",rxbuf[i]); /* print results of data transfer */

  printf("\n hit RETURN to exit the program\n");
  getchar();

}
```

BOPB.C

This sample program demonstrates transmitting and receiving on Port B over a V.25 interface using the libbop.a library. If this program is run against bopa.c, they must both be run in background mode.

```
#include <stdio.h>
#include <cham.h>
main()
{
  extern    long        _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  unsigned char atrans[100],rxbuf[128]; /* transmit & recieve arrays */
  int result,err,i;
  unsigned char c;

  /* SET THE ACTIVE PORT TO "B" */
  if( (err=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", err);
    exit(0);
  }

  if ( (err=initp1(DTE,NRZ,9600l,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", err);
    exit(0);
  }

  printf("\nWaiting to receive data on port 'B'\n");
  RxLEN=0;

  /* WAIT FOR DATA */
  while (rxlen==0) {
    receive(rxbuf); /* if rxlen=0 then no data was received */
    if( (c=getch(_stdvt)) == 'q' ||  c == 'Q' ) exit(0); /* fail safe */
  }

  for(i=0;i<rxlen;i++)
    printf("%x   ",rxbuf[i]); /* print results of data transfer */

  for (i=0;i<=99;i++)
    atrans[i]=0x77; /* store hex 77 into transmit array */

  printf("\n hit RETURN to send the data out of port 'B'\n");
  getchar();

  result=transmit(GOOD_CRC,atrans,100); /* transmit 100 hex 77 & get result */
  printf("\nresult of transmit=%d\n",result);

  printf("\n hit RETURN to exit the program\n");
  getchar();

}
```

B.3 LAPD LIBRARY

Introduction

The LAPD Library is valid for the CCITT automatic Q.921 frame level. It is called *liblapd.a* and it is in the *\lib* directory. The LAPD library fully simulates AT&T Specification 5E4 LAPD and includes the following features:

- Supports up to three SAPIs and three TEIs, plus the broadcast TEI
- Automatically responds with received SAPI and TEI
- Transmits and receives XID and UI frames

The LAPD library functions are described on the pages below.

Refer to Appendix B.1 for a list of common library functions.

Automated Functions

When a command frame is received with any combination of the three user-defined SAPI's and TEI's, or one of the three SAPI's and the broadcast TEI, the response is automatically given a matching SAPI/TEI combination. Thus, you no longer have to simulate support of more than one TEI.

Since this version of LAPD Simulation can receive a number of different frame types from a number of sources that contain an information field, a ***frame status byte*** has been added at the beginning of every data packet passed to your program. Refer to the *receive* function for the frame status byte interpretation.

You can also set and display the status configuration byte using function in the LAPD library. Refer to the **set_sconfig** and **get_sconfig** library functions for more information and for an interpretation of the configuration status byte.

The simulator is configured to assume that an XID frame without an I-field is the link monitor and automatically responds. Therefore no status is in the reception buffer when an XID frame without an I-field is received.

Timeouts

The library timeouts vary from 15 seconds to 45 seconds. When you receive a timeout return from any of the functions, you should stop and reset the Front End Processor, using the function p1reset(1) or initp1(). If you continue to experience problems, reset the Chameleon 32.

Defining Links

To define a link, the following four functions are used:

- SET_TEI
- SET_SAPI
- SET_RNTEI
- SET_RSAPI

In the Single Link LAPD library, you use SET_TEI and SET_SAPI to set the TEI and SAPI values for the link.

SET_RNTEI and SET_RSAPI are used in conjunction with SET_TEI and SET_SAPI. The SET_RSAPI and SET_RNTEI commands allow you to select up to three user-defined SAPIs and TEIs. One of the defined SAPIs and TEI can then be selected as the transmit value for the link. The three user-defined SAPIs are referred to as RSAPI0, RSAPI1, and RSAPI2. Likewise, the three user-defined TEIs are referred to as RNTEI0, RNTEI1, and RNTEI2.

To communicate on a given link, use SET_SAPI and SET_TEI to select the RSAPI and RNTEI with the desired SAPI and TEI values. For example, if you use SET_RSAPI to select 0, 16, and 63 as the user-defined SAPIs, and SET_RNTEI to select 0, 10 and 20 as the user-defined TEIs, you would have the following array of values available:

	RTEI0 = 0	RTEI1 = 10	RTEI2 = 20
RSAPI0 = 0	0,0	0,10	0,20
RSAPI1 = 16	16,0	16,10	16,20
RSAPI2 = 63	63,0	63,10	63,20

You would then use SET_SAPI and SET_TEI to make one of these links active. For example:

SET_SAPI(63) Sets the transmit SAPI value to 63

SET_TEI(127) Sets the transmit TEI value to 127

Note

Only one link at a time can be in the multi-frame alignment state.

GET_MOD

Declaration int get_mod ()

Description This function returns the current modulus. The modulo may change when receiving SABMs or SABMEs.

Returns

0 Mod8
1 Mod128

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from get_mod==%d\n", get_mod ());
}
...
```

GET_RNTEI

Declaration

```
int get_rntei (val)
int val;
```

Range

val 0 - 2

Description

This function returns the value of a user-defined TEI used for reception.

The TEI (Terminal Endpoint Identifier) is a value assigned to and may be associated with a single terminal and a given point-to-point data link connection. At any time, a given terminal endpoint (TE) may contain one or more TEIs.

This value may be assigned by the carrier at the time of equipment installation, or may be automatically assigned on a call-by-call basis. The broadcast value is associated with all user-side data link entities with the same SAPI, regardless of other assigned value(s).

Returns

0 - 127	(Value of TEI *val*)
-1	*val* outside of range

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from get_rntei==%d\n", get_rntei (0));
}
...
```

GET_RSAPI

Declaration

```
int get_rsapi (val)
int val;
```

Range

val 0 -2.

Description

This function returns the value of a user-defined SAPI used for reception. The SAPI (Service Access Point Identifier) indicates the layer two service type requested or supported.

Returns

0 - 63	Receive SAPI value
-1	*val* outside of range

See also the global error codes on page B.1-1.

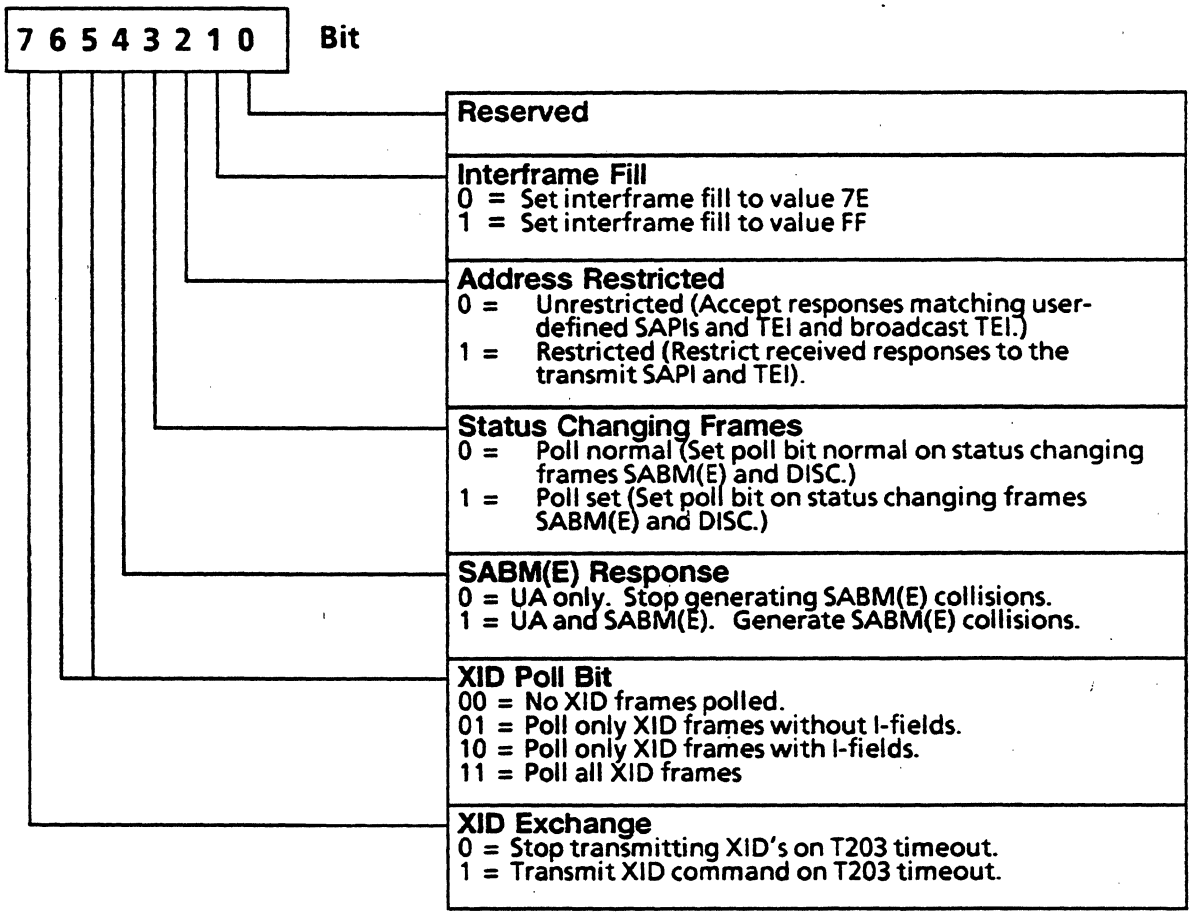
Sample Usage

```
...
{
        printf ("\nResult from get_rsapi==%d\n", get_rsapi (0));
}
...
```

GET_SCONFIG

Declaration int get_config ()

Description This function returns the status configuration byte, interpreted as shown below.



Returns Status configuration byte.

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
    printf ("\nResult from get_sconfig==%x\n", get_sconfig ());
}
...
```

GET_SIM

Declaration int get_sim ()

Description This function returns the side being simulated.

Returns

0 Network
1 Subscriber

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from get_sim==%d\n", get_sim ());
}
...
```

INITP1

Declaration

```
int   initp1(interface, station, encode, bitrate)
int   interface, station, encode;
long  bitrate;
```

Range

interface	0	DCE (V-type)
	1	DTE (V-type)
	2	ISDN (Valid only if the Primary Rate or Basic Rate Interface is physically installed)
station	0	Network
	1	Subscriber
encode	0	NRZ
	1	NRZI
bitrate	Integer value in the range 50-64000	

Description

This function initializes the Front End Processor and loads its simulation software.

Returns

0	Successful
-1	Parameter error

See also the global error codes on page B.1-1.

Sample Usage*

```
...
{
      printf ("\nResult from initp1==%d\n", initp1(0,1,0,16000L));
}
...
```

*NOTE: The program must include cham.h for the definition.

RECEIVE

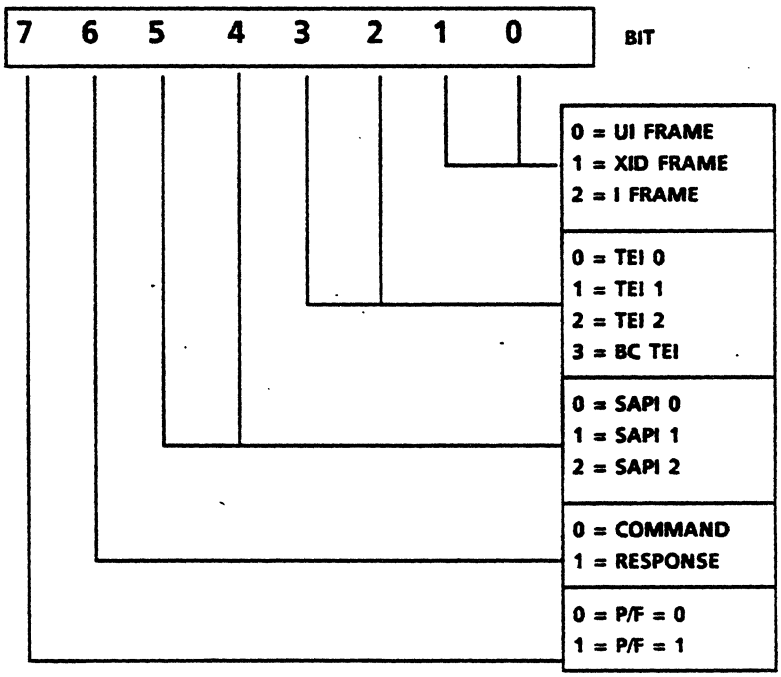
Declaration

```
extern int rxlen;
int  receive (rloc)
char  *rloc;
```

Description

This function receives an I-field from the data link layer, and places it starting at the address pointed to by the passed variable *rloc*. A status byte added in front of the received information which can be interpreted as shown below.

The external global variable ***rxlen*** will be set to the length of the received frame. If *rxlen* = 0, no I-frame was received.



Returns

0	Successful or no frame waiting
2	**initp1** not performed
4	P1 is busy

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
     char rxbuf [512];
     int a;
     extern int rxlen;
     do   {      /*If rxlen = 0, then no data was received)  */
          a=receive (&rxbuf[0]);
     } while (rxlen==0);

          printf("Receive len==%d, dl status==%x, Data Status==%x\n", rxlen, a, rxbuf [0]);
          for(a=1; a < rxlen; a++)
               printf("%x " , rxbuf [a]);
     puts("\n");
     }
     ...
```

RESTARTSIM

Declaration int restartsim()

Description This is equivalent to p1reset(0). This function restarts P1 simulation. The restart function will bring the link down, as if the **initp1** command had been carried out.

Returns

0 Successful
1 Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from Restartsim==%d\n", restartsim ());
}
...
```

SETFLG

Declaration

```
int setflg (flag)
int flag;
```

Range

0	Fill with FF
Non-zero	Fill with 7E

Description

This function changes the idle fill pattern.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from setflg==%d\n", setflg (0));
}
...
```

SET_BIT_RATE

Declaration

```
int set_bit_rate (rate)
long rate;
```

Range

rate 50 - 64000

Description

This function sets the bit rate. No check of the range is done. It is your responsibility to verify that you enter a valid bit rate.

Returns

0 Successful
1 Error

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
    printf ("\nResult from  set_bit_rate==%d\n", set_bit_rate (9600L));
}
...
```

SET_MOD

Declaration

```
int  set_mod (val)
int  val;
```

Range

val	0	Mod8
	1	Mod 128

Description

This function sets the modulus of N(S) and N(R).

Returns

0	Successful
-1	*val* outside the range 0 - 1
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_mod==%d\n", set_mod (0));
}
...
```

S_N200

Declaration

```
int s_n200(val)
int val;
```

Range

val = 1 to 512

Description

This function sets the value of N200, the number of retransmissions. This value defines the maximum number of frame retransmissions that can occur after successive lapses of the T201 timer before declaring the link unattainable, and sending disconnects.

NOTE

This function is also represented by **set_n1** in the sample programs on pages B.3-43 and B.3-45. This representation of the S_N200 function is included in the LAPD Library to ease the transition from HDLC to LAPD programming and aid in the conversion of HDLC programs to run LAPD.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage...

```
{
        printf ("\nResult from  s_n200==%d\n", s_n200 (3));
}
...
```

S_N201

Declaration

```
int s_n201(val)
int  val;
```

Range

val 1 - 512

Description

This function sets the value of N201, the maximum size of a packet in bytes. When the system receives a frame, it checks the value of the N201 variable. If the system receives a frame longer than N201, it automatically sends a frame reject (FRMR).

NOTE

This function is also represented by **set_n2** in the sample programs on pages B.3-43 and B.3-45. This representation of the S_N201 function is included in the LAPD Library to ease the transition from HDLC to LAPD programming and aid in the conversion of HDLC programs to run LAPD.

Returns

0	Successful
-1	*val* outside the range 1 - 512
1	Time out

See also the global error codes on page B.1-1.

Sample Usage...

```
{
        printf.("\nResult from  s_n201==%d\n"; s_n201 (128));
}
...
```

SET_NET

Declaration int set_net ()

Description This function sets simulation of a network. When the Chameleon 32 emulates a network, it sends commands with the C/R bit set to one, and responds with the C/R bit set to zero. It sends the selected SAPI and TEI with the C/R bit automatically set in accordance with CCITT Q. 921.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_net==%d\n", set_net ());
}
...
```

SET_RNTEI

Declaration

```
int set_rntei (val, tei)
int val, tei;
```

Range

val 0 - 2

tei 0 - 255

Description

This function specifies up to three TEI values that can be received by the RECEIVE function. Three user-defined TEIs and one broadcast TEI (127) can be defined at any one time. Once set, one of the defined TEIs can be selected as the transmit TEI using the SET_TEI function.

To disable a user-defined TEI, use SET_RNTEI to assign an invalid value to the TEI, or assign a TEI value that is already in use.

Returns

0	Successful
1	Time out
-1	Invalid value

See also the global error codes on page B.1-1.

Sample Usage

(at program initiation)

```
init( )
{
    setport(PORTA);
    set_rntei(0,PHONE_A);
    set_rntei(1,TA_A);
...
}
...
```

SET_RSAPI

Declaration

```
int set_rsapi (val, sapi)
int val, sapi;
```

Range

val 0 - 2 SAPI number

sapi 0 - 63 SAPI value

Standard SAPI values are:

0	Call control procedures
16	Packet communications procedures
63	Management procedures

Description

This function selects 1 - 3 RECEIVE SAPI values. The SAPI (Service Access Point Identifier) indicates the layer two service type requested or supported. One of these defined SAPIs can be selected as the transmit SAPI using the SET_SAPI function.

To disable a user-defined SAPI, use SET_RSAPI to assign an invalid value to the SAPI, or assign a SAPI value that is already in use.

Returns

0	Successful
1	Time out
-1	Invalid *val*

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_rsapi==%d\n", set_rsapi (1,63));
}
...
```

SET_SAPI

Declaration

```
int  set_sapi(val)
int  val;
```

Range

val 0 - 63

Description

This function sets the supported SAPI for transmission. The SAPI (Service Access Point Identifier) indicates the layer two service type requested or supported.

Standard SAPI values are:

0	Call Control procedures
16	Packet Communication procedures
63	Management procedures

Returns

0	Successful
-1	*val* outside of range
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_sapi==%d\n", set_sapi (0));
}
...
```

SET_SCONFIG

Declaration

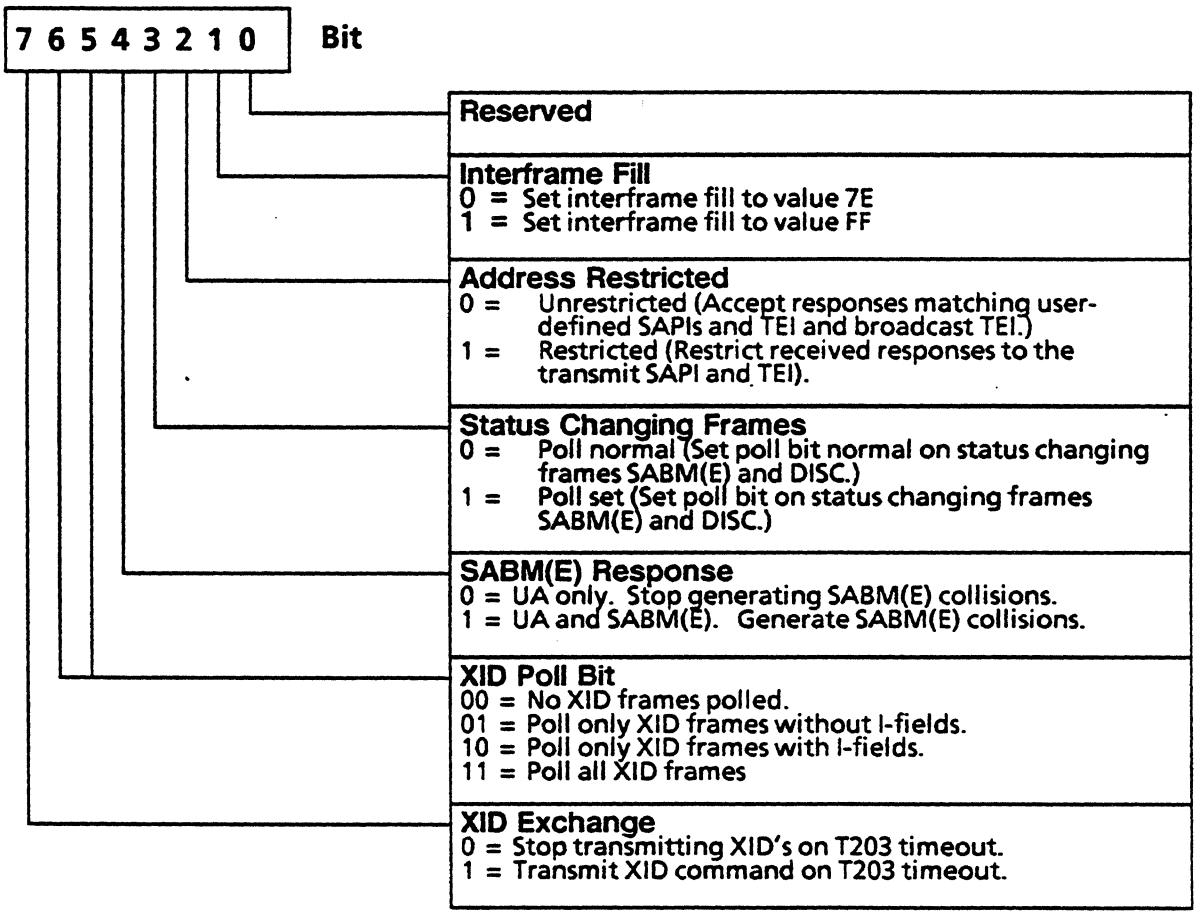
```
int set_sconfig (byte)
int byte;
```

Range

byte 0 - 255

Description

This function sets the status configuration byte which is a bit-mapped control configuration byte shown in the figure below.



Returns

0 Successful
1 Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_sconfig==%d\n", set_sconfig (0x7A));
}
...
```

The hex value 7A is represented in binary as:

0111,1010

which sets the status configuration byte as follows:

Bit 7 = 0 Stop transmitting XID frames on T203 timeout
Bit 6,5 = 11 Poll all XID frames
Bit 4 = 1 Generate SABM(E) collisions
Bit 3 = 1 Poll set
Bit 2 = 0 Unrestricted address
Bit 1 = 1 Interframe fill = FF
Bit 0 Reserved

SET_SUB

Declaration int set_sub ()

Description This function sets simulation of subscriber. When the Chameleon 32 emulates a SIMP/L LAPD subscriber, it sends commands with the C/R bit set to zero, and responses with the C/R bit set to one. It sends the selected SAPI and TEI with the C/R bit automatically set in accordance with CCITT Q. 921.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_sub==%d\n", set_sub ());
}
...
```

S_T200

Declaration

```
s_t200(val)
int  val;
```

Range

val In units of seconds; any int value (0 - 0xFFFF)

Description

This function sets the value of the T200 frame level timer and defines the maximum timeout period permitted between sending a frame and receiving an acknowledgment. The timer is started by the station when it sends any command frames. If T200 expires before the station receives a response, it retransmits the message and restarts the timer and decrements N200. When the T200 expires, the frame is retransmitted N200 - 1 times at T200 intervals. Following an N200 - 1 number of retransmissions, the station takes appropriate recovery action. If T200 expires and outstanding frames remain unacknowledged, the station re-arms T200 and sends an appropriate Supervisory Command Frame with the poll bit set.

T200 is expressed in units of seconds. The T200 timer can be disabled by setting it to zero. Disabling the T200 timer enables you to test a device without receiving a link status check or retransmitted frame while performing a manual operation or test measurement.

NOTE

This function is also represented by **t1** in the sample programs on pages B.3-43 and B.3-45. This representation of the S_T200 function is included in the LAPD Library to ease the transition from HDLC to LAPD programming and aid in the conversion of HDLC programs to run LAPD.

Returns

0 Successful
1 Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  s_t200==%d\n", s_t200 (4));
}
...
```

S_T203

Declaration

```
s_t203(val)
int  val;
```

Range

val In units of seconds; any int value (0 - 0xFFFF)

Description

This function sets the value of the T203 frame level timer. The T203 variable defines the maximum amount of time allowed between the transmission of frames. If this timer expires, the Chameleon 32 tests the link conditions by transmitting an RR, RNR, REJ or XID command, depending on the current state and configuration.

The T203 timer can be disabled by setting it to zero. Disabling the T203 timer enables you to test a device without receiving a link status check or retransmitted frame while performing a manual operation or test measurement.

NOTE

This function is also represented by **set_t2** in the sample programs on pages B.3-43 and B.3-45. This representation of the S_T203 function is included in the LAPD Library to ease the transition from HDLC to LAPD programming and aid in the conversion of HDLC programs to run LAPD.

Returns

0 Successful
1 Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  s_t203==%d\n", s_t203 (100));
}
...
```

SET_TEI

Declaration

```
int  set_tei (val)
int  val;
```

Range

val 0 - 127

Description

This function sets the transmit TEI. The TEI (Terminal Endpoint Identifier) is a value assigned to and may be associated with a single terminal and a given point-to-point data link connection. At any time, a given terminal endpoint (TE) may contain one or more TEIs.

This value may be assigned by the carrier at the time of equipment installation, or may be automatically assigned on a call-by-call basis. The broadcast value is associated with all user-side data link entities with the same SAPI, regardless of other assigned value(s).

Returns

0	Successful
-1	*val* outside of range
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_tei==%d\n", set_tei (1));
}
...
```

SET_WINDOW

Declaration

```
int  set_window (val)
int  val;
```

Range

val 1 - 7

Description

This function sets the window size for the frame level. The WINDOW variable defines the maximum number of sequentially numbered I-frames that the transmitting side can have outstanding (unacknowledged) at any given time.

Return

0	Successful
-1	*val* outside the range 1 - 7
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  set_window==%d\n", set_window (3));
}
...
```

SLOF

Declaration int slof()

Description This function disconnects the link at the frame level by sending a DISConnect.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  slof==%d\n", slof ());
}
...
```

SLON

Declaration int slon()

Description This function attempts to establish a link at the frame level by sending a SABM or SABME, depending on the selected modulus.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  slon==%d\n", slon ());
}
...
```

STATUS

Declaration int status()

Description This function returns a value indicating the status of the frame level.

Returns

0	Disconnected
1	Link connection requested
2	Packet reject state
3	Link disconnection
4	Information transfer state
5	Local station busy
6	Remote station busy
7	Local & remote station busy
8	Remote station not responding

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  status==%d\n", status ());
}
...
```

STOPSIM

Declaration int stopsim()

Description This function stops P1 simulation which is similar to a hardware reset.

Returns

0	Successful
1	Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from  stopsim==%d\n", stopsim ());
}
...
```

TRANS

Declaration

```
int trans (stat, frame, len)
char *frame;
int stat, len;
```

Range

stat	0x80	I-Frame
	0x81	UI frame
	0x82	XID Command frame
	0x83	XID Response frame

**frame* any valid pointer
len 0 - 511

Description

This function transmits a frame where the type of frame is identified by *stat*. *frame is any valid pointer and len is in the range

Returns

0 Successful
1 P1 busy (transmitting previous packet)
2 **initp1** not performed
3 Link not established for an I-Frame
5 Time out
Value returned from **trxcni()** is returned if an XID command frame is transmitted with *len* = 0
Value returned from **trxrni()** is returned if an XID response frame is transmitted with *len* = 0

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from trans==%d\n", trans (0x80,"ABCD",4));
}
...
```

TRANSMIT

Declaration

```
int transmit (packet,length)
char  *packet;
int   length;
```

Description

This function transmits the number of bytes specified by the passed variable *length,* starting at the address pointed to by the passed variable ***packet.***

Returns

Transmit calls and returns the value returned by this call:

trans(IFRAME, packet, length).

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from transmit==%d\n", transmit ("ABCD",4));
}
...
```

TRUI

Declaration

```
trui (xloc, xlen)
char *xloc;
int xlen;
```

Description

This function transmits an unnumbered I-frame. *xloc* is the location of data. *xlen* is the length of the data field.

In keeping with Q.921, a **UI** frame can be transmitted without first setting multiple frame mode. In other words, it is not necessary to exchange **SABM(E)** and **UA** frames before transmitting a **UI** frame.

Returns

TRUI calls and returns the value returned by this call:

trans(UI, packet, length)

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from trui==%d\n", trui (&buffer,4));
}
...
```

TRXCNI

Declaration int trxcni ()

Description This function transmits an XID command frame without an I-field.

Returns

0 Successful
1 Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from trxcni==%d\n", trxcni ());
}
...
```

TRXRNI

Declaration int trxrni ()

Description This function transmits an XID response frame without an I-field.

Returns Values returned are as follows:

0 Successful
1 Time out

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from trxrni==%d\n", trxrni ());
}
...
```

TRXIDC

Declaration

```
int trxidc (xloc, xlen)
char *xloc;
int xlen;
```

Description

This function transmits an XID command frame with the I-Field defined by the user-program, located at *xloc* in the memory and of *xlen* length.

Returns

TRXIDC calls and returns the value returned by this call:

trans(XIDC, xloc, xlen)

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from trxidc==%d\n", trxidc ("ABCD",4));
}
...
```

TRXIDR

Declaration

```
int trxidr (xloc, xlen)
char *xloc;
int xlen;
```

Description

This function transmits an XID response frame with the I-Field defined by the user-program, located at *xloc* in the memory and of *xlen* length.

This response frame is under control of the user-program because of the indefinite nature of the I-Field. The received data must be closely monitored to generate a timely response to the XID command.

This TRXIDR command transmits an XID response frame with a copy of the last received SAPI and TEI. In this way, the process of preparing the SAPI-C/R combination takes less time and the remote unit receives the response it is expecting.

In light of this, it is highly recommended that SIMP/L LAPD application programs be designed to monitor received frames and respond immediately when an XID command is received. If an XID response is sent at a different time, the remote unit may treat it as a command, if the last received frame was a response.

If an XID command is received without an I-field, it is assumed to be a link monitor frame. In this case, an XID response with matching SAPI and TEI is automatically transmitted without an I-field. This relieves the programmer of having to generate responses in applications which use XID frames to test the physical link.

Returns

TRXIDR calls: **trans(XIDR, xloc, xlen)**

See also the global error codes on page B.1-1.

Sample Usage

```
...
{
        printf ("\nResult from trxidr==%d\n", trxidr ("ABCD",4));
}
...
```

Sample Program

There are three sample programs provided on the C Sample Programs Disk for the LAPD Library. These are:

- LAPDA.C
- LAPDB.C
- LAPDAB.C

LAPDA.C

This program initializes the Chameleon's Port A for layer 3 transmission over a V-type interface.

```
#include    <stdio.h>
#include    <video.h>
#include    <cham.h>
#define     MOD128      1

main ()
{
extern long _stdvt;
extern unsigned int rxlen;
int i,result, err;
unsigned char rxbuf[100], txbuf[100], c;

   /*********** SETUP PORT "A" ***********/
   if ( (err=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n", err);
     exit(0);
   }
    puts(CLEARS); /* clear the screen */
    printf ("\n\n");

    /*****************************************/
    /* configure LAPD parameters & set link on */
    /*****************************************/
    printf ("\nResult from initp1 == %d",initp1(DCE,NETWORK,NRZ,96001));
    printf ("\nResult from setflg == %d",setflg(FILLFF));
    printf ("\nResult from set_mod == %d",set_mod (MOD128));
    printf ("\nResult from s_n200 == %d",s_n200 (3));
    printf ("\nResult from s_n201 == %d",s_n201 (260));
    printf ("\nResult from set_sapi == %d",set_sapi (0));
    printf ("\nResult from set_tei == %d",set_tei (10));
    printf ("\nResult from s_t200 == %d",s_t200 (10));
    printf ("\nResult from s_t203 == %d",s_t203 (20));
    printf ("\nResult from set_window == %d",set_window (3));
    printf ("\nResult from set_sconfig == %d",set_sconfig (0x00));
    printf ("\nResult from set_rntei == %d\n",set_rsapi (1,0));
    printf ("\nResult from set_rntei == %d\n",set_rntei (1,10));
```

```
   while ((status()) != 4) /* check for information transfer state */
           {
           puts(HOME); /* move cursor to the upper left corner of the screen */
           printf ("\nResult from status ==%d\n",status());
           if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0);
           }

    for(i=0;i<=25;i++)
      txbuf[i] = 65 + i; /* fill transmit buffer with upper case letters */

    puts(CLEARS);       /* clear the screen */
    printf("\n press RETURN to transmit\n");
    getchar();

    /*************************/
    /* TRANSMIT OUT SOME DATA */
    /*************************/
    result=transmit(txbuf,i); /* transmit txbuf & get result */
    printf("\nresult=%d\n",result); /* result of transmit */

    printf("\n waiting to receive\n");

    /*******************************/
    /* WAIT FOR THE INCOMMING DATA */
    /*******************************/
    do {
       receive(rxbuf); /* if rxlen=0 then no data was received */
       if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0);
    } while (rxlen==0);

    printf("\nrxlen = %d\n", rxlen);

    /*********************************************************/
    /* THIS FIRST BYTE FROM THE RECEIVE IS THE STATUS BYTE */
    /*********************************************************/
    printf("\nThe status from the receive command = %x\n",rxbuf[0]);

    for(i=1;i<rxlen;i++)
      printf("%c   ",rxbuf[i]); /* print results of data transfer */

    printf("\n press RETURN to exit program\n");
    getchar();
}
```

LAPDB.C

This program initializes the Chameleon's Port B for layer 3 transmission over a V-type interface.

In this program, set_t1, set_t2, set_n1 and set_n2 are alternate representations for, respectively, T200, T203, N200, and N203.

```
#include    <stdio.h>
#include    <video.h>
#include    <cham.h>
#define     MOD128      1

main ()
{
extern long _stdvt;
extern unsigned int rxlen;
int i,result, err;
unsigned char rxbuf[100], txbuf[100], c;

   /*********** SETUP PORT "B" ***********/
   if ( (err=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n", err);
     exit(0);
   }
    puts(CLEARS); /* clear the screen */
    printf ("\n\n");

    /*******************************************/
    /* configure LAPD parameters & set link on */
    /*******************************************/
    printf ("\nResult from initp1 == %d",initp1(DTE,SUBSCRIBER,NRZ,96001));
    printf ("\nResult from setflg == %d",setflg(FILLFF));
    printf ("\nResult from set_mod == %d",set_mod (MOD128));
    printf ("\nResult from set_n2 == %d",set_n2 (3));
    printf ("\nResult from set_n1 == %d",set_n1 (260));
    printf ("\nResult from set_sapi == %d",set_sapi (0));
    printf ("\nResult from set_tei == %d",set_tei (10));
    printf ("\nResult from set_t1 == %d",set_t1 (10));
    printf ("\nResult from set_t2 == %d",set_t2 (20));
    printf ("\nResult from set_window == %d",set_window (3));
    printf ("\nResult from set_sconfig == %d",set_sconfig (0x08));
    printf ("\nResult from set_rntei == %d\n",set_rsapi (1,0));
    printf ("\nResult from set_rntei == %d\n",set_rntei (1,10));
    printf ("\nResult from slon == %d",slon ());
```

```
  while ((status()) != 4) /* check for information transfer state */
          {
          puts(HOME); /* move cursor to the upper left corner of the screen */
          printf ("\nResult from status ==%d\n",status());
          if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
          if((status())==0)
            {
            printf ("\n Link Not Established\n");
            slof(); /* bring link down */
            exit(0);
            }
          }

   puts(CLEARS);
   printf("\n waiting to receive\n");

   /*******************************/
   /* WAIT FOR THE INCOMMING DATA */
   /*******************************/
   do {
       receive(rxbuf); /* if rxlen=0 then no data was received */
       if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
   } while (rxlen==0);

   printf("\nrxlen = %d\n", rxlen);

   /*********************************************************/
   /* THIS FIRST BYTE FROM THE RECEIVE IS THE STATUS BYTE */
   /*********************************************************/
   printf("\nThe status from the receive command = %x\n",rxbuf[0]);

   for(i=1;i<rxlen;i++)
     printf("%c   ",rxbuf[i]); /* print results of data transfer */

   for(i=0;i<=25;i++)
     txbuf[i] = 97 + i; /* fill transmit buffer with lower case letters */

   printf("\n press RETURN to transmit\n");
   getchar();

   /*************************/
   /* TRANSMIT OUT SOME DATA */
   /*************************/
   result=transmit(txbuf,i); /* transmit txbuf & get result */
   printf("\nresult=%d\n",result); /* result of transmit */

   slof(); /* bring link down */

   printf("\n press RETURN to exit program\n");
   getchar();
}
```

LAPDAB.C

This program initializes the Chameleon for Dual Port layer 3 transmission over a V-type interface.

In this program, set_t1, set_t2, set_n1 and set_n2 are alternate representations for, respectively, T200, T203, N200, and N203.

```
#include    <stdio.h>
#include    <video.h>
#include    <cham.h>
#define     MOD128      1

config_lapd()
{
    /*****************************************/
    /* configure LAPD parameters & set link on */
    /*****************************************/
    printf ("\nResult from setflg == %d",setflg(FILLFF));
    printf ("\nResult from set_mod == %d",set_mod (MOD128));
    printf ("\nResult from set_n2 == %d",set_n2 (3));
    printf ("\nResult from set_n1 == %d",set_n1 (260));
    printf ("\nResult from set_sapi == %d",set_sapi (0));
    printf ("\nResult from set_tei == %d",set_tei (10));
    printf ("\nResult from set_t1 == %d",set_t1 (10));
    printf ("\nResult from set_t2 == %d",set_t2 (20));
    printf ("\nResult from set_window == %d",set_window (3));
    printf ("\nResult from set_rntei == %d\n",set_rsapi (1,0));
    printf ("\nResult from set_rntei == %d\n",set_rntei (1,10));
}

init_ports()
{
unsigned int err;

   /********** SETUP PORT "A" **********/
   if ( (err=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n", err);
     exit(0);
   }
   printf("\n%s***** PORT A *****\n%s",FYELLOW,FBLUE);
   printf ("\nPort 'A' initp1 == %d",initp1(DCE,NETWORK,NRZ,96001));
   printf ("\nResult from set_sconfig == %d",set_sconfig (0x00));
   config_lapd();
   printf("\n%spress RETURN to continue\n%s",FRED,FBLUE);
   getchar();
```

```
   /*********** SETUP PORT "B" ***********/
   if ( (err=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n", err);
     exit(0);
   }
   printf("\n%s***** PORT B *****\n%s",FYELLOW,FBLUE);
   printf ("\nPort 'B' initp1 == %d",initp1(DTE,SUBSCRIBER,NRZ,96001));
   printf ("\nResult from set_sconfig == %d",set_sconfig (0x08));
   config_lapd();
   printf("\n%spress RETURN to continue\n%s",FRED,FWHITE);
   getchar();
}


send_data(port,buf,count)
unsigned char port, *buf, count;
{
   setport(port); /* select the port to use */

   printf(FRED);
   if(port)  /* port A is 0 and port B is 1 */
     printf("\nPress RETURN to transmit data from port-B to port-A\n");
   else
     printf("Press RETURN to transmit data from port-A to port-B\n");
   printf(FWHITE);

   getchar(); /* wait for a carrage return */

   setport(port); /* select the port */

   /*************************/
   /* TRANSMIT OUT SOME DATA */
   /*************************/
   printf("Transmit result=%d\n",transmit(buf,count));/* result of transmit */

}

wait_for_data(port)
unsigned char port;
{
unsigned char c,i,rxbuf[100];
extern long _stdvt;
extern unsigned int rxlen;

   setport(port);

   if(port)
     printf("%sReceive: on port B\n%s",FGREEN,FWHITE);
   else
     printf("%sReceive: on port A\n%s",FCYAN,FWHITE);
```

```
   /*******************************/
   /* WAIT FOR THE INCOMMING DATA */
   /*******************************/
   do {
     receive(rxbuf); /* if rxlen=0 then no data was received */
     if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
   } while (rxlen==0);

   printf("rxlen = %d\n", rxlen);

   /*********************************************************/
   /* THIS FIRST BYTE FROM THE RECEIVE IS THE STATUS BYTE */
   /*********************************************************/
   printf("The status from the receive command = %x\n",rxbuf[0]);

   for(i=1;i<rxlen;i++)
     printf("%c   ",rxbuf[i]); /* print results of data transfer */
}

main ()
{
extern long _stdvt;
int i;
unsigned char txbuf[100], c;

   /*************************************************/
   /* INITIALIZE THE DUAL PORT CHAMELEON FOR LAPD */
   /*************************************************/
   puts(CLEARS); /* clear the screen */
   init_ports();

   /*****************************************************/
   /* ESTABLISH A LINK BETWEEN THE PORTS (from port B) */
   /*****************************************************/
   printf ("\nResult from slon == %d",slon ());

   /*******************************/
   /* WAIT FOR LINK CONFIRMATION */
   /*******************************/
   while ((status()) != 4) /* check for information transfer state */
     {
      puts(CLEARS);
      puts(HOME); /* move cursor to the upper left corner of the screen */
      printf ("Result from status ==%d\n",status());
      if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0);
      if((status())==0)
        {
        printf ("\n Link Not Established\n");
        slof(); exit(0);
        }
}
```

```
    /*************************************************/
    /* SETUP THE DATA TO BE TRANSFERED, THEN SEND IT */
    /*************************************************/
    for(i=0;i<=25;i++)
      txbuf[i] = 65 + i; /* fill transmit buffer with upper case letters */
    send_data(PORTA,txbuf,i);

    /*************************************************/
    /* WAIT TO RECEIVE THE DATA ON OPPOSITE PORT */
    /*************************************************/
    wait_for_data(PORTB);

    /*************************************************/
    /* SETUP THE DATA TO BE TRANSFERED, THEN SEND IT */
    /*************************************************/
    for(i=0;i<=25;i++)
      txbuf[i] = 97 + i; /* fill transmit buffer with lower case letters */
    send_data(PORTB,txbuf,i);

    /*************************************************/
    /* WAIT TO RECEIVE THE DATA ON OPPOSITE PORT */
    /*************************************************/
    wait_for_data(PORTA);

    slof(); /* bring link down */

    printf("\n%sPress RETURN to exit program\n%s",FRED,FWHITE);
    getchar();
}
```

B.4 AUTO HDLC SIMULATION C LIBRARY

Introduction

The HDLC Simulation **C** Library is called *libhdlc.a* and is in the \lib directory. The Auto HDLC functions are listed in the table below so that you can locate them quickly within this section. Following the index, the functions are in alphabetical order with one function per page.

Also refer to Appendix B.1 for a description of common library functions and error codes.

A sample program demonstrating the use of the HDLC library is provided at the end of this section.

INITP1

Declaration

```
int   initp1(type1, type2, encode, bitrate)
int   type1;
int   type2;
int   encode;
unsigned long  bitrate;
```

Range

type1	0	DCE
	1	DTE
	2	ISDN
type2	0	Network
	1	Subscriber
encode	0	NRZ
	1	NRZI
bitrate	50L - 64000L	

Description

This function initializes the Front End Processor and loads its simulation software.

Returns

0	Successful
-1	Parameter error
-2	P1 program file could not be loaded
-3	Port is busy by the Analysis application (port should be off)

See also the global error codes on page B.1-1.

RECEIVE

Declaration

```
int   receive(packet)
chár  *packet;
```

Description

This function receives an I-frame from P1 and places the I-field frame starting at the address pointed to by the passed variable ***packet***.

The external global variable ***rxlen*** will be set to the length of the received frame. If *rxlen* = 0, then no I-frame was received.

Returns

0	Successful
1	Link not established
2	**initp1** not performed

See also the global error codes on page B.1-1.

Example

```
...
do {
    receive(frame);
} while (rxlen == 0);
...
```

SET_N1

Declaration

```
int  set_n1(val)
int  val;
```

Range

val 1 - 512

This function sets the value of N1 (the maximum size of a received frame in bytes).

Returns

0	Successful
-1	*val* outside of range

See also the global error codes on page B.1-1.

Example

```
...
char RxBuf[128];
...
set_n1(sizeof(RxBuf)+2);
...
```

SET_N2

Declaration

```
int  set_n2(val)
int  val;
```

Range

val 1 - 255

This function sets the value of N2 (number of re-transmissions).

Returns

0	Successful
-1	*val* outside of range

See also the global error codes on page B.1-1.

SET_T1

Declaration

```
int  set_t1(val)
int  val;
```

Range

val 1 - 255 seconds

Description

This function sets the value of the T1 frame level timer in units of seconds.

Returns

0	Successful
-1	*val* outside of range

See also the global error codes on page B.1-1.

SET_WINDOW

Declaration

```
int  set_window(val)
int  val;
```

Range

val 1 - 7

Description

This function sets the window size for the frame level.

Returns

0	Successful
-1	*val* outside of range

See also the global error codes on page B.1-1.

SLOF

Declaration int slof()

Description This function disconnects the link at the frame level by sending a **DISCONNECT**. Be sure to check the link status for the result of this command.

Returns See the global error codes on page B.1-1.

SLON

Declaration int slon()

Description This function attempts to establish a link at the frame level by sending a **SABM**. Verify that the link is established by using the **status** function before you transmit data.

Returns See the global error codes on page B.1-1.

STATUS

Declaration int status()

Description This function returns a value indicating the status of the frame level.

Returns

0	Disconnected
1	Link connection requested
2	Frame reject state
3	Link disconnection requested
4	Information Transfer State
5	Local Station Busy
6	Remote Station busy
7	Local and remote stations busy

See also the global error codes on page B.1-1.

TRANSMIT

Declaration

```
int transmit (packet,length)
char  *packet;
int   length;
```

Description

This function transmits an I-frame with the I-field set to the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer ***packet***.

Returns

0	Successful
1	P1 busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established

See also the global error codes on page B.1-1.

Sample Programs

There are three sample HDLC programs on the C Sample Program Disk. They are:

- hdlca.c
- hdlcb.c
- hdlcab.c

HDLCA.C

This sample program demonstrates transmitting and receiving on Port A over a V.24 interface using the libhdlc.a library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>

main()
{
  extern long _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  int i,result;
  unsigned char rxbuf[100],c; /* recieve frame data */

  /*****************/
  /* SETUP PORT A */
  /*****************/
  if ( (result=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  /*************************************/
  /* INITIALIZE THE FRONT END PROCESSOR */
  /*************************************/
  if ( (result=initp1(DCE,NETWORK,NRZ,9600l)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  /*******************************/
  /* CONFIGURE THE INPUT/OUTPUT */
  /*******************************/
  if( (result=set_n1(512)) ) printf("ERROR: set_n1=%d\n",result);
  if( (result=set_n2(5)) ) printf("ERROR: set_n2=%d\n",result);
  if( (result=set_t1(255)) ) printf("ERROR: set_t1=%d\n",result);
  if( (result=set_window(2)) ) printf("ERROR: set_window=%d\n",result);
```

```
  /**********************/
  /* WAIT FOR THE LINK */
  /**********************/
  while(status()!=4) /* wait until link is up  */
    {
    puts(CLEARS);
    printf("link status=%d\n",status());
    if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); } /* fail safe */
    }

  printf("\n%sHit RETURN to send data\n%s",FRED,FWHITE);
  getchar();

  /**********************/
  /* SEND OUT SOME DATA */
  /**********************/
  result=transmit("WXYZ",4); /* transmit ABCD & get result */
  printf("\nresult=%d\n",result); /* result of transmit */

  printf("\nWaiting to receive\n");

  /*****************************************/
  /* WAIT FOR SOME DATA FROM ANOTHER DEVICE */
  /*****************************************/
  do {
    receive(rxbuf); /* if rxlen=0 then no data was received */
    if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%x   ",rxbuf[i]); /* print results of data transfer */

  printf("\n%sHit RETURN to exit program\n%s",FRED,FWHITE);
  getchar();

  slof(); /* bring link down */

}
```

HDLCB.C

This sample program demonstrates transmitting and receiving on Port B over a V.24 interface using the libhdlc.a library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>

main()
{
  extern long _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  int i,result;
  unsigned char rxbuf[100],c; /* recieve frame data */

  /****************/
  /* SETUP PORT B */
  /****************/
  if ( (result=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  /**************************************/
  /* INITIALIZE THE FRONT END PROCESSOR */
  /**************************************/
  if ( (result=initp1(DTE,SUBSCRIBER,NRZ,9600l)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  /*******************************/
  /* CONFIGURE THE INPUT/OUTPUT */
  /*******************************/
  if( (result=set_n1(512)) ) printf("ERROR: set_n1=%d\n",result);
  if( (result=set_n2(5)) ) printf("ERROR: set_n2=%d\n",result);
  if( (result=set_t1(255)) ) printf("ERROR: set_t1=%d\n",result);
  if( (result=set_window(2)) ) printf("ERROR: set_window=%d\n",result);

  /********************/
  /* ESTABLISH A LINK */
  /********************/
  slon();
```

```
  /**********************/
  /* WAIT FOR THE LINK */
  /**********************/
  while(status()!=4) /* wait until link is up  */
    {
    puts(CLEARS);
    printf("link status=%d\n",status());
    if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
    if(status()==0) /* link cannot be established - exit routine */
      {
      printf("\nLink has been disconnected\n");
      slof(); exit(0);
      }
    }

  printf("\nWaiting to receive\n");

  /*****************************************/
  /* WAIT FOR SOME DATA FROM ANOTHER DEVICE */
  /*****************************************/
  do {
    receive(rxbuf); /* if rxlen=0 then no data was received */
    if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%x   ",rxbuf[i]); /* print results of data transfer */

  printf("\n%sHit RETURN to send data\n%s",FRED,FWHITE);
  getchar();

  /***********************/
  /* SEND OUT SOME DATA */
  /***********************/
  result=transmit("ABCD",4); /* transmit ABCD & get result */
  printf("\nresult=%d\n",result); /* result of transmit */

  printf("\n%sHit RETURN to exit program\n%s",FRED,FWHITE);
  getchar();

  slof(); /* bring link down */

}
```

HDLCAB.C

This sample program demonstrates transmitting and receiving on Dual Port machine over a V.24 interface using the libhdlc.a library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>

config()
{
int result;
  printf(FRED);
  if( (result=set_n1(512)) ) printf("ERROR: set_n1=%d\n",result);
  if( (result=set_n2(5)) ) printf("ERROR: set_n2=%d\n",result);
  if( (result=set_t1(255)) ) printf("ERROR: set_t1=%d\n",result);
  if( (result=set_window(2)) ) printf("ERROR: set_window=%d\n",result);
  printf(FWHITE);
}

init_ports()
{
int result;
  /*****************/
  /* SETUP PORT A */
  /*****************/
  printf("%sSetting up PORT A\n%s",FGREEN,FWHITE);
  if ( (result=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  if ( (result=initp1(DCE,NETWORK,NRZ,96001)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  config();  /* CONFIGURE THE PORT */

  /*****************/
  /* SETUP PORT B */
  /*****************/
  printf("%sSetting up PORT B\n%s",FGREEN,FWHITE);
  if ( (result=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }
```

```
  if ( (result=initp1(DTE,SUBSCRIBER,NRZ,9600l)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  config();  /* CONFIGURE THE PORT */
}

main()
{
  extern long _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  int i,result;
  unsigned char rxbuf[100],c,txbuf[25]; /* recieve frame data */

  puts(CLEARS); /* clear the screen */

  /*****************************************/
  /* INITIALIZE THE FRONT END PROCESSORS */
  /*****************************************/
  init_ports();

  /*********************/
  /* ESTABLISH A LINK */
  /*********************/
  slon(); /* since the last setport was PORTB, PORTB establishes the link */

  /**********************/
  /* WAIT FOR THE LINK */
  /**********************/
  while( (result=status()!=4) ) /* wait until link is up  */
    {
    puts(CLEARS);
    printf("link status=%d\n",result);
    if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); exit(0);}/* fail safe */
    if(status()==0) /* link cannot be established - exit routine */
      {
      printf("\nLink has been disconnected\n");
      slof(); exit(0);
      }
    }

  for(i=0;i<=24;i++)
    txbuf[i] = 33 + i; /* fill transmit buffer with characters */

  printf("\n%sHit RETURN to send data: PORTA to PORTB\n%s",FYELLOW,FWHITE);
  getchar();

  setport(PORTA); /* select the port to use */
```

```
  /**********************/
  /* SEND OUT SOME DATA */
  /**********************/
  result=transmit(txbuf,25); /* transmit data & get result */
  printf("%s\nTransmit result=%d\n%s",FBLUE,result,FWHITE); /* result of transmit */

  setport(PORTB); /* select the port to use */

  /***********************************/
  /* WAIT FOR SOME DATA FROM PORTA */
  /***********************************/
  do {
    receive(rxbuf); /* if rxlen=0 then no data was received */
    if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%c   ",rxbuf[i]); /* print results of data transfer */

  printf("\n%sHit RETURN to send data: PORTB to PORTA\n%s",FYELLOW,FWHITE);
  getchar();

  /**********************/
  /* SEND OUT SOME DATA */
  /**********************/
  result=transmit(txbuf,25); /* transmit data & get result */
  printf("%s\nTransmit result=%d\n%s",FBLUE,result,FWHITE); /* result of transmit */

  setport(PORTA); /* select the port to use */

  /***********************************/
  /* WAIT FOR SOME DATA FROM PORTB */
  /***********************************/
  do {
    receive(rxbuf); /* if rxlen=0 then no data was received */
    if((c=getch(_stdvt)) == 'Q' || c == 'q') { slof(); exit(0); }
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%c   ",rxbuf[i]); /* print results of data transfer */

  printf("\n%sHit RETURN to exit program\n%s",FRED,FWHITE);
  getchar();

  slof(); /* bring link down */

}
```

B.5 SDLC SIMULATION C LIBRARY

Introduction

The SDLC Simulation **C** Library is valid for the SNA automatic frame level. It is called *libsdlc.a* and is in the *\lib* directory. The functions are described in alphabetical order, one function per page, beginning on the following page.

Also refer to Appendix B.1 for a description of common library functions.

Note

N1 (maximum packet size in bytes) is set to 512 and cannot be changed.

A sample program using the SDLC library is provided at the end of this section.

INITP1

Declaration

```
int  initp1(type1,type2,encode,bitrate)
int      type1;
int      type2;
int      encode;
unsigned long   bitrate;
```

Range

type1	0	DCE
	1	DTE
	2	ISDN
type2	0	Primary
	1	Secondary
encode	0	NRZ
	1	NRZI
bitrate	50L - 64000L	

Description

This function initializes the Front End Processor and loads its simulation software.

Returns

0	Successful
-1	Parameter error
-2	Front End Processor simulation programs cannot be loaded

See also the global error codes on page B.1-1.

RECEIVE

Declaration

```
int   receive(packet)
char  *packet;
```

Description

This function receives an I-frame from P1 and places the I-field frame starting at the address pointed to by the passed variable ***packet***.

The external global variable ***rxlen*** will be set to the length of the received frame. If *rxlen* = 0, then no I-frame was received.

Returns

0	Successful
1	Link not established
2	**initp1** not performed

See also the global error codes on page B.1-1.

Example

```
...
do {
    receive(frame);
} while (rxlen == 0);
...
```

SET_ADR

Declaration

```
int set_adr(val)
int val;
```

Range

val 0 - 255

Description

This function sets the transmit and receive address. It should be used before transmitting or receiving frames.

Returns

0	Successful
-1	Parameter error

See also the global error codes on page B.1-1.

SET_N2

Declaration

```
int  set_n2(val)
int  val;
```

Range

val 1 - 255

This function sets the value of N2 (number of re-transmissions). This function is available only if the Chameleon 32 is configured as a primary station.

Returns

0	Successful
-1	*val* outside of range
5	Not configured as a primary station

See also the global error codes on page B.1-1.

SET_T1

Declaration

```
int  set_t1(val)
int  val;
```

Range

val 1 - 255 seconds

Description

This function sets the value of the T1 frame level timer in units of seconds. This function is available only if the Chameleon 32 is configured as a primary station.

Returns

0	Successful
-1	*val* outside of range
5	Not configured as a primary station

See also the global error codes on page B.1-1.

SET_T2

Declaration

```
int set_t2(val)
int val;
```

Range

val 0 - 255 seconds

Description

This function sets the value of the T2 frame level timer, which is the maximum number of seconds allowed between the transmission of frames. This function is valid only when the Chameleon is simulating a Primary station (see **initp1**).

Returns

0	Successful
-1	Error
5	Not configured as a primary station

See also the global error codes on page B.1-1.

SLOF

Declaration int slof()

Description This function disconnects the link at the frame level by sending a **DISCONNECT**. Be sure to check the link status for the result of this command.

Returns

0 Successful
5 Not configured as a primary station

See also the global error codes on page B.1-1.

SLON

Declaration int slon()

Description This function attempts to establish a link at the frame level by sending a **SARM**. This function is available only if the Chameleon 32 is configured as a primary station. Be sure to use STATUS to ascertain that the link is established before you use TRAN to transmit.

Returns

0 Successful
5 Not configured as a primary station

See also the global error codes on page B.1-1.

STATUS

Declaration int status()

Description This function returns a value indicating the status of the frame level.

Returns If the Chameleon 32 is configured as a primary station, STATUS returns the following values:

0 Normal Disconnected Mode
1 Link Request State
2 Disconnect Request State
3 Information Transfer State
4 Local Station Busy
5 Remote station busy
6 Local and remote stations busy

If the Chameleon 32 is configured as a secondary station, STATUS returns the following values:

0 Normal Disconnected Mode
1 Initialization Mode
2 Frame Reject Mode
3 Information Transfer State
4 Local Station Busy
5 Remote Station busy
6 Local and remote stations busy

See also the global error codes on page B.1-1.

TRANSMIT

Declaration

```
transmit (packet,length)
char  *packet;
int   length;
```

Description

This function transmits an I-frame with the I-field set to the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer ***packet**.

Returns

0	Successful
1	P1 busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established
4	Length error (if length > 510)

See also the global error codes on page B.1-1.

TRSIFR

Declaration

```
trsifr (packet,length)
char  *packet;
int   length;
```

Description

This function transmits a sequenced I-frame with the I-field set to the number of bytes specified by the passed variable *length,* starting at the address pointed to by the passed pointer ***packet**.

Returns

0	Successful
1	P1 busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established
4	Length error (if length > 510)

See also the global error codes on page B.1-1.

TRNSI

Declaration

```
trnsi (packet,length)
char  *packet;
int   length;
```

Description

This function transmits a non-sequenced I-frame with the I-field set to the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer ***packet***.

Returns

0	Successful
1	P1 busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established
4	Length error (if length > 510)

See also the global error codes on page B.1-1.

TRTST

Declaration

```
trtst (packet,length)
char  *packet;
int   length;
```

Description

This function transmits a test frame with the I-field set to the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer ***packet***. This function is valid only when the Chameleon is simulating a Primary station (see **initp1**).

Returns

0	Successful
1	P1 busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established
4	Length error (if length > 510)
5	Not configured as a primary station

See also the global error codes on page B.1-1.

TRUI

Declaration

```
trui (packet,length)
char  *packet;
int   length;
```

Description

This function transmits an unnumbered I-frame with the I-field set to the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer ***packet***.

Returns

0	Successful
1	P1 busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established
4	Length error (if length > 510)

See also the global error codes on page B.1-1.

XID

Declaration

```
extern char ident[ ];    / * 6 bytes*/
int   xid();
```

Description

This function transmits an XID frame containing the data in the externally available character array ident[ ], starting at ident[1], if an XID command is received. This array will be used to transmit the XID response.

Note

This function can be used only when the link is in the Normal Disconnected mode (the link is not running).

Returns

0	Successful
1	P1 not initialized
2	P1 fails to responds
3	Not in normal response mode
4	Illegal frame (if secondary)

See also the global error codes on page B.1-1.

Sample Programs

There are three sample SDLC programs on the C Sample Program Disk. They are:

- sdlca.c
- sdlcb.c
- sdlcab.c

SDLCA.C

This sample program demonstrates transmitting and receiving on Port A over a V.24 interface using the libsdlc.a library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>


main()
{
  extern long _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  int i,result;
  unsigned char rxbuf[100],c; /* receive frame data */

  /*****************/
  /* SETUP PORT A */
  /*****************/
  if ( (result=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  /*************************************/
  /* INITIALIZE THE FRONT END PROCESSOR */
  /*************************************/
  if ( (result=initp1(DCE,PRIMARY,NRZ,96001)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  /*******************************/
  /* CONFIGURE THE INPUT/OUTPUT */
  /*******************************/
  set_adr(1); /* sets transmit & receive addresses */
  set_n2(10); /* set number of retransmissions to 10 */
  set_t1(4); /* set t1 frame level timer to 4ms */
```

```
set_t2(40); /* set t2 frame level timer to 40 */

/*********************/
/* ESTABLISH A LINK */
/*********************/
slon();

/*********************/
/* WAIT FOR THE LINK */
/*********************/
while(status()!=3) /* wait until link is up  */
{
puts(CLEARS);
printf("status = %d ",status());
if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); exit(0);}/* fail safe */
}

for(;;)
 {
  /*********************/
  /* SEND OUT SOME DATA */
  /*********************/
  printf("\n%sSelect which type of transmit function to use\n%s",FRED,FWHITE);
  printf("\t 1. Transmit\n\t2. Trsifr\n");
  printf("%s\t 3. EXIT THE PROGRAM\n%s",FCYAN,FWHITE);

  while( (c=getcwt(_stdvt)-48) > 3 && c < 1 )
    ; /* wait for a valid choice */

  switch(c)
    {
    case 1:
      result=transmit("transmit",8); /* transmit data & get result */
      printf("%s\nTransmit Result=%d\n%s",FYELLOW,result,FWHITE);
      break;
    case 2:
      result=trsifr("trsifr",6); /* transmit data & get result */
      printf("%s\nTrsifr Result=%d\n%s",FYELLOW,result,FWHITE);
      break;
    case 3:
      slof();
      exit(0);
    }

  /*****************************/
  /* WAIT TO RECEIVE SOME DATA */
  /*****************************/
  printf("\nWaiting to receive\n");
  printf("%sPress 'q' to return to the MENU\n%s",FRED,FWHITE);
```

```
    do{
      receive(rxbuf); /* if rxlen=0 then no data was received */
      if((c=getch(_stdvt)) == 'Q' || c == 'q') break; /* fail safe */
      } while (rxlen==0);

    for(i=0;i<rxlen;i++)
      printf("%c   ",rxbuf[i]); /* print results of data transfer */

  } /* end forever */
}
```

SDLCB.C

This sample program demonstrates transmitting and receiving on Port B over a V.24 interface using the libsdlc.a library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>


main()
{
  extern long _stdvt;
  extern unsigned int rxlen;   /* global variable for receive data length */
  int i,result;
  unsigned char rxbuf[100],c; /* receive frame data */

  /*****************/
  /* SETUP PORT B */
  /*****************/
  if ( (result=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  /***************************************/
  /* INITIALIZE THE FRONT END PROCESSOR */
  /***************************************/
  if ( (result=initp1(DTE,SECONDARY,NRZ,96001)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  /*******************************/
  /* CONFIGURE THE INPUT/OUTPUT */
  /*******************************/
  set_adr(1); /* sets transmit & receive addresses */

  /***********************/
  /* WAIT FOR THE LINK */
  /***********************/
  while(status()!=3) /* wait until link is up  */
    {
    puts(CLEARS);
    printf("status = %d ",status());
    if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0); /* fail safe */
    }
```

```
  for(;;)
    {
    /*****************************/
    /* WAIT TO RECIEVE SOME DATA */
    /*****************************/
    printf("\nWaiting to receive\n");
    printf("%sPress 'q' to return to the MENU\n%s",FRED,FWHITE);

    do
      {
      receive(rxbuf); /* if rxlen=0 then no data was received */
      if((c=getch(_stdvt)) == 'Q' || c == 'q') break; /* fail safe */
    } while (rxlen==0);

    for(i=0;i<rxlen;i++)
      printf("%c   ",rxbuf[i]); /* print results of data transfer */

    /***********************/
    /* SEND OUT SOME DATA */
    /***********************/
    printf("\n%sSelect which type of transmit function to use\n%s",FRED,FWHITE);
    printf("\t 1. Transmit\n\t2. Trsifr\n");
    printf("%s\t 3. EXIT THE PROGRAM\n%s",FCYAN,FWHITE);

    while( (c=getcwt(_stdvt)-48) > 3 && c < 1 )
      ; /* wait for a valid choice */

    switch(c)
      {
      case 1:
        result=transmit("transmit",8); /* transmit data & get result */
        printf("%s\nTransmit Result=%d\n%s",FYELLOW,result,FWHITE);
        break;
      case 2:
        result=trsifr("trsifr",6); /* transmit data & get result */
        printf("%s\nTrsifr Result=%d\n%s",FYELLOW,result,FWHITE);
        break;
      case 3:
        exit(0);
      }

  } /* end forever */
}
```

SDLCAB.C

This sample program demonstrates transmitting and receiving on a Dual Port machine over a V.24 interface using the libsdlc.a library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>

init_ports()
{
int result;
  /*****************/
  /* SETUP PORT A */
  /*****************/
  if ( (result=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  /***************************************/
  /* INITIALIZE THE FRONT END PROCESSOR */
  /***************************************/
  if ( (result=initp1(DCE,PRIMARY,NRZ,96001)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }

  /*****************/
  /* SETUP PORT B */
  /*****************/
  if ( (result=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit(0);
  }

  /***************************************/
  /* INITIALIZE THE FRONT END PROCESSOR */
  /***************************************/
  if ( (result=initp1(DTE,SECONDARY,NRZ,96001)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit(0);
  }
}
```

```
send_data(port)
unsigned char port;
{
unsigned char c;
int result;

    setport(port); /* select which port is active */

    if(port)
      printf("SENDING FROM PORT B\n");
    else
      printf("SENDING FROM PORT A\n");

    printf("\n%sSelect which transmit function to use.\n%s",FRED,FWHITE);
    printf("\t 1. Transmit\n\t2. Trsifr\n");
    printf("%s\t 3. EXIT THE PROGRAM\n%s",FCYAN,FWHITE);

    while( (c=getcwt(_stdvt)-48) > 3 && c < 1 )
      ; /* wait for a valid choice */

    switch(c)
      {
      case 1:
        result=transmit("transmit",8); /* transmit data & get result */
        printf("%s\nTransmit Result=%d\n%s",FYELLOW,result,FWHITE);
        break;
      case 2:
        result=trsifr("trsifr",6); /* transmit data & get result */
        printf("%s\nTrsifr Result=%d\n%s",FYELLOW,result,FWHITE);
        break;
      case 3:
        slof();
        exit(0);
      }
}

get_data(port)
unsigned char port;
{
unsigned char rxbuf[100],c; /* receive frame data */
extern unsigned int rxlen;   /* global variable for receive data length */
int i;

    setport(port); /* select which port is active */

    if(port)
      printf("\n\nWaiting to receive on port B\n");
    else
      printf("\n\nWaiting to receive on port A\n");

    printf("%sPress 'q' to return to the MENU\n%s",FRED,FWHITE);
```

```
    do{
      receive(rxbuf); /* if rxlen=0 then no data was received */
      if((c=getch(_stdvt)) == 'Q' || c == 'q') break; /* fail safe */
    } while (rx en==0);

    for(i=0;i<rxlen;i++)
      printf("%c   ",rxbuf[i]); /* print results of data transfer */

    printf("\n\n");
}

main()
{
unsigned char c;

  /*************************/
  /* INITIALIZE BOTH PORTS */
  /*************************/
  init_ports();

  setport(PORTA); /* select which port is active */

  /*******************************/
  /* CONFIGURE THE INPUT/OUTPUT */
  /*******************************/
  set_adr(1); /* sets transmit & receive addresses */
  set_n2(10); /* set number of retransmissions to 10 */
  set_t1(4); /* set t1 frame level timer to 4ms */
  set_t2(40); /* set t2 frame level timer to 40 */

  /*********************/
  /* ESTABLISH A LINK */
  /*********************/
  slon();

  /*********************/
  /* WAIT FOR THE LINK */
  /*********************/
  while(status()!=3) /* wait until link is up  */
   {
   puts(CLEARS);
   printf("status = %d ",status());
   if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); exit(0);}/* fail safe */
   }

   for(;;)
   {
    /*********************************/
    /* SEND OUT SOME DATA FROM PORT A */
    /*********************************/
    send_data(PORTA);
```

```
    /*****************************************/
    /* WAIT TO RECEIVE SOME DATA ON PORT B */
    /*****************************************/
    get_data(PORTB);

    /***********************************/
    /* SEND OUT SOME DATA FROM PORT B */
    /***********************************/
    send_data(PORTB);

    /*****************************************/
    /* WAIT TO RECEIVE SOME DATA ON PORT A */
    /*****************************************/
    get_data(PORTA);

  } /* end forever */
}
```

B.6 BASIC RATE INTERFACE LIBRARY

Introduction

The Basic Rate Library enables you to use the Chameleon 32 ISDN Basic Rate Interface hardware with the C environment. The library is called **libbri.a** and is in the **/lib** directory.

In general, your application should include BRI library routines which enable you to set up and modify the Basic Rate Interface from within your application. However, when you start a C application from the Applications Selection menu, you must set up the BRI before you can access the Applications Selection menu to start the application. This can be accomplished in two ways:

- You can access the BRI Setup menu before starting the C application. This is done by pressing ***F7 Physicl*** in the main configuration menu.
- You can save the BRI Setup menu as part of the DEFAULT configuration file. The BRI is then automatically set up when the Chameleon is booted or reset.

When the C application is then started, it uses the BRI Setup menu parameters. Your application can then use other routines to modify the setup or use BRI functions, as required by your test. If the application contains a BRI library setup routine, the application setup overrides the menu setup.

When starting an application from the C Shell, if your application contains a valid BRI setup routine, it is not necessary to access the BRI Setup menu prior to starting your application.

Chapter 3.1 describes he procedure for starting applications from the C Shell and from the Applications Selection menu

Note

You **cannot** use the C BRI library and the BASIC application at the same time. (The BASIC application enables the user to monitor and modify the BRI at run time.) The BASIC application is useful for Monitoring applications and non-C simulation applications, but results in an error message when used simultaneously with the C BRI library. The BASIC application is described in the *Chameleon Protocol Interpretation Manual, Chapter 12.*

The BRI functions are described on the following pages:

Also refer to Appendix B.1 for a description of common library functions and error codes.

Several sample programs using the Basic Rate Interface library are provided at the end of this section.

Bas_version

Declaration char *Bas_version()

Description This function returns a pointer to a string which indicates the date of the Basic Rate C Library version.

SetBasic

Declaration

```
int  SetBasic(cmdblock, resblock)
int cmdblock [5];
int resblock [5];
```

Description

This function has two parameters blocks that are integer arrays. The minimum size currently needed is 4.

The *cmdblock* (command block) parameter contains the input values needed. The first item in *cmdblock* (index 0) is the command. The arguments, if any, are given as subsequent entries in the table.

The *resblock* (result block) parameter is an array containing the results, if any, of the operation requested in the *cmdblock*.

The first entry in the result block (*resblock[0]*) indicates if the command in *cmdblock* was completed successfully. The error code in *resblock[0]* is the same for all Basic Rate Library commands and is listed on the next page. Other values that are returned are given in subsequent result block elements, and are described with each command.

When the requested operation cannot be done because of an existing selection, the currently selected values are returned.

This is illustrated in the figure below:

COMMAND
Argument 1
Argument 2
etc.

RESULT
Result 1
Result 2
etc.

The commands listed below are available as the first item in *cmdblock*. Note that the commands are different for Basic Rate Interface boards 0 and 1 to accomodate a dual port Chameleon 32 with two Basic Rate Interface boards installed.

There are some hardware differences between board 0 and board 1. If you have only one Basic Rate Interface board installed on your Chameleon 32, you must use the commands that correspond to the board you are using. Generally, if you have only one Basic Rate Interface board installed, it will be board 0.

Board 0	Board 1	Command
1	101	Setup
2	102	Reactivate
3	103	Reset
4	104	Channel functions
5	105	Signal functions
6	106	Get status
9		Select trace option
10	110	NT Power
11	111	BRI Board Version
12	112	Bit Inversion
13	113	DTMF Tone Selection
14	114	Generate DTMF Tone

The commands are described on the following pages.

resblock[0] Error Codes

The error codes for *resblock[0]* are the same for all Basic Rate Library commands and are listed below.

Code	Meaning
00	Successful
01	Hardware has already been set up
02	Requested function is not available for this configuration
03	Requested channel is invalid (for B1, B2 and D)
04	Requested function is not available for this channel
05	Invalid command or request
06	DTMF tone position out of range
07	BRI parameters not set
09	BRI Menu is running (cannot access BRI from the BRI C library)
10	Basic Rate Interface board is not installed

cmdblock[0] = 1
cmdblock[0] = 101
Setup

This command must be given before any other commands can be used.

cmdblock[1] *mode*
- **1** Monitor
- **2** Simulate NT
- **3** Simulate TE

resblock[0] See page B.6-4.
resblock[1] Returns current mode, if unsuccessful

cmdblock[0] = 2
cmdblock[0] = 102
Reactivate

This command reactivates the line.

Argument None

resblock[0] See page B.6-4.

cmdblock[0] = 3
cmdblock[0] = 103
Reset

This command resets the current setup and BRI board.

Argument None

resblock[0] See page B.6-4.

cmdblock[0] = 4
cmdblock[0] = 104
Channel Functions

cmdblock[1] *mode*
- **0** Do not override current setup. Requested action is done only if it does not conflict with an existing selection.
- **1** Override current setup. If there is a conflicting setup, it is reset.

cmdblock[2] *channel*
- **1** B1 channel
- **2** B2 channel
- **3** D channel

cmdblock[3] *selection*
- **1** System
- **2** Milliwatt
- **3** Codec
- **4** External interface
- **5** Idle

resblock[0] See page B.6-4.

resblock[1] Channel as defined above (If resblock[0] ≠ 0)
resblock[2] Selection as defined above (If resblock[0] ≠ 0)

cmdblock[0] = 5
cmdblock[0] = 105
Signal Functions

cmdblock[1] For NT
- **1** Deactivate request
- **2** Send info-2
- **3** Send info-4
- **4** Activate NT
- **5** Reserved
- **6** Send single pulses
- **7** Send continuous pulses
- **8** Send info-2, test loop 2
- **9** Send info-4, test loop 2

For TE
- **1** Deactivate
- **2** Activate at priority 8
- **3** Activáte at priority 10
- **4** Activate TE
- **5** Reserved
- **6** Reserved
- **7** Reset PEB 2080
- **8** Send single pulses
- **9** Send continuous pulses
- **10** Activate test loop 3

resblock[0] See page B.6-4.

cmdblk[0] = 6
cmdblk[0] = 106
Get Status

Argument None

resblock[0] See page B.6-4.
resblock[1] Control byte received from PEB 2080.

If Simulating an NT:

resblock[2]
1 No clock signal
2 Lost signal level
3 Receiver not synchronous
4 Error
5 Info-1 received
6 Receiver synchronized
7 Deacitvation complete
8 Undefined

If Simulating a TE:

resblock[2]
1 Power up
2 Deactivate request
3 Slip detected
4 Disconnected
5 Error
6 Resynchronizing
7 Info-2 received
8 Test mode
9 Level received during test loop
10 Info-4 received, D channel priority 8 or 9
11 Info-4 received, D channel priority 10 or 11
12 Quiescent state
13 Undefined

If in Monitor mode:

resblock[1] Control byte received from PEB 2080.
resblock[2] same as resblock[2] from NT
resblock[3] same as resblock[2] from TE

cmdblk[0] = 9
Select Trace Option

This function is useful for debugging your programs.

cmdblock[1]	0	Turns off the trace.
	1	Command/result display
	2	Detailed trace

resblock[0] See page B.6-4

cmdblk[0] = 10
cmdblk[0] = 110
NT Power

This function enables you to specify the type of power provided from the NT to the TE.

cmdblock[1]	Mode	
	1	Power source 1 under normal conditions
	2	Power source 1 under emergency conditions (reverses polarity)
	3	Power source 2 under normal conditions
	4	Power source 2 under emergency conditions (reverses polarity)
	5	Off (NT power lines are off)

cmdblk[0] = 11
cmdblk[0] = 111
Board Version

This function returns the version number of the BRI board, if available.

resblock[0]	See page B.6-4
resblock[1]	BRI board version number

cmdblk[0] = 12
cmdblk[0] = 112
Bit Inversion

This function inverts the data bits on the B-Channel.

cmdblock[1]	1	Bit inversion on
	2	Bit inversion off

resblock[0] See page B.6-4

cmdblk[0] = 13
cmdblk[0] = 113
DTMF Tone Selection

This is used in conjunction with Generate DTMF Tone (see below) to generate the Dual Tone Multi-Frequency tones when using Codec on a B-channel. This function sets up the DTMF tone array. You can select a maximum of 20 digits. **The final digit must be zero.**

cmdblock[1] Position of the tone in the array.
cmdblock[2] DTMF tone (valid ASCII digit)

resblock[0] See page B.6-4

cmdblk[0] = 14
cmdblk[0] = 114
Generate DTMF Tone

This is used in conjunction with the DTMF Tone Selection function to generate the Dual Tone Multi-Frequency tones when using Codec on a B-channel. This function dials the numbers in the DTMF tone array.

resblock[0] See page B.6-4

Sample Programs

There are three sample BRI programs on the C Sample Program Disk. They are:

- bria.c
- brib.c
- briab.c

BRIA.C

This sample program demonstrates sending and receiving data on Port A using the BRI and the liblapd.c library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>
#define  MOD128      1

int resblock [5];
int cmdblock [5];

init_basic_rate()
{

   /*********************************/
   /* set basic rate to simulate NT */
   /*********************************/
   cmdblock[0] = 1;   /* port A */
   cmdblock[1] = 2;   /* NT type */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to NT = %d\n",resblock[0]);

   /*************************************/
   /*       set channel functions to D-channel */
   /*************************************/
   cmdblock[0] = 4;  /* channel */
   cmdblock[1] = 0;  /* keep current setup */
   cmdblock[2] = 3;  /* use D channel */
   cmdblock[3] = 1;  /* system */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to D-Channel = %d\n",resblock[0]);
}

resetbasic()
{
   cmdblock[0]=3;  /* reset current setup */
   SetBasic(cmdblock,resblock);
}
```

```
init_lapd()
{
int result;

   /*********** SETUP PORT "A" ***********/
   if ( (result=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n",result);
     resetbasic();
     exit(0);
   }

   if( (result=initp1(ISDN,NETWORK,NRZ,160001)) != 0)
      printf("ERROR: initp1 = %d\n",result);

   /* configure LAPD parameters */
   if( (result=setflg(FILLFF)) != 0 )
     printf("result from setflag = %d\n",result);
   if( (result=set_net()) != 0 )
     printf("ERROR: result from set_net = %d\n",result);
   if( (result=set_mod (MOD128)) != 0 )
     printf("ERROR: result from set_mod == %d\n",result);
   if( (result=set_n2 (3)) != 0 )
     printf("ERROR: result from set_n2 == %d\n",result);
   if( (result=set_n1 (260)) != 0 )
     printf("ERROR: result from set_n1 == %d\n",result);
   if( (result=set_sapi (0)) != 0 )
     printf("ERROR: result from set_sapi == %d\n",result);
   if( (result=set_tei (10)) != 0 )
     printf("ERROR: result from set_tei == %d\n",result);
   if( (result=set_t1 (10)) != 0 )
    printf("ERROR: result from set_t1 == %d\n",result);
   if( (result=set_t2 (20)) != 0 )
     printf("ERROR: result from set_t2 == %d\n",result);
   if( (result=set_window (3)) != 0 )
     printf("ERROR: result from set_window == %d\n",result);
   if( (result=set_rsapi (1,0)) != 0 )
     printf("ERROR: result from set_rntei == %d\n",result);
   if( (result=set_rntei (1,10)) != 0 )
     printf("ERROR: result from set_rntei == %d\n",result);
}
```

```
main()
{
unsigned char rxbuf[100], txbuf[30], c;
extern unsigned int rxlen;
int result,i;
char frame [10];

   for(i=0;i<10;i++)
     frame[i] = i + 48;

   /*************************************/
   /* Initialize the Basic Rate Interface */
   /*************************************/
   init_basic_rate();

   /*****************************************************/
   /* Initialize the Front End Processor (FEP) for LAPD */
   /*****************************************************/
   init_lapd();

   /*********************************/
   /* Wait for the link to come up */
   /*********************************/
   while ((status()) != 4) /* check for information transfer state */
   {
    puts(CLEARS);
    printf ("\nResult from status ==%d\n",status());
    if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); resetbasic(); exit(0);}
   }

   /*****************************************************/
   /* Transmit a UI frame after the link is established */
   /* to see if their is a fixed TEI value .            */
   /*****************************************************/
   printf("\nPress RETURN to transmit a UI frame\n");
   getchar();
   if( (result=trans(UI,frame,10)) != 0 )
     printf("Result from the UI transmit = %d\n",result);

    /*****************/
    /* transmit data */
    /*****************/
    for(i=0;i<=25;i++)     /* upper case letters */
      txbuf[i] = i + 65;

    printf("\nPress RETURN to send data from port A\n");
    getchar();

    if( (result=transmit(txbuf,i)) != 0 ) /* transmit data & get result */
      printf("TRANSMIT ERROR: result=%d\n",result);
```

```
    /*****************/
    /* receive data */
    /*****************/
    printf("\nWaiting to receive data on port A\n");

    do {
       receive(rxbuf); /* if rxlen=0 then no data was received */
       if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); resetbasic(); exit(0);}
    } while (rxlen==0);

    if(rxlen != 0)
     printf("\nReceive status = %d\n",rxbuf[0]);

    for(i=1;i<rxlen;i++)
      printf("%c   ",rxbuf[i]); /* print results of data transfer */

    slof(); /* bring link down */
    resetbasic();

    printf("\nPress RETURN to end the program\n");
    getchar();

}
```

BRIB.C This sample program demonstrates sending and receiving data on Port B using the BRI and the liblapd.c library.

```
#include <stdio.h>
#include <video.h>
#include    <cham.h>
#define     MOD128      1

int resblock [5];
int cmdblock [5];

init_basic_rate()
{

   /**********************************/
   /* set basic rate to simulate TE */
   /**********************************/
   cmdblock[0] = 101; /* port B */
   cmdblock[1] = 3;   /* TE type */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to NT = %d\n",resblock[0]);

   /**************************************/
   /*      set channel functions to D-channel */
   /**************************************/
   cmdblock[0] = 104;  /* channel */
   cmdblock[1] = 0;    /* keep current setup */
   cmdblock[2] = 3;    /* use D channel */
   cmdblock[3] = 1;    /* system */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to D-Channel = %d\n",resblock[0]);
}

resetbasic()
{
   cmdblock[0]=103;  /* reset current setup */
   SetBasic(cmdblock,resblock);
}
```

```
init_lapd()
{
int result;

   /*********** SETUP PORT "B" ***********/
   if ( (result=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n",result);
     resetbasic();
     exit(0);
   }

   if( (result=initp1(ISDN,SUBSCRIBER,NRZ,160001)) != 0)
      printf("ERROR: initp1 = %d\n",result);

   /* configure LAPD parameters */
   if( (result=setflg(FILLFF)) != 0 )
     printf("result from setflag = %d\n",result);
   if( (result=set_mod (MOD128)) != 0 )
     printf("ERROR: result from set_mod == %d\n",result);
   if( (result=set_n2 (3)) != 0 )
     printf("ERROR: result from set_n2 == %d\n",result);
   if( (result=set_n1 (260)) != 0 )
     printf("ERROR: result from set_n1 == %d\n",result);
   if( (result=set_sapi (0)) != 0 )
     printf("ERROR: result from set_sapi == %d\n",result);
   if( (result=set_tei (10)) != 0 )
     printf("ERROR: result from set_tei == %d\n",result);
   if( (result=set_t1 (10)) != 0 )
     printf("ERROR: result from set_t1 == %d\n",result);
   if( (result=set_t2 (20)) != 0 )
     printf("ERROR: result from set_t2 == %d\n",result);
   if( (result=set_window (3)) != 0 )
     printf("ERROR: result from set_window == %d\n",result);
   if( (result=set_rsapi (1,0)) != 0 )
     printf("ERROR: result from set_rntei == %d\n",result);
   if( (result=set_rntei (1,10)) != 0 )
     printf("ERROR: result from set_rntei == %d\n",result);
}

main()
{
unsigned char rxbuf[100], txbuf[30], c;
extern unsigned int rxlen;
int result,i,j;

   /*****************************************/
   /* Initialize the Basic Rate Interface */
   /*****************************************/
   init_basic_rate();
```

```
/*******************************************************/
/* Initialize the Front End Processor (FEP) for LAPD */
/*******************************************************/
init_lapd();

/*************************************/
/* establish information transfer state */
/*************************************/
if( (result=slon()) != 0 )
  printf("result from slon() = %d\n",result);

/*********************************/
/* Wait for the link to come up */
/*********************************/
while ((status()) != 4) /* check for information transfer state */
{
 puts(CLEARS);
 printf ("\nResult from status ==%d\n",status());
 if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); resetbasic(); exit(0);}
}

for(j=0;j<2;j++) {
   /****************/
   /* receive data */
   /****************/
   printf("\nWaiting to receive data on port B\n");

   do {
      receive(rxbuf); /* if rxlen=0 then no data was received */
      if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); resetbasic(); exit(0);}
   } while (rxlen==0);

   if(rxlen != 0)
     printf("\nReceive status = %d\n",rxbuf[0]);


   for(i=1;i<rxlen;i++)
     printf("%c   ",rxbuf[i]); /* print results of data transfer */
}

 /*****************/
 /* transmit data */
 /*****************/
 for(i=0;i<=25;i++)     /* lower case letters */
   txbuf[i] = i + 97;

 printf("\nPress RETURN to send data from port B\n");
 getchar();

 if( (result=transmit(txbuf,i)) != 0 ) /* transmit data & get result */
   printf("TRANSMIT ERROR: result=%d\n",result);

 resetbasic();

 printf("\nPress RETURN to end the program\n");
 getchar();

}
```

BRIAB.C

This sample program demonstrates sending and receiving data using a Dual Port Machine with the BRI and the liblapd.c library.

```
#include <stdio.h>
#include <video.h>
#include <cham.h>
#define  MOD128      1

int cmdblock [5], resblock [5];
int result;

init_basic_rate()
{
   /*****************************/
   /* set port A to simulate NT */
   /*****************************/
   cmdblock[0] = 1;   /* port A */
   cmdblock[1] = 2;   /* NT type */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to NT = %d\n",resblock[0]);

   /*************************************/
   /*       set channel functions to D-channel */
   /*************************************/
   cmdblock[0] = 4;  /* channel */
   cmdblock[1] = 0;  /* keep current setup */
   cmdblock[2] = 3;  /* use D channel */
   cmdblock[3] = 1;  /* system */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to D-Channel = %d\n",resblock[0]);

   /*****************************/
   /* set port B to simulate TE */
   /*****************************/
   cmdblock[0] = 101; /* port B */
   cmdblock[1] = 3;   /* TE type */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to NT = %d\n",resblock[0]);
```

```
   /*************************************/
   /*       set channel functions to D-channel */
   /*************************************/
   cmdblock[0] = 104;  /* channel */
   cmdblock[1] = 0;    /* keep current setup */
   cmdblock[2] = 3;    /* use D channel */
   cmdblock[3] = 1;    /* system */
   SetBasic(cmdblock,resblock);
   if( resblock[0] != 0 )
     printf("ERROR: result from SetBasic to D-Channel = %d\n",resblock[0]);
}

resetbasic()
{
   cmdblock[0]=3;    /* reset port A's current setup */
   SetBasic(cmdblock,resblock);
   cmdblock[0]=103;  /* reset port B's current setup */
   SetBasic(cmdblock,resblock);
}

config_lapd()
{
   /*****************************/
   /* configure LAPD parameters */
   /*****************************/
   if( (result=setflg(FILLFF)) != 0 )
     printf("result from setflag = %d\n",result);
   if( (result=set_mod (MOD128)) != 0 )
     printf("ERROR: result from set_mod == %d\n",result);
   if( (result=set_n2 (3)) != 0 )
     printf("ERROR: result from set_n2 == %d\n",result);
   if( (result=set_n1 (260)) != 0 )
     printf("ERROR: result from set_n1 == %d\n",result);
   if( (result=set_sapi (0)) != 0 )
     printf("ERROR: result from set_sapi == %d\n",result);
   if( (result=set_tei (10)) != 0 )
     printf("ERROR: result from set_tei == %d\n",result);
   if( (result=set_t1 (10)) != 0 )
     printf("ERROR: result from set_t1 == %d\n",result);
   if( (result=set_t2 (20)) != 0 )
     printf("ERROR: result from set_t2 == %d\n",result);
   if( (result=set_window (3)) != 0 )
     printf("ERROR: result from set_window == %d\n",result);
   if( (result=set_rsapi (1,0)) != 0 )
     printf("ERROR: result from set_rntei == %d\n",result);
   if( (result=set_rntei (1,10)) != 0 )
     printf("ERROR: result from set_rntei == %d\n",result);
   if( (result = set_sub()) !=0)
     printf("ERROR: result from set_sub == %d\n",result);
}
```

```
init_lapd()
{
   /*********** SETUP PORT "A" ***********/
   if ( (result=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n",result);
     resetbasic();
     exit(0);
   }

   if( (result=initp1(ISDN,NETWORK,NRZ,16000)) != 0)
      printf("ERROR: initp1 = %d\n",result);

   if( (result=set_net()) != 0 )
     printf("ERROR: result from set_net = %d\n",result);

   config_lapd();

   /*********** SETUP PORT "B" ***********/
   if ( (result=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n",result);
     resetbasic();
     exit(0);
   }

   if( (result=initp1(ISDN,SUBSCRIBER,NRZ,16000)) != 0)
      printf("ERROR: initp1 = %d\n",result);

   config_lapd();
}


send_data(port,buf,buf_size)
unsigned char port, *buf;
int buf_size;
{
    setport(port);              /* select which port to use */
    if(port)
      printf("\n\n\nPress RETURN to send data from port B\n");
    else
      printf("\n\n\nPress RETURN to send data from port A\n");

    getchar();
    puts(CLEARS); /* clear the screen */
    if( (result=transmit(buf,buf_size)) != 0 )/* transmit data & get result */
      printf("TRANSMIT ERROR: result=%d\n",result);
}
```

```
get_data(port)
unsigned char port;
{
unsigned char rxbuf[100], c;
extern unsigned int rxlen;
int i;

    setport(port);                /* select which port to use */
    if(port)
      printf("\nData received on port B:\n");
    else
      printf("\nData received on port A:\n");

    do {
       receive(rxbuf); /* if rxlen=0 then no data was received */
       if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); resetbasic(); exit(0);}
    } while (rxlen==0);

    if(rxlen != 0)
      printf("\nReceive status = %d\n",rxbuf[0]);

    for(i=1;i<rxlen;i++)
      printf("%c   ",rxbuf[i]); /* print results of data transfer */
}

main()
{
unsigned char txbuf[30], c;
int i;
char frame [10];

   for(i=0;i<10;i++) /* setup the data for the UI frame, ascii 0 - 9 */
     frame[i] = i + 48;

   /*****************************************/
   /* Initialize the Basic Rate Interface */
   /*****************************************/
   init_basic_rate();

   /*********************************************************/
   /* Initialize the Front End Processor (FEP) for LAPD */
   /*********************************************************/
   init_lapd();

   /*************************************************************/
   /* The subscriber establishs the information transfer state */
   /*************************************************************/
   if( (result=slon()) != 0 )
     printf("result from slon() = %d\n",result);
```

```
   /*********************************/
   /* Wait for the link to come up */
   /*********************************/
   while ((status()) != 4) /* check for information transfer state */
   {
    puts(CLEARS);
    printf ("\nResult from status ==%d\n",status());
    if((c=getch(_stdvt)) == 'Q' || c == 'q') {slof(); resetbasic(); exit(0);}
   }

   /*****************************************************/
   /* Transmit a UI frame after the link is established */
   /* from port A to port B.                            */
   /*****************************************************/
   printf("\nPress RETURN to transmit a UI frame from port A\n");
   getchar();

   setport(PORTA);
   if( (result=trans(UI,frame,10)) != 0 )
     printf("Result from the UI transmit = %d\n",result);

   get_data(PORTB); /* Receive the UI Frame on port B */

    /***************************************/
    /* Transmit data from port A to port B */
    /***************************************/
    for(i=0;i<=25;i++)     /* upper case letters */
      txbuf[i] = i + 65;

    send_data(PORTA,txbuf,i);

    get_data(PORTB); /* receive data on port B */

    /***************************************/
    /* Transmit data from port B to port A */
    /***************************************/
    for(i=0;i<=25;i++)     /* lower case letters */
      txbuf[i] = i + 97;

    send_data(PORTB,txbuf,i);

    get_data(PORTA); /* receive data on port A */

    slof(); /* bring link down */
    resetbasic();

    printf("\n\n\nPress RETURN to end the program\n");
    getchar();
}
```

B.7 BSC C LIBRARY

Introduction

The BSC C Library (libbsc.a) is valid for IBM's Binary Synchronous Communications protocol. It is in the \lib directory.

The functions are described on the following pages:

Also refer to Appendix B.1 for a description of common library functions and error codes.

A sample program using the BSC library is provided at the end of this section.

IDLE_MODE

Declaration

```
#include <cham.h>
int idle_mode(mode)
int mode;
```

Range

mode	IDLE or 0	Transmits the idle character (FF)
	SYNC or 1	Transmits the SYN character

Description

This function specifies the character to be transmitted while the line is idle.

Returns

0 Successful

See also the global error codes on page B.1-1.

INITP1

Declaration

```
int  initp1(type, encode, bitrate, crc, data)
int type;
struct BSC_CTRL encode;
unsigned long bitrate;
int crc;
int data;
```

Range

type

0	DCE
1	DTE

encode This is a structure that defines the control characters for the BSC protocol. It is defined as follows:

```
struct BSC_CTRL
  {
  unsigned char eot;
  unsigned char syn;
  unsigned char dle;
  unsigned char stx;
  unsigned char etx;
  unsigned char soh;
  unsigned char etb;
  unsigned char itb;
  unsigned char enq;
  };
```

bitrate 50L - 64000L

crc

0	CRC16 block check algorithm
1	CCITT-CRC block check algorithm

data

0x10	EBCDIC data
0x04	ASCII data (no parity)
0X01	ASCII data (even parity)
0x00	ASCII data (odd parity)

Description This function initializes P1 and loads its simulation software.

Returns

0	Successful
-1	One or more parameter errors
-2	Front End Processor program could not be loaded

See also the global error codes on page B.1-1.

RECEIVE

Declaration

```
int   receive(frame)
char  *frame;
```

Description

This function receives a frame from P1 and places the frame starting at the address pointed to by the passed variable *frame*.

The external global variable *rxlen* is set to the length of the received frame. If *rxlen* = 0, then no frame was received.

Returns

0	Good BCC or no frame waiting
1	Bad BCC
2	**initp1** not performed
3	Overflow

See also the global error codes on page B.1-1.

Example

```
...
do {
    receive(frame);
} while (rxlen == 0);
...
```

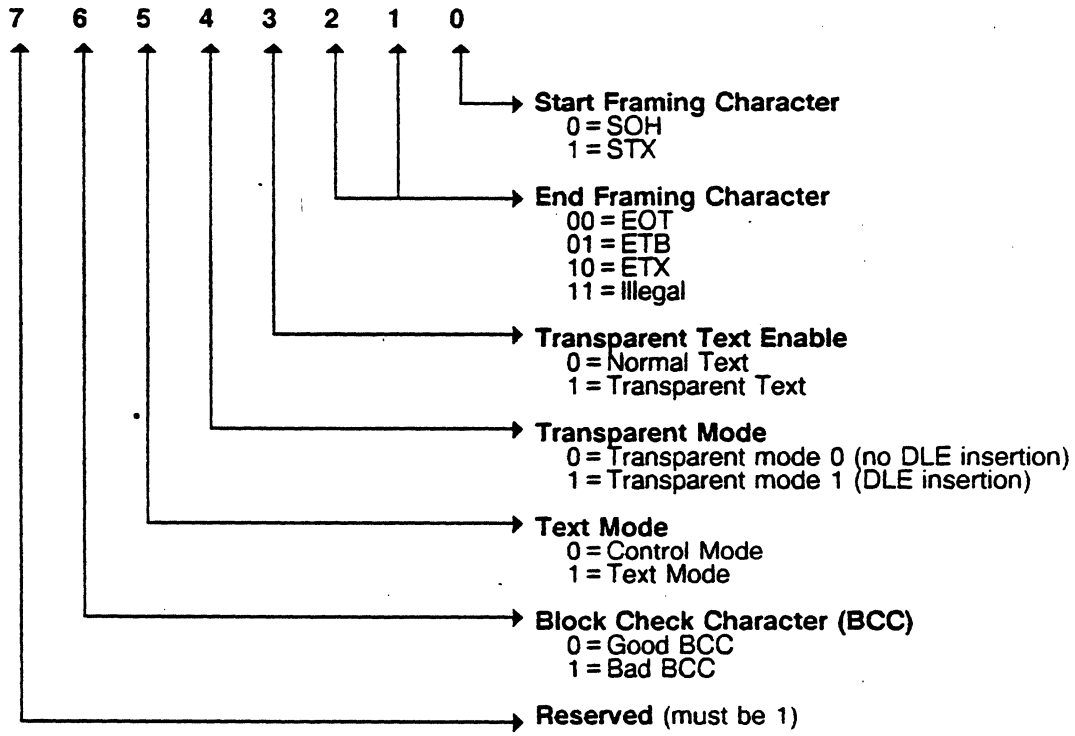
TRANSMIT

Declaration

```
int   transmit(mode, frame, length)
int   mode;
char  *frame;
int   length;
```

Description

This function transmits the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer **frame*, with the control characters and block check as specified by the passed value *mode*. The mode parameter is defined as follows:



Returns

0	Successful
1	P1 busy (transmitting previous frame)
2	**initp1** not performed
3	Parameter error
4	Buffer overflow

See also the global error codes on page B.1-1

TREADY

Declaration int tready()

Description This function returns the status of the Front End Processor transmitter.

Returns

0	Transmitter is ready for next frame
1	Transmitter is busy (sending previous frame)
2	**initp1** not performed
3	Overflow

See also the global error codes on page B.1-1

Sample Programs

There are three sample BSC programs on the C Sample Program Disk. They are:

- bsca.c
- bscb.c
- bscab.c

BSCA.C

This sample program demonstrates transmitting and receiving on Port A over a V.24 interface using the libbsc.a library.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
struct BSC_CTRL;

main()
{
char        ch,crc;
struct      BSC_CTRL    encode;
extern unsigned int rxlen;   /* global variable for receive data length */
unsigned char atrans[100],rxbuf[128]; /* transmit & revieve arrays */
int result,i;

   encode.eot    =   55;
   encode.syn    =   50;
   encode.dle    =   16;
   encode.stx    =   02;
   encode.etx    =   03;
   encode.soh    =   01;
   encode.etb    =   38;
   encode.itb    =   31;
   encode.enq    =   45;

   /*******************************/
   /* SET THE ACTIVE PORT TO "A" */
   /*******************************/
   if( (result=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }

   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DCE,&encode.eot,9600l,CCITT,ASCII_EVEN_DATA)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }
```

```
   for (i=0;i<=99;i++)
   atrans[i]=0x66; /* store hex 66 into transmit array */

   /*********************************************/
   /* WAIT UNITL THE SYSTEM IS READY TO TRANSMIT */
   /*********************************************/
   while((tready())!=0){            /* loop while transmitter not ready */
      puts(CLEARS);  /* clear screen */
      printf("PRESS ANY KEY TO ABORT \n");
      printf("TREADY STATUS =%d\n",tready());
      if((ch = getch (_stdvt)) != -1)       /* check for key pressed */
         exit(0);
   }

   /***********************/
   /* SEND OUT SOME DATA */
   /***********************/
   printf("\n hit RETURN to send the data out of port 'A'\n");
   getchar();

   result=transmit(0x80,atrans,100); /* transmit 100 hex 66 & get result */
   printf("\nRESULT OF TRANSMIT=%d\n",result);

   /*********************************************/
   /* WAIT FOR SOME DATA FROM THE OTHER DEVICE */
   /*********************************************/
   printf("\nWaiting to receive\n");

   do {
       receive(rxbuf); /* if rxlen=0 then no data was received */
   } while (rxlen==0);

   for(i=0;i<rxlen;i++)
     printf("%d)=%x   ",i,rxbuf[i]); /* print results of data transfer */

   printf("\nPress RETURN to exit the program\n");
   getchar();

}
```

BSCB.C

This sample program demonstrates transmitting and receiving on Port B over a V.24 interface using the libbsc.a library.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
struct   BSC_CTRL;

main()
{
char        ch,crc;
struct      BSC_CTRL    encode;
extern unsigned int rxlen;   /* global variable for receive data length */
unsigned char atrans[100],rxbuf[128]; /* transmit & revieve arrays */
int result,i;

encode.eot  =           55;
encode.syn  =           50;
encode.dle  =           16;
encode.stx  =           02;
encode.etx  =           03;
encode.soh  =           01;
encode.etb  =           38;
encode.itb  =           31;
encode.enq  =           45;

   /*******************************/
   /* SET THE ACTIVE PORT TO "B" */
   /*******************************/
   if( (result=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }

   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DTE,&encode.eot,96001,CCITT,ASCII_EVEN_DATA)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }

   for (i=0;i<=99;i++)
     atrans[i]=0x22; /* store hex 22 into transmit array */
```

```
   /*********************************************/
   /* WAIT UNITL THE SYSTEM IS READY TO TRANSMIT */
   /*********************************************/
   while((tready())!=0){            /* loop while transmitter not ready */
      puts(CLEARS);  /* clear screen */
      printf("PRESS ANY KEY TO ABORT \n");
      printf("TREADY STATUS =%d\n",tready());
      if((ch = getch (_stdvt)) != -1)       /* check for key pressed */
         exit(0);
   }

   /*********************************************/
   /* WAIT FOR SOME DATA FROM THE OTHER DEVICE */
   /*********************************************/
   printf("\nWaiting to receive\n");

   do {
      receive(rxbuf); /* if rxlen=0 then no data was received */
   } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%d)=%x   ",i,rxbuf[i]); /* print results of data transfer */

   /***********************/
   /* SEND OUT SOME DATA */
   /***********************/
   printf("\n hit RETURN to send the data out of port 'B'\n");
   getchar();

   result=transmit(0x80,atrans,100); /* transmit 100 hex 22 & get result */
   printf("\nRESULT OF TRANSMIT=%d\n",result);

   printf("\nPress RETURN to exit the program\n");
   getchar();

}
```

BSCAB.C

This sample program demonstrates transmitting and receiving on a Dual Port machine over a V.24 interface using the libbsc.a library.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
struct   BSC_CTRL;

/*
 * SETUP BOTH OF THE PORTS
 */
init_ports()
{
struct      BSC_CTRL    encode;
int result;

   encode.eot          =          55;
   encode.syn          =          50;
   encode.dle          =          16;
   encode.stx          =          02;
   encode.etx          =          03;
   encode.soh          =          01;
   encode.etb          =          38;
   encode.itb          =          31;
   encode.enq          =          45;

   /*******************************/
   /* SET THE ACTIVE PORT TO "A" */
   /*******************************/
   if( (result=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }

   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DCE,&encode.eot,96001,CCITT,ASCII_EVEN_DATA)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }

   /*******************************/
   /* SET THE ACTIVE PORT TO "B" */
   /*******************************/
   if( (result=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }
```

```
   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DTE,&encode.eot,9600l,CCITT,ASCII_EVEN_DATA)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }
}

/*
 * RECEIVE DATA
 */
get_data(port)
unsigned char port;
{
extern unsigned int rxlen;   /* global variable for receive data length */
int i;
unsigned char rxbuf[128];

   setport(port); /* determine which port to use */
   printf("\nWaiting to receive\n");

   do {
      receive(rxbuf); /* if rxlen=0 then no data was received */
   } while (rxlen==0);

   for(i=0;i<rxlen;i++) {
     if( i % 15 == 0 ) printf("\n");
     printf("%x   ",rxbuf[i]); /* print results of data transfer */
   }
}

/*
 * TRANSMIT DATA OUT
 */
send_data(port,buf,buf_size)
unsigned char port,*buf;
int buf_size;
{
   setport(port); /* determine which port to use */
   printf("\nRESULT OF TRANSMIT=%d\n",transmit(0x80,buf,buf_size));
}
```

```
main()
{
char        ch;
unsigned char atrans[100]; /* transmit array */
int i;

   /*************************/
   /* INITIALIZE BOTH PORTS */
   /*************************/
   init_ports();

   setport(PORTA);

   /*********************************************/
   /* WAIT UNITL THE SYSTEM IS READY TO TRANSMIT */
   /*********************************************/
   while((tready())!=0){              /* loop while transmitter not ready */
      puts(CLEARS);  /* clear screen */
      printf("PRESS ANY KEY TO ABORT \n");
      printf("TREADY STATUS =%d\n",tready());
      if((ch = getch (_stdvt)) != -1)       /* check for key pressed */
         exit(0);
   }

   /***********************/
   /* SEND OUT SOME DATA */
   /***********************/
   for (i=0;i<=99;i++)
     atrans[i]=0x66; /* store hex 66 into the transmit array */

   printf("\nPress RETURN to send data from port 'A' to port 'B'\n");
   getchar();

   send_data(PORTA,atrans,i);

   /***********************/
   /* WAIT FOR SOME DATA */
   /***********************/
   get_data(PORTB);

   /***********************/
   /* SEND OUT SOME DATA */
   /***********************/
   for (i=0;i<=99;i++)
     atrans[i]=0x22; /* store hex 22 into the transmit array */

   printf("\nPress RETURN to send data from port 'B' to port 'A'\n");
   getchar();

   send_data(PORTB,atrans,i);
```

```
    /**********************/
    /* WAIT FOR SOME DATA */
    /**********************/
    get_data(PORTA);

    printf("\nPress RETURN to exit the program\n");
    getchar();

}
```

B.8 PRIMARY RATE INTERFACE LIBRARY

Introduction

The Primary Rate Library enables you to use the Chameleon 32 Primary Rate Interface hardware with the C environment. It is valid for both the ANSI and CEPT Primary Rate Interfaces. The library is called **libpri.a** and is in the **/lib** directory.

In general, your application should include PRI library routines which enable you to set up and modify the Primary Rate Interface from within your application. However, when you start a C application from the Applications Selection menu, you must set up the PRI before you can access the Applications Selection menu to start the application. This can be accomplished in two ways:

- You can access the PRI Setup menu before starting the C application. This is done by pressing ***F7 Physicl*** in the main configuration menu.
- You can save the PRI Setup menu as part of the DEFAULT configuration file. The PRI is then automatically set up when the Chameleon is booted or reset.

When the C application is then started, it uses the PRI Setup menu parameters. Your application can then use other routines to modify the setup or use PRI functions, as required by your test. If the application contains a PRI library setup routine, the application setup overrides the menu setup.

When starting an application from the C Shell, if your application contains a valid PRI setup routine, it is not necessary to access the PRI Setup menu prior to starting your application.

Chapter 3.1 describes he procedure for starting applications from the C Shell and from the Applications Selection menu

Note

You **cannot** use the C PRI library and the PRIMARY application at the same time. (The PRIMARY application enables the user to monitor and modify the PRI at run time.) The PRIMARY application is useful for Monitoring applications and non-C simulation applications, but results in an error message when used simultaneously with the C PRI library. The PRIMARY application is described in the *Chameleon Protocol Interpretation Manual, Chapter 11.*

The functions are described on the following pages:

Also refer to Appendix B.1 for a description of common library functions and error codes.

Several sample programs using the PRI library are provided at the end of this section.

Pri_version

Declaration char *Pri_version()

Description This function returns a pointer to a string which indicates the date of the Primary Rate Interface library version.

SetPrimary

Declaration

```
int  SetPrimary(cmdblock, resblock)
int cmdblock [14];
int resblock [14];
```

Description

This function has two parameters that are integer arrays. The size of the arrays must be at least 14.

The *cmdblock* (command block) parameter contains the input values needed. The first item in *cmdblock* (index 0) is an integer specifying the requested function. The remaining entries in the command block are the arguments required for the function. Choices for selections which are not available in a given configuration are ignored (eg., Data Rate in CEPT). For details of the Primary Rate Interface, refer to the Chameleon 32 Monitoring Reference Manual, Chapter 11.

The *resblock* (result block) parameter is an array containing the results, if any, of the operation requested in the *cmdblock*.

The first entry in the result block (index 0) indicates whether or not the command in *cmdblock* was completed successfully. Zero indicates a successful completion. Other values that are returned are given in the next result block elements.

When the requested operation cannot be done because of an existing selection, the currently selected values are returned.

This is illustrated in the figure below:

COMMAND
Argument 1
Argument 2
etc.

RESULT
Result 1
Result 2
etc.

Figure B.8-1: SetPrimary Function

The commands listed below are available as the first item in *cmdblock*. Note that the commands are different for Primary Rate Interface boards A and B. Currently only board A is available.

Board A	Board B	Command
1	101	Setup
2	102	Resynchronize
3	103	Reset
4	104	Channel functions
5	105	Signal functions
6	106	Get status
7	107	Change status
8	108	Reserved
9	-	Enable trace

Each command is described on the following pages.

Channel Number/ Time Slot

When selecting a channel number (ANSI) or time slot (CEPT) for a command, the following convention applies:

- Enter the channel number itself to select a channel or time slot for line 1 (T1 or R1). For example, for ANSI, the argument *24* would select channel number 24 on line 1.
- Enter the channel number + 100 to select a channel or time slot for line 2 (T2 or R2) For example, for ANSI, the argument *124* would select channel number 24 on line 2.

Error Codes resblock[0]

The error codes for *resblock[0]* are the same for all Primary Rate Interface Library commands, and are listed below.

If a command is successful, the current configuration is returned as resblock[1] - resblock[11]. The result reflects the structure of the cmdblock[1] - cmdblock[11] of the Setup command.

If a command fails because of an invalid parameter, the invalid parameter is returned in cmdblock[1]. If a command fails for something other than an invalid parameter, the current configuration is returned.

CODE	MEANING	NOTES
0	Successful	resblock[1] - resblock[11] return the current configuration
1	Primary Rate Interface board not installed	
2	Setup already done	To change the setup, first use the Reset command to reinitialize, and then use the Setup command
3	Invalid channel number/time slot	resblock[1] returns the invalid channel or time slot
4	Selection already in use	resblock[1] - resblock[11] return the current configuration
5	Channel already assigned	resblock[1] - resblock[11] return the current configuration
10	Command not implemented	

Figure B.8-2: resblock[0] Error Codes

Setup
cmdblock[0] = 1
cmdblock[0] = 101 This command is used to initialize all parameters. The **Reset** command (cmdblock[0] = 3/103) must be used before using **Setup** again.

cmdblock[1] *mode*
1 Monitor
2 Simulate

cmdblock[2] *framing*
1 D4
2 ESF
3 SL96
4 CEPT

cmdblock[3] *idle data*
8 bit value

cmdblock[4] *idle signal*
2 or 4 bit value

cmdblock[5] *DS0x receive*
Channel/time slot

cmdblock[6]* *Codec receive*
Channel/time slot

cmdblock[7]* *DS0y receiver/transmitter*
Channel/time slot

cmdblock[8]* *Codec transmitter*
Channel/time slot (This parameter is irrelevant for Monitor mode and is ignored.)

cmdblock[9]* *Milliwatt transmitter*
Channel/time slot (This parameter is irrelevant for Monitor mode and is ignored.)

cmdblock[10]* *status line 1*
One byte (See Figure B.8-3 on the next page)

cmdblock[11] *status line 2*
One byte (See Figure B.8-3 on the next page)

* These functions are available on Line 1 only.

NOTE: When the system is in Monitor Mode, no Transmit facilities are available

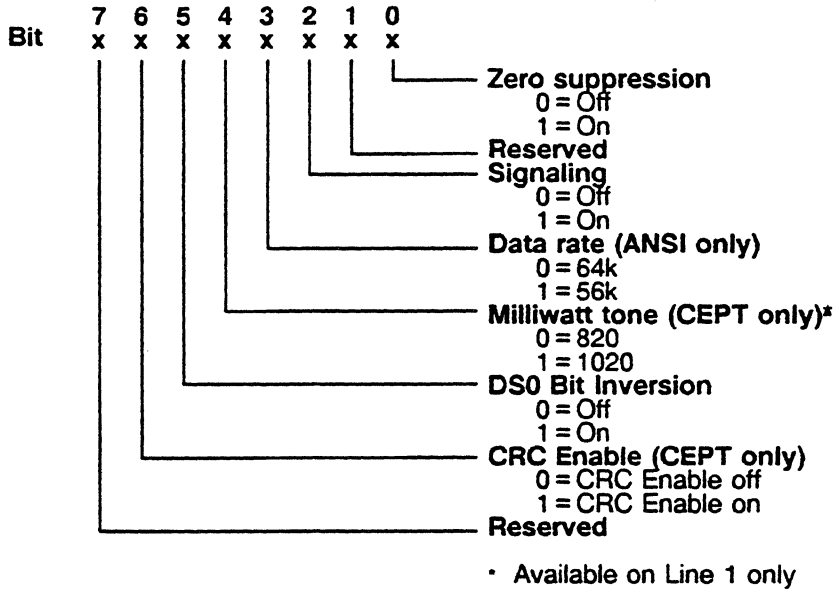


Figure B.8-3: Status Byte Interpretation

If you use the **Setup** command and the setup has already been done, you will receive the error resblock[0] = 2 (Setup already done). In this case, use the Reset command (cmdblock[0] = 3) to reinitialize, and then use the Setup command with the new configuration.

Resynchronize
cmdblock[0] = 2
cmdblock[0] = 102 Argument None

This command resets the Primary Rate Interface and reconfigures with the current setup.

Reset
cmdblock[0] = 3
cmdblock[0] = 103 Argument None

This command resets the Primary Rate Interface and puts the lines in repeater mode. The interface can then be reconfigured using the **Setup** command.

Channel Functions
cmdblock[0] = 4
cmdblock[0] = 104

cmdblock[1] *mode*	**0**	Retain current setup.
	1	Override current setup. (See below)
cmdblock[2] *selection*	**1**	DS0x receive
	2	Codec receive
	3	DS0y transmit
	4	DS0y receive
	5	Codec transmit
	6	Milliwatt transmit
	7	Reset transmit channel
	8	Reset receive channel
	9	Idle data
	10	Idle signal

cmdblock[3] *channel number* (if cmdblock[2] = 1 - 8 *)*

1 - 24	D4/ESF line 1
1 - 31	CEPT line 1

cmdblock[3] *Idle bits* (if cmdblock[2] = 9 or 10*)*

8, 4, 2 bits

If there is an existing selection which conflicts with the new request, cmdblock[1] (*mode*) determines whether the command is executed. For example, if the request is for Codec transmitter on channel 2, but channel 2 is already being used by Milliwatt transmitter, the following occurs:

- If mode 1 is selected, the old selection is reset. For this example, the Milliwatt transmitter is reset, and channel 2 assigned to Codec transmitter.
- If mode 0 is selected, the old selection is retained and the command request is not executed. For this example, Milliwatt transmitter is retained on channel 2 and the error code resblock[0] = 4 (Selection already in use) and the current configuration are returned.

In general, if you wish to execute channel functions regardless of their previous selection or non-selection, use mode 1 (cmdblock[1] = 1).

cmdblock[0] = 5
cmdblock[0] = 105
Signal Functions cmdblock[1] *selection*

1 Resynchronize
This selection resets and resynchronizes the line.

2 Normal
This selection puts the line back to normal mode of operation after selecting any other choice of this command.

3 Repeater
This selection loops the receive line back to the transmit line.

4 Alarm
This selection transmits the Yellow alarm signal in D4/ESF mode, or the Remote alarm in CEPT.

5 Transparency (for line 1)
This selection causes all channels from the received line to be looped back as channels in the transmit line. For channels with transmit functions (codec, Milliwatt, Data Y), the incoming traffic is replaced by the signal from the selected transmitter.

cmdblock[2] *line number*

1 Line 1
2 Line 2

Get Status
cmdblock[0] = 6
cmdblock[0] = 106 Argument None

This command returns the status of the line.

resblock[1] *0* Synchronized
1 Loss of signal
2 Yellow Alarm
4 Loss of framing

Change Status
cmdblock[0] = 7
cmdblock[0] = 107

cmdblock[1] *line 1 or 2*

cmdblock[2] *status* xxxxxxxx (see below)

This command changes the operating modes of different selections, for example, zero suppression and DS0 bit inversion. The choices are coded in bits, as shown in Figure B.8-4.

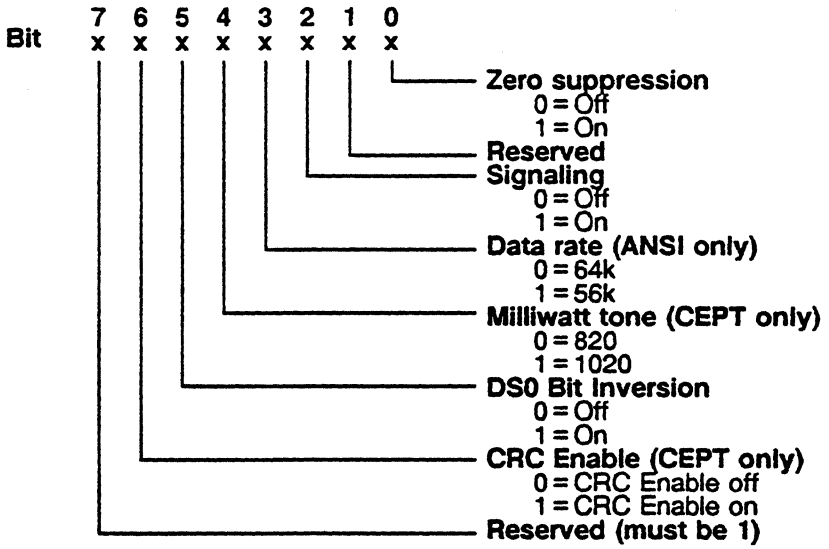
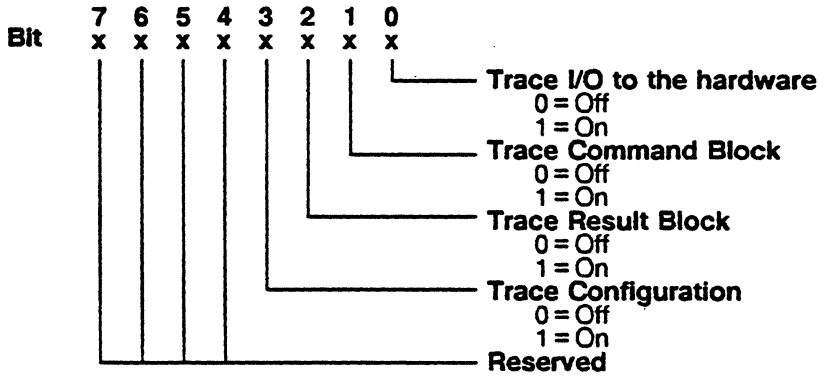


Figure B.8-4: Status Byte Interpretation

Enable trace
cmdblock[0] = 9 This command is useful for debugging your programs.

cmdblock[1] *8 trace bits (see below)*



Sample Programs

There are four sample PRI programs on the C Sample Program Disk. They are:

- pria.c — Sets up an ANSI PRI on Port A.
- prib.c — Sets up an ANSI PRI on Port B.
- cepta.c — Sets up a CEPT PRI on Port A.
- ceptb.c — Sets up a CEPT PRI on Port B.

PRIA.C

This sample program demonstrates transmitting and receiving over a PRI on Port A using the libbop.a and the libpri.a libraries.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
#include <mtosux.h>

            /* initalize cmd array for set primary function - setup as follows:
1           mode          =         simulate
2           framing       =         D4
3           idle data     =         hex 55
4           idle signal =           11 binary
5           DS0x rec      =         channel 15
6           codec rec     =         channel 1
7           DS0y xmit     =         channel 15
8           codec xmit    =         channel 1
9           millawatt     =         channel 4
10 & 11     status #1&2 =           hex 01
            zero suppression on
            signaling off
            data rate 64000
            millawatt tone 820hz
            DS0 bit inversion off
            CRC enable off
            line 2 off

                 0  1  2  3     4  5   6  7   8  9  10    11    12 13 */

   int cmd []= {1, 2, 1, 0x55, 3, 15, 1, 15, 1, 4, 0x01, 0x01, 0, 0 };
   int rsp []= {0, 0, 0, 0x00, 0, 00, 0, 00, 0, 0, 0x00, 0x00, 0, 0 };

exit_program()
{
  printf("\n\nPress RETURN to end this program\n");
  getchar();
  exit(0);
}
```

```
main()
{
extern unsigned int rxlen;   /* global variable for receive data length */
int i,result;
unsigned char atrans[30],rxbuf[128],c; /* transmit & revieve arrays */

  /*********************************************************/
  /* initialize the ISDN PRIMARY RATE (layer 1) interface */
  /*********************************************************/
  SetPrimary (cmd,rsp);
  if(rsp[0] != 0) {
    printf("\nERROR: RESULT SETPRIMARY  = %d\n", rsp[0]);
    exit_program();
  }

  pause(125+MS); /* wait for the primary (layer 1) to initialize */

  /*************************************************/
  /* initialize the front end processor for port A */
  /*************************************************/
  if ( (result=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit_program();
  }
  if( (result=initp1(ISDN,NRZ,64000l,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit_program();
  }

  /***************************************/
  /* store data into the transmit array */
  /***************************************/
  for(i=0;i<=25;i++)
    atrans[i] = 65 + i; /* fill transmit buffer with upper case letters */

  /*********************************/
  /* transmit data & get result */
  /*********************************/
  printf("Press Return to send data from port A\n");
  getchar();

  if( (result=transmit(GOOD_CRC,atrans,i)) != 0 ) {
    printf("\nERROR: result=%d\n",result);
    exit_program();
  }

  /*********************************************************/
  /* receive the data from the the other device or port */
  /*********************************************************/
  printf("\nWaiting to receive data, (press 'q' to quit)\n");
```

```
  do {
     receive(rxbuf);        /* if rxlen=0 then no data was received */
     if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0);
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%c   ",rxbuf[i]); /* print results of data transfer */

  exit_program();
}
```

PRIB.C

This sample program demonstrates transmitting and receiving over a PRI on Port B using the libbop.a and the libpri.a libraries.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
#include <mtosux.h>

            /* initalize cmd array for set primary function - setup as follows:
1           mode            =       simulate
2           framing         =       D4
3           idle data       =       hex 55
4           idle signal =           11 binary
5           DS0x rec        =       channel 15
6           codec rec       =       channel 1
7           DS0y xmit       =       channel 15
8           codec xmit  =           channel 1
9           millawatt       =       channel 4
10 & 11     status #1&2 =           hex 01
            zero suppression on
            signaling off
            data rate 64000
            millawatt tone 820hz
            DS0 bit inversion off
            CRC enable off
            line 2 off

                    0  1  2  3   4  5   6  7   8  9  10    11    12 13 */

   int cmd []= {101, 2, 1, 0x55, 3, 15, 1, 15, 1, 4, 0x01, 0x01, 0, 0 };
   int rsp []= {0, 0, 0, 0x00, 0, 00, 0, 00, 0, 0, 0x00, 0x00, 0, 0 };

exit_program()
{
  printf("\n\nPress RETURN to terminate this program\n");
  getchar();
  exit(0);
}
```

```
main()
{
extern unsigned int rxlen;   /* global variable for receive data length */
int i,result;
unsigned char atrans[30],rxbuf[128],c; /* transmit & revieve arrays */

  /*********************************************************/
  /* initialize the ISDN PRIMARY RATE (layer 1) interface */
  /*********************************************************/
  SetPrimary (cmd,rsp);
  if(rsp[0] != 0) {
    printf("\nERROR: RESULT SETPRIMARY  = %d\n", rsp[0]);
    exit_program();
  }

  pause(125+MS); /* wait for the primary (layer 1) to initialize */

  /*****************************************************/
  /* initialize the front end processor for port B */
  /*****************************************************/
  if ( (result=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit_program();
  }
  if( (result=initp1(ISDN,NRZ,64000l,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit_program();
  }

  /*********************************************************/
  /* receive the data from the the other device or port */
  /*********************************************************/
  printf("\nWaiting to receive data, (press 'q' to quit)\n");

  do {
     receive(rxbuf);       /* if rxlen=0 then no data was received */
     if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0);
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%c   ",rxbuf[i]); /* print results of data transfer */

  /***************************************/
  /* store data into the transmit array */
  /***************************************/
  for(i=0;i<=25;i++)
    atrans[i] = 97 + i; /* fill transmit buffer with lower case letters */
```

```
  /*******************************/
  /* transmit data & get result */
  /*******************************/
  printf("\n\nPress Return to send data from port B\n");
  getchar();

  if( (result=transmit(GOOD_CRC,atrans,i)) != 0 )
    printf("\nERROR: result=%d\n",result);

  exit_program();
}
```

CEPT.C

This sample program demonstrates transmitting and receiving over a CEPT PRI using the libbop.a and the libpri.a libraries.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
#include <mtosux.h>

            /* initialize cmd array for set primary function - setup as follows:
1           mode          =         simulate
2           framing       =         CEPT
3           idle data     =         hex 55
4           idle signal =           11 binary
5            DS0x rec     =         channel 15
6            codec rec    =         channel 1
7           DS0y xmit     =         channel 15
8           codec xmit  =           channel 1
9           millawatt     =         channel 4
10 & 11     status #1&2 =           hex 01
            zero suppression on
            signaling off
            data rate 64000
            millawatt tone 820hz
            DS0 bit inversion off
            CRC enable off
            line 2 off

                  0  1  2  3     4  5   6  7   8  9  10    11    12 13 */

   int cmd []= {1, 2, 3, 0x55, 3, 15, 1, 15, 1, 4, 0x01, 0x01, 0, 0 };
   int rsp []= {0, 0, 0, 0x00, 0, 00, 0, 00, 0, 0, 0x00, 0x00, 0, 0 };

exit_program()
{
  printf("\n\nPress RETURN to end this program\n");
  getchar();
  exit(0);
}
```

```
main()
{
  extern unsigned int rxlen;   /* global variable for receive data length */
  int i,result;
  unsigned char atrans[30],rxbuf[128],c; /* transmit & revieve arrays */

  /*********************************************************/
  /* initialize the ISDN PRIMARY RATE (layer 1) interface */
  /*********************************************************/
  SetPrimary (cmd,rsp);
  if(rsp[0] != 0) {
    printf("\nERROR: RESULT SETPRIMARY  = %d\n", rsp[0]);
    exit_program();
  }

  pause(125+MS); /* wait for the cept to initialize */

  /*************************************************/
  /* initialize the front end processor for port A */
  /*************************************************/
  if ( (result=setport(PORTA)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit_program();
  }
  if( (result=initp1(ISDN,NRZ,64000l,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit_program();
  }

  /***************************************/
  /* store data into the transmit array */
  /***************************************/
  for(i=0;i<=25;i++)
    atrans[i] = 65 + i; /* fill transmit buffer with upper case letters */

  /*********************************/
  /* transmit data & get result */
  /*********************************/
  printf("\nPress Return to send data from port A\n");
  getchar();

  if( (result=transmit(GOOD_CRC,atrans,i)) != 0 ) {
    printf("\nERROR: result=%d\n",result);
    exit_program();
  }
```

```
  /*********************************************************/
  /* receive the data from the the other device or port */
  /*********************************************************/
  printf("\nWaiting to receive data, (press 'q' to quit)\n");

  do {
     receive(rxbuf);       /* if rxlen=0 then no data was received */
     if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0);
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%c   ",rxbuf[i]); /* print results of data transfer */

  exit_program();
}
```

CEPTB.C

This sample program demonstrates transmitting and receiving over a CEPT PRI on Port B using the libbop.a and the libpri.a libraries.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
#include <mtosux.h>

            /* initialize cmd array for set primary function - setup as follows:
1           mode          =       simulate
2           framing       =       CEPT
3           idle data     =       hex 55
4           idle signal   =       11 binary
5           DS0x rec      =       channel 15
6           codec rec     =       channel 1
7           DS0y xmit     =       channel 15
8           codec xmit    =       channel 1
9           millawatt     =       channel 4
10 & 11     status #1&2   =       hex 01
            zero suppression on
            signaling off
            data rate 64000
            millawatt tone 820hz
            DS0 bit inversion off
            CRC enable off
            line 2 off

                  0  1  2  3   4  5   6  7   8  9  10    11    12 13 */

   int cmd []= {101, 2, 3, 0x55, 3, 15, 1, 15, 1, 4, 0x01, 0x01, 0, 0 };
   int rsp []= {0, 0, 0, 0x00, 0, 00, 0, 00, 0, 0, 0x00, 0x00, 0, 0 };

exit_program()
{
  printf("\n\nPress RETURN to terminate this program\n");
  getchar();
  exit(0);
}
```

```
main()
{
extern unsigned int rxlen;   /* global variable for receive data length */
int i,result;
unsigned char atrans[30],rxbuf[128],c; /* transmit & revieve arrays */

  /*********************************************************/
  /* initialize the ISDN PRIMARY RATE (layer 1) interface */
  /*********************************************************/
  SetPrimary (cmd,rsp);
  if(rsp[0] != 0) {
    printf("\nERROR: RESULT SETPRIMARY  = %d\n", rsp[0]);
    exit_program();
  }

  pause(125+MS); /* wait for the cept to initialize */

  /*****************************************************/
  /* initialize the front end processor for port B */
  /*****************************************************/
  if ( (result=setport(PORTB)) != 0 ) {
    printf("ERROR: setport = %d\n", result);
    exit_program();
  }
  if( (result=initp1(ISDN,NRZ,640001,FILL7E)) != 0 ) {
    printf("ERROR: initp1 = %d\n", result);
    exit_program();
  }

  /*********************************************************/
  /* receive the data from the the other device or port */
  /*********************************************************/
  printf("\nWaiting to receive data, (press 'q' to quit)\n");

  do {
     receive(rxbuf);       /* if rxlen=0 then no data was received */
     if((c=getch(_stdvt)) == 'Q' || c == 'q') exit(0);
  } while (rxlen==0);

  for(i=0;i<rxlen;i++)
    printf("%c   ",rxbuf[i]); /* print results of data transfer */

  /*****************************************/
  /* store data into the transmit array */
  /*****************************************/
  for(i=0;i<=25;i++)
    atrans[i] = 97 + i; /* fill transmit buffer with lower case letters */
```

```
  /*******************************/
  /* transmit data & get result */
  /*******************************/
  printf("\n\nPress Return to send data from port B\n");
  getchar();

  if( (result=transmit(GOOD_CRC,atrans,i)) != 0 )
    printf("\nERROR: result=%d\n",result);

  exit_program();
}
```

B.9 ASYNC C LIBRARY

Introduction

The Async C Library (libasc.a) is valid for asynchronous simulation. It is in the \lib directory.

The functions are described on the following pages:

Also refer to Appendix B.1 for a description of the common library functions and error codes.

INITP1

Description This function initializes the Front End Processor and loads its simulation software.

Declaration

```
int   initp1(type, encode)
int   type;
struct ASC_CTRL   *encode;
```

Range

type

0	DCE
1	DTE

encode This is a structure that defines the control characters for the Async protocol. It it defined as follows:

```
struct ASC_CTRL
  {
  int bitrate;
  int parity;
  int stop;
  int data;
  int duplex;
  int block;
  unsigned char eob;
  };
```

bitrate

1	50	7	1200
2	75	8	2400
3	110	9	4800
4	150	10	9600
5	300	11	19200
6	600		

parity

0	None
1	Odd
2	Even

stop

0	1	Stop bit
1	1.5	Stop bits
2	2	Stop bits

data

5	5 Data bits
6	6 Data bits
7	7 Data bits
8	8 Data bits

duplex 0 Full Duplex
1 Half Duplex

block 0 Block mode
1 Character mode

eob (End of block character)
Range: 0 - 0xFF

Returns

0 Successful
-1 One or more parameter errors
-2 Front End Processor program could not be loaded
-3 Port is busy

See also the global error codes on page B.1-1.

RECEIVE

Declaration

```
int   receive(frame)
char  *frame;
```

Description

This function receives a block or a character from P1, and places the frame starting at the address pointed to by the passed variable *frame*. The external global variable *rxlen* is set to the length of the received frame. If *rxlen* = 0, then no data was received.

The *block* parameter of the *initp1* function enables you to select either block or character mode. In block mode, a block of data is returned only when the end of block character is received. The end of block character is defined with the *eob* parameter in the *initp1* function. If the end of block character is not received, *rxlen* = zero.

Returns

0	Good BCC or no frame waiting
1	Bad BCC
2	*initp1* not performed
3	Overflow

See also the global error codes on page B.1-1.

Example

```
...
do {
    receive(frame);
} while (rxlen == 0);
...
```

TBREAK

Declaration int tbreak()

Description This function transmits a break sequence.

Returns See the global error codes on page B.1-1.

TRANSMIT

Declaration

```
int   transmit(frame, length)
char  *frame;
int   length;
```

Description

This function transmits the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer **frame*.

Returns

0	Successful
1	Front End Processor busy (transmitting previous frame)
2	**initp1** not performed
3	Parameter error
4	Buffer overflow

See the global error codes on page B.1-1.

TREADY

Declaration int tready()

Description This function returns the status of the Front End Processor transmitter.

Returns

0 Transmitter is ready for next frame
1 Transmitter is busy (sending previous frame)
2 **initp1** not performed
3 Overflow

See the global error codes on page B.1-1.

SAMPLE PROGRAMS

There are three sample Async programs on the C Sample Program Disk. They are:

- ascach.c
- ascbch.c
- ascasch.c

ASCACH.C

This sample program demonstrates transmitting and receiving in character mode on Port A over a V.24 interface using the libasc.a library.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
struct   ASC_CTRL;

main()
{
char        ch;
struct      ASC_CTRL    encode;
extern unsigned int rxlen;   /* global variable for receive data length */
unsigned char atrans[30],rxbuf[2]; /* transmit & receive arrays */
int result,i;

  encode.bitrate        =          B9600;
  encode.parity         =          EVEN;
  encode.stop           =          STOP1;
  encode.data           =          DATABIT7;
  encode.duplex         =          HALF;
  encode.block          =          CHARMODE;
  encode.eob            =          0x40;       /* only used in block mode */

   rxbuf[1] = '\0'; /* initialize position 1 */

   /*******************************/
   /* SET THE ACTIVE PORT TO "A" */
   /*******************************/
   if( (result=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }

   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DCE,&encode)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }
```

```
   /*************************************************/
   /* WAIT UNITL THE SYSTEM IS READY TO TRANSMIT */
   /*************************************************/
   while( tready()!=0 ){            /* loop while transmitter not ready */
      puts(CLEARS);                  /* clear screen */
      printf("TREADY STATUS =%d\n",tready());
      printf("PRESS 'q' TO ABORT \n");
      if((ch = getch (_stdvt)) == 'q')       /* fail safe */
         exit(0);
            }

   /***********************/
   /* SEND OUT SOME DATA */
   /***********************/
   for (i=0;i<=25;i++)
     atrans[i]= 65 + i; /* store upper case letters into the transmit array */
   atrans[i]=0x40;      /* sentinel with an @ character */

   printf("\n hit RETURN to send the data out of port 'A'\n");
   getchar();

   for(i=0;i<=26;i++) {
     while(tready() != 0 ){         /* wait for the previous frame to send */
       if((ch = getch (_stdvt)) == 'q')       /* fail safe */
         exit(0);
     }
     printf("RESULT OF TRANSMIT %d)=%d\n",i,transmit(atrans+i,1));
     }

   /*************************************************/
   /* WAIT FOR SOME DATA FROM THE OTHER DEVICE */
   /*************************************************/
   printf("\nWaiting to receive\n");

   do {
      do {
         receive(rxbuf);           /* if rxlen=0 then no data was received */
         if((ch = getch (_stdvt)) == 'q')       /* fail safe */
           exit(0);
      } while (rxlen==0);
               printf("%s   ",rxbuf);    /* print results of data transfer */
   } while (rxbuf[0] != '#');   /* transfer complete ? */

   printf("\nPress RETURN to exit the program\n");
   getchar();

}
```

ASCBCH.C

This sample program demonstrates transmitting and receiving in character mode on Port B over a V.24 interface using the libasc.a library.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
struct   ASC_CTRL;

main()
{
char        ch;
struct      ASC_CTRL    encode;
extern unsigned int rxlen;   /* global variable for receive data length */
unsigned char atrans[30],rxbuf[2]; /* transmit & receive arrays */
int result,i;

   encode.bitrate       =          B9600;
   encode.parity        =          EVEN;
   encode.stop          =          STOP1;
   encode.data          =          DATABIT7;
   encode.duplex        =          HALF;
   encode.block         =          CHARMODE;
   encode.eob           =          0x23;          /* only used in block mode */

   rxbuf[1] = '\0'; /* initialize position 1 */

   /*******************************/
   /* SET THE ACTIVE PORT TO "B" */
   /*******************************/
   if( (result=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }

   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DTE,&encode)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }
```

```
   /*************************************************/
   /* WAIT UNITL THE SYSTEM IS READY TO TRANSMIT */
   /*************************************************/
   while( tready()!=0 ){              /* loop while transmitter not ready */
      puts(CLEARS);                    /* clear the screen */
      printf("PRESS 'q' TO ABORT \n");
      printf("TREADY STATUS =%d\n",tready());
      if((ch = getch (_stdvt)) == 'q')
         exit(0);
   }

   /*************************************************/
   /* WAIT FOR SOME DATA FROM THE OTHER DEVICE */
   /*************************************************/
   printf("\nWaiting to receive\n");

   do {
      do {
         receive(rxbuf);            /* if rxlen=0 then no data was received */
            if((ch = getch (_stdvt)) == 'q')       /* fail safe */
              exit(0);
      } while (rxlen==0);
      printf("%s   ",rxbuf);     /* print results of data transfer */
   } while (rxbuf[0] != '@');       /* transfer complete ? */

   /***********************/
   /* SEND OUT SOME DATA */
   /***********************/
   for (i=0;i<=25;i++)
     atrans[i]= 97 + i; /* store lower case letters into the transmit array */
   atrans[i]=0x23; /* sentinel with a # character */

   printf("\n hit RETURN to send the data out of port 'B'\n");
   getchar();

   for(i=0;i<=26;i++) {
     while(tready() != 0 ){          /* wait for the previous frame to send */
       if((ch = getch (_stdvt)) == 'q')       /* fail safe */
            exit(0);
     }
     printf("RESULT OF TRANSMIT %d)=%d\n",i,transmit(atrans+i,1));
  }

   printf("\nPress RETURN to exit the program\n");
   getchar();

}
```

ASCABCH.C

This sample program demonstrates transmitting and receiving in character mode on a Dual Port machine over a V.24 interface using the libasc.a library.

```
#include <stdio.h>
#include <cham.h>
#include <video.h>
struct   ASC_CTRL;

init_ports()
{
struct      ASC_CTRL    encode;
int result;

   encode.bitrate       =          B9600;
   encode.parity        =          EVEN;
   encode.stop          =          STOP1;
   encode.data          =          DATABIT7;
   encode.duplex        =          HALF;
   encode.block         =          CHARMODE;
   encode.eob           =          0x40;       /* only used in block mode */

   /*******************************/
   /* SET THE ACTIVE PORT TO "A" */
   /*******************************/
   if( (result=setport(PORTA)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }

   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DCE,&encode)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }

   /*******************************/
   /* SET THE ACTIVE PORT TO "B" */
   /*******************************/
   if( (result=setport(PORTB)) != 0 ) {
     printf("ERROR: setport = %d\n", result);
     exit(0);
   }
```

```
   /*************************************/
   /* INITIALIZE THE FRONT END PROCESSOR */
   /*************************************/
   if( (result=initp1(DTE,&encode)) ) {
     printf("ERROR: initp1 = %d\n", result);
     exit(0);
   }
}

send_data(port,buf,buf_size)
unsigned char port, *buf;
int buf_size;
{
int i,result;
unsigned char ch;

   setport(port);                    /* the port sending the data */
   for(i=0;i<=buf_size;i++) {
     while(tready() != 0 ){          /* wait for the previous frame to send */
       if((ch = getch (_stdvt)) == 'q')       /* fail safe */
           exit(0);
     }
     if( (result=transmit(buf+i,1)) != 0 )
       printf("ERROR: RESULT OF TRANSMIT %d)=%d\n",i,result);
   }
}

get_data(port)
unsigned char port;
{
extern unsigned int rxlen;   /* global variable for receive data length */
unsigned char ch, rxbuf[2]; /* receive array */

   rxbuf[1] = '\0'; /* initialize position 1 */

   printf("\nWaiting to receive\n");

   setport(port);                   /* the port receiving the data */

   do {
      do {
         receive(rxbuf);           /* if rxlen=0 then no data was received */
           if((ch = getch (_stdvt)) == 'q')       /* fail safe */
             exit(0);
      } while (rxlen==0);
      printf("%s   ",rxbuf);    /* print results of data transfer */
   } while (rxbuf[0] != '@');   /* transfer complete ? */
}
```

```
main()
{
char        ch;
unsigned char atrans[30]; /* transmit array */
int i;

   /*************************/
   /* INITIALIZE BOTH PORTS */
   /*************************/
   init_ports();

   /***********************************************/
   /* WAIT UNITL THE PORT A IS READY TO TRANSMIT */
   /***********************************************/
   setport(PORTA);
   while( tready()!=0 ){              /* loop while transmitter not ready */
      puts(CLEARS);                    /* clear screen */
      printf("TREADY STATUS =%d\n",tready());
      printf("PRESS 'q' TO ABORT \n");
      if((ch = getch (_stdvt)) == 'q')       /* fail safe */
        exit(0);
   }

   /*********************************/
   /* SEND OUT SOME DATA OUT PORT A */
   /*********************************/
   for (i=0;i<=25;i++)
     atrans[i]= 65 + i; /* store upper case letters into the transmit array */
   atrans[i]=0x40;       /* sentinel with an @ character */

   printf("\n hit RETURN to send the data out of port 'A'\n");
   getchar();

   send_data(PORTA,atrans,i);

   /*********************************/
   /* WAIT FOR SOME DATA FROM PORT A */
   /*********************************/
   get_data(PORTB);

   /*********************************/
   /* SEND OUT SOME DATA OUT PORT B */
   /*********************************/
   for (i=0;i<=25;i++)
     atrans[i]= 97 + i; /* store lower case letters into the transmit array */
   atrans[i]=0x40;       /* sentinel with an @ character */

   printf("\n hit RETURN to send the data out of port 'B'\n");
   getchar();

   send_data(PORTB,atrans,i);
```

```
    /***********************************/
    /* WAIT FOR SOME DATA FROM PORT B */
    /***********************************/
    get_data(PORTA);

    printf("\nPress RETURN to exit the program\n");
    getchar();

}
```

B.10 ANALYSIS LIBRARY

Introduction

The C Analysis Library enables you to design custom analysis programs using the C language on the Chameleon 32. The name of the library is **libanal.a** and is located in the **/lib** directory.

The functions are described on the following pages:

Also refer to Appendix B.1 for a description of the common library functions and error codes.

A sample program is provided following the functions.

INIT_ANAL

Declaration

```
#include <cham.h>
int init_anal(port, protocol, par)
int port, protocol;
union PARBLOCK *par;
```

Description

This function initializes the hardware and loads the analysis software. Events are returned only for the port(s) selected by *port*.

Range

Port	0	Port A
	1	Port B
	2	Port A and B
Protocol	1	BOP
	2	ISDN
	7	ASYNC
	8	BSC

Par:

Note

union PARBLOCK and the parameters are defined in a:\include\cham.h

```
union PARBLOCK  {           /* BOP parameter block */
        struct {
            unsigned short encode;
        } pbop;

        struct  {               /* Bisync parameter block */
            unsigned short  table;
            unsigned short  bcc;
            char  sync1;
            char  sync2;
            unsigned short  parity;
        }pbisync;

        struct  {               /* Async parameter block */
            unsigned short  baud;
            unsigned short  parity;
            unsigned short  databit;
        }pasync;
};
```

BOP/ISDN

If Protocol = 1 (BOP) OR 2 (ISDN), the following parameter must be initialized:

par->pbop.encode	0	NRZ
	1	NRZI

ASYNC

If Protocol = 7 (Async), the following three parameters must be initialized:

par->pasync.baud	2	75 baud rate
	3	110
	5	300
	6	600
	7	1200
	8	2400
	9	4800
	10	9600
	11	19200
par->pasync.parity	0	None
	1	Odd
	2	Even
par->pasync.databit 5	5	data bit
	6	6 data bits
	7	7 data bits
	8	8 data bits

BSC

If Protocol = 8 (BSC), the following parameters must be initialized:

par->pbsync.table	0	ASCII
	1	EBCDIC
par->pbsync.bcc	0	CRC16
	1	LRC
	2	CCITT

par->pbsync.sync1 Range: 0 - 0xff
par->pbsync.sync2 Range: 0 - 0xff

AND if par->pbsync.table is initialized to ASCII the following parameter must also be initialized:

par->pbsync.parity	0	None
	1	Odd
	2	Even

Returns

0	Successful
-1	Parameter error
-2	Port B not available (Not a Dual Port machine)
-3	Cannot load analysis files
-4	Simulation is running
-5	Port is busy

See also the global error codes on page B.1-1.

GETEVENT

Declaration

```
#include <cham.h>

int getevent(pevent)
event *pevent;
```

Description

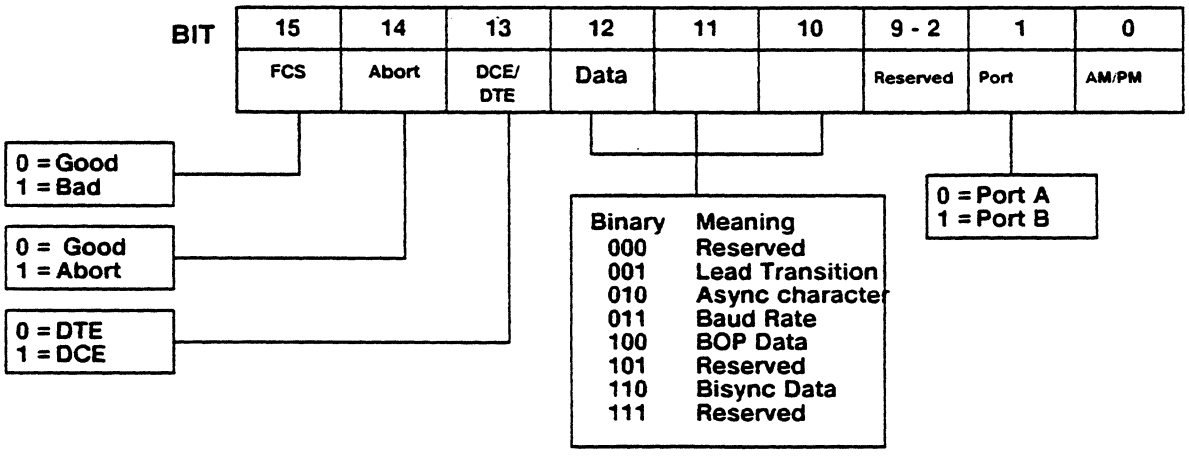
This function gets an event from the line, if available. The structure below will be filled with the event information upon returning from the function call.

```
typedef struct {
        unsigned short type;
        unsigned short length;
        unsigned short buflen;
        unsigned char *pdata;
        long seconds;
        long ms20;
        unsigned short special;
        unsigned short crc;
        unsigned short flags;
} event;
```

Note

The getevent function copies event.buflen bytes of the frame data to the user data buffer (event.pdata). event->pdata and event->buflen must be initialized before calling the routine. If the length of the frame data received is greater than the data buffer length, the data will be truncated.

event.type A bit-mapped information element described in the figure below.



event.length The length of the data

event.pdata Data buffer address that points to the frame (must be initialized by user)

event.ampm The time stamp flag for morning or evening

event.seconds The number of seconds elapsed since 12:00 midnight or noon

event.ms20 The number of 20 microsecond units elapsed since the second, which wraps around at 50,000

event.buflen Data buffer length (must be initialized by user)

event.crc Contains the crc value of the frame

event.flags For BOP only, contains the number of flags.

event.special Contains different information based on bits 10 - 12, as follows;

If a baud rate event, the baud rate change event will contain the new baud rate value.

If a lead transition event, the bits are interpreted as follows:

Bits	7	6	5	4	3	2	1	0
DCE	CTS	DSR	DCD	RI	SDCD			
DTE	RTS	DTR						

Otherwise the value in special should be ignored.

Returns

0	Successful
-1	No new events
-2	Data overwitten (buffer wrapped)
-3	Wrong port selected

See also the global error codes on page B.1-1.

RESET_ANAL

Declaration

```
int reset_anal(port)
int port;
```

Range

port	0	Port A
	1	Port B
	2	Ports A and B

Description

This function resets the acquisition processor (Front End Processor board).

Returns

See the global error codes on page B.1-1.

Sample Program

The program below (**bscanal**) configures the Chameleon 32 to analyze BSC traffic over the Basic Rate Interface. When an event is received, the main program calls the **dispframe** function, which interprets and displays the event. It displays the following information:

- Port (A or B) is displayed in white
- DCE (red) or DTE (green) event
- Event type in hex
- Timestamp
- Length, if data frame > 0
- Data in hex, if any

```
#include    <cham.h>
#include    <video.h>
#include    <stdio.h>

extern long _stdvt;
extern char  getch();
char  ch;
int   len;

#define     SP        SetBasic(c,rsp);
#define setup()    c[0]=1;c[1]=1; SP
#define chan_func()   c[0]=4;c[1]=0;c[2]=3;c[3]=1;  SP
#define reset()       c[0]=3; SP
main()
{
event *pevent, eventst;
char  buffer[256];
int   val;
int   c [5], rsp [5];
      union PARBLOCK    *par;

      reset();
      setup();
      reset_anal();
      chan_func();

      printf("setting SET-UP ..."\n);
      disablecur(_stdvt);     /* disables cursor  */
      pevent = &eventst;      /*  allocates event space  */
      val=init_anal(PORTA,ISDN,&par);
            if ( val != 0 )  {      /*  check return status  */
                  printf("init_anal failed, error code =%d\n",val);
                  exit(0);}
      puts("Hit <q> to abort, hit space bar to halt\n");
      pevent->pdata = (unsigned char*)buffer;       /*  initialize buffer address  */
      pevent->buflen = 250;       /*  initialize buffer length  */
```

```
        while ((ch = getch(_stdvt)) !='q' ){        /*  If q is pressed, aborted  */
                    if ( ch== ' ')
                        getchar();
                    if ( getevent (pevent)== 0)  {
                        dispframe (pevent);
                    }
        }
        reset_anal(PORTA);
        enablecur (_stdvt);
}

dispframe (pe)
event *pe;
{
        if ( (pe->type & EM_SIDE) == EM_SIDE )
              puts("\033[32m\nDCE Event");     /*  displays DCE events in red  */

              else
                    puts ("\033[31m\nDTE Event");     /*  displays DTE events in green  */

        printf("Type = %04x   Sec %5ld  Usec(20) %5ld\n", pe->type, pe->seconds, pe->ms20);

        if (pe-> length){   /*  determines if event was a data frame  */
              printf("length = %d\n", pe->length);
        len=((unsigned int)pe->length > (unsigned int)pe->buflen) ? pe->buflen :   pe->length;
              hex_dump(len, pe->pdata);
        }
        printf("\033[0m\n");    /*  restores the screen color  */
}
        hex_dump(len, data)         /*  displays data in ASCII  */
        int len;
        unsigned char *data;
{
              unsigned int a,b;
              for (a=0 ,b=0; a!=len; a++)  {
                    if (b == 0)
                        printf("\n%4x\t" , a);
                    printf("%02x ",data[a]);
                    if (++b == 16)
                        b = 0;
              }
}
```

B.11 MULTI-LINK LAPD LIBRARY

The Multi-Link LAPD library is an optional C library which must be purchased in addition to the Chameleon 32 C Development System.

The Multi-Link LAPD library supports a total of 64 logical links. The library is named **libmlapd.a** and is located in the a:\lib directory of the hard disk.

Note

Some functions in the Multi-Link LAPD library provide a degree of compatibility with the single-link LAPD library. These functions enable you to quickly upgrade your existing code to take advantage of the Multi-Link LAPD features without having to rewrite your program. However, these functions should not be used in new programs, and should be eventually taken out of upgraded program, as they may be removed in some future library version.

TGI Byte

Multi-Link LAPD includes the option of using the TGI (Terminal Group Identifier) address byte. (If used, the TGI is the third byte of the LAPD Address field.) The set_tgi() function assigns a TGI value to the currently selected link in the range 0 - 14. The simulator handles the TGI byte as follows:

- If a valid TGI value is assigned to a link, the TGI byte is used.
- If a TGI value > 14 is assigned to a link, the TGI byte is not used.
- If an invalid TGI value (0 or 14, depending on LAPD implementation) is assigned to a link, it enables you to test the recovery of the Device Under Test to an invalid TGI value.

Link Selection

The Multi-Link LAPD library includes functions which enable you to control the use of 64 logical links. Each of the 64 links is referred to by a unique link number in the range 0 - 63. Each of the 64 logical links has its own SAPI and TEI value, which are assigned as follows:

1. Select one of the 64 links (0 - 63) using set_link. All links default to state 9, disabled.
2. Assign the link a SAPI value using set_sapi.
3. Assign the link a TEI value using set_tei.

4. If applicable, assign the link a TGI value using the set_tgi function.

5. When you select a link using set_link, you can then use the other functions to set the link on (slon), set the link off (slof), and transmit and receive messages.

Frame Status Word

A two-byte frame status (frstat) which is attached at the beginning of each received message provides access to the following information:

- Frame type
- Number of link which received the frame
- Command or response frame
- Poll/Final bit value

The get_rxstat function returns the low order byte of *frstat*. The get_rlink function returns the high order byte of *frstat*.

General Notes

If two or more links have the same address (SAPI/TEI or SAPI/TEI/TGI combination), received frames will be considered to belong to the highest link number matching that address.

A link is disabled by selecting the link and setting the SAPI or TEI to an invalid value. You should ensure that the link is in the disconnected state before you disable it. If a link is disabled while in a connected (multi-frame) state, the device under test will see it as a Layer 1 failure.

Note

Using an invalid TGI value disables the use of the TGI byte. It does not disable the link.

Setting the SAPI and/or TEI value to an invalid value sets the link to the disabled state (9). This provides an easy means of testing lost link recovery and providing a means of ignoring unused links.

There are two functions which get the state of a link. The status function gets the state of the selected link. The link_stat function gets the state of any specified link.

get_freelink returns the number of the lowest numbered disabled link. search_link returns the number of the lowest link matching a specified SAPI/TEI combination. find_link returns the number of the lowest link matching a specified SAPI/TEI/TGI combination.

Functions

The following functions are in the Multi-Link LAPD library. Also refer to the common functions and error codes described in Appendix B.1. Programming tips and examples are provided beginning on page B.11-52.

find_link()

Declaration

```
int find_link(sapi,tei,tgi)
int sapi, tei, tgi;
```

sapi SAPI value of the link, in the range 0 - 255 (SAPI > 63 is invalid, resulting in *don't care* value)

tei TEI value of the link, in the range 0 - 255 (TEI > 127 is invalid, resulting in *don't care* value)

tgi TGI value of the link, in the range 0 - 255 0 - 255 (TGI > 15 is invalid, resulting in *don't care* value)

Description

This function returns the number of the lowest link matching the SAPI/TEI/TGI values specified. An invalid SAPI, TEI, or TGI value is treated as a *don't care* value for that parameter, so that setting a parameter to an invalid value returns the first link matching the valid parameters.

Returns

0 - 63	Matching link number
-1	No match found

get_freelink()

Declaration int get_freelink()

Description This function gets the link number (0 - 63) of the first disabled link.

Returns

0 - 63	Disabled link number
-1	No free links available
-2	initp1 not performed

get_fwaiting

Declaration

```
int get_fwaiting (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the number of I-frames waiting to be transmitted on link *lnkn*.

Returns

0 - 7 Number of I-frames waiting to be sent by link lnkn

See also the global error codes on page B.1-1.

get_link()

Declaration int get_link()

Description This function gets the number of the link which is currently under user control.

Returns

0 - 63	Current link number
-1	initp1 not performed

get_lnksapi

Declaration

```
int get_lnksapi (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the SAPI value for link lnkn.

Returns

0 - 63	SAPI value assigned to link lnkn
> 63	SAPI value disabling link lnkn

See also the global error codes on page B.1-1.

get_lnktei

Declaration

```
int get_lnktei (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the TEI value for link lnkn.

Returns

0 - 127	TEI value assigned to link lnkn
> 127	TEI value disabling link lnkn

See also the global error codes on page B.1-1.

get_lnktgi

Declaration

```
int get_lnktgi (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the TGI value for link lnkn.

Returns

0 - 14	TGI value assigned to link lnkn
15 - 255	TGI value disabling use of TGI on lnkn (the link is not disabled)

See also the global error codes on page B.1-1.

get_meswaiting

Declaration int get_meswaiting ()

Description This function gets the number of messages waiting to be received from the Front End Processor (FEP).

Note This function returns the number of messages buffered by the FEP. The library buffers one additional message.

Returns

0 - 32	Number of messages waiting to be received from the FEP

See also the global error codes on page B.1-1.

get_rlink()

Declaration int get_rlink()

Description This function gets the number of the link which sent the last received message. This is the high order byte of the frame status word **frstat** passed by the FEP.

Returns

0 - 63	Current link number
-1	No messages received yet
-2	initp1 not performed

get_rntei

Declaration

```
int get_rntei (val)
int val;
```

Range

Range of val is untested

Description

This is a dummy function to maintain compatibility with existing single link LAPD programs that are being upgraded to Multi-Link LAPD. Refer to the LAPD library in Appendix B.3 for more information.

Returns

This function always returns zero.

get_rsapi

Declaration

```
int get_rsapi (val)
int val;
```

Range

Range of ***val*** is untested

Description

This is a dummy function to maintain compatibility with the existing single link LAPD programs that are being upgraded to Multi-Link LAPD. Refer to the LAPD library in Appendix B.3 for more information.

Returns

This function always returns zero.

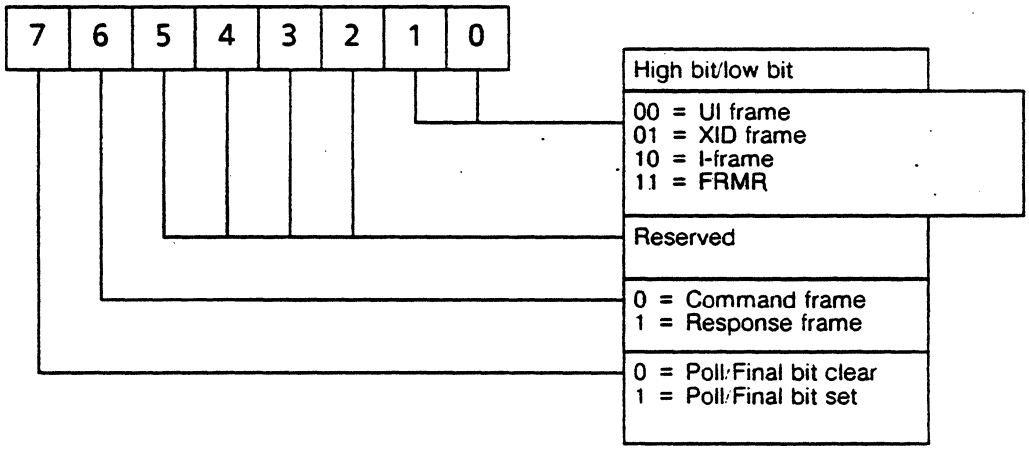
get_rxstat()

Declaration char get_rxstat()

Description This function gets the low order byte of the frame status word **frstat**, which contains the frame type, C/R bit and P/F bit of the last received message.

Returns

0 - 0xC3	frstat value (interpreted as shown below)
0xFF	No messages received yet
0xFE	initp1 not performed



Examples

0x41	Non-final XID response
0x02	I-frame command
0xC3	Final FRMR response

get_sapi()

Declaration int get_sapi()

Description This function gets the SAPI value of the link currently under user control.

Returns

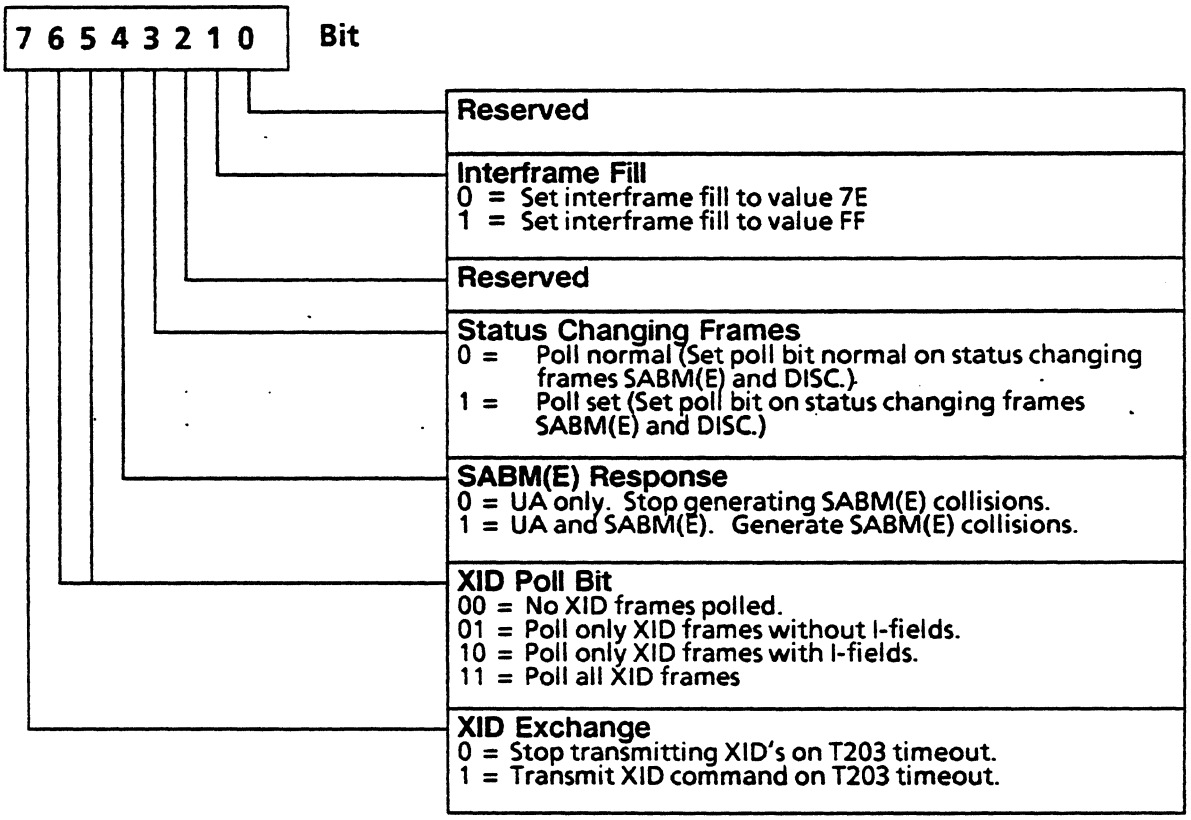
0 - 255	SAPI for current link

Also see global error codes on page B.1-1.

get_sconfig ()

Declaration int get_sconfig ()

Description This function returns a copy of the current control configuration byte, which can be interpreted as shown in the figure below.



get__sim ()

Declaration int get__sim ()

Description This function returns a copy of of the network/subscriber selection.

Returns

0	Network
1	Subscriber

get_tei()

Declaration int get_tei()

Description This function gets the TEI of the link currently under user control.

Returns 0 - 255 TEI for current link number

Also see global error codes on page B.1-1.

get_tgi()

Declaration int get_tgi()

Description This function returns the TGI value of the link currently under user control.

Returns

0 - 14	TGI for current link number
15 - 255	TGI value disabling use of TGI on the link (the link is not disabled)

Also see global error codes on page B.1-1.

get_window

Declaration

```
int get_window (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the number of outstanding I-frames on link number **lnkn.**

Returns

0 - 7 Number of unacknowledged I-frames of link lnkn

See also the global error codes on page B.1-1.

initp1

Declaration

```
int initp1 (interface, sta, encode, bitrt)
int interface, sta, encode;
long  bitrt;
```

Description

initp1 loads the Front End Processor (FEP) code for the library and starts simulation. Predefined values exist in mlklib.h to aid in setting up the call to this function. **sta** is the station type and selects the initial sense of the command/response bit. The library permits reselection of the station type at any time. **encode** selects the physical data encoding. **bitrt** sets the data rate when simulating a DCE device.

Note

This function is identical to and interchangeable with the *start_sim* function. It has been included in the Multi-Link LAPD library for downward compatibility with the single link LAPD library.

Ranges

interface	0	V-type interface (DCE)
	1	V-type interface (DTE)
	2	ISDN interface
sta	0	NETWORK
	1	SUBSCRIBER
encode	0	NRZ
	1	NRZI
bitrt	Any long integer value from 50 - 64000.	

Returns

See the global error codes on page B.1-1.

link_stat

Declaration

```
int link_stat(n)
char n;
```

Range

n 0 - 63

Description

This function gets the current state of link n.

Returns

0 - 9 Current state of link (see table below)

See also the global error codes on page B.1-1.

STATE	LINK STATUS
0	Link Disconnected
1	Link Connection Requested
2	Frame Rejected
3	Disconnect Requested
4	Information Transfer
5	Local Station Busy
6	Remote Station Busy
7	Local and Remote Station Busy
8	Remote Station not Responding
9	Link Disabled

receive

Declaration

```
int receive(dest_addr)
char *dest_addr;
```

Description

This function receives a message from the FEP by performing the following tasks:

- It polls the FEP to see if any received messages are available
- It transfers the message contents to the user defined buffer pointed to by **dest_addr**
- The total length of the message (including the frame status bytes ***frstat***) is placed in the global variable **rxlen**

  The frstat word is accessible by calling **get_rlink** and **get_rxstat** so that you can interpret and respond to a message quickly. The frstat bytes are attached to the beginning of each received message so that several messages may be received, sorted, interpreted, and individual responses made.

It is up to the user to ensure that the destination buffer is long enough to contain the message. Generally, a length equal to N201 + 2 is adequate.

s_n200

Declaration

```
int s_n200 (val)
int val;
```

Range

val 1 - 255

Description

This function sets the maximum number of retries (N200).

Returns

0 Successful

See also global error codes on page B.1-1.

s_n201

Declaration

```
int s_n201 (val)
int val;
```

Range

val 1 - 512

Description

This function sets the maximum length for an I-frame (N201).

Returns

0 Successful

See also global error codes on page B.1-1.

s_t200

Declaration

```
int s_t200 (val)
int val;
```

Range

val 0 - 255

Description

This function sets the time allowed for the remote station to respond (T200). Setting this value to 0 disables the T200 timer.

Returns

0 Successful

See also global error codes on page B.1-1.

s_t203

Declaration

```
int s_t203 (val)
int val;
```

Range

val 0 - 255

Description

This function sets the maximum time between frames (T203). On time out, a polled RR or XID command is transmitted, depending on the configuration selection. Setting this value to 0 disables the T203 timer.

Returns

0 Successful

See also global error codes on page B.1-1.

set_link

Declaration

```
int set_link(n)
char n;
```

Range

n 0 - 63

Description

This function puts link n under user control. Only one link at a time can be under user control.

Returns

0	Successful
-1	Parameter out of range
-2	initp1 not performed
-3	Timeout

set_net ()

Declaration int set_net ()

Description This function sets the simulation side to NETWORK. The Chameleon can simulate either a network or subscriber device.

When the Chameleon 32 emulates a network, it sends commands with the C/R bit set to one, and responds with the C/R bit set to zero. It sends the selected SAPI and TEI with the C/R bit automatically set in accordance with CCITT Q. 921.

set_rntei

Declaration

```
int set_rntei (val, tei)
int val, tei;
```

Range

Range of val and tei is untested

Description

This is a dummy function to maintain compatibility with existing LAPD single-link programs that are being upgraded to Multi-Link LAPD. Refer to the LAPD library in Appendix B.3 for more information.

Returns

This function always returns zero.

set_rsapi

Declaration

```
int set_rsapi (val, sapi)
int val, sapi;
```

Range

Range of ***val*** and ***sapi*** is untested

Description

This is a dummy function to maintain compatibility with existing LAPD programs that are being upgraded to Multi-Link LAPD. Refer to the LAPD library in Appendix B.3 for more information.

Returns

This function always returns zero.

set_sapi

Declaration

```
int set_sapi(v)
char v;
```

Range

Accepted range of *v* is 0 - 255. A value over 63 disables the selected link.

Description

This function sets the SAPI value for the link under user control. The SAPI (Service Access Point Identifier) indicates the layer two service type requested or supported. Normal values are:

0	Call Control procedures
16	Packet communication procedures
63	Management procedures
64 - 255	Disable link

Returns

0	Successful
-1	Parameter out of range
-2	initp1 not performed
-3	Timeout

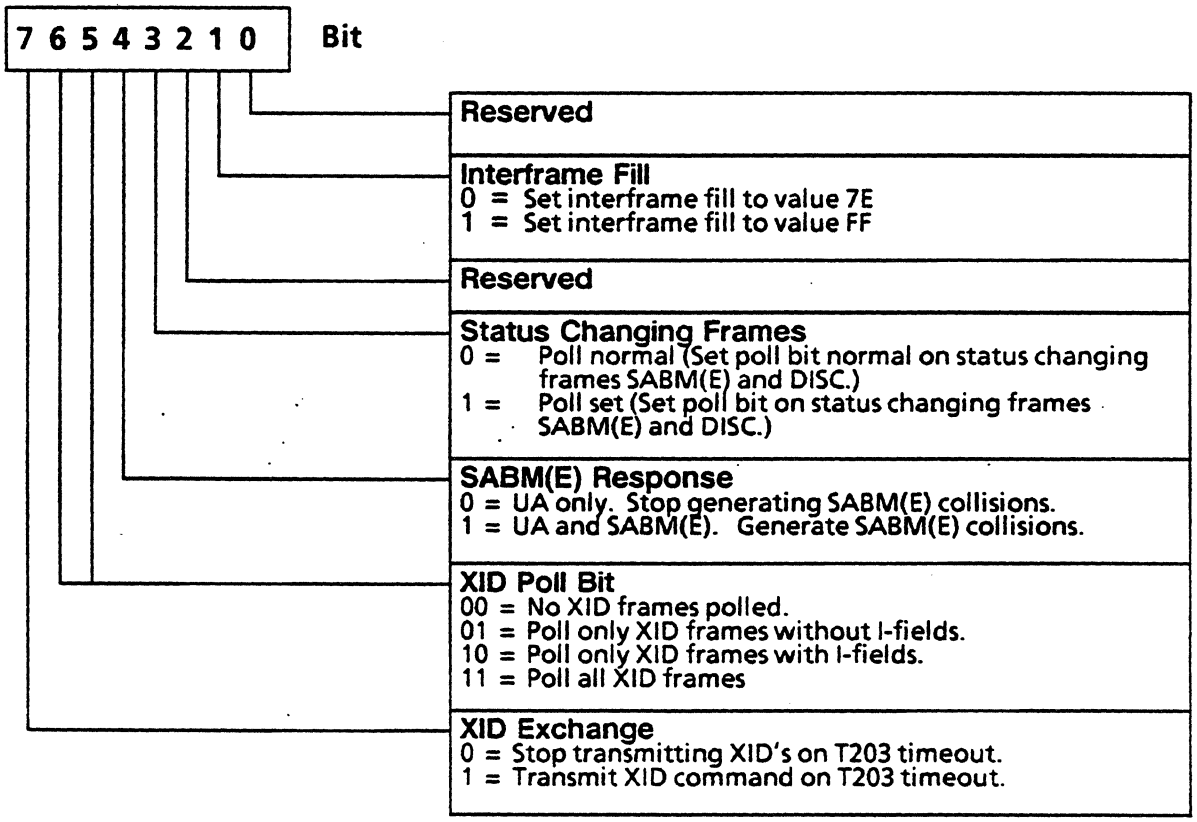
set_sconfig

Declaration

```
int set_sconfig (byte)
int byte;
```

Description

This function sets the value of the control configuration byte, interpreted as shown in the figure below.



Returns

0 Successful

See also global error codes on page B.1-1.

set_sub ()

Declaration int set_sub ()

Description This function sets the simulation side to SUBSCRIBER. The Chameleon can simulate either a network or subscriber device.

When the Chameleon 32 emulates a LAPD subscriber, it sends commands with the C/R bit set to zero, and responds with the C/R bit set to one. It sends the selected SAPI and TEI with the C/R bit automatically set in accordance with CCITT Q. 921.

set_tei

Declaration

```
int set_tei(value)
char value;
```

Range

value — The TEI value to use for the link, as follows:

0 - 127	Valid TEI values
128 - 255	Invalid TEI value, which causes the link to be disabled

Description

This function sets the TEI value for the link under user control. The TEI (Terminal Endpoint Identifier) is a value assigned to and may be associated with a single terminal and a given point-to-point data link connection. At any time, a given terminal endpoint (TE) may contain one or more TEIs.

This value may be assigned by the carrier at the time of equipment installation, or may be automatically assigned on a call-by-call basis. The broadcast value is associated with all user-side data link entities with the same SAPI, regardless of other assigned value(s).

Normal values are:

0 - 63	Non-Automatically assigned values
64 - 126	Automatically assigned values
127	Broadcast value
128 - 255	Disable link

Returns

0	Successful
-1	Parameter out of range
-2	initp1 not performed
-3	Timeout

set_tgi

Declaration

```
int set_tgi(value)
char value;
```

Range

value — The TGI value to use for the link, as follows:

0 - 14	Valid TGI values
15 - 255	Disables the use of the TGI byte

Description

This function sets the TGI (Terminal Group Identifier) value for the link under user control. If used, the TGI is the third byte of the LAPD Address field.

If an invalid TGI value (0 or 14, depending on LAPD implementation) is assigned to a link, it enables you to test the recovery of the Device Under Test to an invalid TGI value.

Returns

0	Successful
-1	Parameter out of range
-2	initp1 not performed
-3	Timeout

set_window

Declaration

```
int set_window (val)
int val;
```

Range

val 1 - 7

Description

This function sets the maximum number of outstanding frames on each link.

Note

The total of outstanding frames + the number of frames passed to the FEP waiting to be transmitted + the number of messages over 16 bytes long waiting to be received from the FEP may not exceed 80.

Returns

0 Successful

See also global error codes on page B.1-1.

setflg

Declaration

```
int setflg (flag)
int flag;
```

Range

flag	1	0x7E fill
	0	0xFF fill

Description

This function selects an interframe fill pattern.

Returns

0 Successful

See also global error codes on page B.1-1.

slof ()

Declaration int slof ()

Description This function sends a DISC and waits for a UA frame. This is equivalent to the CCITT primitive **DL RELEASE**.

Returns 0 Successful

Also see global error codes on page B.1-1.

slon ()

Declaration int slon ()

Description This function sends a SABME and waits for a UA frame. This is equivalent to the CCITT primitive **DL ESTABLISH.**

Returns 0 Successful

Also see global error codes on page B.1-1.

srch_lnk

Declaration

```
int srch_lnk(sapi,tei)
int sapi, tei;
```

Description

This function returns the number of lowest link matching the specified SAPI/TEI. If you set one parameter to an invalid value, it returns the first link matching the valid parameter. In other words, any invalid value is a *don't care*.

Note

To search for a link by specifying SAPI, TEI, and TGI, refer to the find_link function.

Returns

0 - 63	Number of lowest link matching parameters
-1	No match found

start_sim

Declaration

```
int start_sim (interface, sta, encode, bitrt)
int interface, sta, encode;
long  bitrt;
```

Description

start_sim loads the Front End Processor (FEP) code for the library and starts simulation. Predefined values exist in mlklib.h to aid in setting up the call to this function. **sta** is the station type and selects the initial sense of the command/response bit. The library permits reselection of the station type at any time. **encode** selects the physical data encoding. **bitrt** sets the data rate when simulating a DCE device.

Note

This function is identical to and interchangeable with the *initp1* function. initp1 is included for downward compatibility with the single link LAPD library.

Ranges

interface	0	V-type interface (DCE)
	1	V-type interface (DTE)
	2	ISDN interface
sta	0	NETWORK
	1	SUBSCRIBER
encode	0	NRZ
	1	NRZI
bitrt	Any long integer value from 50 - 64000.	

Returns

See the global error codes on page B.1-1.

status()

Declaration int status()

Description This function gets the current state of link under user control.

Returns 0 - 9 Current state of link (see table below)

See also the global error codes on page B.1-1.

STATE	LINK STATUS
0	Link Disconnected
1	Link Connection Requested
2	Frame Rejected
3	Disconnect Requested
4	Information Transfer
5	Local Station Busy
6	Remote Station Busy
7	Local and Remote Station Busy
8	Remote Station not Responding
9	Link Disabled

trans

Declaration

```
int trans (frame,address,len)
int frame, len;
char *address;
```

Description

This function transmits a frame, as follows:

frame selects type of frame to transmit:

0x80	I-frame	Sequenced (numbered) I-frame
0x81	UI	Unnumbered I-frame (NSI)
0x82	XIDC	XID command frame
0x83	XIDR	XID response frame

address is a pointer to the first byte of the message to be transmitted.

len is the actual length of the message to be transmitted. There are two restrictions on the message length:

- I-frames should not exceed the value set in N201 (maximum length of an I-frame)
- The total length of the frame cannot exceed 512 bytes.

Returns

0 Successful

Also see global error codes on page B.1-1.

transmit

Declaration

```
int transmit (xloc, xlen)
char *xloc;
int xlen;
```

Description

This function transmits a message in a sequenced (numbered) I-frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. There are two restrictions on the message length:

- I-frames should not exceed the value set in N201 (maximum length of an I-frame)
- The total length of the frame cannot exceed 512 bytes.

Note

The *transmit* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trui

Declaration

```
int trui (xloc, xlen)
char *xloc;
int  xlen;
```

Description

This function transmits a message in an unnumbered I-frame (UI frame).

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trui* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trxcni

Declaration int trxcni ()

Description This function transmits an XID command frame with no data field.

Returns 0 Successful

See also global error codes on page B.1-1.

trxidc

Declaration

```
int trxidc (xloc, xlen)
char *xloc;
int  xlen;
```

Description

This function transmits a message in an XID command frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trxidc* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trxidr

Declaration

```
int trxidr (xloc, xlen)
char *xloc;
int xlen;
```

Description

Transmit a message in an XID response frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trxidc* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trxrni

Declaration int trxrni ()

Description This function transmits an XID response frame with no data field.

Returns 0 Successful

See also global error codes on page B.1-1.

PROGRAMMING NOTES AND EXAMPLES

This section provides general information about using the Multi-Link LAPD library. Following these general notes is a section on converting current LAPD C programs to Multi-Link LAPD programs.

General Notes

In your program, specify the Chameleon port being used by a call to the set_port function. This is not necessary when using a Single Port Chameleon, but it should be done to make your application portable.

A call to initp1 must be made to start the Front End Processor (FEP). This loads the FEP operating code and starts the simulation. The Chameleon is then ready to begin testing.

Before frames can be transmitted or received, at least one link must be enabled. This is done by:

a. Selecting a link (default is 0) using the set_link function

b. Setting the SAPI and TEI to valid values. This is done by calling set_sapi and set_tei.

Interpreting Received Messages

To interpret a received message, you must know the SAPI value and the frame type of the received message. This information is available to you from the frame status bytes, and can be accessed using the technique shown below:

```
receive(pointr);
if(rxlen==0)                    /*exit if no message received*/
      return(0);
frtype=(get_rxsta()&3);         /*get the frame type status bits*/
lnk=get_rlink();                /*get number of the link sending message*/
sapval=get_lnksap(lnk);         /*get the SAPI for that link*/
```

In this example, the frame type may be interpreted from ***frtype*** as follows:

0 = UI frame
1 = XID frame
2 = I-frame
3 = FRMR

sapval is the SAPI value assigned to the link on which the message was received.

Optimizing Transmit Speed

To optimize speed for applications such as load generators, follows these guidelines:

- The trans function is faster than the other transmit functions such as ***transmit***, ***trui***, ***trxidr***, and ***trxidc***.
- If you are not concerned with the contents of the received messages, use the following technique to keep the receive buffer empty. This is faster than a call to the ***receive*** function in the transmit loop.

```
if(get_meswtg()>=6)
    flush();
```

- Minimize input, output, and screen print operations in all tasks. Since the processor that runs the C shell also manages many of the I/O tasks, this will result in your simulation program running faster.
- Run your program in background mode. This is done by adding an ampersand (&) at the end of the file name when starting the test from the C shell. For example:

  test&

  will run the program **test** in background mode. This causes the program to run at a higher priority and frees the C shell for other uses. This technique also reduces the number of windows available for other tasks, since a separate window is opened for each program running in background mode.

Transmitting Responses

In a test environment, the actual information content of a given message type is often fixed. In such cases, only the message type must be known in order to select the proper response. To simplify responding to message, predefine the content of responses in a message array.

In the following program fragment, the following is assumed:

- A set of responses and pointers to the responses has been defined earlier
- SAPI/TEI combinations have been set up

The program fragment uses a defined value TYPEOS, which is the offset to the byte in the message containing the message type. The call to fix_cref extracts the call reference value from the received message and copies it into the selected response message.

```
respond()
{
      char mestyp,*resp;
      int resp_len;
      rxlen=0; /* prepare to loop until message received */
      while(!rxlen)
            receive(&rxmes[0]);

      if((get_rxstat()&3)=2) /*only respond to Iframes*/
      {
      mestyp=rxmes[TYPEOS]; /*get message type from rcvd message*/
      switch(mestyp)
            {
            case  CALL_SU:       /*if msg is call setup*/
                                 /*respond setup ack*/
                  resp=&su_ack;
                  resp_len=SU_ACK_LEN;
                  break;

      •
      •
      •

            case RELEASE:        /* if msg is release*/
                                 /*respond release complete*/
                  resp=&rel_cmplt;
                  resp_len=REL_CMPLT_LEN;

            }
      fix_cref(resp); /*set response call ref value*/
      set_link(get_rlink()); /*select link to send response*/
      transmit(resp,resp_len); /* send response*/
      }

fix_cref(dest)
      char *dest;
{
      int ref_val;
      refval=rxmes[CREF_OS];
      *(dest+CREF_OS-2)=refval; /*-2 to allow for frstat bytes*/
}
```

Simulating an ASP

The following program fragment demonstrates how to use the Multi-Link LAPD library to simulate an Assignment Source Point (ASP). The fragment selects a random TEI value and returns it to be assigned to the Device Under Test.

This assignment function consists of two nested loops. The inner loop requests a random number until it gets one greater than 63 (64 to 126 are available for auto-assignment). It then exits to the outer loop where a search is made to see if the value is in use. This assumes that each assigned TEI is set in an active link on the Chameleon.

```
assign_tei()
{
char tei_val;

      while(1) /* start outer loop */
      {
         while(1) /* start inner loop */
         {
            tei_val=rnd(126);
            if(tei_val>63) /* exit if good value */
                  break;
         }
         if(srch_lnk(64,tei_val)<0) /* invalid sapi=don't care */
            break; /* exit if no match, tei is ok to use */
      }
return(tei_val);
}
```

UPGRADING PROGRAMS TO MULTI-LINK LAPD

Upgrading existing LAPD programs to Multi-Link LAPD has been made as simple as possible. Many such programs can be run with little or no modification. This section contains hints and suggestions to aid in modifying existing user programs that have problems with the Multi-Link LAPD library, and upgrading to take advantage of the new library features.

Troubleshooting

This section lists typical problems and solutions when converting LAPD C program to Multi-Link LAPD. It is assumed that the program will run when relinked with the LAPD library.

Problem You cannot establish a link because a SABME is not transmitted by the Chameleon on a slon() call.

Solution 1: The selected link is probably disabled. Multi-Link LAPD defaults all SAPI/TEI combinations to an invalid value, thus disabling the link. With the LAPD library, the SAPI and TEI are assigned default values of 0. To enable a link, add set_sapi and set_tei calls using valid values.

Solution 2: If you are using a V-type interface, and simulating a DTE, the Chameleon may not be receiving a clock from the Device Under Test. To determine this, set the interframe fill (setflg) to 0x7E. If the clock is being received, the Chamelelon front panel red and green Data LEDs should both illuminate.

Problem You cannot establish a link because the Chameleon does not respond when a SABME is received.

Solution 1: The SAPI/TEI combination is not assigned to a link. This occurs in programs that set receive SAPI (RSAPI) and receive TEI (RTEI) values without setting the transmit SAPI and TEI to the same value. A call to set_link(get_freelnk()) will select an unused link. Then use set_sapi and set_tei to assign the SAPI/TEI combination to that link. This must be done for each SAPI/TEI combination that might be received.

Solution 2: The subscriber/network selection is incorrect. Your program may change this selection as desired, but remember that it is a global selection which affects all 64 links.

Problem

The Chameleon rejects frames on some SAPI/TEI combinations because of an incorrect N(r) value.

Solution 1: More than one link is assigned the same SAPI/TEI, and the library is transmitting on one while the FEP is receiving on another. This can be avoided by adding a call to srch_lnk before assigning SAPI and TEI values to determine whether the combination already exists.

Solution 2: There is a modulus mismatch. The Multi-Link LAPD FEP codes run only Mod 128.

Problem

Timeout problems. Timeout (T200,T203) problems will nearly always be caused by changing the SAPI/TEI combination on a link after it is established and in multi-frame mode.

Solution 1: The preferred solution is to assign all SAPI/TEI combinations to different links. Subsequent SAPI/TEI changes can then be replaced with set_link calls.

Solution 2: A second option is to disable the T200 and T203 timers in the Chameleon and the Device Under Test, and then re-establish the link at intervals to purge unacknowledged I-frames.

This solution is not good practice and should be used temporarily while debugging a program.

Problem

The Chameleon sends response messages on the wrong link.

Cause 1: The correct link was not selected prior to transmitting the response.

Cause 2: The link was selected based on the single-link LAPD frstat interpretation.

Cause 3: The link selection was based on the wrong received message.

There are two possible solutions. If your program receives and responds to messages one at a time, either solution will work. If a number of messages are received, then interpreted as time permits, use the second solution.

Solution 1: When a message is received (and a response is needed), call set_link(get_rlink()). This selects the link from which the last message was received.

Solution 2: Set the link to the value of the first byte of frstat in the received message. This selects the proper link regardless of the number of messages received after the one being interpreted.

SAMPLE PROGRAM 1: ASSIGNMENT SOURCE POINT (ASP) SIMULATOR

This sample program causes the Chameleon to simulate an Assignment Source Point (ASP) which generates random TEI values and assigns them on demand. The program has three major parts:

- Part one starts the Multi-Link LAPD simulator and initializes the Front End Processor (FEP).
- Part two is the actual ASP simulator.
- Part three is an exit routine that scans the state of all 64 links, sends a disconnect to any that are in a multiple frame state, removes any assigned TEI's, and then stops simulation.

The ASP Simulator

Fully simulating an ASP is more complex than it first appears. A number of tasks must be performed which, in this program, involve nested loops and subroutines. The outermost loop tests for user input, specifically the letter **Q**, and exits when it is typed and the current cycle is finished. Until this happens, it prepares the next inner loop by setting a one second resolution countdown timer to 30 seconds.

The second loop contains a third loop which waits for the timer to count down to zero. While it is waiting for the timer, it is also waiting to receive a message from the FEP. If a message is received, a message interpreter routine is called to decide what to do with it.

When the timer reaches zero, another loop is entered that scans links 1 - 63 for valid TEI values. For each valid TEI, an ID check message is sent. If an ID check response is received, a second wait loop is entered to see if another response is received. If two responses are received, the RI and AI values are compared. If the AIs match and the RIs do not, two users have the same TEI, so that TEI is removed.

The Message Interpreter

There are three calls to this subroutine, which returns values indicating the received message type. The actions taken depend on the message type, as follows:

- If the message is not from link zero, it is not from a UI frame, or it has a MEI other than 15, it is ignored.
- If it is an id request, the assign() function is called.

- If it is an id check response, the following occurs:
  - If the call is from the thirty second time loop, an id check response message is ignored. This is based on the asumption that it was either sent in error, or was sent so late that, by the time the loop has started, the timeout routine has already removed the TEI.
  - At call 2, the RI value is extracted from the message and saved, then a second timed receive loop is entered. If another id check response is received at call 3, the RI value is compared with the one from call 2, and if it is different, the TEI is in use by two subscribers and it is removed.

Removing a TEI

To remove or unassign a TEI, the following occurs:

- An ID remove message is sent.
- The local link to which the TEI was assigned is disabled by giving it an invalid TEI.

The Assignment Subroutine

To assign a TEI, the following sequence occurs:

- A call is made to get_freelnk() to see if there is a link available for assignment. If a link is not available, the request is denied.
- The AI value is examined. If the AI is 127, a counter is started in the range 64 to 126.
- A search is made to see if this value has been assigned to any link, and if so, another number is generated.
- Once an unassigned TEI value is found, it is assigned to the link obtained by the get_freelnk() call and then sent in an ID assigned message.
- If the AI value is less than 127, a request for a specific TEI, a check is made to see if it was already assigned. If so, it is denied, otherwise it is assigned.

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>


#define T201 20

extern int rxlen; /*this is where receive puts the message length*/
char rmes[260],tmes[5];         /*Messages are stored in character arrays
                                because it is easier to interpret the message and
                                make a response.*/

main()
{
int reti,lnkno;

      reti=startup();    /*call the startup routine below*/
          if(reti)   /*if any problems, quit*/
            exit(0);

/*while all the links are still disabled, set up the layer 2 parameters*/

      reti=s_t200(0);
      reti=s_t203(0);
/*timers t200,t203 disabled to reduce irrelevent traffic*/

      reti=s_n200(3);
      reti=s_n201(260);
      reti=set_window(3);
/*Initialize the xmit message array.  This will be used for id check and remove*/
      tmes[0]=15;
      tmes[1]=0;
      tmes[2]=0;

/* Set up 1 link for broadcast management procedures*/

      reti=set_link(0);
      reti=set_sapi(63);
      reti=set_tei(127);

/* go do the ASP simulation until the user wants to quit */
      aspsim();

/* go stop things as gracefully as possible */
      shut_down();
}


startup()
{
int rets;
```

```
/*attempts to start the FEP on port A and returns the results.  Setup is
DCE simulation, network, NRZ encoding, 16000 BPS.  */

      rets=setport(PORTA); /*run this program on port a*/
      return(initp1(0,0,0,16000L));
}

aspsim()
{
int rets,rival1,rival2;
char lnkno,answer,tei,ai1,ai2,stop_flag;

      rets=inittime(); /*initialize the timers*/
      stop_flag=1;

      while (stop_flag) /*loop until 'Q' typed*/
      {
         settimer(2,T201);
         while(timer(2)&&stop_flag)
            {
            if (toupper(getch(_stdvt))=='Q')/*stop loop if 'Q' typed*/
                  stop_flag=0;
            rets=receive(&rmes[0]);
            if(rxlen)
                  {
                  rets=interp();
                  }
            }

/*scan links for assigned TEIs and see if they are still in use*/
         for(lnkno=1;lnkno<64;lnkno++)
            {
            tei=get_lnktei(lnkno);
            if(tei>63 && tei<127)/*if link has a valid TEI */
                  {
  /*build and send an id check message*/
                  tmes[3]=4;/*id check message type*/
                  tmes[4]=(tei<<1)+1;
                  settimer(0,500);
                  rets=1;
                  while(rets&&timer(0))
                  rets=trui(&tmes[0],5);/*send check message*/

   /*wait 2 seconds for an id response*/

                  answer=0;
                  settimer(2,2); /*2 seconds is long enough*/
                     while(timer(2))
                      {
                      rets=receive(&rmes[0]); /*see if any messages*/
                  rets=0;
                       if(rxlen)
                        {
                        rets=interp();/* if so, see what to do*/
                        }
```

```
                    if(rets) /*if id check response*/
                       {
                       rival1=rmes[4]+256*rmes[3];
                       ai1=rmes[6]; /* get ai byte*/
                       answer=1;    /*flag TEI  in use*/

                       }
                    if(rets==1) /*if id check response*/
                       break; /*then exit first timer loop*/
                     }
/*resend check message if no response received*/
            if(!answer)
                {
                settimer(0,500);
                rets=1;
                while(rets&&timer(0))
                rets=trui(&tmes[0],5);
                }

  /* start second wait loop*/
                  settimer(2,2); /*2 seconds is long enough*/
                     while(timer(2))
                      {
                      rets=receive(&rmes[0]); /*see if any messages*/
                       if(rxlen)
                         rets=interp();/* if so, see what to do*/
                     if(rets==1) /*if id check response*/
                        {
                        answer++;/* add 1 to answer*/
                        rival2=rmes[4]+256*rmes[3];/*get RI value*/
                        ai2=rmes[6]; /* get ai byte*/
                        }
                     if(rets==1) /*if id check response*/
                        break; /*then exit first timer loop*/
                      }

            switch(answer)
                  {
                  case  0:/*no response, TEI  no longer in use*/
                        rmvtei(get_lnktei(lnkno));
                        break;
                  case  1:/*1 response, TEI  in use*/
                        break;
                  case  2:/*2 responses*/
 /*if different ri values and same ai, two entities have same TEI */
                        if((rival1!=rival2)&&(ai1==ai2))
                        rmvtei(get_lnktei(lnkno));
                  }
               }
            }
      }
}


rmvtei(tei)
```

```
char tei;
{
int rets;
      tmes[3]=6;/*id remove message type*/
  /*ri = tei shifted left 1 bit and the lsb set*/
      tmes[4]=(tei<<1)+1;
      set_link(srch_lnk(255,tei));
      if(get_link()>0)
            {
            set_tei(255);
            set_sapi(255);
            set_link(0);/*back to management link*/
            xyplot(0,0);
            printf("removing tei %d    \n",tei);
            settimer(0,500);
            rets=1;
            while(rets&&timer(0))
            rets=trui(&tmes[0],5);/*send id remove message*/
            }
}

interp()
{
int rets;

  /*return 0 if message not from link 0, not a UI frame, or not ASP entity id code*/
      if(get_rlink()||(get_rxstat()&3)||rmes[2]!=15)
                  return(0);
      switch(rmes[5])
            {
            case  1: /*id request*/
                     rets=assign();
                     break;

            case  5: /*id check response*/
                     rets=1;
                     break;

            default:   rets=0;
            }
      return(rets);
}


xyplot (x, y)
{
      printf ("\033[%d;%df", y, x);
      fflush (stdout);
}


assign()
{
int rets;
char mestyp,ai,lnkno;
```

```
      ai=(rmes[6]&0xfe)>>1;/*get requested TEI  value*/
      lnkno=get_freelink();/*get a link to assign to the requestor*/

      if(lnkno<0) /*if no links available*/
            {
            retmes(3,ai);/*deny TEI  assignment*/
            return(0); /* and leave*/
            }
      if(ai<127) /*if specific TEI  value requested*/
            {
            if(srch_lnk(255,ai)<0)/*if requested TEI  not in use*/
                  {
                  give_tei(lnkno,ai);/*go make assignment*/
                  return(0);
                  }
            /*if requested TEI  is already in use*/
            retmes(3,ai);/*deny TEI  assignment*/
            return(0);/* and leave*/
            }
/*at this point, the request is for any TEI , and since there is a
link free, there are at least 2 TEI  values not in use.  All we have to
do is find one.  While random number generation is more in complience
with CCITT recomendations, the following code is simpler and yields
more repeatable results.*/

      for(ai=64;ai<127;ai++)/*loop until an unassigned TEI  value reached*/
            {
            rets=srch_lnk(255,ai);/*see if TEI  in use*/
            if(rets<0)/*if not, break out of loop*/
                  break;
            }
      give_tei(lnkno,ai);/*go make assignment*/
      return(0);
}

give_tei(lnkno,tei)
char lnkno,tei;
{
int rets;

      /*setup the local link first*/
            rets=set_link(lnkno);
            rets=set_sapi(0);
            rets=set_tei(tei);

      /*next, assign the TEI  to the device under test and exit*/
xyplot(0,0);
printf("assigning tei %d  \n",tei);
            rets=set_link(0);
            retmes(2,tei);
/*          return();*/
}

retmes(mestyp,ai)
```

```
char mestyp,ai;
{
int rets;
/* send return message generated by inserting new message type and
ai values into the last received message.  This saves transferring the
ri value into a new message.*/

      rmes[5]=mestyp;
      rmes[6]=(ai<<1)+1;
      settimer(0,500);
      rets=1;
      while(rets&&timer(0))
      rets=trui(&rmes[2],5);
}

shut_down()
{
int rets;
char lnkno;

      for(lnkno=1;lnkno<64;lnkno++)
            {
            rets=set_link(lnkno);
            if(status()>0&&status()<9)
                  slof();
            if(get_tei()<127)
                  rmvtei(get_tei());

            }
}
```

SAMPLE PROGRAM 2: ASSIGNMENT SOURCE POINT (ASP) TESTER

This program (**asp2**) displays a menu which enables the user to select one of the following:

- Request a TEI
- Discontinue use of randomly selected TEIs
- Simulate two subscribers with the same TEI
- Discontinue use of all TEIs

This program is intended to be run against sample program 1 so that you can see the Multi-Link LAPD sample program in action. For this reason, the startup() routine sets up the program to run on Port B, so that the two programs can be run against each other on a Dual Port machine.

However, this program can be used as an independent test in an actual test environment. To use the program on Port A, or on a single port Chameleon, simply change the port selection in the startup routine.

In this program, the following sequence occurs:

- The Front End Processor (FEP) is initialized
- A loop is entered that assigns a pseudo-random RI value to each link. This value is made up of the link number in the MSB and a random number in the LSB. This accomplishes two things:
  - Each RI is guaranteed to be unique without a second loop to verify it
  - It gives each RI a tag so the user can more easily track the results of a test run.

The User Interface

A selection menu is printed on the upper part of the screen. The 'getch(_stdvt)' call scans the keyboard for any key stroke. The lower part of the screen has two fields, the most recent user selection is described in the upper one, and the lower field describes the latest TEI assignment, denial, or removal.

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>

#define T201 30

extern int rxlen; /*this is where receive puts the message length*/

char rmes[260],tmes[5],lnk_ri[64];
 /*Character arrays are used for storing the messages because it is easier to see what is being
done to interpret the message and make aresponse.*/

main()
{
int reti,lnkno;

      reti=startup();    /*call the startup routine*/
          if(reti)   /*if any problems, quit*/
            exit(0);

/*while all the links are still disabled, setup the layer 2 parameters*/

      reti=s_t200(0);
      reti=s_t203(0);
/*timers t200,t203 disabled to reduce irelevent traffic*/

      reti=s_n200(3);
      reti=s_n201(260);
      reti=set_window(3);

/*initialize the xmit message array.  this will be used for id messages.*/
      tmes[0]=15;
      tmes[1]=0;
      tmes[2]=0;

/* setup 1 link for broadcast management procedures*/

      reti=set_link(0);
      reti=set_sapi(63);
      reti=set_tei(127);

/* go do the ASP test until the user wants to quit */
      asptes();

/* go undo everything before simulation is stopped */
      shut_down();


}


startup()
```

```
{
int rets;
/*attempt to start the FEP on port B and return the results.  Setup is
DTE simulation, subscriber, NRZ encoding, 16000 BPS.  */

      rets=setport(PORTB); /*run this program on port b*/
      return(initp1(1,1,0,16000L));
}

asptes()
{
int rets,rival1;
char lnkno,answer,tei,teslnk,act_flag,quit_flag;

/* set up RI value table */
         for(lnkno=1;lnkno<64;lnkno++)
            lnk_ri[lnkno]=5+2*lnkno;

/*list user selections*/
print_men();

quit_flag=1;
while(quit_flag) /*loop until quit selected by user*/
   {

      answer=toupper(getch(_stdvt));/*get user command */

        rets=receive(&rmes[0]);
      if(rxlen)/*if message received*/
            {
            rets=interp();/*see what kind*/
            tei=(rmes[6]&0xfe)>>1;
            xyplot(5,12);
            switch(rets)
            {
            case 0: /*message does not concern us*/
                  break;

            case 1:/*tei assigned message*/
                  printf("TEI %d assigned to link %d       \n",tei,lnkno);
                  set_link(lnkno);
                  set_sapi(0);
                  set_tei(tei);
                  set_link(0);
                  break;

            case 2:/*tei denied message*/
                  printf("TEI %d denied for link %d       \n",tei,lnkno);
                  break;

            case 3:/*id check message*/
                  lnkno=srch_lnk(255,tei);
                  if(!(lnkno<0))
                  {
                  tmes[1]=lnkno;
```

```
            tmes[2]=lnk_ri[lnkno];
            tmes[3]=5;
            tmes[4]=(tei<<1)+1;
            settimer(0,500); /* if fep busy, keep trying 1/2 sec*/
            rets=1;
            while(rets&&timer(0))
            rets=trui(&tmes[0],5);
      /*if user selection D, fake 2 users on same tei*/
            if(act_flag==1)
                  {
                  tmes[1]=0;
                  tmes[2]=lnk_ri[0];
                  tmes[3]=5;
                  tmes[4]=(tei<<1)+1;
                  settimer(0,500); /* if fep busy, keep trying 1/2 sec*/
                  rets=1;
                  while(rets&&timer(0))
                  rets=trui(&tmes[0],5);
                  act_flag=0;
                  }
            }
            break;

      case 4:/*tei removed message*/
            printf("TEI %d removed            \n",tei);
            set_link(srch_lnk(255,tei));
            if(get_link()>0)
                  set_tei(255);
            set_link(0);
            break;
      }
     }

if((answer>='A')&&(answer<='H'))
{

xyplot(5,10);
switch(answer)
  {
  case 'A':/*request any tei*/
        lnkno=get_freelink();
        tmes[1]= (lnkno<0? 90:lnkno);
        tmes[2]=(lnkno<0? 90:lnk_ri[lnkno]);
        tmes[3]=1;
        tmes[4]=255;
printf("requesting any tei for link %d                  \n",tmes[1]);
        settimer(0,500); /* if fep busy, keep trying 1/2 sec*/
        rets=1;
        while(rets&&timer(0))
        rets=trui(&tmes[0],5);
        break;
  case 'B':/*request specific tei*/
        for(tei=64;tei<127;tei++)
            {
            if(srch_lnk(255,tei)<0)
```

```
                    break;
                }
            lnkno=get_freelink();
            tmes[1]= (lnkno<0? 90:lnkno);
            tmes[2]=(lnkno<0? 90:lnk_ri[lnkno]);
            tmes[3]=1;
            tmes[4]=(tei<<1)+1;
printf("requesting tei %d for link %d                    \n",tei,tmes[1]);
            settimer(0,500); /* if fep busy, keep trying 1/2 sec*/
            rets=1;
            while(rets&&timer(0))
            rets=trui(&tmes[0],5);
                    break;
  case 'C':/*request an existing tei*/
            for(tei=126;tei>63;tei--)
                {
                if(srch_lnk(255,tei)>=0)
                    break;
                }
            lnkno=get_freelink();
            tmes[1]= (lnkno<0? 90:lnkno);
            tmes[2]=(lnkno<0? 90:lnk_ri[lnkno]);
            tmes[3]=1;
            tmes[4]=(tei<<1)+1;
printf("requesting existing tei %d for link %d              \n",tei,lnkno);
            settimer(0,500); /* if fep busy, keep trying 1/2 sec*/
            rets=1;
            while(rets&&timer(0))
            rets=trui(&tmes[0],5);
                    break;

  case 'D':/*simulate 2 users with same tei*/
            act_flag=1;
printf("Next ID check will get 2 different responses\n");
            break;

  case 'E':/*kill 1 tei*/
            for(tei=126;tei>63;tei--)
                {
                lnkno=srch_lnk(255,tei);
                if(lnkno>=0)
                    break;

                }
            if(lnkno<0)
          printf("No TEI's assigned                          \n");
            else
                {
          printf("TEI %d no longer in use                    \n",tei);
                set_link(lnkno);
                set_tei(255);
                set_link(0);
                }
            break;
```

```
        case 'F':/* kill all TEI's*/
              for(lnkno=1;lnkno<64;lnkno++)
                  {
                  set_link(lnkno);
                  set_tei(255);
                  }
             printf("All TEI's no longer in use                    \n");
             set_link(0);
             break;

      case 'G': /*stop test*/
             quit_flag=0;
             break;

      case 'H': /* print help menu*/
             print_men();

        }
      }
   }

}

xyplot (xpos, ypos)
int xpos,ypos;
{
      printf ("\033[%d;%df",ypos,xpos);
      fflush (stdout);
}

clear_screen()
{
      printf ("\033[2J");
      fflush (stdout);
}

print_men()
{

clear_screen();
printf("A - request any TEI\n");
printf("B - request specific TEI\n");
printf("C - request a TEI that is in use\n");
printf("D - simulate 2 users with same TEI\n");
printf("E - stop using a TEI\n");
printf("F - stop using all TEI's\n");
printf("G - stop test\n");
printf("H - reprint screen\n");
      xyplot(5,10);
}

interp()
{
int rets;
```

```
  /*return 0 if message not from link 0 ,not a UI frame,or not ASP
    entity id code*/
      if(get_rlink()||(get_rxstat()&3)||rmes[2]!=15)
                        return(0);
      switch(rmes[5])
            {
            case  2: /*id assigned*/
                     rets=1;
                     break;

            case  3: /*id denied*/
                     rets=2;
                     break;

            case  4: /*id check*/
                     rets=3;
                     break;

            case  6: /*id removed*/
                     rets=4;
                     break;

            default:   rets=0;
            }
      return(rets);
}

retmes(mestyp,ai)
char mestyp,ai;
{
int rets;
/* send return message generated by inserting new message type and
ai values into the last received message.*/

      rmes[5]=mestyp;
      rmes[6]=(ai<<1)+1;
      settimer(0,500); /* if fep busy, keep trying 1/2 sec*/
      rets=1;
      while(rets&&timer(0))
      rets=trui(&rmes[2],5);
}

shut_down()
{
int rets;
char lnkno;

      for(lnkno=1;lnkno<64;lnkno++)
            {
            rets=set_link(lnkno);
            if(status()>0 && status()<9)
                  rets=slof();
            }

}
```

B.12 V.120 LIBRARY

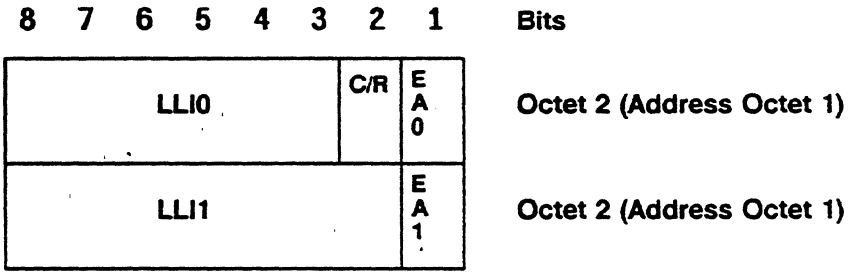
The V.120 library is an optional C library which must be purchased in addition to the Chameleon 32 C Development System. It supports a total of 64 independent links and adheres to the following V.120 protocol requirements:

- The C/R bit is set to 0 for all commands, and to 1 for all responses, regardless of the sending station
- I-Frames can be command or response frames
- When a command reject (CMDR) occurs the link is automatically restarted

The library is named ***libv120.a*** and is located in the a:\lib directory of the hard disk.

V.120 Address

The format of the V.120 address field can be viewed as a single 13-bit Logical Link Identifier (LLI) field or as two separate fields (LLI0 and LLI1). This is shown as follows:



The LLI0 is the high order 6 bits of the LLI. The LLI1 is the low order 7 bits of the LLI. The LLI is a concatenation of the LLI0 field with the LLI1 field. The LLI can take on values in the range 0 - 8191, with the following reserved values:

LLI (In Decimal)	FUNCTION
0	In-channel signaling
1 - 255	Reserved for future standardization
256	Default LLI
257-2047	For LLI assignment
2048-8190	Reserved for future standardization
8191	In-channel layer management

LLI Values

EA0 is the octet 2 address extension bit, which is set to 0. EA1 is the octet 3 address extension bit, which is set to 1 for two octet address field.

Link Selection

With the V.120 library, you select one of the 64 links (0 - 63) using the set_link function. You then assign the link an LLI with the set_lli function as a single decimal value as shown in the figure on the previous page. All links default to state 9, disabled.

When you select a link using set_link, you can then use the other library functions to set the link on (slon), set the link off (slof), and transmit and receive messages.

Frame Status Word

A two-byte frame status (frstat) which is attached at the beginning of each received message provides access to the following information:

- Link number over which message was received
- Frame type
- Command or response frame
- Poll/Final bit value

The get_rxstat function returns the low order byte of *frstat*. The get_rlink function returns the high order byte of *frstat*.

Functions

The V.120 library provides the functions listed below. Also refer to the common functions and error codes described in Appendix B.1.

get_freelink()

Declaration int get_freelink()

Description This function gets the number of the first disabled link.

Returns

0 - 63	Disabled link number
-1	No free links available
-2	initp1 not performed

get_fwaiting

Declaration

```
int get_fwaiting (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the number of I-frames waiting to be transmitted on link *lnkn*.

Returns

0 - 7 Number of I-frames waiting to be sent by link lnkn

See also the global error codes on page B.1-1.

get_link()

Declaration int get_link()

Description This function gets the number of the link which is currently under user control.

Returns

0 - 63	Current link number
-1	initp1 not performed

See also the global error codes on page B.1-1.

get_lli()

Declaration int get_lli()

Description This function gets the LLI of the link currently under user control.

Returns

0 - 0x1FFF	LLI of current link
-1	initp1 not performed

See also the global error codes on page B.1-1.

get_lnklli

Declaration

```
int get_lnklli (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the LLI value for link lnkn.

Returns

0 - 0x1FFF	LLI value assigned to link n
> 0x1FFF	Link lnkn is disabled

See also the global error codes on page B.1-1.

get_meswaiting

Declaration int get_meswaiting ()

Description This function gets the number of messages waiting to be received from the Front End Processor (FEP).

Note An additional received message is buffered by the library.

Returns

0 - 32	Number of messages waiting to be received from the FEP

See also the global error codes on page B.1-1.

get_rlink()

Declaration int get_rlink()

Description This function gets the number of the link which sent the last received message. This is the high order byte of the frame status word **frstat** passed by the FEP.

Returns

0 - 63	Current link number
-1	No messages received yet
-2	initp1 not performed

See also the global error codes on page B.1-1.

get_rxstat()

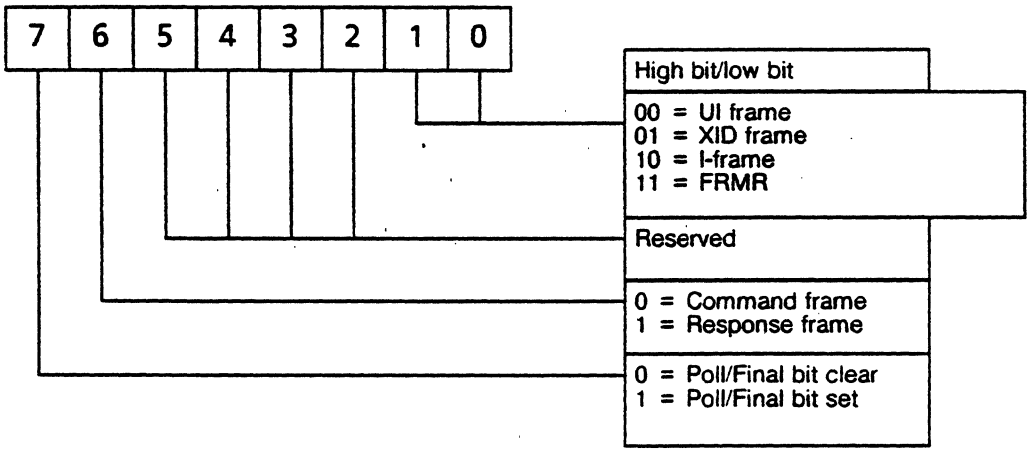
Declaration char get_rxstat()

Description This function gets the low order byte of the frame status word **frstat**, which contains the frame type etc. for the last received message.

Returns

0 - 0xC3	frstat value (interpreted as shown below)
0xFF	No messages received yet
0xFE	initp1 not performed

See also the global error codes on page B.1-1.



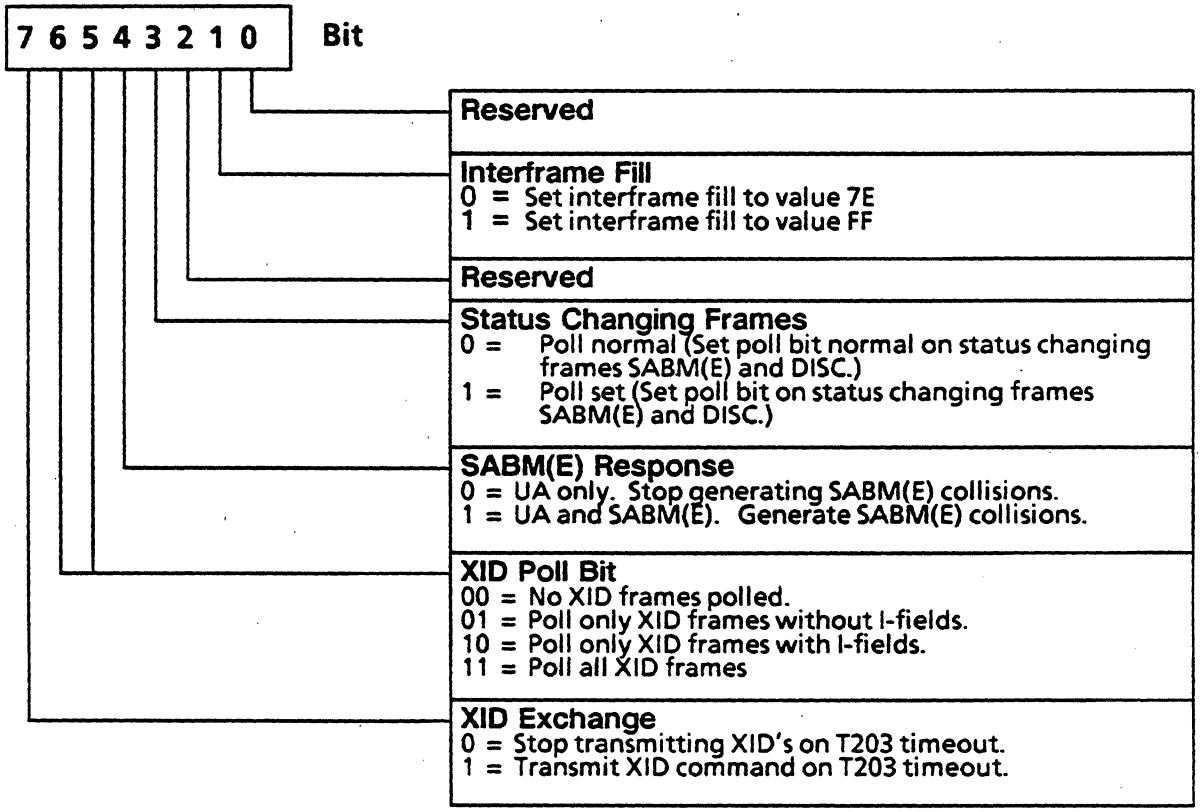
Examples

0x41	Non-final XID response
0x02	I-frame command
0xC3	Final FRMR response

get_sconfig ()

Declaration int get_sconfig ()

Description This function returns a copy of the current control configuration byte, which can be interpreted as shown in the figure below.



get_window

Declaration

```
int get_window (lnkn)
char lnkn;
```

Range

lnkn 0 - 63

Description

This function gets the number of outstanding I-frames on link number **lnkn.**

Returns

0 - 7 Number of unacknowledged I-frames of link lnkn

See also the global error codes on page B.1-1.

initp1

Declaration

```
int initp1 (interface, sta, encode, bitrt)
int interface, sta, encode;
long  bitrt;
```

Description

initp1 loads the Front End Process (FEP) code for the selected library and starts simulation. Predefined values exist in mlklib.h to aid in setting up the call to this function. **sta** is the station type and selects the initial sense of the command/response bit. The library permits reselection of the station type at any time. **encode** selects the physical data encoding. **bitrt** sets the data rate when simulating a DCE device.

Note

This function is identical to and interchangeable with the *start_sim* function. It is included to provide downward compatibility with the single link LAPD library.

Ranges

interface	0	V-type interface (DCE)
	1	V-type interface (DTE)
	2	ISDN interface
sta	0	V.120
encode	0	NRZ
	1	NRZI
bitrt	Any long integer value from 50 - 64000.	

Returns

See the global error codes on page B.1-1.

link_stat

Declaration int link_stat(n)
char n;

Range n 0 - 63

Description This function gets the current state of link n.

Returns 0 - 9 Current state of link (see table below)

See also the global error codes on page B.1-1.

STATE	LINK STATUS
0	Link Disconnected
1	Link Connection Requested
2	Frame Rejected
3	Disconnect Requested
4	Information Transfer
5	Local Station Busy
6	Remote Station Busy
7	Local and Remote Station Busy
8	Remote Station not Responding
9	Link Disabled

receive

Declaration

```
int receive(dest_addr)
char *dest_addr;
```

Description

This function receives a message from the FEP by performing the following tasks:

- It polls the FEP to see if any received messages are available
- It transfers the message contents to the user defined buffer pointed to by **dest_addr**
- The total length of the message (including the frame status bytes ***frstat***) is placed in the global variable **rxlen**

  The frstat word is accessible by calling **get_rlink** and **get_rxstat** so that you can interpret and respond to a message quickly. The frstat bytes are attached to the beginning of each received message so that several messages may be received, sorted, interpreted, and individual responses made.

It is up to the user to assure that the destination buffer is long enough to contain the message. Generally, a length equal to N201 + 2 is adequate.

s_n200

Declaration

```
int s_n200 (val)
int val;
```

Range

val 1 - 255

Description

This function sets the maximum number of retries (N200).

Returns

0 Successful

See also global error codes on page B.1-1.

s_n201

Declaration

```
int s_n201 (val)
int val;
```

Range

val 1 - 512

Description

This function sets the maximum length for an I-frame (N201).

Returns

0 Successful

See also global error codes on page B.1-1.

s_t200

Declaration

```
int s_t200 (val)
int val;
```

Range

val 0 - 255

Description

This function sets the time allowed for the remote station to respond (T200). Setting this value to 0 disables the T200 timer.

Returns

0 Successful

See also global error codes on page B.1-1.

s_t203

Declaration

```
int s_t203 (val)
int val;
```

Range

val 0 - 255

Description

This function sets the maximum time between frames (T203). On time out, a polled RR or XID command is transmitted, depending on the configuration selection. Setting this value to 0 disables the T203 timer.

Returns

0 Successful

See also global error codes on page B.1-1.

set_link

Declaration

```
int set_link(n)
char n;
```

Range

n 0 - 63

Description

This function puts link n under user control. Only one link at a time can be under user control. Initp1 must be performed prior to this function.

Returns

0	Successful
-1	Parameter out of range
-2	initp1 not performed
-3	Timeout

See also the global error codes on page B.1-1.

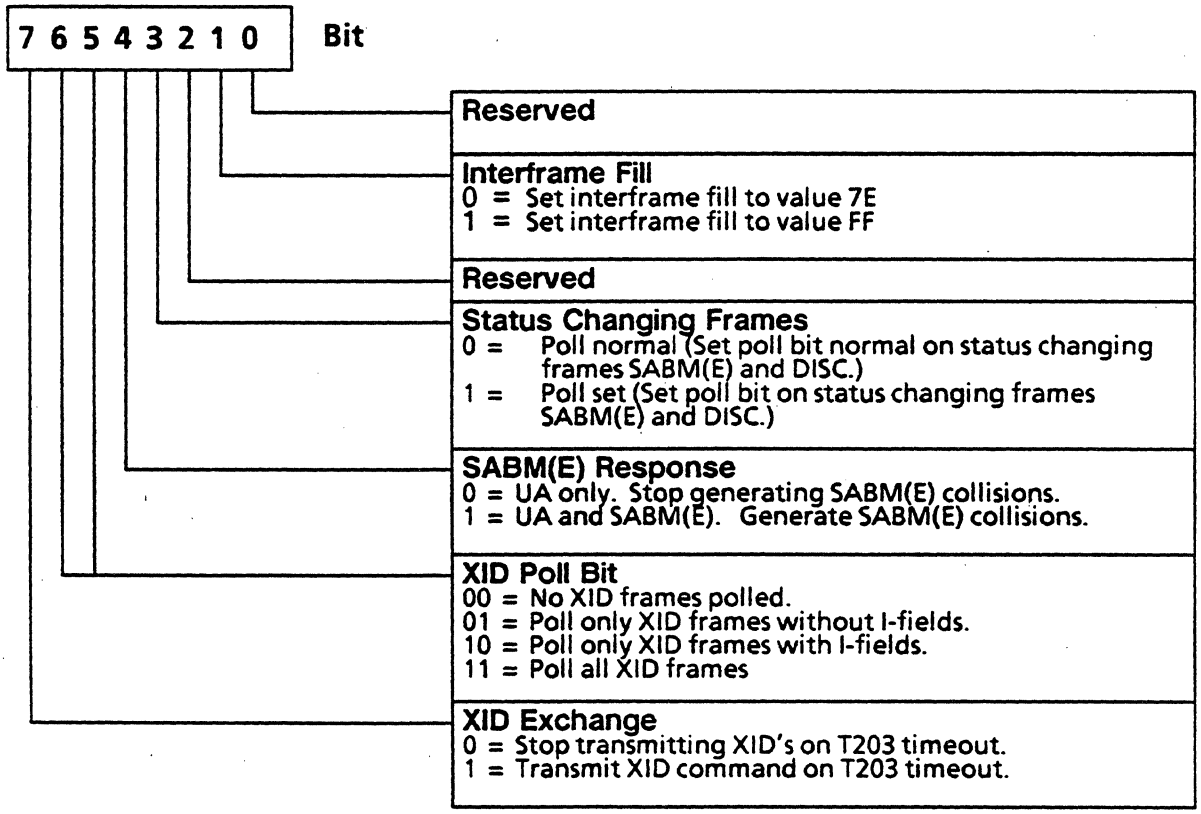
set_sconfig

Declaration

```
int set_sconfig (byte)
int byte;
```

Description

This function sets the value of the control configuration byte, interpreted as shown in the figure below.



Returns

0	Successful

See also global error codes on page B.1-1.

set_lli

Declaration

```
int set_lli(val)
int val;
```

Range

val	0x00 - 0xFFFF hex	
	A value > 0x1FFF disables the link	

Description

This function sets the LLI (Logical Link Identifier) value for the link under user control.

Returns

0	Successful
-1	Parameter out of range
-2	initp1 not performed
-3	Timeout

See also the global error codes on page B.1-1.

set_window

Declaration

```
int set_window (val)
int val;
```

Range

val 1 - 7

Description

This function sets the maximum number of outstanding frames on each link.

Note

The total of outstanding frames + the number of frames passed to the FEP waiting to be transmitted + the number of messages over 16 bytes long waiting to be received from the FEP may not exceed 80.

Returns

0 Successful

See also global error codes on page B.1-1.

setflg

Declaration

```
int setflg (flag)
int flag;
```

Range

flag	0	0x7E fill
	1	0xFF fill

Description

This function selects an interframe fill pattern.

Returns

0 Successful

See also global error codes on page B.1-1.

slof ()

Declaration int slof ()

Description This function sends a DISC and waits for a UA frame. This is equivalent to the CCITT primitive **DL RELEASE**.

Returns 0 Successful

Also see global error codes on page B.1-1.

slon ()

Declaration int slon ()

Description This function sends a SABME and waits for a UA frame. This is equivalent to the CCITT primitive **DL ESTABLISH**.

Returns 0 Successful

Also see global error codes on page B.1-1.

srch_lnk

Declaration

```
int srch_lnk(lli)
int lli;
```

Description

This function returns the number of highest link matching the specified LLI.

Returns

0 - 63	Number of highest link matching parameters
-1	No match found

See also the global error codes on page B.1-1.

start_sim

Declaration

```
int start_sim (interface, sta, encode, bitrt)
int interface, sta, encode;
long  bitrt;
```

Description

start_sim loads the Front End Process (FEP) code for the selected library and starts simulation. Predefined values exist in mlklib.h to aid in setting up the call to this function. **sta** is the station type and selects the initial sense of the command/response bit. The library permits reselection of the station type at any time. **encode** selects the physical data encoding. **bitrt** sets the data rate when simulating a DCE device.

Note

This function is identical to and interchangeable with the *initp1* function. The *initp1* function is provided for downward compatibility with the single link LAPD library.

Ranges

interface	0	V-type interface (DCE)
	1	V-type interface (DTE)
	2	ISDN interface
sta	0	V.120
encode	0	NRZ
	1	NRZI
bitrt	Any long integer value from 50 - 64000.	

Returns

See the global error codes on page B.1-1.

status()

Declaration int status()

Description This function gets the current state of link under user control.

Returns 0 - 9 Current state of link (see table below)

See also the global error codes on page B.1-1.

STATE	LINK STATUS
0	Link Disconnected
1	Link Connection Requested
2	Frame Rejected
3	Disconnect Requested
4	Information Transfer
5	Local Station Busy
6	Remote Station Busy
7	Local and Remote Station Busy
8	Remote Station not Responding
9	Link Disabled

trans

Declaration

```
int trans (frame,address,len)
int frame, len;
char *address;
```

Description

This functions transmits a frame as follows:

frame selects type of frame to transmit:

0x80	I-frame	Sequenced (numbered) I-frame
0x81	UI	Unnumbered I-frame (NSI)
0x82	XIDC	XID command frame
0x83	XIDR	XID response frame
0x84	RESP_IFRAME	IFRAME response

address is a pointer to the first byte of the message to be transmitted.

len is the actual length of the message to be transmitted. There are two restrictions on the message length:

- I-frames must not exceed the value set in N201
- The total length of the frame cannot exceed 512 bytes

Returns

0 Successful

Also see global error codes on page B.1-1.

transmit

Declaration

```
int transmit (xloc, xlen)
char *xloc;
int xlen;
```

Description

This function transmits a message in a sequenced (numbered) I-frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The length of an I-frame must not exceed the value set in N201.

Note

The *transmit* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trans_resp

Declaration

```
int trans_resp (xloc, xlen)
char *xloc;
int  xlen;
```

Description

This function transmits a message in a sequenced (numbered) I-frame response.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The length of an I-frame must not be more than the value set in N201.

Note

The *trans_resp* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trui

Declaration

```
int trui (xloc, xlen)
char *xloc;
int xlen;
```

Description

Transmit a message in an unnumbered I-frame (UI frame).

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The lenght of an I-frame must not exceed the value set in N201.

Note

The *trui* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trxcni

Declaration int trxcni ()

Description This function transmits an XID command frame with no data field.

Returns 0 Successful

See also global error codes on page B.1-1.

trxidc

Declaration

```
int trxidc (xloc, xlen)
char *xloc;
int  xlen;
```

Description

This function transmits a message in an XID command frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes

Note

The *trxidc* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trxidr

Declaration

```
int trxidr (xloc, xlen)
char *xloc;
int  xlen;
```

Description

Transmit a message in an XID response frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trxidc* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0 Successful

Also see global error codes on page B.1-1.

trxrni

Declaration int trxrni ()

Description This function transmits an XID response frame with no data field.

Returns 0 Successful

See also global error codes on page B.1-1.

Sample Programs

The following are two sample SIMP/L V.120 programs which can be run against each other on a Dual Port Chameleon 32. Program test1.c uses Port B; test2.c uses Port A. The two programs interact as follows:

- Test1 sets up 64 links and randomly transmits on one of the links.
- Test2 receives the message and returns the message on the same link as was received.

The program terminates when the letter Q is pressed, or the link is lost.

Makefile

The following is the makefile for the two sample programs:

```
tests : test1 test2
test1 : test1.o
        cc -o test1 test1.o -lv120
test2 : test2.o
        cc -o test2 test2.o -lv120
```

Test1.c

The sample program test1.c is as follows:

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>

char tmsg[48],rmsg[48];

main()
{
        char insit,tempc,answer,*tadr,*radr;
        extern char;
        extern int rxlen;
        int result,tempi;
        long int count;

        printf("C program started.\n");
        tadr=&tmsg[0];
        radr=&rmsg[0];
        s_start();

/***** make and display transmit message *****/
```

```
        for(insit=0;insit<=16;insit++)
              tmsg[insit] = 15 - insit;

        printf("transmit message = ");
        for(insit=0;insit<=16;insit++)
        printf(" %0x ",tmsg[insit]);
        printf("\n");

/***** do frame level setup *****/

         result=set_mod(1);
         result=s_t200(0);
         result=s_t203(0);
         result=s_n201(20);
         result=set_window(3);

/***** setup lli for 64 links *****/

        for(;;)
        {
              tempi=get_freelink();
              if(tempi<0)
                    break;
              printf("free link = %d\n",tempi);
              result=set_link(tempi);
              result=set_lli(tempi*3);
        }

        while(status()<4);
        result=set_link(2);
        result=trans(0x80,tadr,16);
        count=0;
        result=get_link();
        clear_screen();
        printf("Current link is %d\n",result);

/***** send and receive until link state is less than 4 *****
****** or user types "Q" ****/

        for(;;)
          {

              answer = toupper(getch(_stdvt));
              if((status()<4)||(answer=='Q'))
                    break;

              result=receive(radr);
              if(rxlen>0)
                 {
```

```
                    xyplot(5,5);
                    printf("received message = ");
                    for(insit=0;insit<rxlen;insit++)
                          printf(" %0x ",rmsg[insit]);
                    printf("\n");
                      printf("Reclink = %d, frstat = %x
\n",get_rlink(),get_rxstat());

/* select a link at random and, if it is not busy, send a message */

      tempc=63*rand();
      result=set_link(tempc);
      if(status()<5)
            result=trans(0x80,tadr,16);
            count++;
            printf("That's %ld.\n",count);
              }
      }

/***** if link still connected, disconnect it, then exit *****/

      if(status()>0)
      {
            result=slof();
            printf("slof result %0d\n",result);
            while(status()>1);
      }
}

xyplot (xpos, ypos)
int xpos,ypos;
{
      printf ("\033[%d;%df",ypos,xpos);
      fflush (stdout);
}

clear_screen()
{
      printf ("\033[2J");
      fflush (stdout);
}

s_start()
      {
      int rets,ret2;

      rets=setport(PORTB);
      printf("Trying to start P1");
      rets= initp1(1,1,0,6400L);
      printf(" result = %0x\n",rets);
      }
```

Test2.c

The sample program test2.c is as follows:

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>

/* make buffers for tx and rx messages */
char tmsg[48],rmsg[48];

main()
{
      char insit,answer,*tadr,*radr;
      extern char;
      extern int rxlen;
      int result,tempi;
      long int count;

      printf("C program started.\n");
      tadr=&tmsg[0];
      radr=&rmsg[0];
      s_start();

/***** make and display transmit message *****/

      for(insit=0;insit<=16;insit++)
                  tmsg[insit] = insit;

      printf("transmit message = ");
      for(insit=0;insit<=16;insit++)
          printf(" %0x ",tmsg[insit]);
      printf("\n");

/***** do frame level setup *****/

       result=set_mod(1);
       result=s_t200(0);
       result=s_t203(0);
       result=s_n201(20);
       result=set_window(3);

/***** setup lli for 64 links and set them on *****/

      for(;;)
      {
            tempi=get_freelink();
            if(tempi<0)
                break;
            printf("free link = %d\n",tempi);
            result=set_link(tempi);
            result=set_lli(tempi*3);
```

```
            result=slon();
            while(status()<4);
      }

/**** test get link stats ***/

      for(tempi=0;tempi<64;tempi++)
      {
            result=link_stat(tempi);
            printf(" link= %d ",tempi);
            printf(" state = %d.\n",result);
            result=get_window(tempi);
            printf(" window = %d.\n",result);
            result=get_fwaiting(tempi);
            printf("frames to be sent  = %d.\n",result);
            result=get_meswaiting();
            printf(" messages to be received = %d.\n",result);
            result=get_lnklli(tempi);
            printf(" lli = %d.\n",result);
      }

/***** test link search function
search 12 should return 4
search 66 should return 22
search 189 should return 63
search 11 should return 'not found' ****/

       printf("result of search 12 = %d\n",result);
      result=srch_lnk(66);
      printf("result of search 66 = %d\n",result);
      result=srch_lnk(189);
      printf("result of search 189 = %d\n",result);
      result=srch_lnk(11);
      printf("result of search 11 = %d\n",result);

/***** send and receive untill link state is less than 4 *****
****** or user types "Q" ****/

      result=set_link(2);
      tempi=0;
      count=0;
      result=get_link();
      clear_screen();
      printf("Current link is %d\n",result);

      for(;;)
       {
            answer = toupper(getch(_stdvt));
            if((status()<4)||(answer=='Q'))
                              break;
            result=receive(radr);
            if(rxlen>0)
            {
                  xyplot(5,5);
                  printf("received message = ");
```

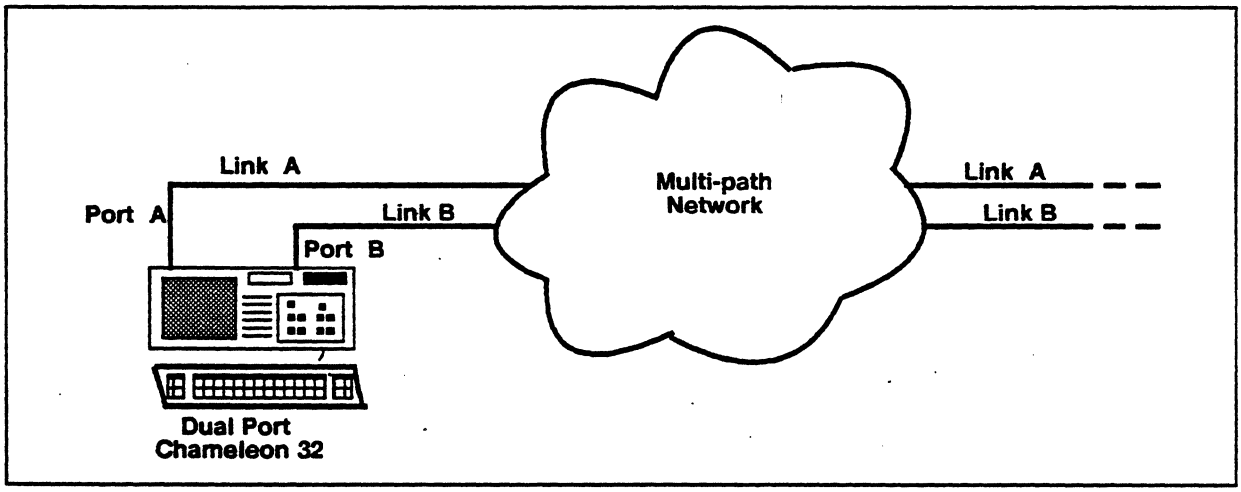
```
                    for(insit=0;insit<rxlen;insit++)
                          printf(" %0x ",rmsg[insit]);
                    printf("\n");
                    printf("Reclink = %d, frstat = %x
\n",get_rlink(),get_rxstat());
                        result=set_link(get_rlink());
                        if(status()<5)
                      result=trans(0x80,tadr,15);
                      count++;
                      printf("That's %ld.\n",count);
              }
        }

/***** if link still connected, disconnect it, then exit *****/

        if(status()>0)
        {
              result=slof();
              printf("slof result %0d\n",result);
              while(status()>1);
        }
}

xyplot (xpos, ypos)
int xpos,ypos;
{
        printf ("\033[%d;%df",ypos,xpos);
        fflush (stdout);
}

clear_screen()
{
        printf ("\033[2J");
        fflush (stdout);
}


s_start()
        {
        int rets,ret2;

        rets=setport(PORTA);
        printf("Trying to start P1");
        rets= initp1(0,0,0,6400L);
        printf(" result = %0x\n",rets);
        }
```

B.13 MULTI-LINK HDLC C LIBRARY

Introduction

The Multi-Link HDLC C Library (libmhdlc.a) is located in the \lib directory. It can be used only on a Dual Port Chameleon 32.

The Multi-Link HDLC library enables a Dual Port Chameleon 32 to simulate two links using the HDLC protocol. With the Multi-Link HDLC library, each port is configured as a permanent virtual circuit, providing two links for testing a multi-path network.



The Multi-Link HDLC library is an enhanced version of the HDLC library described in Appendix B.3. The HDLC library provides single link simulation, whereas the Multi-Link HDLC library provides dual link simulation

Applications

The library provides a high degree of flexibility while providing solutions to the problems of simulating an HDLC system with multiple physical links. The library provides functions for initializing both ports with the automatic layer 2 FEP software, establishing links, and transmitting, and receiving data on either port.

Some applications of the library include:

- Simulating a multi-circuit device by connecting the Chameleon to two Permanent Virtual Circuits (PVCs) of a multi-path packet switching network.

- Testing a single circuit by inserting the Chameleon into the middle of a single circuit and running simulation in both directions to the devices on either end. This provides a divide and diagnose test to find a circuit fault.

- Use the Chameleon as a emergency adaptor. This library was not intended for such an application; however, it is possible to use the Chameleon as an adaptor between two incompatible HDLC devices.

  For example, use the Chameleon between two DTE terminals or as a rate adaptor between two devices that run at different data rates. In this context, it is possible to connect an ISDN Basic Rate Interface to an ISDN Primary Rate Interface.

- Use the Chameleon to compare two devices or networks. The Chameleon can be set to transmit in redundant mode so that the same data is transmitted over both ports. The responses of the two devices/networks can then be evaluated.

Also refer to Appendix B.1 for a description of common library functions and error codes.

Sample programs using the Multi-Link HDLC library are provided at the end of this section.

Simulator Initialization

The library provides three functions for initializing the Chameleon ports to run the Multi-Link HDLC simulator. These functions are:

initp1 This function initializes both Ports A and B with identifical configuration parameters with a single call. For example, both ports are initialized as DCEs with a bit rate of 56000. For most testing applications, both ports will be the same.

init_a
init_b If the testing application requires that Port A and Port B configuration differ, each port can be initialized independently using the init_a and init_b functions.

There are some unique applications for the init_a()/init_b()functions. Programs with an automatic send/receive/respond function can test themselves by temporarily inserting init_a() and init_b() calls and connecting the two ports together as DCE and DTE, network and subscriber.

It is also possible, using a fairly simple receive/resend program, to use the Chameleon 32 as a data rate or side translator to link incompatible equipment such as two DCE devices or two devices with no common data rate selection.

Note

When configured as a DTE or for ISDN, timeout errors may occur when the Chameleon is configured for a data rate that is significantly higher than what is actually being received. This is more likely to occur when very long packets are being exchanged because the timeout interval is adjusted to the data rate to minimize delays. If unwarranted timeout errors occur consistently, initialize the port using a lower data rate.

Transmitting Data

There are two transmit functions in the library:

transmit This function transmits a packet over a specified port.

mlh_trans This function transmits a packet to a port depending on a specified distribution pattern which is defined using one of the following functions:

- set_ratio specifies the distribution pattern of the packets being transmitted over the two ports. The ratio can be set so that all, none, or a specified percentage of packets is transmitted over Port A or Port B.
- set_pat specifies a user-defined distribution pattern if set_ratio does not provide the distribution required by the testing application.

Each time a call is made to mlh_trans, the simulator determines over which port to transmit the packet.

mlh_trans() should be used to transmit information packets only. Other packet types should be sent using the transmit() function.

Global Variables

When the two physical ports are used for different Logical Channel Numbers (LCNs), it is important that the application has a means of determining which port will transmit the next packet. This can be determined using two global variables and a pointer:

rec_port This is a variable which Indicates which port provided the last received data packet:

0	Port A
1	Port B

tpat_ptr This counter is used by the function ***mlh_trans*** to determine which port will be used to transmit the next packet. This counter is the offset into the pattern table pointed to by ***pat_loc***.

pat_loc — This is a pointer to the pattern table used by mlh_trans() to determine how transmitted packets are to be distributed to Ports A and B. Reading the character located at *(pat_loc + tpat_ptr) provides a means of determining which port will transmit the next packet. The possible values of the character are:

1	Port A
2	Port B
0	End of pattern table

The following program fragment demonstrates the use of these variables:

```
char next_port;

next_port = *(pat_loc + tpat_ptr);

if (!next_port) /*if end of pattern flag */
next_port = *patloc; /* use start of pattern */

switch(next_port){
      case 1: /* port A is next */
            tx_lcgn = lcgn_a;
            tx_lcn = lcn_a;
            tx_ps = ps_a++;
            tx_pr = pr_a;
            break;

      case 2: /* port B is next */
            tx_lcgn = lcgn_b;
            tx_lcn = lcn_b;
            tx_ps = ps_b++;
            tx_pr = pr_b;
            break;

      default: oh_oh_error(BAD_PATRN);
            }
```

The packet is then assembled using the tx_ components and transmitted using mlh_trans().

Reception

When the Front End Processor (FEP) for either port receives a frame, the simulator automatically generates a response frame, if applicable. If an information frame is received, the packet is passed to a FIFO (first in-first out) message buffer and is given to the program by calling one of the following receive functions:

receive — This function causes the Chameleon to check the reception buffer of the specified port.

mlh_receive — This function checks both ports for received packets and returns one packet on each call, if one is available. It remembers which port provided the last packet, and gives preference to the other port on the next call. If the preferred port has no packet available, the other port is then queried.

For example, assume three packets were received on Port A and one packet on Port B. If the current preferred port is A, successive calls to mlh_receive will return packets in this order:

Port A	packet 1
Port B	packet 1
Port A	packet 2
Port A	packet 3

After Port A packet 2 is returned, Port B is queried, and since it has no packet available, Port A packet 3 is returned.

With either receive function, the following occurs:

- The packet is transferred from the message reception buffer to a user defined buffer
- The global variable **rxlen** is set to the length of the received packet
- If mlh_receive was called to receive the packet, it sets the global variable **rec_port** to indicate from which port the packet was received

In applications where packets are transmitted in redundant mode by the remote device, receive() should be used. This simplifies the process of comparing a packet from Port A with the same packet from Port B.

In applications where packets are being distributed over both physical links by the remote unit, using mlh_receive() makes it easier to quickly generate response packets and send them on the correct port.
For applications where the Chameleon-32 is being used as a rate adapter, DTE/DTE or DCE/DCE translator, or an interface translator (eg. V.35 to BRI), using receive() provides a quick turnaround, as shown in the following program fragment:

```
while(1) {                    /* endless loop, there should be some
                              exit code included such as break if the
                              'q' is typed or if the state of one
                              link goes to 0. */
receive (PORT_A,*buffer);              /* receive on port A */
if (rxlen)                             /* if packet received */
transmit (PORT_B,*buffer,rxlen);                  /* resend it on
                                                  port B */
receive(PORT_B,*buffer);               /* now receive port B */
if (rxlen)                             /* if packet received */
transmit (PORT_A,*buffer,rxlen)        ;          /* resend it on
                                                  port A */
}
```

Set Functions

There are two types of functions which enable you to change protocol parameters:

mlh_set_*xxx* — These functions set the parameter for the specified port. For example, the mlh_set_t1 sets the value of the T1 timer for the specified port.

set_xxx — These functions set the parameter to an identical value for both ports. For example, set_t1 sets the value of the T1 timer to the same value for both ports.

flush

Declaration flush ()

Description This function clears the receive buffer of the currently selected port.

The set_port function enables you to select a port. (See Appendix B.1 for a description of set_port.)

Returns None

See Also set_port(), flush_all(), mlh_flush()

flush_all

Declaration flush_all()

Description This function clears the reception buffer of both ports.

Returns None

See Also flush(), mlh_flush()

init_a

Declaration

```
int init_a (interface, sta, encode, bitrt)
int   interface, sta, encode;
long  bitrt;
```

Range

interface	0	DCE
	1	DTE
	2	ISDN
sta	0	Network
	1	Subscriber
encode	0	NRZ
	1	NRZI
bitrt	50 to 64000 bps	

The bit rate selection is used to optimize some functions, so it should be set to the correct value even when the Chameleon is simulating a DTE or when an ISDN interface is being used. If you are unsure of the appropriate bit rate, use 50.

Description

This function initializes Port A according to the passed parameters. Port B must then be initialized using the init_b function. If both Ports A and B are to be initialized using identical parameters, use the **initp1** function.

Returns

0	Operation successfuly completed
-1	Parameter error

Also see the common error codes in Appendix B.1.

See Also

initp1(), init_b()

init_b

Declaration

```
int init_b (interface, sta, encode, bitrt)
int   interface, sta, encode;
long  bitrt;
```

Range

interface	0	DCE
	1	DTE
	2	ISDN
sta	0	Network
	1	Subscriber
encode	0	NRZ
	1	NRZI
bitrt	50 to 64000 bps	

The bit rate selection is used to optimize some functions, so it should be set to the correct value even when the Chameleon is simulating a DTE or when an ISDN interface is being used. If you are unsure of the appropriate bit rate, use 50.

Description

This function initializes Port B according to the passed parameters. Port A must then be initialized using the init_a function. If both Ports A and B are to be initialized using identical parameters, use the initp1 function.

Returns

0	Operation successfuly completed
-1	Parameter error

Also see the common error codes in Appendix B.1.

See Also

initp1(), init_a()

initp1

Declaration

```
int initp1 (interface, sta, encode, bitrt)
int   interface, sta, encode;
long  bitrt;
```

Range

interface	0	DCE
	1	DTE
	2	ISDN
sta	0	Network
	1	Subscriber
encode	0	NRZ
	1	NRZI
bitrt	50 to 64000 bps	

The bit rate selection is used to optimize some functions, so it should be set to the correct value even when the Chameleon is simulating a DTE or when an ISDN interface is being used. If you are unsure of the appropriate bit rate, use 50.

Description

This function initializes both Ports A and B according to the passed parameters. If the two ports must be initialized with different parameters, use init_a and init_b to set each port independently.

Returns

0	Operation successfuly completed
-1	Parameter error

Also see the common error codes in Appendix B.1.

See Also

init_a(), init_b()

mlh_flush

Declaration

```
mlh_flush (port)
int port;
```

Range

port	0	Port A
	1	Port B

Description

This function clears the receive buffer of the specified port.

Returns

None

See Also

flush(), flush_all()

mlh_receive

Declaration

```
int mlh_receive (loc)
char *loc;
```

Range

loc A pointer to the user defined buffer.

Description

This function causes the Chameleon to check for a received packet. An internal flag is maintained to indicate which port supplied the last packet. This flag gives priority to the other port on the next call so that one port does not dominate the system during heavy traffic. If the currently preferred port does not have a received packet to pass, the port that received the last packet is then queried.

For example, if a packet is received from Port A, the next call checks the Port B reception buffer. If Port B does not have a packet, it then queries Port A.

When a packet is detected in the reception buffer:

- The packet is transferred from the message reception buffer to a user defined buffer pointed to by *loc.
- The global variable **rxlen** is set to the length of the received packet.
- It sets the global variable **rec_port** to indicate from which port the packet was received, as follows:

  0 No packet was received
  1 Packet received from Port A
  2 Packet received from Port B

Returns

0	No packet in the reception buffer
2	FEP not initialized
128	Packet received

Also see the common error codes in Appendix B.1.

See Also

receive()

mlh_set_n1

Declaration

```
int mlh_set_n1 (port,val)
int port,val;
```

Range

port
0 Port A
1 Port B

val N1 value in the range 1 to 512.

Description

This function sets the N1 value for the specified port.

The set_n1 function sets both ports to the identical N1 value.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

set_n1()

mlh_set_n2

Declaration

```
int mlh_set_n2 (port,val)
int port,val;
```

Range

port	0	Port A
	1	Port B
val	N2 value in the range 1 to 512.	

Description

This function sets the N2 value for the specified port.

The set_n2 function sets both ports to the identical N2 value.

Returns

0	Successful
-1	Parameter error

Also see the common error codes in Appendix B.1.

See Also

set_n2()

mlh_set_net

Declaration

```
int mlh_set_net (port)
int port,val;
```

Range

port	0	Port A
	1	Port B

Description

This function sets the specified port to act as a network.

The set_net function configures both ports to act as networks.

Returns

0 Successful

Also see the common error codes in Appendix B.1.

See Also

set_net()

mlh_set_t1

Declaration

```
int mlh_set_t1 (port,val)
int port,val;
```

Range

port
0 Port A
1 Port B

val T1 value in the range 1 to 255 seconds.

Description

This function sets the value of the T1 timer for the specified port.

The set_t1 function sets both ports to the identical T1 value.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

set_t1()

mlh_set_t2

Declaration

```
int mlh_set_t2 (port,val)
int port,val;
```

Range

port	0	Port A
	1	Port B
val	T2 value in the range 1 to 255 seconds.	

Description

This function sets the value of the T2 timer for the specified port.

The set_t2 function sets both ports to the identical T2 value.

Returns

0	Successful
-1	Parameter error

Also see the common error codes in Appendix B.1.

See Also

set_t2()

mlh_slof

Declaration

```
int mlh_slof (port)
int port;
```

Range

port	0	Port A
	1	Port B

Description

This function disconnects the link on the specified port by sending a DISC frame.

The slof function disconnects the link on both ports.

Returns

See the common error codes in Appendix B.1.

See Also

slof()

mlh_slon

Declaration

```
int mlh_slon (port)
int port;
```

Range

port	0	Port A
	1	Port B

Description

This function attempts to establish a link on the specified port by sending a SABM.

The slon function attempts to establish links on both ports.

Returns

See the common error codes in Appendix B.1.

See Also

slon()

mlh_status

Declaration

```
int mlh_status (port)
int port;
```

Range

port	0	Port A
	1	Port B

Description

This function returns the link status of the specified port.

Returns

0	Disconnected
1	Link connection requested
2	Frame reject state
3	Link disconnection requested
4	Information transfer state
5	Local station busy
6	Remote station busy
7	Local and remote stations busy

Also see the common error codes in Appendix B.1.

See Also

status()

mlh_set_sub

Declaration

```
int mlh_set_sub (port)
int port;
```

Range

port	0	Port A
	1	Port B

Description

This function sets the specified port to act as a subscriber.

The set_sub function sets both ports to act as subscribers.

Returns

0 Successful

Also see the common error codes in Appendix B.1.

See Also

set_sub()

mlh_set_window

Declaration

```
int mlh_set_window (port,val)
int   port,val;
```

Range

port
0 Port A
1 Port B

val Window size in the range 1 to 7 frames

Description

This function sets the window size (maximum number of outstanding unacknowledged frames) for the specified port.

The set_window function sets the window size of both ports to the identical value.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

set_window()

mlh_trans

Declaration

```
int mlh_trans (xloc,xlen)
char *xloc;
int  xlen;
```

Range

xloc Pointer to the user defined packet to be transmitted

xlen Length of the packet to be transmitted

Description

This function transmits a data packet on Port A or B as determined by the distribution pattern set by a call to the ***set_pat*** or the ***set_ratio*** function.

mlh_trans() should be used to transmit information packets only. Other packet types should be sent using the transmit() function, which has a passed argument selecting the port. The only time it is safe to send non-info packets using mlh_trans() is when the distribution pattern is specified as redundant (set_ratio(-1) is selected).

Returns

0 Successful

Also see the common error codes in Appendix B.1.

See Also

transmit(), set_pat(), set_ratio()

receive

Declaration

```
int receive (port,loc)
char *loc;
int  port;
```

Range

port
0 Receive from Port A
1 Receive from Port B

loc A pointer to the user defined destination buffer.

Description

This function checks the reception buffer of the specified port for a received packet. If a packet has been received:

- The packet is transferred from the message reception buffer to a user defined buffer
- The global variable **rxlen** is set to the length of the received packet

The mlh_receive function is another receive function. It alternates between Ports A and B when called. Calling receive() does not affect the alternating pattern of mlh_receive.

Returns

0	No packet in the reception buffer
2	FEP not initialized
128	Packet received

Also see the common error codes in Appendix B.1.

See Also

mlh_receive()

set_n1

Declaration

```
int set_n1 (val)
int val;
```

Range

val N1 value in the range 1 to 512

Description

This function sets the N1 value for both ports.

The mlh_set_n1 function sets the N1 value of a specified port.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

mlh_set_n1()

set_n2

Declaration

```
int set_n2 (val)
int val;
```

Range

val N2 value in the range 1 to 512

Description

This function sets the N2 value for both ports.

The mlh_set_n2 function sets the N2 value of a specified port.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

mlh_set_n2()

set_net

Declaration int set_net

Description This function configures both ports to act as networks.

The mlh_set_net function configures the specified port to act as a network.

Returns 0 Successful

Also see the common error codes in Appendix B.1.

See Also mlh_set_net()

set_pat

Declaration

```
int set_pat (pat_ptr)
char *pat_ptr;
```

Range

pat_ptr A pointer to a user defined table which specifies the distribution pattern the for mlh_trans() transmit function

Description

This function enables you to specify a user defined distribution pattern for transmitting packets using the mlh_trans function.

The distribution pattern is defined in a table which contains the following values:

0	End of table
1	Send on Port A
2	Send on Port B

The table may be of any length, but it must be a character (byte) oriented table which contains at least one port selection code and terminates with an end of table code (0).

With each packet transmitted, mlh_trans increments a pointer into the pattern table. When the pointer lands on a zero entry in the table, or a new pattern selection is made, the pointer is reset.

The set_ratio function provides an alternate means of defining a distribution pattern.

Returns

See the common error codes in Appendix B.1.

See Also

mlh_trans(), set_ratio(), transmit()

set_ratio

Declaration

```
int set_ratio (pct_a)
int pct_a;
```

Range

pct_a The percentage of packets to be transmitted over Port A. Valid values are 0 to 100 in increments of 10, and -1.

- -1 All packets are transmitted over both Ports A and B. This is also referred to as redundant mode.
- 0 0% of the packets are transmitted over Port A. (All packets are transmitted over Port B.)
- 10 10% of the packets are transmitted over Port A. 90% are transmitted over Port B.
- 20 20% on Port A, 80% on Port B.
- ⋮
- 90 90% on Port A, 10% on Port B.
- 100 All packets sent on Port A

Description

This function selects a distribution pattern for transmitting packets using the mlh_trans function. It specifies the percentage of packets to be transmitted over Port A.

The set_pat function provides an alternate means of defining a distribution pattern.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

mlh_trans(), set_pat()

set_sub

Declaration int set_sub ()

Description This function configures both ports to act as subscribers.

The mlh_set_sub function configures the specified port to act as a subscriber.

Returns 0 Successful

Also see the common error codes in Appendix B.1.

See Also mlh_set_sub()

set_t1

Declaration

```
int set_t1 (val)
int val;
```

Range

val T1 timeout value in the range 1 to 255 seconds.

Description

This function sets the T1 timer to an identical value for both ports.

The mlh_set_t1 function sets the value of the T1 timer of the specified port.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

mlh_set_t1()

set_t2

Declaration

```
int set_t2 (val)
int val;
```

Range

val T2 timeout value in the range 1 to 255 seconds.

Description

This function sets the T2 timer to an identical value for both ports.

The mlh_set_t2 function sets the value of the T2 timer of the specified port.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

mlh_set_t2()

set_window

Declaration

```
int set_window (val)
int val;
```

Range

val Window size in the range 1 to 7 frames.

Description

This function sets the window size (maximum number of outstanding unacknowledged frames) to an identical value for both ports.

The mlh_set_window function sets the window value of the specified port.

Returns

0 Successful
-1 Parameter error

Also see the common error codes in Appendix B.1.

See Also

mlh_set_window()

slof

Declaration int slof ()

Description This function disconnects the link on both ports by sending a DISC frame.

The mlh_slof function disconnects the link on a specified port.

Returns See the common error codes in Appendix B.1.

See Also mlh_slof()

slon

Declaration int slon ()

Description This function attempts to establish a link on both ports by sending a SABM.

The mlh_slon function attempts to establish a link on a specified port.

Returns See the common error codes in Appendix B.1.

See Also mlh_slon()

status

Declaration int status ()

Description This function returns the link status of the currently selected port.

Use the set_port function to select a port (Appendix B.1).

Returns

0	Disconnected
1	Link connection requested
2	Frame reject state
3	Link disconnection requested
4	Information transfer state
5	Local station busy
6	Remote station busy
7	Local and remote stations busy

Also see the common error codes in Appendix B.1.

See Also set_port(), mlh_status()

transmit

Declaration

```
int transmit (port,xloc,xlen)
char   port,*xloc;
int    xlen;
```

Range

port 0 Port A
1 Port B

xloc Pointer to the user defined packet to be transmitted

xlen Length of the packet to be transmitted

Description

This function transmits a packet over the specified port.

The mlh_trans function is an alternate function which transmits packets over Ports A and B as determined by a specified pattern. The transmit function does not affect the mlh_trans pattern.

Returns

0 Successful

Also see the common error codes in Appendix B.1.

See Also

mlh_trans()

Sample Program 1

This program starts port to port simulation on a Dual Port Chameleon. To use the program, connect Port A to Port B. The program initializes the ports and runs continuous transmit/receive on alternate ports until stopped by the user or the link goes down. It counts and displays the content and number of frames received on each port.

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>

char tmsg[48],rmsg[48];

main()
{
char answer,insit,*tadr,*radr;
extern char *malloc();
extern int rxlen;
int result,tempi;
long int count,countb;

printf("C program started.\n");
tadr=&tmsg[0];
radr=&rmsg[0];
s_start(0);
s_start(1);

/***** make and display transmit message *****/

for(insit=0;insit<=16;insit++)
      tmsg[insit] = insit;

printf("message = ");
for(insit=0;insit<=16;insit++)
printf(" %0x ",tmsg[insit]);
printf("\n");

/***** do frame level setup *****/

       result=set_n1(0x50);
printf("set n1 result= %d\n",result);
       result=set_t1(10);
       result=set_n2(4);
       result=set_window(3);

      mlh_set_net(0);   /* Port A is network  */
      mlh_set_sub(1);   /* Port B is subscriber  */
```

```
printf("Type Q to exit\n");

/****start link from port B, wait for port A state 4****/
      mlh_slon(1);
      while(mlh_status(0)<4);

/***** send and receive untill port A link state is less than 4
or user types 'Q' key   *****/

      count=countb=0;

for(;;){
      if(status()<4) break;

   answer = getch(_stdvt);
   if (toupper(answer)=='Q')
               break;

      result=trans(1,0x80,tadr,10);
      result=receive(0,radr);
            if(rxlen>0)
                  {

            printf("port A received message = ");
            for(insit=0;insit<rxlen;insit++)
                  printf(" %0x ",rmsg[insit]);
            printf("\n\n");
            count++;
                  }
      result=trans(0,0x80,tadr,10);
      result=receive(1,radr);
            if(rxlen>0)
                  {

            printf("port B received message = ");
            for(insit=0;insit<rxlen;insit++)
                  printf(" %0x ",rmsg[insit]);

                  }
  if(mlh_status(1)==4)
      result=trans(1,0x80,tadr,15);
      countb++;

      printf("\033[%d;%df",11,15);      /* xyplot */
      printf("\033[31mThat's %ld on A,  %ld on b.\n",count,countb);
      }

for(count=0;count<0xfff;count++);

/***** if link still connected, disconnect it, then exit *****/
```

```
        if(status()>0)
        {
        result=slof();
        while(status()>0);
        }
}

s_start(port)
int port;
{
int rets,ret2,waiter;
long int rate;

rate=64000;
printf("Trying to start P1");
if(!port)
rets= init_a(0,0,0,rate);
else
rets= init_b(1,1,0,rate);

printf(" result = %0d\n",rets);
for(ret2=0;ret2<0xffff;ret2++)
        for(rets=0;rets<0xffff;rets++)
        for(waiter=0;waiter<0xffff;waiter++);

}
```

Sample Program 2

This program starts port to port simulation. It runs continuous transmit/receive on both ports using mlh_trans and set_ratio until stopped by the user or the link goes down. It counts and displays the content, number of frames, and ratio of frames received on each port. Note that the displayed ratio is only aproximate, as it is calculated using integers.

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>
#include <math.h>

char tmsg[48],rmsg[48];

main()
{
char insit,answer,*tadr,*radr;
extern char *malloc(),rec_port,pat_tbl,*pat_loc,tpat_ptr;
extern int rxlen;
int ab_ratio,result,tempi;
long int count,countb;
char *place;

tadr=&tmsg[0];
radr=&rmsg[0];
s_start(0);
s_start(1);

/***** make and display transmit message *****/

for(insit=0;insit<=16;insit++)
            tmsg[insit] = insit;

printf("message = ");
for(insit=0;insit<=16;insit++)
printf(" %0x ",tmsg[insit]);
printf("\n");

/***** do frame level setup *****/

       result=set_n1(0x50);
printf("set n1 result= %d ",result);
       result=set_t1(10);
printf("set t1 result= %d ",result);
       result=set_n2(4);
printf("set n2 result= %d ",result);
```

```
        result=set_window(3);
printf("set window result= %d\n",result);

      mlh_set_net(0);
printf("set net port A result= %d ",result);
      mlh_set_sub(1);
printf("set sub port B result= %d\n",result);
      ab_ratio=-1;
      set_ratio(ab_ratio);
printf("set ratio result= %d\n",result);

printf("Press space bar to change port ratio, Q to exit\n");

/****start link from port B, wait for both ports state 4 ****/

      mlh_slon(1);
      while(mlh_status(0)<4);
      while(mlh_status(1)<4);

/***** send and receive untill port A link state is less than 4 *****/

      count=countb=0;

for(;;){
      if(status()<4) break;
      result=mlh_trans(tadr,12);
      result=mlh_receive(radr);
            if(rxlen>0)
            {
            if(rec_port==1)
                  {
            printf("port A received message = ");
            for(insit=0;insit<rxlen;insit++)
                  printf(" %0x ",rmsg[insit]);
            printf("\n\n");
if((count>0)&&(countb>0))
{
printf("Ratio of frames on ports A:B = ");

count>countb? printf("%ld:1     \n",count/countb):printf("1:%ld
\n",countb/count);
}

            count++;
                  }

            if(rec_port==2)
                  {
            printf("\nport B received message = ");
            for(insit=0;insit<rxlen;insit++)
                  printf(" %0x ",rmsg[insit]);
```

```
                printf("\n");
                countb++;
                      }

          }

       answer = getch(_stdvt);

       if (toupper(answer)=='Q')
          break;

      if(answer==' ')
{
if(ab_ratio==-1)
          ab_ratio=0;
else
          ab_ratio+=10;

if(ab_ratio>100)
          ab_ratio=-1;

count=countb=0;
flush_all();
          printf("\nset ratio result %d  \n",set_ratio(ab_ratio));

printf("\nRatio selection = %d\n",ab_ratio);
}

          printf("\033[%d;%df",11,13);
          printf("\033[31mThat's %ld on A,  %ld on B.
          \n",count,countb);
          }

/***** if link still connected, disconnect it, then exit *****/

          if(status()>0)
          {
          result=slof();
          while(status()>0);
          }
}

s_start(port)
int port;
{
int rets,ret2,waiter;
long int rate;

rate=64000;
printf("Trying to start P1");
if(!port)
```

```
rets= init_a(0,0,0,rate);
else
rets= init_b(1,1,0,rate);

printf(" result = %0d\n",rets);
for(ret2=0;ret2<0xffff;ret2++)
      for(rets=0;rets<0xffff;rets++)
      for(waiter=0;waiter<0xffff;waiter++);
}
```

Sample Program 3

This program demonstrates the use of many of the Multi-Link HDLC library functions. To run the program as a Chameleon self-test, connect Port A to Port B on a Dual Port Chameleon.

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>
#include <math.h>

char tmsg[48],rmsg[48],answer;

main()
{
char insit,*tadr,*radr;
extern int rxlen;
int result;
long int count,countb;

tadr=&tmsg[0];
radr=&rmsg[0];
s_start(0);
s_start(1);

/***** make and display transmit message *****/

for(insit=0;insit<=16;insit++)
            tmsg[insit] = insit;

printf("message = ");
for(insit=0;insit<=16;insit++)
printf(" %0x ",tmsg[insit]);
printf("\n");

/***** do frame level setup *****/

       result=set_n1(0x50);
printf("set n1 result= %d ",result);
       result=set_t1(10);
printf("set t1 result= %d ",result);
       result=set_n2(4);
printf("set n2 result= %d ",result);
       result=set_window(3);
printf("set window result= %d\n",result);

      result=mlh_set_net(0);
printf("set net port A result= %d ",result);
```

```
      result=mlh_set_sub(1);
printf("set sub port B result= %d\n",result);

/***** run test until finished or user quits *****/
while (1)
{
re_paint();
if (toupper(answer)=='Q')
break;

printf("\033[40m\033[%d;%df",11,13);/* xyplot */
printf("\033[35mmlh_slon/mlh_slof test. Next test: flush_all");

printf("\033[%d;%df",13,13);/* xyplot */
printf("\033[35mResult of mlh_slon/mlh_slof test: \n");

while(1)
      {
      mlh_slon(0);
      while(mlh_status(0)==1);
      if (mlh_status(0)==0)
            {
            no_good();
            break;
            }
      mlh_slof(0);
      settimer(2,2);
      while (timer(2));
      if (mlh_status(0)!=0)
            {
            no_good();
            break;
            }
      mlh_slon(1);
      while(mlh_status(1)==1);
      if (mlh_status(1)==0)
            {
            no_good();
            break;
            }
      mlh_slof(1);
      settimer(2,2);
      while (timer(2));
      if (mlh_status(1)!=0)
            {
            no_good();
            break;
            }
      test_passed();
      break;
      }
```

```
re_paint();

printf("\033[%d;%df",11,13);/* xyplot */

printf("\033[40m\033[35mflush_all test Next test: mlh_flush(port)");

printf("\033[%d;%df",13,13);/* xyplot */
printf("\033[35mResult of flush_all test: \n");

set_ratio(-1); /*so 1 trans call will send on both ports*/
mlh_slon(1);
while(mlh_status(1)==1);
        result=mlh_trans(tadr,12);
flush_all();
        result=mlh_receive(radr);
          if(rxlen>0) /*s/b no frame on either port*/
       no_good();
  else
       test_passed();

re_paint();

printf("\033[%d;%df",11,13);/* xyplot */
printf("\033[40m\033[35mmlh_flush(port) Last test");

printf("\033[%d;%df",13,13);/* xyplot */
printf("\033[35mResult of mlh_flush(port) test: \n");


while(1)
       {
result=mlh_trans(tadr,12);/* send frames,both ports*/
       mlh_flush(0); /*flush port A*/
result=receive(0,radr);
       if(rxlen) /* fail if port A not flushed */
             {
             no_good();
             printf("port A failed to flush");
             break;
             }

result=receive(1,radr);
       if(!rxlen) /* fail if port B is flushed */
             {
             no_good();
             printf("port B flushed with A");
             break;
             }

result=mlh_trans(tadr,12);
```

```
      mlh_flush(1); /*flush port B*/
result=receive(1,radr);
      if(rxlen) /* fail if port B not flushed */
            {
            no_good();
            printf("port B failed to flush");
            break;
            }


result=receive(0,radr);
      if(!rxlen) /* fail if port A is flushed */
            {
            no_good();
            printf("port A flushed with port B");
            break;
            }
      test_passed();
      break;
      }

printf("\033[%d;%df",9,13);/* xyplot */

printf("\033[40m\033[34mPress Q to exit, space to restart  \n");
answer=0;
while ((answer!=' ')&&(toupper(answer)!='Q'))
      answer = getch(_stdvt);
      if (toupper(answer)=='Q')
      break;
printf("\033[0J");

}

/***** if link still connected, disconnect it, then exit *****/

      if(status()>0)
      {
      result=slof();
      while(status()>0);
      }
}

re_paint()
{
printf("\033[%d;%df",9,13);/* xyplot */

printf("\033[40m\033[34mPress space bar to run test          \n");
answer=0;
while (answer!=' ')
      {
      answer = getch(_stdvt);
```

```
        }
printf("\033[0J");
}
no_good()
{

printf("\033[%d;%df",14,13);/* xyplot */
printf("\033[30m\033[41mFAILED ");
}

test_passed()
{
printf("\033[%d;%df",14,13);/* xyplot */
printf("\033[30m\033[42mPASSED ");
}

s_start(port)
int port;
{
int rets,ret2,waiter;
long int rate;

rate=64000;
printf("Trying to start P1");
if(!port)
rets= init_a(0,0,0,rate);
else
rets= init_b(1,1,0,rate);

printf(" result = %0d\n",rets);
for(ret2=0;ret2<0xffff;ret2++)
      for(rets=0;rets<0xffff;rets++)
      for(waiter=0;waiter<0xffff;waiter++);
}
```

B.14 U–INTERFACE LIBRARY

Introduction

The U–interface library function enables you to use the Chameleon 32 U–interface hardware (the U–board) in the C environment. The library is called **libu.a**, and is in the **\lib** directory. The function call described below interfaces the U–interface library from C programs.

Two sample programs are given beginning on page B.14–14.

SetU

Declaration

```
int SetU (cmdblock, resblock);
char cmdblock[ ];
char resblock[ ];
```

int SetU is the function call for accessing the U–board from the C Shell.

Description

This function has two parameters blocks that are character (char) arrays. The size depends upon the requested function.

The *cmdblock* (command block) parameter contains the input parameters needed. The first item, *cmdblock [0],* is a character specifying the requested function. The remaining items are function–specific parameters.

The *resblock* (result block) parameter contains the results of the operation requested in the *cmdblock.* The first item, *resblock[0],* is a character indicating whether or not the command was completed successfully. The remaining items are function–specific results.

The commands listed on the next page are available as the first item in *cmdblock.* The commands are different for U–boards 0 and 1 to distinguish between the two boards installed in your Chameleon. However, if you have only one U–board in your Chameleon, you must use the commands that correspond to that board.

Board 0	Board 1	Command
0	100	Initialize Interface
1	101	Configure
2	102	Set Transceiver State
3	103	Get Transceiver State
4	104	Set Transceiver Activation
5	105	Get Transceiver Connection
6	106	Set Transceiver Errors
7	107	Get Transceiver Errors
8	108	Get HW Version
9	109	Get Link Status
11	111	Transceiver Transmit
12	112	Transceiver Receive
13	113	EOC Processing
14	114	EOC Mode Control
15	115	M4 Mode Control
16	116	M5/6 Mode Control
30	130	Shutdown Interface

The commands are described on the following pages.

resblock[0] Error Codes

The error codes for *resblock[0]* are the same for all U–Interface Library commands.

Code	Meaning
00	Successful
01	Invalid Command
02	Invalid Command Parameters
03	Requested board is not responding
04	U–board physical error
05	U–board interface is not initialized
10	Requested board is not installed

cmdblock[0] = 0
cmdblock[0] =100
Initialize

This command is used to initialize the C Library interface. This includes actions such as allocating memory, getting unit identifiers, enabling reception, etc. This command should be the first command issued when using the U Interface.

The response parameter for this function is *resblock[0]*. See **Error Codes**.

cmdblock[0] = 1
cmdblock[0] = 101
Configure

This command is used to set up the U board for monitoring or simulating the M channel and the 2B+D channels. It also specifies the routing of the 2B+D–channel data.

Configuration for the M channel:

cmdblock[1]	Mode of operation	
	00000000	Customer Equipment (NT) Simulation.
	00000001	Network Equipment (LT) Simulation.
	00000010	Monitoring

Configuration for the 2B+D channels:

cmdblock[2]	Mode of operation	
	00000000	Customer Equipment (NT) Simulation.
	00000001	Network Equipment (LT) Simulation.
	00000010	Monitoring
cmdblock[3]	Codec encode	
	00000000	μ–Law
	00000001	A–Law
cmdblock[4]	Clocking (Valid only in LT simulation)	
	00000000	External
	00000001	Internal
	00000010	NT–Recovered
cmdblock[5]	B1–Channel Routing	
	00000000	None
	00000001	Port A
	00000010	Port B
	00000011	Idle pattern
	00000100	X1
	00000101	X2
	00000110	Codec Handset

00000111 Codec 600–Ohm.

cmdblock[6] B2–Channel Routing
00000000 None
00000001 Port A
00000010 Port B
00000011 Idle pattern
00000100 X1
00000101 X2
00000110 Codec Handset
00000111 Codec 600–Ohm.

cmdblock[7] D–Channel Routing
00000000 None
00000001 Port A
00000010 Port B
00000011 Idle pattern
00000100 X1
00000101 X2

cmdblock[8] B1–Channel Idle Pattern

cmdblock[9] B2–Channel Idle Pattern

cmdblock[10] D–Channel Idle Pattern

The response parameter for this function is *resblock[0]*. See **Error Codes.**

cmdblock[0] = 2
cmdblock[0] = 102
Set Transceiver State

This command is used to set up the state of the specified transceiver.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] Transceiver state
- 1 Reset
- 2 Power down
- 3 Absolute
- 4 Normal

The response parameter for this function is *resblock[0].* See **Error Codes.**

cmdblock[0] = 3
cmdblock[0] = 103
Get Transceiver State

This command is used to set up the state of the specified transceiver.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

The response parameters for this function are:

resblock[0] See Error Codes

resblock[1] Transceiver state
- 0 Reset
- 1 Power down
- 2 Absolute
- 3 Normal

cmdblock[0] = 4
cmdblock[0] = 104
Set Transceiver Activation

This command is used to start transceiver activation or deactivation.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] Transceiver activation
- 1 Start activation
- 2 Start deactivation

The response parameter for this function is *resblock[0]*. See **Error Codes**.

cmdblock[0] = 5
cmdblock[0] = 105
Get Transceiver Connection

This command is used to get the connection status of the specified transceiver.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

The response parameters for this function are:

resblock[0] See **Error Codes**

resblock[1] Transceiver connection
- 0 None
- 1 Port A
- 2 Port B
- 3 Ports A and B

cmdblock[0] = 6
cmdblock[0] = 106
Set Transceiver Errors

This command is used to reset the error counters of the specified transceiver.

cmdblock[1] Transceiver specifier
0 NT Xcvr
1 LT Xcvr

The response parameter for this function is *resblock[0]*. See **Error Codes**.

cmdblock[0] = 7
cmdblock[0] = 107
Get Transceiver Errors

This command is used to retrieve the 32–bit error counters of the specified transceiver.

cmdblock[1] Transceiver specifier
0 NT Xcvr
1 LT Xcvr

The response parameters for this function are:

resblock[0]	See **Error Codes**
resblock[1–4]	32–bit FEBE count. MSBs followed by LSBs.
resblock[5–8]	32–bit NEBE count. MSBs followed by LSBs.
resblock[9–12]	32–bit NoSyn count. MSBs followed by LSBs.
resblock[13–16]	32–bit NoAct count. MSBs followed by LSBs.

cmdblock[0] = 8
cmdblock[0] = 108
Get HW Version

This command is used to get the version numbers of the transceiver pair.

The response parameters for this function are:

resblock[0]	See **Error Codes**
resblock[1]	NT transceiver version number.
resblock[2]	LT transceiver version number.

cmdblock[0] = 9
cmdblock[0] = 109
Get Link Status

This command is used to get the link status of the transceiver pair.

The response parameters for this function are:

resblock[0]	See **Error Codes**
resblock[1]	NT Link Status bit0 link up bit1 superframe sync recognized bit2 transceiver activation in progress bit3 error indicator
resblock[2]	LT Link Status bit0 link up bit1 superframe sync recognized bit2 transceiver activation in progress bit3 error indicator

There is no function cmdblock [0] = 10, 110

cmdblock[0] = 11
cmdblock[0] = 111
Transceiver Transmit

This command is used to transmit data on the M channel. Four types of information are allowed: EOC, M4, M5 and M6.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] Channel specifier
- 1 EOC
- 2 M4
- 3 M5/M6

EOC Message

cmdblock[3] EOC address, EOC DM bit.

cmdblock[4] EOC Information

M4 Message

cmdblock[5] M4 Information

M5/6 Messages

cmdblock[6] M5/6 Information

The response parameter for this function is *resblock[0]*. See **Error Codes**.

cmdblock[0] = 12
cmdblock[0] = 112
Transceiver Receive

This command is used to receive the M–Channel information for a superframe, including two EOC messages and one M4 and M5/M6 message each. Table B.6.2–1 illustrates the coding/decoding format of the M–Channel superframe and how this corresponds to response parameters of this function.

cmdblock[1] Transceiver specifier
0 NT Xcvr
1 LT Xcvr

The response parameters for this function are:

resblock[0] See **Error Codes**

resblock[1] Message Length
0 – No data available
1 – 6 data bytes follow

resblock[2] EOC address, EOC DM bit (byte 0 in the table).

resblock[3] EOC Information (byte 1 in the table).

resblock[4] EOC address, EOC DM bit (byte 2 in the table).

resblock[5] EOC Information (byte 3 in the table).

resblock[6] M4 Information (byte 4 in the table).

resblock[7] M5/6 Information (byte 5 in the table).

BYTE# \ BIT#	7	6	5	4	3	2	1	0
0 (resblock[2])	x*	x	x	x	a3	a2	a1	dm
1 (resblock[3])	i8	i7	i6	i5	i4	i3	i2	i1
2 (resblock[4])	x	x	x	x	a3	a2	a1	dm
3 (resblock[5])	i8	i7	i6	i5	i4	i3	i2	i1
4 (LT→NT) [6]	ACT	DEA	RSV*	RSV	RSV	RSV	UOA	AIB
4 (NT→LT) [6]	ACT	PS1	PS2	NTM	CSO	RSV	SAI	RSV
5 (resblock[7])	M50	M60	M51	FEBE	x	x	x	x

Notes: * = spare bit
RSV = reserved bit

Table B.6.2–1: M–Channel Superframe Byte/Bit Format.

cmdblock[0] = 13
cmdblock[0] = 113
EOC Processing

This command is used to configure the U board for automatic EOc processing.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] Automatic Processing Mode
- 0 No action
- 1 Operate 2B+D Loopback
- 2 Operate B1 Loopback
- 3 Operate B2 Loopback
- 4 Send Corrupted CRC
- 5 Return to Normal

The response parameter for this function is *resblock[0].* See **Error Codes.**

cmdblock[0] = 14
cmdblock[0] = 114
EOC Mode Control

This command is used to configure the U board for EOC reception.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] EOC Reception Mode
- 0 No action
- 1 Handle every EOC
- 2 Handle EOC passing trinal checks
- 3 Handle EOC passing trinal checks with automatic EOC processing

The response parameter for this function is *resblock[0].* See **Error Codes.**

cmdblock[0] = 15
cmdblock[0] = 115
M4 Mode Control

This command is used to configure the U board for M4 reception.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] M4 Reception Mode
- 0 No action
- 1 Handle Dual–Consecutive M4 with Verified act/dea.
- 2 Handle Dual–Consecutive M4
- 3 Handle Delta M4
- 4 Handle every M4

The response parameter for this function is *resblock[0].* See **Error Codes.**

cmdblock[0] = 16
cmdblock[0] = 116
M5/6 Mode Control

This command is used to configure the U board for M5/6 reception.

cmdblock[1] Transceiver specifier
- 0 NT Xcvr
- 1 LT Xcvr

cmdblock[2] M5/6 Reception Mode
- 0 No action
- 1 Handle Dual–Consecutive M5/6
- 3 Handle Delta M5/6
- 4 Handle every M5/6

The response parameter for this function is *resblock[0].* See **Error Codes.**

cmdblock[0] = 30
cmdblock[0] = 130
Shutdown

This command is used to shut the C–Library interface down. This includes freeing memory, disabling reception capability, etc. When done with the U interface, this command should be the last one issued.

The response parameter for this function is *resblock[0]*. See **Error Codes**.

Sample Programs

There are two sample U–Interface programs on the C Sample Program Disk:

- **testlt.c**
- **testnt.c**

TESTLT.C

This program verifies activation of the LT U transceiver and reception of EOC messages: i.e., that there is data flowing over the M channel.

```
#include <stdio.h>
#include <mtosux.h>

#define          NT              0
#define          LT              1
#define          LINKUP          1
#define          ERROR           8
#define          ERR_NOSTAT      6

unsigned char Command[20],Response[20];
long     febe,nebe;

/*
 * init_U
 * Initializes and configs U-Board for NT simulation
 *
 */
init_U()
{
        Command[0] = 0;             /* Initialize U Library */
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Init = %x\n",Response[0]);


        Command[0] = 1;             /* Config command */
        Command[1] = 0;             /* NT simulation */
        Command[2] = 0;             /* NT simulation */
        Command[3] = 0;             /* U-law */
        Command[4] = 1;             /* Internal Clocking */
        Command[5] = 0;             /* No B1 routing */
        Command[6] = 0;             /* No B2 routing */
        Command[7] = 0;             /* No D routing */
        Command[8] = 0;             /* Idle patterns */
        Command[9] = 0;
        Command[10] = 0;
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Config = %x\n",Response[0]);

}
```

```
/*
 * reset_U
 * Shuts down U-Board
 *
 */
reset_U()
{
        Command[0] = 30;  /* shut down U Library */
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Shut Down = %x\n",Response[0]);
}

/*
 * main
 * Entry point
 *
 */
main()
{

        /*
        * Initialize for NT simulation
         */
        init_U();

        /*
         * Update receive EOC for trinal checks only
         */
        Command[0] = 14;
        Command[1] = NT;
        Command[2] = 2;
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:EOC Processing = %x\n",Response[0]);


        /*
         * Issue Activation Command
         */
        printf("Activating...");
        fflush(stdout);
        Command[0] = 4;
        Command[1] = NT;
        Command[2] = 1;
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Activate = %x\n",Response[0]);

        /*
         * Wait 'till Link is up or Error
         */
        Command[0] = 9;
        do {
                SetU(Command,Response);
                if (Response[0]==ERR_NOSTAT)
```

```
                continue;
        if (Response[0]) {
                printf("ERROR:Link Status = %x\n",Response[0]);
                printf("Abort\n");
                reset_U();
                exit(0);
        };
} while (!(Response[1]&(LINKUP|ERROR)));


/*
 * Failure
 */
if (Response[1]&ERROR) {
        printf("Failure\n");
        reset_U();
        exit(0);
}

printf("Successful\n");

pause (SEC+1);

/*
 * Flush activation EOCs
 */
Command[0] = 12;
do {
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Receive = %x",Response[0]);
        if (Response[1])
                printf("Rx'edEOC[%x%x]\n",Response[2],Response[3]);
} while (Response[1]!=0);

/*
 * Issue Deactivation Command
 */
printf("Deactivating\n");
Command[0] = 4;
Command[1] = NT;
Command[2] = 2;
SetU(Command,Response);
if (Response[0])
        printf("ERROR:Deactivate = %x",Response[0]);

/*
 * Shut down U Interface and boogie
 */
reset_U();

exit(0);

}
```

TESNT.C This program verifies activation of the NT U transceiver and reception of EOC messages: i.e., that there is data flowing over the M channel.

```
#include <stdio.h>
#include "mtosux.h"
#define          NT              0
#define          LT              1
#define          LINKUP          1
#define          ERROR           8
#define          ERR_NOSTAT      6

unsigned char Command[20],Response[20];

/*
 * init_U
 * Initializes and configs U-Board for LT simulation
 *
 */
init_U()
{
        Command[0] = 0;             /* Initialize U Library */
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Init = %x\n",Response[0]);


        Command[0] = 1;             /* Config command */
        Command[1] = 1;             /* LT simulation */
        Command[2] = 1;             /* LT simulation */
        Command[3] = 0;             /* U-law */
        Command[4] = 1;             /* Internal Clocking */
        Command[5] = 0;             /* No B1 routing */
        Command[6] = 0;             /* No B2 routing */
        Command[7] = 0;             /* No D routing */
        Command[8] = 0;             /* Idle patterns */
        Command[9] = 0;
        Command[10] = 0;
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Config = %x\n",Response[0]);

}

/*
 * reset_U
 * Shuts down U-Board
 *
 */
reset_U()
{
        Command[0] = 30;  /* shut down U Library */
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Shut Down = %x\n",Response[0]);
}
```

```
/*
 * main
 * Entry point
 *
 */
main()
{

        /*
        * Initialize for LT simulation
         */
        init_U();

        /*
         * Update receive EOC for trinal checks only
         */
        Command[0] = 14;
        Command[1] = LT;
        Command[2] = 2;
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:EOC Processing = %x\n",Response[0]);

        /*
         * Issue Activation Command
         */
        printf("Activating...");
        fflush(stdout);
        Command[0] = 4;
        Command[1] = LT;
        Command[2] = 1;
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Activate = %x\n",Response[0]);

        /*
         * Wait 'till Link is up or Error
         */
        Command[0] = 9;
        do {
                SetU(Command,Response);
                if (Response[0]==ERR_NOSTAT)
                        continue;
                if (Response[0]) {
                        printf("ERROR:Link Status = %x\n",Response[0]);
                        printf("Abort\n");
                        reset_U();
                        exit(0);
                }
        } while (!(Response[2]&(LINKUP|ERROR)));


        /*
         * Failure
         */
        if (Response[2]&ERROR) {
```

```
        printf("Failure\n");
        reset_U();
        exit(0);
}

printf("Successful\n");

/*
* Send a 2B+D Loopback EOC
 */
printf("Transmitting EOC [150]\n");
Command[0] = 11;
Command[1] = LT;
Command[2] = 1;
Command[3] = 1;             /* EOC address,dm */
Command[4] = 0x50;          /* EOC info */
SetU(Command,Response);
if (Response[0])
        printf("ERROR:Transmit = %x\n",Response[0]);

/*
* Send a Return to Normal EOC
 */
printf("Transmitting EOC [1FF]\n");
Command[0] = 11;
Command[1] = LT;
Command[2] = 1;
Command[3] = 1;             /* EOC address,dm */
Command[4] = 0xFF;          /* EOC info */
SetU(Command,Response);
if (Response[0])
        printf("ERROR:Transmit = %x\n",Response[0]);

pause(SEC+1);

/*
 * Flush the received EOCs (two of which are 150 and 1FF)
 */
Command[0] = 12;
do {
        SetU(Command,Response);
        if (Response[0]) {
                printf("ERROR:Receive = %x",Response[0]);
                break;
        }
        if (Response[1])
                printf("Rx'edEOC[%x%x]\n",Response[2],Response[3]);
} while (Response[1]!=0);

/*
 * Issue Deactivation Command
 */
printf("Deactivating\n");
Command[0] = 4;
Command[1] = LT;
Command[2] = 2;
```

```
        SetU(Command,Response);
        if (Response[0])
                printf("ERROR:Deactivate = %x",Response[0]);

        /*
         * Shut down U Interface
         */
        reset_U();

        exit(0);

}
```

B.15 ETSI LIBRARY

The ETSI library is an optional C library which must be purchased in addition to the Chameleon 32 'C' Development System. This library supports a total of 64 logical links. The library is named **libetsi.a** and resides in the a:\lib directory of the hard disk.

Link Selection

The ETSI library includes functions which enable you to control the use of 64 logical links. Each of the 64 links is referred to by a unique link number in the range 0 – 63. Each of the 64 logical links has its own SAPI and LIC values, which are assigned as follows:

When you select a link using *set_link*, you can then use the other functions to set the link on (*slon*), set the link off (*slof*), and transmit and receive messages.

1. Use *set_link* to select one of the 64 links (0 – 63). All links default to state 9, disabled.
2. Use *set_sapi* to assign the link a SAPI value.
3. Use *set_lic1, 2* and *3* to assign the link a LIC value.

Frame Status Word

A two–byte, frame status (frstat) word is attached to the beginning of each received message. This field provides the following information:

- Frame type
- Number of link which received the frame
- Command or response frame
- Poll/Final bit value

The *get_rxstat* function returns the low-order byte of *frstat*. The *get_rlink* function returns the high-order byte of *frstat*.

Addressing

This library allows variable-length frame addressing, The range, conforming to CCITT Q.921 standards, is two (2) to four (4) octets. You can freely mix addressing modes in the 64 available logical links.

To select 2-octet addressing,
set LIC2 lto any invalid value (>127)

To select 3-octet addressing,
set LIC3 to any invalid value

To select 4-octet addressing,
set all LICs to *valid* values.

If two or more links have the same address (SAPI/LIC combination), received frames will be considered to belong to the highest link number matching that address.

A link is disabled by selecting the link and setting the SAPI or LIC1 to an invalid value. You should ensure that the link is in the disconnected state before you disable it. If a link is disabled while in a connected (multi–frame) state, the device under test will see it as a Layer 1 failure.

Setting the SAPI and/or LIC1 value to an invalid value sets the link to the disabled state (9). This provides an easy means of testing lost-link recovery and ignoring unused links.

The Frame Address format may be modified by the values in LIC2 and LIC3. If an invalid value (>127) is set in LIC2, only the SAPI and LIC1 will be used. If LIC3 is set to an invalid value and LIC2 is valid, SAPI, LIC1 and LIC2 will be used.

There are two functions which get the state of a link: *status()* gets the state of the selected link; *link_stat* gets the state of any specified link.

get_freelink returns the number of the lowest-numbered, disabled link. *find_link* returns the number of the lowest link matching a specified SAPI/LIC combination.

SIMAN

This library functions with Simultaneous Analysis (SIMAN) only if your Chameleon is equipped with a P5 board of revision level K33 or higher. If your Chameleon has an older P5 board (rev. K32 or lower), this feature is automatically disabled. You can obtain an upgrade kit from Tekelec which – when installed – will enable these older machines to run SIMAN with this library.

Functions

The following functions are in the ETSI library. Also refer to the common functions and error codes described in Appendix B.1. Programming tips and examples are provided beginning on page B.15–51.

find_link()

Declaration

```
int find_link(sapi, lic1, lic2, lic3)
int sapi, tei, tgi;
```

sapi SAPI value of the link, in the range 0 – 63 (SAPI > 63 is invalid, resulting in *don't care* value)

lic LICs 1, 2 and 3 values of each link, in the range 0 – 127 (LIC*n*> 127 is invalid, resulting in *don't care* value)

Description

This function returns the number of the lowest link matching the specified SAPI/LIC values. An invalid SAPI or LIC value is treated as a *don't care* for that parameter and returns the first link matching the valid parameters.

Returns

0 – 63	Matching link number
–1	No match found

get_freelink()

Declaration int get_freelink()

Description This function gets the link number (0 – 63) of the first disabled link.

Returns

0 – 63	Disabled link number
–1	No free links available
–2	initp1 not performed

get_fwaiting

Declaration

```
int get_fwaiting (lnkn)
char lnkn;
```

Range

lnkn	0 – 63

Description

This function gets the number of I–frames waiting to be transmitted on link *lnkn.*

Returns

0 – 7	Number of I–frames waiting to be sent by link lnkn

See also the global error codes on page B.1–1.

get_lic1()

Declaration int get_lic1()

Description This function gets the LIC1 of the link currently under user control. Any value greater than 127 disables the link.

Returns

0 – 127	LIC1 for current link number
128 – 255	LIC1 value that disables link.

Also see global error codes on page B.1–1.

get_lic2()

Declaration int get_lic2()

Description This function returns the LIC2 value of the link currently under user control.

Returns

0 – 127	LIC2 for current link number
128 – 255	LIC2 value that disables use of LIC2 on the link (the link is not disabled)

Also see global error codes on page B.1–1.

get_lic3()

Declaration int get_lic3()

Description This function returns the LIC3 value of the link currently under user control.

Returns

0 – 127	LIC3 for current link number
128 – 255	LIC3 value disabling use of LIC3 on the link (the link is not disabled)

Also see global error codes on page B.1–1.

get_link()

Declaration int get_link()

Description This function gets the number of the link which is currently under user control.

Returns

0 – 63	Current link number
–1	initp1 not performed

get_lnksapi

Declaration

```
int get_lnksapi (lnkn)
char lnkn;
```

Range

lnkn	0 – 63

Description

This function gets the SAPI value for link lnkn.

Returns

0 – 63	SAPI value assigned to link lnkn
> 63	SAPI value disabling link lnkn

See also the global error codes on page B.1–1.

get_lnklic1

Declaration

```
int get_lnklic1(lnkn)
char lnkn;
```

Range

lnkn	0 – 63

Description

This function gets the LIC1 value for link lnkn.

Returns

0 – 127	LIC1 value assigned to link lnkn
> 127	LIC1 value disabling link lnkn

See also the global error codes on page B.1–1.

get_lnklic2

Declaration

```
int get_lnklic2(lnkn)
char lnkn;
```

Range

lnkn	0 – 63

Description

This function gets the LIC2 value for link lnkn. If an invalid value (>127) is set in LIC2, only the SAPI and LIC1 (i.e., 2-byte addressing) will be used.

Returns

0 – 127	valid values
>127	invalid values.

See also the global error codes on page B.1–1.

get_lnklic3

Declaration

```
int get_lnklic3(lnkn)
char lnkn;
```

Range

lnkn	0 – 63

Description

This function gets the LIC3 value for link lnkn. If LIC3 is set to an invalid value and LIC2 is valid, SAPI, LIC1 and LIC2 (i.e., 3-byte addressing) will be used.

Returns

0 – 127	valid values
>127	invalid values

See also the global error codes on page B.1–1.

get_meswaiting

Declaration int get_meswaiting ()

Description This function gets the number of messages waiting to be received from the Front End Processor (FEP).

Note This function returns the number of messages buffered by the FEP. The library buffers one additional message.

Returns

0 – 32	Number of messages waiting to be received from the FEP

See also the global error codes on page B.1–1.

get_rlink()

Declaration int get_rlink()

Description This function gets the number of the link which sent the last received message. This is the high order byte of the frame status word **frstat** passed by the FEP.

Returns

0 – 63	Current link number
–1	No messages received yet
–2	initp1 not performed

get_rxstat()

Declaration char get_rxstat()

Description This function gets the low order byte of the frame status word **frstat**, which contains the frame type, C/R bit and P/F bit of the last received message.

Returns

0 – 0xC3	frstat value (interpreted as shown below)
0xFF	No messages received yet
0xFE	initp1 not performed

Examples

0x41	Non–final XID response
0x02	I–frame command
0xC3	Final FRMR response

get_sapi()

Declaration int get_sapi()

Description This function gets the SAPI value of the link currently under user control.

Returns

0 – 255	SAPI for current link

Also see global error codes on page B.1–1.

get_sconfig ()

Declaration int get_sconfig ()

Description This function returns a copy of the current control configuration byte, which can be interpreted as shown in the figure below.

get_sim ()

Declaration int get_sim ()

Description This function returns a copy of of the network/subscriber selection.

Returns

0	Network
1	Subscriber

get_window

Declaration

```
int get_window (lnkn)
char lnkn;
```

Range

lnkn	0 – 63

Description

This function gets the number of outstanding I–frames on link number **lnkn.**

Returns

0 – 7	Number of unacknowledged I–frames of link lnkn

See also the global error codes on page B.1–1.

initp1

Declaration

```
int initp1 (interface, sta, encode, bitrt)
int interface, sta, encode;
long  bitrt;
```

Description

initp1 loads the Front End Processor (FEP) code for the library and starts simulation. Predefined values exist in mlklib.h to aid in setting up the call to this function. **sta** is the station type and selects the initial sense of the command/response bit. The library permits reselection of the station type at any time. **encode** selects the physical data encoding. **bitrt** sets the data rate when simulating a DCE device.

Note

This function is identical to and interchangeable with the *start_sim* function. It has been included in the ETSI library for downward compatibility with the single link LAPD library.

Ranges

interface	0	V–type interface (DCE)
	1	V–type interface (DTE)
	2	ISDN interface
sta	0	NETWORK
1	SUBSCRIBER	
encode	0	NRZ
	1	NRZI
bitrt	Any long integer value from 50 – 64000.	

Returns

See the global error codes on page B.1–1.

link_stat

Declaration

```
int link_stat(n)
char n;
```

Range

n	0 – 63

Description This function gets the current state of link n.

Returns

0 – 9	Current state of link (see table below)

See also the global error codes on page B.1–1.

receive

Declaration

```
int receive(dest_addr)
char *dest_addr;
```

Description

This function receives a message from the FEP by performing the following tasks:

- It polls the FEP to see if any received messages are available
- It transfers the message contents to the user defined buffer pointed to by **dest_addr**
- The total length of the message (including the frame status bytes *frstat*) is placed in the global variable **rxlen**

  The frstat word is accessible by calling **get_rlink** and **get_rxstat** so that you can interpret and respond to a message quickly. The frstat bytes are attached to the beginning of each received message so that several messages may be received, sorted, interpreted, and individual responses made.

It is up to the user to ensure that the destination buffer is long enough to contain the message. Generally, a length equal to N201 + 2 is adequate.

s_n200

Declaration

```
int s_n200 (val)
int val;
```

Range

val	1 – 255

Description This function sets the maximum number of retries (N200).

Returns

0	Successful

See also global error codes on page B.1–1.

s_n201

Declaration

```
int s_n201 (val)
int val;
```

Range

val	1 – 512

Description This function sets the maximum length for an I–frame (N201).

Returns

0	Successful

See also global error codes on page B.1–1.

s_t200

Declaration

```
int s_t200 (val)
int val;
```

Range

val	0 – 255

Description

This function sets the time allowed for the remote station to respond (T200). Setting this value to 0 disables the T200 timer.

Returns

0	Successful

See also global error codes on page B.1–1.

s_t203

Declaration

```
int s_t203 (val)
int val;
```

Range

val	0 – 255

Description

This function sets the maximum time between frames (T203). On time out, a polled RR or XID command is transmitted, depending on the configuration selection. Setting this value to 0 disables the T203 timer.

Returns

0	Successful

See also global error codes on page B.1–1.

set_lic1

Declaration

```
int set_lic1(value)
char value;
```

Range

value — The LIC1 value to use for the link, as follows:

0 – 127	Valid LIC1 values
128 – 255	Invalid LIC1 value, which causes the link to be disabled

Description

This function sets the LIC1 value for the link under user control. LIC1 is a value assigned to, and may be associated with, a single link and a given point–to–point data link connection. At any time, a given terminal endpoint (TE) may contain one or more LICs.

Normal values are:

0 –127	Valid
128 – 255	Disable link

Returns

0	Successful
–1	Parameter out of range
–2	initp1 not performed
–3	Timeout

set_lic2

Declaration

```
int set_lic2(value)
char value;
```

Range

value	The LIC2 value to use for the link, as follows:	
	0 – 127	Valid LIC2 values
	128 – 255	Disables the use of the LIC2 and LIC3 bytes

Description

This function sets the LIC2 value for the link under user control. If used, the LIC2 is the third byte of the LAPD Address field. Any value greater than 127 disables both LIC2 and LIC3.

Returns

0	Successful
–1	Parameter out of range
–2	initp1 not performed
–3	Timeout

set_lic3

Declaration

```
int set_lic3(value)
char value;
```

Range

value — The LIC3 value to use for the link, as follows:

0 – 127 Valid LIC3 values

128 – 255 Disables the use of the LIC3 byte

Description

This function sets the LIC3 value for the link under user control. If used, the LIC3 is the fourth byte of the LAPD Address field. Any value greater than 127 disables LIC3.

Returns

0	Successful
–1	Parameter out of range
–2	initp1 not performed
–3	Timeout

set_link

Declaration

```
int set_link(n)
char n;
```

Range

n	0 – 63

Description

This function puts link n under user control. Only one link at a time can be under user control.

Returns

0	Successful
–1	Parameter out of range
–2	initp1 not performed
–3	Timeout

set_net ()

Declaration int set_net ()

Description This function sets the simulation side to NETWORK. The Chameleon can simulate either a network or subscriber device.

When the Chameleon 32 emulates a network, it sends commands with the C/R bit set to one, and responds with the C/R bit set to zero. It sends the selected SAPI and LIC with the C/R bit automatically set in accordance with CCITT Q. 921.

set_sapi

Declaration

```
int set_sapi(v)
char v;
```

Range

Accepted range of *v* is 0 – 255. A value over 63 disables the selected link.

Description

This function sets the SAPI value for the link under user control. The SAPI (Service Access Point Identifier) indicates the layer two service type requested or supported. Normal values are:

0	Call Control procedures
16	Packet communication procedures
63	Management procedures
64 – 255	Disable link

Returns

0	Successful
–1	Parameter out of range
–2	initp1 not performed
–3	Timeout

set_sconfig

Declaration

```
int set_sconfig (byte)
int byte;
```

Description

This function sets the value of the control configuration byte, interpreted as shown in the figure below.

Returns

0	Successful

See also global error codes on page B.1–1.

set_sub ()

Declaration int set_sub ()

Description This function sets the simulation side to SUBSCRIBER. The Chameleon can simulate either a network or subscriber device.

When the Chameleon 32 emulates a LAPD subscriber, it sends commands with the C/R bit set to zero, and responds with the C/R bit set to one. It sends the selected SAPI and LIC with the C/R bit automatically set in accordance with CCITT Q. 921.

set_tei

Declaration

```
int set_tei(value)
char value;
```

Range

value — The TEI value to use for the link, as follows:

0 – 127	Valid TEI values
128 – 255	Invalid TEI value, which causes the link to be disabled

Description

This function sets the TEI value for the link under user control. TEI is a value assigned to, and may be associated with, a single link and a given point–to–point data link connection. At any time, a given terminal endpoint (TE) may contain one or more TEIs.

This function is provided for the convenience of those users whose applications use the CCITT Q.921 2-field address (SAPI TEI). Its use is completely interchangeable with *set_lic1()*.

Normal values are:

0 –127	Valid
128 – 255	Disable link

Returns

0	Successful
–1	Parameter out of range
–2	initp1 not performed
–3	Timeout

set_window

Declaration

```
int set_window (val)
int val;
```

Range

val	1 – 7

Description

This function sets the maximum number of outstanding frames on each link.

Note

The total of outstanding frames + the number of frames passed to the FEP waiting to be transmitted + the number of messages over 16 bytes long waiting to be received from the FEP may not exceed 80.

Returns

0	Successful

See also global error codes on page B.1–1.

setflg

Declaration

```
int setflg (flag)
int flag;
```

Range

flag	1	0x7E fill
	0	0xFF fill

Description This function selects an interframe fill pattern.

Returns

0	Successful

See also global error codes on page B.1–1.

slof ()

Declaration int slof ()

Description This function sends a DISC and waits for a UA frame. This is equivalent to the CCITT primitive **DL RELEASE.**

Returns

0	Successful

Also see global error codes on page B.1–1.

slon ()

Declaration int slon ()

Description This function sends a SABME and waits for a UA frame. This is equivalent to the CCITT primitive **DL ESTABLISH**.

Returns

0	Successful

Also see global error codes on page B.1–1.

start_sim

Declaration

```
int start_sim (interface, sta, encode, bitrt)
int interface, sta, encode;
long  bitrt;
```

Description

start_sim loads the Front End Processor (FEP) code for the library and starts simulation. Predefined values exist in mlklib.h to aid in setting up the call to this function. **sta** is the station type and selects the initial sense of the command/response bit. The library permits reselection of the station type at any time. **encode** selects the physical data encoding. **bitrt** sets the data rate when simulating a DCE device.

Note

This function is identical to and interchangeable with the *initp1* function. initp1 is included for downward compatibility with the single link LAPD library.

Ranges

interface	0	V–type interface (DCE)
	1	V–type interface (DTE)
	2	ISDN interface
sta	0	NETWORK
	1	SUBSCRIBER
encode	0	NRZ
	1	NRZI
bitrt		Any long integer value from 50 – 64000.

Returns

See the global error codes on page B.1–1.

status()

Declaration int status()

Description This function gets the current state of link under user control.

Returns 0 – 9 Current state of link (see table below)

See also the global error codes on page B.1–1.

trans

Declaration

```
int trans (frame,address,len)
int frame, len;
char *address;
```

Description

This function transmits a frame, as follows:

frame selects type of frame to transmit:

0x80	I–frame	Sequenced (numbered) I–frame
0x81	UI	Unnumbered I–frame (NSI)
0x82	XIDC	XID command frame
0x83	XIDR	XID response frame

address is a pointer to the first byte of the message to be transmitted.

len is the actual length of the message to be transmitted. There are two restrictions on the message length:

- I–frames should not exceed the value set in N201 (maximum length of an I–frame)
- The total length of the frame cannot exceed 512 bytes.

Returns

0 Successful

Also see global error codes on page B.1–1.

transmit

Declaration

```
int transmit (xloc, xlen)
char *xloc;
int xlen;
```

Description

This function transmits a message in a sequenced (numbered) I–frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. There are two restrictions on the message length:

- I–frames should not exceed the value set in N201 (maximum length of an I–frame)
- The total length of the frame cannot exceed 512 bytes.

Note

The *transmit* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0	Successful

Also see global error codes on page B.1–1.

trui

Declaration

```
int trui (xloc, xlen)
char  *xloc;
int  xlen;
```

Description

This function transmits a message in an unnumbered I–frame (UI frame).

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trui* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0	Successful

Also see global error codes on page B.1–1.

trxcni

Declaration int trxcni ()

Description This function transmits an XID command frame with no data field.

Returns

0	Successful

See also global error codes on page B.1–1.

trxidc

Declaration

```
int trxidc (xloc, xlen)
char *xloc;
int  xlen;
```

Description

This function transmits a message in an XID command frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trxidc* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0	Successful

Also see global error codes on page B.1–1.

trxidr

Declaration

```
int trxidr (xloc, xlen)
char *xloc;
int xlen;
```

Description

Transmit a message in an XID response frame.

xloc is a pointer to the first byte of the message to be transmitted.

xlen is the actual length of the message to be transmitted. The total length of the frame must not exceed 512 bytes.

Note

The *trxidc* function is provided for user convenience. If extremely high data rates are required, the *trans* function should be used, as it is somewhat faster.

Returns

0	Successful

Also see global error codes on page B.1–1.

trxrni

Declaration int trxrni ()

Description This function transmits an XID response frame with no data field.

Returns

0	Successful

See also global error codes on page B.1–1.

PROGRAMMING NOTES AND EXAMPLES

This section provides general information about using the ETSI library.

General Notes

In your program, specify the Chameleon port being used by a call to the *set_port* function. This is not absolutely necessary when using a Single-Port Chameleon (Ch20), but should be done to make your application portable.

A call to initp1 must be made to start the Front End Processor (FEP). This loads the FEP operating code and starts the simulation. The Chameleon is then ready to begin testing.

Before frames can be transmitted or received, at least one link must be enabled. To do this:

a. Use *set_link* to select a link (default is 0)

b. Set the SAPI and LIC1 to valid values by calling *set_sapi* and *set_lic*

c. LIC2 and LIC3 must be set to a valid value if they are to be used.

Interpreting Received Messages

To interpret a received message, you must know the SAPI value and the frame type of the received message. This information is available to you from the frame status bytes, and can be accessed using the technique shown below:

```
receive(pointr);
if(rxlen = = 0)          /*exit if no message received*/
        return(0);
frtype=(get_rxsta()&3);      /*get the frame type status bits*/
lnk=get_rllnk();             /*get number of the link sending message*/
sapval=get_lnksap(lnk);          /*get the SAPI for that link*/
```

In this example, the frame type may be interpreted from *frtype* as follows:

0 = UI frame
1 = XID frame
2 = I–frame
3 = FRMR

sapval is the SAPI value assigned to the link on which the message was received.

Optimizing Transmit Speed

To optimize speed for applications such as load generators, follows these guidelines:

- The trans function is faster than the other transmit functions such as *transmit*, *trui*, *trxidr*, and *trxidc*.
- If you are not concerned with the contents of the received messages, use the following technique to keep the receive buffer empty. This is faster than a call to the *receive* function in the transmit loop.

```
if(get_meswtg()>=6)
    flush();
```

- Minimize input, output, and screen print operations in all tasks. Since the processor that runs the C shell also manages many of the I/O tasks, this will result in your simulation program running faster.
- Run your program in background mode. This is done by adding an ampersand (&) at the end of the file name when starting the test from the C shell. For example:

```
test&
```

will run the program **test** in background mode. This causes the program to run at a higher priority and frees the C shell for other uses. This technique also reduces the number of windows available for other tasks, since a separate window is opened for each program running in background mode.

Transmitting Responses

In a test environment, the actual information content of a given message type is often fixed. In such cases, only the message type must be known in order to select the proper response. To simplify responding to message, predefine the content of responses in a message array.

In the following program fragment, the following is assumed:

- A set of responses and pointers to the responses has been defined earlier
- SAPI/LIC combinations have been set up

The program fragment uses a defined value TYPEOS, which is the offset to the byte in the message containing the message type. The call to fix_cref extracts the call reference value from the received message and copies it into the selected response message.

```
respond()
{
        char mestyp,*resp;
        int resp_len;
        rxlen=0; /* prepare to loop until message received */
        while(!rxlen)
                receive(&rxmes[0]);

        if((get_rxstat()&3)=2) /*only respond to Iframes*/
        {
        mestyp=rxmes[TYPEOS]; /*get message type from rcvd message*/
        switch(mestyp)
                {
                case  CALL_SU:
                /*if msg is call setup*/
                                /*respond setup ack*/
                        resp=&su_ack;
                        resp_len=SU_ACK_LEN;
                        break;
        •
        •
        •
                case RELEASE:
                /* if msg is release*/
                                /*respond release complete*/
                        resp=&rel_cmplt;
                        resp_len=REL_CMPLT_LEN;
                }
        fix_cref(resp); /*set response call ref value*/
        set_link(get_rlink()); /*select link to send response*/
        transmit(resp,resp_len); /* send response*/
        }

fix_cref(dest)
        char *dest;
{
        int ref_val;
        refval=rxmes[CREF_OS];
        *(dest+CREF_OS-2)=refval; /*-2 to allow for frstat bytes*/
```

Demonstration Programs

You can run the two test programs for ETSI LAPD library port-to-port on a dual-port CH32. You can also run them between Chameleons – CH32–to–CH32. However, in this case you may have to change the port selection in the function *sim_start()* at the end of the program.

To compile and link these programs:

```
cc –o etsia etsia.c –letsi
cc –o etsib etsib.c –letsi
```

Start the 'etsia' program first.

ETSIA.C

etsia.c is one of two sample programs to demonstrate the use of the ETSI lapd library functions.

The Program:

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>
#include <mtosux.h>

char tmsg[48],rmsg[48];

main()
{
char insit,*tadr,*radr,answer;
extern char *malloc();
extern int rxlen;
int result,tempi;

long int count;

printf("C program started.\n");
tadr=&tmsg[0];
radr=&rmsg[0];
sim_start();

/***** make and display transmit message *****/
for(insit=0;insit<=16;insit++)
        tmsg[insit] = insit;

printf("message = ");
for(insit=0;insit<=16;insit++)
        printf(" %0x ",tmsg[insit]);
        printf("\n");

/***** do frame level setup *****/
        result = setflg(0);
        result = set_mod(1);
        result = s_t200(0);
        result = s_t203(0);
        result = s_n201(260);
        result = s_n200(26);
        result = set_window(3);

/***** setup sapi and LICs for 64 links and set them on *****/
for(;;)
{
        tempi = get_freelink();
        if(tempi<0)
                break;
        result = set_link(tempi);
        result = set_sapi(tempi);
        result = set_lic1(tempi);
        result = set_lic2(tempi);
        result = set_lic3(tempi);
        pause(5+512); /* so other tasks can use processor */

}
```

```
                while(status() <4 )
                        pause(5+512); /* so other tasks can use processor */

/***** send and receive until 'q' is entered  *****/
                result = set_link(2);
                count=0;
                result = get_link();
        printf("Current link is %d \n",result);
printf("\033[2J");
printf(" type Q to quit\n");
                 for(;;){
        answer=getch(_stdvt);
                if(answer == 'q') break;
                result = receive(radr);
                        if(rxlen>0)
                                {

                printf("\033[10;7f received message = ");
                for(insit=0;insit<rxlen;insit++)
                        printf(" %0x ",rmsg[insit]);
                printf("\n\n");
        printf("Current link is %d \n",get_rlink());

        result = set_link(get_rlink());
          if(status() ==4 )

                result = trans(0x80,tadr,15);
                count++;
                printf("\033[%d;%df",14,36);
                printf("\033[31mThat's %ld. \n",count);
                                }
                pause(517); /* so other tasks can use processor */
                }
for(count=0;count<0xfff;count++);

/***** if link still connected, disconnect it, then exit *****/
                if(status()>0)
                {
                result = slof();
                }

        }

sim_start()
        {
        int rets,ret2;
        long int rate;

        rate=64000;
        printf("Trying to start P1");
        rets= initp1(0,0,0,rate);
        printf(" result = %0x\n",rets);
        }
```

ETSIB.C

etsib.c is one of two sample programs to demonstrate the use of the ETSI lapd library functions.

The Program:

```
#include <stdio.h>
#include <cham.h>
#include <ctype.h>
#include <fcntl.h>
#include <init.h>
#include <video.h>
char tmsg[48],rmsg[48];

main()
        {
        char tempc,insit,user_in,*tadr,*radr;
        extern int rxlen;
        int result,tempi;
        long int count;
        printf("C program started.\n");
        tadr=&tmsg[0];
        radr=&rmsg[0];
        sim_start();

/***** make and display transmit message *****/
        for(insit=0;insit<=16;insit++)
                tmsg[insit] = 16-insit;
        printf("message = ");
        for(insit=0;insit<=16;insit++)
        printf(" %0x ",tmsg[insit]);
        printf("\n");

/***** do frame level setup *****/
                result=set_mod(1);
                result=s_t200(0);
                result=s_t203(0);
                result=s_n201(20);
                result=set_window(3);

/***** setup sapi and LICs for 64 links and set them on *****/
        for(;;)
        {
                tempi=get_freelink();
                if(tempi<0)
                        break;
        printf("free link = %d\n",tempi);
                result = set_link(tempi);
                result = set_sapi(tempi);
                result = set_lic1(tempi);
                result = set_lic2(tempi);
                result = set_lic3(tempi);
                result = slon();
                while(status() < 4)
                        pause(513); /* so other tasks can use processor */
        }
```

```
/***** send and receive until user types 'Q' *****/
                result=set_link(2);
                count=0;
                result=get_link();
        printf("Current link is %d \n",result);
                result=trans(0x80,tadr,15);

printf("\033[2J");
printf(" type Q to quit, S to restart link, T to transmit Iframe\n");

                for(;;){
                user_in=toupper(getch(_stdvt));
                if(user_in=='T')
                        result=trans(0x80,tadr,15);
                if(user_in=='Q')
                        break;
                if(user_in == 'S')
                        slon();
                result=receive(radr);
                        if(rxlen>0)
                                {
                        printf("\033[10;8f received message = ");
                        for(insit=0;insit<rxlen;insit++)
                                printf(" %0x ",rmsg[insit]);
                        printf("\n\n");
                printf("Current link is %d \n",get_rlink());

                tempc=get_rlink()==63? 0:get_rlink()+1;
                result=set_link(tempc);
                 if(status()==4)
                        result=trans(0x80,tadr,15);
                        count++;
                        printf("\033[%d;%df",14,36);
                        printf("\033[36m That's %ld.\n",count);
                                        }
                pause(513);  /* so other tasks can use processor */
                }
for(count=0;count<0xfff;count++);
/***** if link still connected, disconnect it, then exit *****/
                if(status()>0)
                {
                result=slof();
                printf("slof result %0d\n",result);
                }
        }

sim_start()
        {
        int rets,ret2;
        long int rate;
        rets=setport(PORTB);
        rate=64000;
        printf("Trying to start P1");
        rets= initp1(1,1,0,rate);
        printf(" result = %0x\n",rets);
        }
```

INDEX

A

B

INDEX

D

E

F

G

H

I

J

K

L

M

O

P

T

U

V

INDEX

W

X