TI-99/4A HOME COMPUTER
HEXBUS INTERFACE PERIPHERAL
SOFTWARE FUNCTIONAL SPEC
Version 1.0

Texas Instruments
Personal Computer Division
Date October 18, 1983

TABLE OF CONTENTS

1.0 Introduction

This document is intended to serve as a software functional specification for the HEXBUS Interface Peripheral for the 99/4 and 99/4A Home Computers. All references by this document to the 99/4A will in actually be references to both the 99/4 and the 99/4A unless otherwise noted. Since the Device Service Routine (DSR) is the only software needed to interface the Home Computer with this peripheral, this document is, therefore, also the Functional Spec for the HEXBUS Interface DSR.

1.1 Design Guidelines

The HEXBUS peripherals were originally designed for the Compact Computer Model CC-40. The purpose of the HEXBUS Interface is to adapt these peripherals to the 99/4A and other Home Computers.

There are differences between the 99/4A and the CC-40 File Management Systems. For a 99/4A user, it is impossible to use the CC-40 Peripherals exactly the way he uses the 99/4A peripherals. Several considerations are implemented to make the differences between the two File Management Systems as narrow as possible. These considerations are listed below in the order of design priority.

1. The syntax of Basic Open Commands follow the 99/4A convention as closely as possible. The DSR translates the 99/4A Peripheral Access Block (PAB) into the CC-40 PAB format.

2. CC-40 PAB features that the 99/4A does not have, are implemented, if possible, through switch options in the Open Command.

3. Interrupts can be accessed through GPL and Assembler only.

4. The HEXBUS RS232 Device (not to be confused with the RS232 Interface Card or the RS232 Interface Box used in Home Computer) is accessible through the device name "RS232" or "RS232/1", in addition to the normal name "HEXBUS.20". This will enable TE II to use the HEXBUS RS232 Device without any modification.

5. Function 4 (Shift C) is used to terminate DSR execution provided the Device Code is from 20 to 27, 50 to 57 (RS232 Serial Port or Centronix-type Parallel Port), or 70 to 77 (Hexbus Modem).

6. Slave Mode is supported minimally.

7. Basic users are able to send and receive raw data that form a meaningful HEXBUS message frame. This option will be very useful, if future HEXBUS peripherals contain functions not easily accessible from 99/4A Basic.

1.2 Applicable Documents

1. 99/4 File Management Spec
2. 99/4 Disk Peripheral Software Functional Spec
3. 99/4 RS232C Peripheral Software Functional Spec
4. GPL Interface Specification for the 99/4 Disk Peripheral
5. HEXBUS Peripheral Bus Spec
6. HEXBUS RS232/PO Software Functional Spec
7. HEXBUS Printer Software Functional Spec
8. HEXBUS Wafertape Software Functional Spec

2.0 HEXBUS Protocol Overview

The HEXBUS consists of eight wires: four data lines, two handshaking lines, one ground, and one reserved for future use. All devices are connected to the bus in a parallel manner. Data is transmitted and received in a bit-parallel, nibble-serial scheme.

Normally the bus consists of a master device and several slave devices. The Master device addresses one of the Slave devices in a command message, while the addressed Slave device answers by sending a response message. The command and response messages constitute a message frame.

The two handshaking lines are /BAV and /HSK. The /BAV line signifies the beginning of a message frame and is normally controlled by the master. The Slave device can also use this line to inform the master that it needs immediate service. This line is connected to the interrupt logic and will issue an interrupt request, whenever it goes low. The /HSK line is used by all devices to ensure data is transmitted and received properly.

2.1 Command Message

A command message contains the following data:

Field	No. of Bytes
Device Code	1
Command Code	1
Logical Unit Number(luno)	1
Record Number	2
Buffer Length	2
Data Length	2
Data	variable

Each device connected to the HEXBUS has a unique device code. The device codes are assigned as follows(tentative):

0	all devices
1-8	stringy floppy(wafer tape)
10-17	printer/plotter
20-27	RS232C Interface
30-37	TV Interface (color)
40-47	TV Interface (black and white)
50-57	Centronix Type Parallel Output
60-67	Calculator or Computer in Slave Mode
70-77	Modem
80-87	GPIB Interface
90-97	Bar Code Reader
100-107	Floppy Disk Drive

The command code tells the Slave Device what I/O operartion should be undertaken. The command codes are currently assigned as:

0	Open
1	Close
2	Delete Open File
3	Read Data
4	Write Data
5	Restore File
6	Delete File
7	Return Status
8	Service Request Enable
9	Service Request Disable
10	Service Request Poll
11	You Are the Master
12	Verify Read/Write Operation
13	Format and Certify Media
14	Catalog Directory
15	Set Options
16	Transmit Break
17	Protect/Unprotect File
18	Read Sectors
19	Write Sectors
20	Modify File Name
21	Read File Descriptor
22	Write File Descriptor
23	Read File Sectors
24	Write File Sectors
25	Load
26	Save
27	Inquire Save
28	Home Computer Status
29	Home Computer Verify
254	Null Operation
255	Bus Reset

2.2 <u>Response Message</u>

The Slave device whose device code matches the first byte in a command message should transmit a response message which is of the following format:

Field	No. of Bytes
Data Length	2
Data	variable
Operation Status	1

The Operation Status contains 0 if the I/O operation terminates successfully. Otherwise, it contains the error code. The error codes are listed in Section 8.0.

3.0 General Commands

This section describes the I/O Commands that can be accessed from TI Basic, GPL and Assembler. These commands are the standard commands which all DSR's must handle so that a user can treat a hardware device like a file-oriented data set per the 99/4 File Management Spec. These Commands are Open, Close, Read, Write, Old, Save, Rewind, Delete and Status.

All the commands included in this section provide the same functions for Basic, GPL or Assembler users. Therefore, these commands will be discussed from a Basic user's point of view unless otherwise noted.

3.1 Open Command

To a 99/4A user, all the HEXBUS peripherals are considered as a group of secondary devices to a general device called "HEXBUS". Every secondary device has its own device code. This code is partially defined by the category to which the device is assigned and partially defined by a dip-switch or jumper-wire included in the unit. The device categories and their associated codes are listed in Section 2.1. Thus, each secondary device can be addressed by the string "HEXBUS" followed by its device code. The device codes range from decimal 0 to 255, inclusive. Device code 0 applies to all devices attached to the HEXBUS.

To open a non-disk type HEXBUS peripheral with device code <devcode>, a Basic user types:

```
OPEN #<luno> : "HEXBUS.<devcode>.<option string>",
<access mode>
```

where <luno> is an integer from 1 to 255 and <access mode> is a string specifying the mode under which the device is accessed, such as UPDATE, FIXED 80, SEQUENTIAL, etc. The <option string> is device dependent and the whole string will be sent to the device without any modification. For example, in an RS232 device, the <option string> would look like:

```
   BA=9600.PA=E.DA=8.CR.EC
```

In a Wafertape device, the <option string> should be the name of the file being accessed.

In the HEXBUS, the <luno> is part of the PAB and is always transmitted to the device in the Open Command. In subsequent I/O operations, the <option string> is never resent to the device, but <luno> is still included in the command message. In a non-disk type device, the <luno> in the communication message is functionless and ignored. However, in a disk-type device, where more than one file can be opened at a time, <luno> tells the device which opened file is being accessed and plays an important role when the filename is missing from the communication message.

In the 99/4A File Management System, <luno> is not included in the PAB. Nevertheless, the DSR needs to have a distinct integer associated with each open file, so that the DSR can tell the device to which file the I/O operation applies. To solve this problem, the user is required to open a disk file as:

```
OPEN #<luno> : "HEXBUS.<devcode>.LU=<luno>.<option str>",
<access mode>
```

where .LU stands for HEXBUS Luno and must immediately follow the <devcode>. Although the integer after ".LU= " need not be the same as the opened <luno>, it is a good habit to keep both numbers the same to avoid possible duplication of Logical Unit Numbers. The luno specified by ".LU=" is the luno actually sent to the device.

For a non-disk type device, if ".LU=<luno>" is used right after the <devcode>, the entire sub-string will still be interpreted by the DSR and the <luno> will be sent to the device. However, non-disk devices will ignore the <luno>. This restriction requires that all present and future HEXBUS devices exclude using ".LU" as an internal switch option.

The Default Logical Record Length, legal I/O Access Modes, Switch Options, etc., are dependent on each individual device attached to the HEXBUS. The DSR has no way of knowing these parameters until necessary communication steps are properly carried out.

After a device is successfully opened for access, the File Descriptor in the user's PAB should not be changed until the device is closed, so that the values of <devcode> and <luno> remain the same as those in the Open operation. All GPL and Assembler programs that create the PAB should respect this restriction.

If the device code is from 20 to 27, from 50 to 57, or from 70 to 77, the DSR always issues a Close Command before sending the Open Command. This allows an application to repeatedly open these devices without closing them. This is commonly done in Home Computer GPL or Assembly Language programs, because the Home Computer RS232 Interface has no way of preventing itself from being opened when it is already open.

The default Logical Record Length of the Wafertape (device codes 1 to 8) is 255, instead of 256, used by the

CC-40. This is done because the Maximum Record Length in the Home Computer is only 255.

3.2 Close Command

To close a device from further access, a Basic user types:

CLOSE #<luno>

If the file under operation is a disk file, the disk directory will be updated and the latest changes to the file will be recorded permanently on the media.

3.3 Read Command

To read a record from the HEXBUS, a Basic user types:

INPUT #<luno>: A

where A is a variable.

The Basic File Management System will send a PAB with a Read Opcode to the DSR. Certain HEXBUS devices, whether opened for variable or fixed record length, will always return a fixed length record with unused trailing data padded with 0's. The length of the returned data is the logical record length determined at file open time.

If the device under access is the RS232 Interface, Parallel Output, or Modem, the DSR will scan the keyboard for a user abort (Function 4) at the beginning of each Read operation in order to remain compatible with the Home Computer RS232 Interface Card (or Box).

3.4 Write Command

To write a record to the HEXBUS, a Basic user types:

PRINT #<luno>: A

where A is a variable

This statement causes a PAB with a Write Command to be passed to the DSR. The data contained in variable A will then be sent to the addressed device.

The LIST statement in Basic consists of an Open command, several Write commands, and a Close command.

Therefore, with the Write Command supported in the DSR, Basic can always do a LIST. For example, to obtain a listing from an RS232 device, the user would type:

```
LIST "HEXBUS.<devcode>.BA=1200"
```

where <devcode> is usually 20 for a HEXBUS RS232 device.

If the device under access is the RS232 Interface, Parallel Output, or Modem, the DSR will scan the keyboard for a user abort (Function 4) at the beginning of each Write operation, in order to remain compatible with the Home Computer RS232 Interface Card (or Box).

3.5 Save Command

The Save Command allows a user to save a block of data, usually a program, to a mass storage device. The following scheme was devised to maintain compatibility between the Expansion System Disk Controller and the HEXBUS Disk Controller. The scheme is important because it works with devices which support Load and Save (Disk) and those which do not (Wafertape).

When the HEXBUS Interface receives a Save Command from the Home Computer, it first issues an Inquire Save Command to the device. This is a shortcut (timesaving) to determine if the device supports the Save Command without transmitting the entire saved program. The Inquire Save Command is of the following format:

Field	Data	Source
Device Code	100 to 104	Master
Command Code	>1B	
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	0	.
Data Length	0	Slave
Status	as required	.

If the device supports the Save Command, the HEXBUS Interface issues the Save Command of the following format:

Field	Data	Source
Device Code	100 to 104	Master
Command Code	>1A	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	file name length+1+ program size	.
File name	as required	.
Seperator	00	.
Program data	as required	.
Data Length	0	Slave
Status	as required	.

If the device does not support the HEXBUS Save Command (such as the Wafertape), the HEXBUS Interface performs the following sequence of commands:

1. open the file/device with the user's PAB Record Number field set to the program byte count (2 byte value),
2. write the entire program,
3. close the file/device.

The DSR will perform the above three steps transparent to the user. Thus, a Basic user would type the following statement to save a program to a Wafertape device:

```
SAVE HEXBUS.1.MYFILE
```

where we have assumed the device code of the Wafertape is 1. To save a program to a disk-type device, the user still needs to provide the Logical Unit Number as required in an Open Statement, such as:

```
SAVE HEXBUS.101.LU=15.MYFILE
```

where the Disk is assumed to have device code 101 and Logical Unit Number 15 is not used by any other open disk file.

3.6 Load Command

The Load Command does the reverse of the Save Command. It allows the user to retrieve a block of data, usually a program, from a mass storage device and put it into memory.

When the HEXBUS Interface receives a Load Command from the Home Computer, it issues the HEXBUS Load Command which is of the following form:

Field	Data	Source
Device Code	100 to 104	Master
Command Code	>19	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	as required	.
Data Length	file name length	.
Data	file name	.
Data Length	as required	Slave
Data	program data	.
Status	as required	.

If the device does not support the HEXBUS Load Command (such as the Wafertape), the HEXBUS Interface performs the following sequence of commands:

1. open the file/device for input, with user' PAB Record Number field set to the memory buffer size,
2. read the program into the buffer if it is large enough to hold the data,
3. close the device.

As an example, to load a program from a disk-type device with device code 101, a Basic user might type:

```
OLD HEXBUS.101.LU=15.PROGRAM1
```

where the Disk is assumed to have device code 101 and Logical Unit Number 15 is not used by any other open disk file.

3.7 Rewind Command

The rewind command is mainly used to position to the first record of an open file on mass storage devices. The format of a Rewind Statement in Basic is similar to the Read or Write Statement, namely:

```
RESTORE #<luno>
```

In the 99/4A, if a file is opened for Relative Access in Update or Input Mode, the Rewind Command can also position to any record of the file, if the REC clause is used. For example:

```
RESTORE #<luno>, REC 100
```

will restore the open file to record 100.

This feature will not work, if the addressed device is not designed to support this option.

3.8 Delete Command

In the CC-40 File Management System, there are two types of Delete Commands: Delete Open File and Delete File. To delete an open file, the <luno> assigned to the file at open time is used to tell the device which file to delete. To delete a closed file, the entire file name is sent to the device, together with the Delete File Command.

In the 99/4A, there is only one Delete Command and the DSR has no way of knowing, based on the PAB alone, if the file is open. Whenever a Delete Command is received, the DSR shall first send Delete Open File Command to the HEXBUS. If an error of the type "File Not Open" is received from the device, the DSR then tries to delete the file by sending the Delete File Command.

Examples of some Delete Statements in Basic can be found in the following program:

```
100 OPEN #1:"HEXBUS.101.LU=1.DISKFILE",OUTPUT
110 OPEN #2:"HEXBUS.1.TAPEFILE"
120 DELETE "HEXBUS.101.LU=3.DISKFILE2"
130 CLOSE #1,DELETE
140 DELETE "HEXBUS.1.TAPEFILE"
```

Please note in the above program, we first delete a closed file, and next delete and close two open files.

3.9 Status Command

In the 99/4 File Management System, the Status Command can always be issued whether a file is opened or not. However, the Hexbus Status Command can only be used if the file is opened. A new status command, which always sends the file name in the PAB and can be issued whether a file is opened or not is required for certain software to be compatible. This command defined for 99/4A has the following format:

Field	Data	Source
Device Code	100 to 104	Master
Command Code	>1C	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	1	.
Data Length	file name length	.
Data	file name	.
Data Length	1	Slave
Data	99/4 Status	.
Status	as required	.

The bit definition of the status returned by the controller is the same as the Status byte defined in the 99/4 File Management Spec, namely:

Bit	Data	Description
7(MSB)	0/1	File exists/does not exist
6	0/1	Not Protected/Protected
5	0	Reserved
4	0/1	Display/Internal Data type
3	0/1	Data/Program File
2	0/1	Fixed/Variable Record
1	0/1	End of storage medium not true/true
0	0/1	EOF not true/true

If the HEXBUS device does not support the 99/4A Status Command, the HEX Interface sends a standard HEXBUS Status Command (HEXBUS Opcode 7). Th HEXBUS Interface then converts the HEXBUS Status byte into 99/4A Status format. In this situation, bits 1 and 3 have no meaning.

4.0 Special Commands

All the commands discussed thus far have been limited to those common to both the 99/4A and the CC-40 File Management Systems. There are still some commands which the CC-40 has, but the 99/4A does not. These commands shall be implemented through switch options in the Open or Save statements so that Basic users can still access them. These special commands are: Format Media, Catalog File, Verify Saved File, Transfer Raw Data on Bus, Reset Bus, and Slave Mode.

The six commands mentioned above shall be identified through a switch option between "HEXBUS" and the device code in the Open or Save statement. Namely,

```
OPEN #<luno>: "HEXBUS.<special sw>.<devcode>.<option str>",
<access mode>
```

The <special sw> switch option is defined as:

.FORMAT MEDIA	format media, used in Open
.CA	catalog file, used in Open
.VE	verify saved file, used in Save
.TR	transfer raw data on bus, used in Open
.RB	send Reset Bus Command, used in Open
.SL	Slave Mode, used in Open

Therefore, if the character right after the device name "HEXBUS" is a numeric character, the statement must not be one of the six special types listed above. However, if the character following the device name is not numeric, then one of the above six switches should be used. Otherwise, an error is returned.

Slave Mode will be discussed in a later section. The function of the other five commands is the topic of this section.

To format a device, the string "FORMAT MEDIA" must be completely spelled out. This is required to prevent the user from accidentally formatting a device due to mistyping. The other commands merely use two characters.

4.1 Format Media

To format a Wafertape with device code 1, a Basic user types:

```
OPEN #<luno>: "HEXBUS.FORMAT MEDIA.1"
CLOSE #<luno>
```

To format a Disk Device (Double Sided, Double Density, 40 Tracks Per Side) with device code 101, a Basic user types:

```
OPEN #<luno>: "HEXBUS.FORMAT MEDIA.101"
CLOSE #<luno>
```

The above statements can be performed at command level also. It is a good habit to Close the device immediately after it is formatted successfully.

4.2 Catalog File

The Catalog Command allows a user to inspect certain information of a file stored on a mass storage device such as the Wafertape. Repeated use of this command displays the entire directory of the storage media. Since the disk controller has its own method for cataloging the disk, the ".CA" option is not supported for the disk system.

A sample program to obtain the entire catalog of a Wafertape with device code 1 is listed below. This program reads the catalog of one file after another into an array until there are no files left. The program terminates with an error.

```
200 DIM A$(20)
210 OPEN #1:"HEXBUS.CA.1",INPUT,DISPLAY,FIXED 18
220 I=1
230 INPUT #1,A$(I)
240 I=I+1
250 GOTO 230
```

In the above program, the ".CA" switch option informs the DSR to execute a Catalog Command. The I/O mode, data type and record type must be limited to Input, Display and Fixed. The Logical Record Length, which defines the number of bytes returned by the device in response to the Input Command, is device dependent. For example, in the Wafertape, the total number of bytes returned by the device is 18, consisting of:

Byte #	Content
1	File Number, from 0 to 15
2-13	File Name in ASCII
14-15	Maximum Record Length
16-17	Record Count
18	Flag Byte, defined as: Bit 0 - 0/1 File Nonactive/Active Bit 1 - 0/1 Not/Is Last File in tape Bit 2 0/1 Display/Internal (MSB is Bit 0)

A Basic program, which reads and analyzes the File Catalog of a Wafertape, is shown below:

```
100 CALL CLEAR
110 PRINT "FILE_NAME    MAX_L REC_# FLG"  *print header
120 PRINT
130 OPEN #1:"HEXBUS.CA.1",FIXED 18,INPUT  *open device
140 INPUT #1:A$                           *read file catalog
150 PRINT TAB(1);SEG$(A$,2,12);           *print file name
160 A1=ASC(SEG$(A$,14,1))
170 A2=ASC(SEG$(A$,15,1))
180 PRINT TAB(13);A2*256+A1;              *print max record
190 A1=ASC(SEG$(A$,16,1))
200 A2=ASC(SEG$(A$,17,1))
210 PRINT TAB(19);A2*256+A1;              *print record #
220 A1=ASC(SEG$(A$,18,1))                 *get flag byte
230 ACTIVE$="N"
240 LAST$="N"
250 INTERNAL$="D"
260 IF A1<128 THEN 290                    *go if not active
270 A1=A1-128                             *active file
280 ACTIVE="A"
290 IF A1<64 THEN 320                     *go if not last
300 A1=A1-64                              *last file on tape
310 LAST$="Y"
320 IF A1<16 THEN 320                     *go if Display
330 INTERNAL$="I"                         *Internal
340 PRINT TAB(26);ACTIVE$;INTERNAL$
350 IF LAST$="N" THEN 140                 *go for next file
360 END
```

4.3 Verify File

The Verify Command verifies that the information in memory is the same as the information stored on a mass-storage device. If there is a discrepancy, an I/O error message is returned. An example of saving and verifying a program to a disk device attached to the HEXBUS is shown below.

```
SAVE HEXBUS.101.AFILE
SAVE HEXBUS.VE.101.AFILE
```

The above two statements should be used together to make sure the program is saved correctly. The first statement saves the file and the second statement will verify the previously saved file. Under no circumstances will the second statement both save and verify a file.

The .VE option may be used to verify a SAVED program file as illustrated above or a data file. The second cpapbility was included to allow verification of large Extended Basic programs, which are actually saved as data files. If the Verify Command is used to verify data, the file should be opened in OUTPUT mede. The PRINT statement functions as a verify command when the file is opened in the VERIFY mode. No data can be written to the file while it is opened in the VERIFY mode. Also, a file cannot be opened in the normal mode and VERIFY mode at the same time.

The following program creates a data file and then verifies it.

```
100 OPEN #1:"HEXBUS.101.EXAMPLE",INTERNAL,FIXED 20
110 FOR X=1 TO 10
120 PRINT #1:X,X*2
130 NEXT X
140 CLOSE #1
150 OPEN #10:"HEXBUS.VE.101.EXAMPLE",OUTPUT,INTERNAL,FIXED 20
160 FOR X=1 TO 10
170 PRINT #10:X,X*2
180 NEXT X
190 CLOSE #10
200 DISPLAY "NO ERRORS"
210 END
```

Lines 100-140 open a file called EXAMPLE on a disk drive, print 10 records, and close the file. Lines 150-190 open the save file in the VERIFY mode, verify that the data was written correctly to the file, and close the file. Line 200 displays a message that there were no errors found. If an error is detected, a hardware I/O error occurs.

NE 10 4.4 Transfer Raw Data on Bus

This command provides a Basic user the ability to send and receive raw data on the HEXBUS. With this command, a Basic program can communicate with HEXBUS devices directly. If some future HEXBUS device has certain functions the 99/4A can not easily access, this command could solve that problem.

The function of this command can be best described by an example. To send and receive a frame of raw data between the HEXBUS and an RS232 device, a Basic user could do:

```
   .
   .
200 OPEN #1:"HEXBUS.TR"    *Enable 99/4A to transfer raw data
   .
260 Z$=CHR$(0)
270 ZZ$=Z$&Z$
280 A$=CHR$(20)&ZZ$&ZZ$&CHR$(4)&Z$&CHR$(3)&Z$
290 A$=A$&ZZ$&CHR$(192)    *Build the Open RS232
300 PRINT #1:A$            *Open the RS232
310 INPUT #1:B$            *Read response to the Open
   .
350 PRINT #1:C$            *Write to device
360 INPUT #1:D$            *Read response to the Write
   .
400 PRINT #1:E$            *Close RS232
410 INPUT #1:F$            *Read response to the Close
   .
420 CLOSE #1               *Exit Transfer Raw Data Mode
   .
   .
```

In the above program, A$ contains a series of bytes that are to appear on the bus in the same sequence they are packed into A$. After the last byte in A$ is sent, the 99/4A shall hold the /HSK line low, so that the sending device has to wait for the /HSK to go high before sending any data back to the 99/4A. In order to resume the communication sequence, the user must put an Input Statement immediately following the Print Statement, like Line 300 and Line 310. In this Input Statement, the DSR first releases the /HSK line so that the responding device can send its data back. The DSR then packs the device data into B$ for the main program to analyze.

A pair of properly coupled Print and Input Statements form a communication frame on the HEXBUS. For example, if a user wants to send one line to the RS232 device, he should include three pairs of Print and Input statements in his program. The first pair performs the Open Command, the second pair writes to the device and the third pair closes the device.

4.5 Reset Bus

Sometimes it is useful to reset all the devices on HEXBUS so that all files and devices are closed and the entire bus is known to be in an initial state. To achieve this, a Basic user types:

```
OPEN #1:"HEXBUS.RB.0"
```

Here the device code is 0, meaning the command applies to all devices. No response message is returned by the devices when this command is sent.

5.0 Home Computer Compatibility Commands

The commands included in the following section constitute the GPL subprograms described in the GPL Interface Specification for the 99/4 Disk Peripheral.

5.1 Extended Format Media Command

The Format Media Command described in Section 4.1 does not allow one to pass any parameters to the peripheral. In other words, the message sent by the master contains the Peripheral Access Block (PAB) without any device dependent information.

Currently, there are many different types of floppy disks on the market : single or double density, single or double sided and 35, 40, 77 or 80 tracks. There is no way the Disk Controller knows what types of disk drive or which format the user desires without instructions from the user. To allow the user maximum flexibility in choosing disk drives, the current Format Media Command is redefined as follows :

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>0D	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	as required	.
Data	as required	.
Data Length	2	Slave
# of sectors formatted	as required	.
Status	as required	.

If the Data Length in the master message is 0, the Disk Controller should format the diskette with default options, namely, double density, double sided, 40 tracks per side, interleaving sectors 4 and skewing factors 0. If the Data Length received is 6, the controller should interpret them as (bit 7 is MSB) :

Byte 1	Bit	Meaning
	7	0/1 for single/double density, default SD
	6-0	set to 0's
Byte 2		number of sides to format, default 1
Byte 3,4		number of tracks on each side, Byte 3 is least siginificant byte, default 40
Byte 5		interleaving sectors, default to 4
Byte 6		skewing factors, default to 0

If the Data Length received is less than six, it is assumed that the remaining data not sent to the controller are default values. In the case that more than 6 bytes of data are following the PAB, the controller should treat the first six bytes as defined above and ignore all the other data.

The controller returns the total number of sectors formatted if the operation is successful.

5.2 read/write Sectors Commands

To read data from or write data to a group of contiguous sectors, the master needs to specify the starting sector and the total number of sectors under operation. Both the Starting Sector and the Total Sectors are two byte values, with the Least Significant Bytes issued first.

The Read Sectors Command has the following format:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>12	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	as required	.
Data Length	4	.
Starting Sector	0 to >FFFF	.
Total Sectors	0 to >FFFF	.
Data Length	as required	Slave
Data	from sectors	.
Status	as required	.

The Write Sectors Command has the following format:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>13	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	total data + 4	.
Starting Sector	0 to >FFFF	.
Total Sectors	0 to >FFFF	.
Data	as required	.
-----------	-----------	-------
Data Length	0	Slave
Status	as required	.

Since the controller transfers data to the disk drive faster than the master transfers data to the controller, the controller might have to buffer a full sector of data before sending the data to the drive. If the master is sending more data than the controller can buffer, the controller will have to hold the master waiting until some of the data in the buffer is stored to the diskette and then informs the master to send more data.

When the Read Sectors and the Write Sectors Commands are used, no file should be open.

5.2 Modify File Protection Command

The Modify File Protection Command allows a diskette file to be specified as protected or unprotected. This protection option is part of the File Header information and informs the controller software whether a file can be opened in Update, Write or Append Mode, can be deleted, the file name can be changed, etc. The format of the Modify File Protection Command is defined as:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>11	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	as required	.
Data	as required	.
Data Length	0	Slave
Status	as required	.

The Data Length sent by the master should be the length of the file name plus one. The length of a file name on a diskette should be less than or equal to ten charaters for Home Computer compatibility. A file name with less than ten characters is equivalent to a name of length ten with trailing characters padded with spaces.

The first character in the Data Field sent by the master is 0 for unprotect and >80 for protect. The file name begins as the second byte of data.

5.3 Modify File Name Command

The Modify File Name Command allows a user to change disk file names. The format of the command is shown below.

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>14	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	as required	.
Data	as required	.
Data Length	0	Slave
Status	as required	.

The data field in the master's message should be:

```
Current File Name (10 characters maximum for disk file)
00                (end of  Current File Name indicator)
New File Name     (10 characters maximum for disk file)
```

The Data Length sent by the master is the sum of lengths of the two file names plus one.

5.4 File Copying Commands

In general, a disk file consists of two parts; the recorded data which is common to all mass storage devices, and information to assist in faster data retrieval. This later information is called the File Descriptor Record (FDR) in the 99/4 File Management System. To copy a file, both the recorded data and part of the File Descriptor Record (FDR) must be copied.

For convenience, the Home Computer File Descriptor Record is listed below. Please refer to the TI 99/4 Disk Peripheral Software Specification for more detailed information. Here, the AU (Allocatable Unit) is defined as one sector in the current disk format.

Byte	Description
0-9	ASCII File Name
10	Copy Protection Indicator
11	Reserved, set to 0
12	File Status Flags
13	Number of Records/AU
14,15	Total AU currently allocated
16	EOF position in last AU
17	Logical Record Length
19,18	Fixed Length file: total Logical Records other type of file: same as byte 14, 15
20-27	Reserved, set to 0
28-255	Pointer Chain to AU's of this file

5.4.1 R/W File Descriptor Record

Four new Commands are recommended for the file copy purposes, namely, Read FDR, Write FDR, Read File Sectors and Write File Sectors Commands. Only one file can be copied to another file at any time using these commands.

The Read FDR Command and Write FDR Command informs the controller that a new series of copy operations are to be undertaken on a source file and a destination file, respectively. It is the application software's responsibility to properly pair the Read/Write FDR Commands to achieve the file copy process.

In order to read the FDR information of a file, the Master specifies the file name in the Read FDR Command and the controller returns the FDR contents from Byte 0 to Byte 27. Bytes 28 to 255 in the FDR determine where the file data is stored across the diskette. They are diskette dependent and need not be transmitted by the controller.

The format of the Read File Descriptor Record Command is:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>15	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	28	.
Data Length	length of name	.
Data	file name	.
-------------	-------------	--------
Data Length	28	Slave
Data	Byte 0-27 in FDR	.
Status	as required	.

The counterpart of the Read FDR Command is the Write FDR Command. The master should send the first 28 bytes in the FDR to the Controller. Again, Bytes 28 to 255 in the FDR need not be transmitted, since they are diskette dependent. The format of the Write File Descriptor Record Command is:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>16	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	28	.
Data	1st 28 bytes in FDR	.
-------------	-------------	--------
Data Length	0	Slave
Status	as required	.

The actions taken by the controller upon receiving the Write FDR Command include creating a file with a duplicate File Descriptor Record as transmitted in the command.

5.4.2 R/W File Sectors

After the FDR of a file is obtained from the controller, an application program should know the size of the file. The application program can now read and write all or part of the file (dependent upon the application program's available memory). However, it is the application program's responsibility to keep track of where it is in the copying process.

The Read File Sectors Command and Write File Sectors

Command are defined for the partial file copy purposes. Two parameters must be specified in these commands, one for the Total Sectors and the other for the Starting Sector to transfer. This later parameter is 0 if Starting Sector is the very first sector of the file data. Again, it is the responsibility of the application software to properly pair the Read/Write File Sectors Commands to correctly copy a source file to a destination file piece by piece.

The Read File Sectors Command is defined as:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>17	.
Logical Unit Number	don't care	.
Record Number	don't care	.
Buffer Length	(Total Sec)*256	.
Data Length	4+file name length	.
Starting Sector	0 to >FFFF	.
Total Sectors	0 to >FFFF	.
File Name	as required	.
----------------	----------------	----------
Data Length	(Total Sec)*256	Slave
Data	Sector Data	.
Status	as required	.

If the Total Sectors requested by the master is larger than the file size, the controller should return the actual size of the file in the Data Length field. If there is no more data left in the file, the controller should send zero Data Length.

The format of Write File Sectors Command is fedined as:

Field	Data	Source
Device Code	101 to 104	Master
Command Code	>18	.
Logical Unit Number	as required	.
Record Number	don't care	.
Buffer Length	don't care	.
Data Length	5+256*(Total Sec) +file name length	.
Starting Sector	0 to >FFFF	.
Total Sectors	0 to >FFFF	.
File Name	as required	.
Seperator	00	
Data	Sector Data	.
----------------	----------------	----------
Data Length	0	Slave
Status	as required	.

During the sector reading process, the controller might have to read one sector from the disk, transmit it to the HEXBUS, and go fetch the next sector. If there is a bad sector found, the controller should not terminate the communication process immediately. Instead, the controller must continue sending the amount of data the master is expecting -- even though they could be trash -- and then terminates the communication with a proper Error Code.

During the sector writing process, similar actions must be take care of to maintain communication integrity, when a bad sector is found.

6.0 The RS232 Device Name for TE II Support

Do not confuse the Hexbus RS232 Interface with the Home Computer RS232 Interface Card or the RS232 Interface Box. The former is an RS232 Interface peripheral attached to the HEXBUS Interface and is addressed by "HEXBUS.20", while the later two are attached to the Home Computer directly through the 44-pin I/O connector and are addressed by "RS232/1" or "RS232/2".

The HEXBUS Interface DSR includes in its Header Field three more device names: "RS232", "RS232/1" and "RS232/2". If there is an RS232 Interface Card (or Box)in the system whenever these three RS232 device names are referenced, the RS232 Interface Card will gain control bypassing the HEXBUS Interface. However, if the system does not have an RS232 Interface Card, the HEXBUS DSR will gain control and mimic the functions provided by the RS232 Interface Card DSR.

The purpose of providing some of the RS232 functions in this DSR is to allow TE II, which knows only three device names, to use the HEXBUS RS232 Interface without any modification. Please refer to the 99/4 RS232 Software Functional Spec for more details about the RS232 Interface functions, including Circular Interrupt Mode.

In order to save ROM space, some of the functions found in the RS232 Interface Card (or Box) are not supported. Those RS232 functions supported in the HEXBUS DSR are: Open, Close, Read, Write and Circular Interrupt. The functions not supported are: Load, Save and Parallel I/O.

In the HEXBUS DSR, both device names "RS232" and "RS232/1" correspond to device code 20, while "RS232/2" corresponds to device code 70 (HEXBUS Modem).

7.0 Interrupt Service Routine

The Interrupt Service Routine discussed in this section is mainly the Circular Interrupt for the RS232 input used in TE II. If the user intends to use the HEXBUS Service Request per the HEXBUS Peripheral Bus Spec, he should write his own Interrupt Service Routine and set up a linkage in CPU RAM >8300 as described next.

Upon entry of interrupt, the Interrupt Service Routine will inspect the contents of the word >8300 and take different prodecures depending on the value of the most siginificant 2 bits.

If Bit 0,1 = 0,0 then Circular Interrupt is in effect with data buffer in VDP RAM defined by:

>8300,>8301	Address of Buffer in VDP RAM
>8302	Size of buffer (255 maximum)
>8303	Caller Read offset value
>8304	RS232 DSR Write offset value
>8305	not used

If Bit 0,1 = 0,1 then Circular Interrupt is in effect with data buffer in CPU RAM defined by

>8300,>8301	Least Siginificant 14 Bits times 4 points to a CPU RAM area defined as:

1st, 2nd Bytes	Address of Buffer in CPU RAM
3rd, 4th Bytes	Size of buffer
5th, 6th Bytes	Caller Read offset value
7th, 8th Bytes	RS232 DSR Write offset value

If Bit 0,1 = 1,0 then branch to user Interrupt Service Routine. The entry point of user Interrupt Service Routine is 4 times the value of the remaining 14 least significant bits.

Upon entry of the Interrupt Routine, the Workspace Pointer is set to >83E0. The Interrupt Routine may use from R1 to R8 and R10. R8 must be set to 0 and R12 must be kept at the same value before exit. R11 contains the return address.

8.0 Slave Mode

Normally, the 99/4A is in Master Mode, with all the devices connected to the HEXBUS as Slave Devices. However, if one of the devices on the bus is the CC-40 or another 99/4A, it might be desirable to let the 99/4A behave as a Slave Device and let the CC-40 or another 99/4A be the master.

All the DSR functions provided in this section are designed to only handle communication details between the 99/4A as a slave and the new master. No other implementation has been included to assist the user in any Slave Mode application. However, with the features described in this section, the user can still do the following application:

1. Set the 99/4A in Slave Mode with a certain Device Code.
2. If the Master is addressing the 99/4A, assign Master's message to a string.
3. Analyze the message. During this period, the /HSK line is held low, so that the Master will not time out.
4. Pack the proper responding message in a string, release /HSK line, and send the string back to the Master.

In the above procedures, the user is responsible for everything described in Step 1, analyzing Master's message in Step 3 and packing the response message in Step 4. All the communication steps and handshaking sequences are taken care of by the DSR. With this fundamental design concept, the user has the maximum flexibility in using the 99/4A's vast resources around the CC-40 or between two 99/4As.

To put the 99/4A in Slave Mode, a Basic program should contain:

```
OPEN #<luno>: "HEXBUS.SL.<devcode>"
```

The switch ".SL" informs the DSR that the 99/4A is operating in Slave Mode. The <devcode> is a decimal number denoting the device code of the 99/4A from the new Master's point of view. <devcode> should be from decimal 60 to 67, inclusive. The legal access mode, file type, record type and record length are: Update, Display, Sequential, Variable and 255-byte record length.

When the Slave Mode is in effect, all the Commands in Master Mode and the Circular Interrupt described in the previous sections should not be used. Otherwise, unpredictable results might occur. The DSR will let the 99/4A become Master again, when the Close Command is received. Of course, the master device must be taken out of the Master Mode first to avoid having two Masters on the bus.

Since the 99/4A PAB can handle only 255 bytes per record, the Basic user running Slave Mode can only receive and transmit 255 byte to and from the master at a time. To read and write more than 255 bytes, the user will have to use an Assembly Language program.

After the 99/4A is opened in Slave Mode, in order to participate in the bus communication, the user should use an Input Statement followed by a Print Statement. The Input Statement packs the Master's message into a return string, while the Print Statement passes a string containing the response message to the Master. The DSR handles all the string packing, unpacking and communication sequences. The data contained in these strings will be arranged in the exact order they appear on the HEXBUS. At the end of the Input Statement, the DSR will hold the /HSK line low to keep the Master waiting. Therefore, a time out error will not occur on the Master side. A Basic program which performs the steps described above is shown below:

```
100 OPEN #1:"HEXBUS.SL.61"
 .

300 INPUT #1:A$
310 FOR I=1 TO LEN(A$)       *print each byte on screen
320 PRINT ASC(SEG(A$,1,I));
330 NEXT I
340 PRINT
 .
 .  {analyzing A$}
 .  {packing responding message in B$}
 .
400 PRINT #1:B$              Respond to Master
 .
 .  {more Input & Print Pairs}
 .
500 CLOSE #1                 Exit Slave Mode
```

The 99/4A should be in Slave Mode before the Master can send any data to it. For example, in the above program, if the message from the Master is received by the 99/4A before Statement 100 is completed, the 99/4A will not recognize the message and the Master will experience a time out error. On the other hand, if the message is received by the 99/4A after Statement 100 is executed, whether Statement 300 is pending or not, the communication will be carried through under normal bus conditions.

9.0 Error Messages

According to the 99/4A PAB, there are only 7 Error Codes. However, the HEXBUS File Management System provides many more error codes. The following table illustrates the HEXBUS Error Codes and their corresponding 99/4A Error Codes. After each I/O Operation, the DSR will return the 99/4A Error Code in the users PAB.

99/4A Error Code	HEXBUS Error Type	
0	0	no error/normal operation termination
2	1	device/file option error
2	2	attribute error
7	3	file/device not found error
7	4	file/device not open error
7	5	file/device already open error
6	6	device error
5	7	EOF error
7	8	data/file too long error
1	9	write protect error
	10	not requesting service (response to poll inquiry)
4	11	directory full error
4	12	buffer size error
3	13	unsupported command error
3	14	file not opened for write
3	15	file not opened for read
6	16	data error (checksum)
2	17	relative file not supported
2	18	sequential file not supported ???
2	19	append mode not supported
2	20	output mode not supported
2	21	input mode not supported
2	22	update mode not supported
2	23	internal/display type error
6	24	verify error
6	25	low batteries in peripheral
6	26	uninitialized media
6	27	peripheral bus error (timing error)
4	32	media full
	33	attempted to excede maximum number of lunos
	34	invalid data (too short or incorrect contents
3	254	illegal in slave mode
6	255	bus time out error

The 99/4A Error Codes and error types are:

0	no error
1	write protected
2	bad open attributes
3	illegal operation
4	out of table/buffer space
5	attempt to read past EOF
6	hardware error
7	file error