Application Note

Implementing a Secure Digital Card with a ZNEO Microcontroller

AN032001-1211

Abstract

The ability of embedded devices to store data or a volume of data is widespread across multiple industries. The need for these types of devices was met by the Secure Digital (SD) card, a storage device that is inexpensive, readily available and a popular storage/file transfer medium for PCs and embedded devices such as cameras, cell phones, PSPs and data loggers. With SD cards, files can be written to and retrieved from PCs and embedded devices.

The implementation of an SD card interface requires both a hardware interface and file system to handle the FAT16 and FAT32 types of file allocation; the file system manages how data is organized inside the SD card. This application note discusses how customers can interface Zilog's ZNEO Z16F Series microcontroller with an SD card using both the Secure Digital Standard Card (SDSC) and Secure Digital High Capacity Card (SDHC) standards and managing file system procedures to create files and directories. To demonstrate this interface, the SD card will create, write and read directory entries and read, write and delete files from the contents of ZNEO MCU memory via a console/terminal emulation application such as HyperTerminal.

The Enhanced Serial Peripheral Interface (ESPI) of the ZNEO CPU provides a transmit and receive buffer to support high performance throughput and includes a number of transmit data, receive data and error interrupts. The ESPI function supports SCK clock rates up to one-half the system frequency, which can result in faster SD card read and write operations.

Note: The source code file associated with this application note, AN0320-SC01.zip, is available for download on zilog.com. This source code has been tested with version 5.0.1 of ZDSII for ZNEO MCUs. Subsequent releases of ZDSII may require you to modify the code supplied with this application note.

Features

The feature set for this application includes a file system interface to the SD card using SPI signals. Both the FAT16 and FAT32 file systems are used on the SD card to manage the following functions:

- Reading/writing directory entries
- Creating directory entries
- Reading file contents
- Writing file contents

- Deleting files
- Card memory statistics
- Formatting of the SD card

The application described herein supports the following functions and formats:

- FAT16 and FAT32 (FAT12 is not supported)
- SDSC and SDHC
- FAT16 and FAT32 for SDSC to a maximum of 2GB
- FAT32 for SDHC to a maximum of 32GB
- Short File Name 8.3 format (long file names are not supported)
- Up to 128 directory levels; i.e., E:\FOLD01\FOLD02\.....\FOLD128

Zilog's ZNEO CPU is a 16-bit MCU that meets the continuing demand for faster and more code-efficient microcontrollers. This SD card application is implemented using the ZNEO CPU's ESPI and UART blocks.

Discussion

A Secure Digital (SD) card is a nonvolatile memory card developed by the SD Card Association. It has become the accepted standard for removable memory storage, and has been adapted to many different devices and classes. An SD card standard was introduced by Toshiba, Matsushita Electric and SanDisk in 1999. SD cards are used to store data and to enable the transfer of data between devices. This document discusses two types of SD cards: the Standard Card (SDSC) with 2GB of maximum storage capacity, and the High Capacity Card (SDHC) with 32GB of maximum capacity. See Figure 1.

Figure 1. SanDisk 2GB SD card

The SD card can operate in two modes: SD Mode and SPI Mode. This application employs SPI Mode as its SD card interface protocol; indeed, the SD card is connected to the ZNEO MCU through its Enhanced Serial Peripheral Interface (ESPI). To communicate via this kind of mode interface, the SD card's host machine uses commands to send an SPI packet to the SD card; these commands are then interpreted by the card.

Note: For more information about the SPI interface, see the SPI Mode section of the document titled *SD Specifications, Part 1: Physical Layer Simplified Specification Version 3.01*, published by the SD Card Association, at https://www.sdcard.org/downloads/pls/simplified_specs/Part_1_Physical_Layer_Simplified_Specification_Ver_3.01_Final_100518.pdf.

To learn more about SD card commands and responses, refer to the SD Memory Card Functional Description section of the document linked to above.

The SPI 4-wire synchronous serial protocol is used to interface peripheral devices with microcontrollers such as the ZNEO Z16F Series MCU. As described earlier in this document, the Enhanced SPI block of the ZNEO Z16F Series MCU is used to interface the MCU with the SD card. To use an SD card with the SPI protocol, it must be initialized to SPI Mode by the master thru a simple command. The SD card, acting as the slave, responds to the CMD0 command, followed by a data token, to indicate either a bulk transfer or an error condition.

Figure 2 illustrates the interface between the SD card and its SPI master.

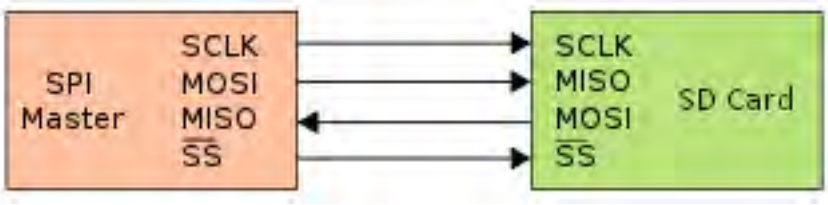

Figure 2. Master to Slave Interface

Through the SPI connection, files and directories can be easily managed, provided that the SD card has been formatted with either the FAT16 or FAT32 file system.

FAT File Systems

Many file systems are employed in embedded devices across the computer industry, all intended to operate within their respective operating systems. The Linux OS, for example, uses the *ext* family (such as ext2, ext3 and ext4), XFS, JFS, ReiserFS and btrf; the Solaris OS uses VxFS and QFS. The file system used in this application note is the Microsoft File System – namely, the FAT16 and FAT32 file systems.

The FAT16 file system offers a total of 65535(2^16) clusters and uses a 16-bit number to identify its cluster addresses; Figure 3 illustrates the FAT16 file map.

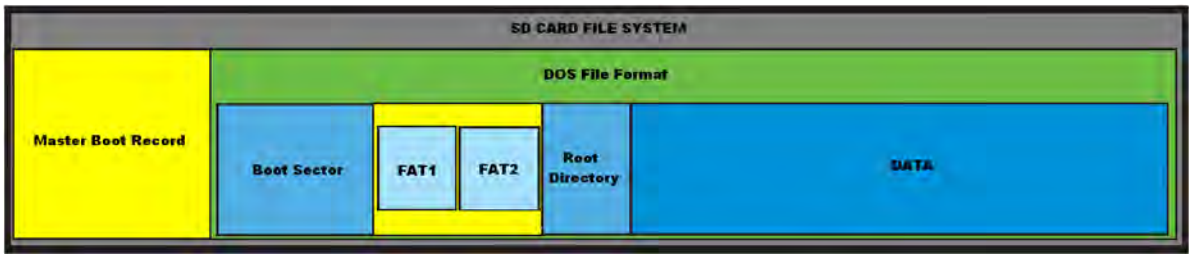

Figure 3. FAT16 File Map

The FAT32 file system offers a total of 4294967296(2^32) clusters and uses a 32-bit number to identify its cluster addresses; Figure 4 illustrates the FAT32 file map.

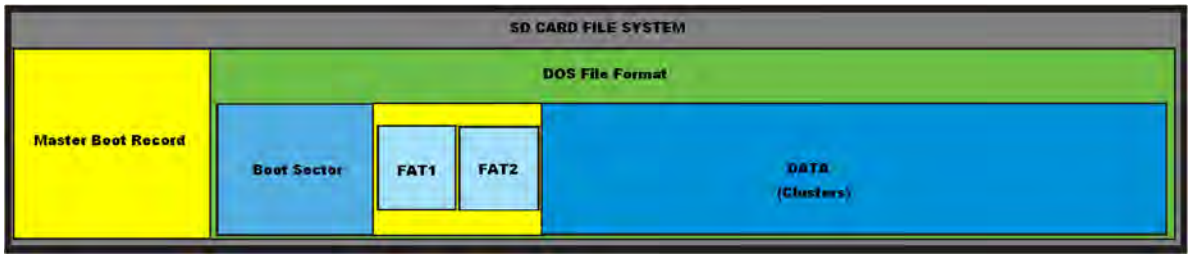

Figure 4. FAT32 File Map

The size of each cluster is defined in the boot sector named *Sectors per Cluster*.

A FAT file system volume is composed of four basic regions, which are laid out in this order on the volume:

- Reserved region (includes unused FAT32 sectors)
- FAT region
- Root directory region (does not exist on FAT32 volumes)
- File and directory data region

These four regions are described in the following paragraphs.

Reserved Region

The Reserved Region is the beginning of the FAT map. In FAT16 file systems, Sector 0 is the boot sector which stores data called the BIOS Parameter Block (BPB). In FAT32 file systems, Sector 0 contains the Master Boot Record (MBR), which stores the addresses or sector number of the boot sector. The BPB contains information about the SD card such as an OEM identifier, the number of file allocation tables, a media descriptor (type of storage device) and the number of sectors per cluster. Starting at offset 36, FAT16 and FAT32 differ in their BPB/boot sector structure.

Note: For more information about this difference in boot sector structures, refer to the *How FAT Works* page on the Microsoft TechNet website at http://technet.microsoft.com/en-us/library/cc776720%28WS.10%29.aspx.

The FAT32 file system contains a file system information structure adjacent to the boot sector. Both this information structure and the boot sector account for redundant file system management in the event of a lost boot sector; i.e., a system may inquire via the Sector 6 and 7 addresses in which it resides to restore the boot sector and the file system information structure at sectors 0 and 1.

In a partitioned SD card, Sector 0 is termed the *Master Boot Record*. A Master Boot Record contains the following parameters:

- A 446 bytes with a value of `0x00`
- Partition data composed of 64 bytes that contain the addresses of four partitions within the boot sector; each of these partitions contains 16 bytes
- A 2-byte trailing end with a value of `0xAA55`

The file system information structure contains the next free cluster and the number of available clusters.

Note: For more information about free clusters within the file system information structure, refer to the *How FAT Works* page on the Microsoft TechNet website at http://technet.microsoft.com/en-us/library/cc776720%28WS.10%29.aspx

File Allocation Table

Often referred to as *FAT*, a a file allocation table manages the cluster locations of stored data. In the FAT16 file system, a cluster entry is in the form of a 16-bit number; address `0xFFFF` indicates the last cluster of the file. In the FAT32 file system, a cluster entry is in the form of a 32-bit number; address `0xFFFFFF0F` indicates the last cluster of the file.

Root Directory Region

A Root Directory Region stores information about all files and directories in the file system; it is only used with FAT16 file systems. In FAT32 systems, the root directory is located at Cluster 2, which is the starting cluster of the data region. This root directory stores 32 bytes of file root information, as described in the Microsoft TechNet page, *How FAT Works* at http://technet.microsoft.com/en-us/library/cc776720%28WS.10%29.aspx.

Figure 5 shows a sample Root Directory entry.

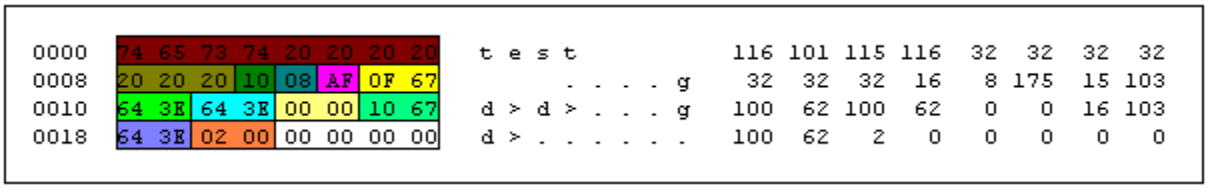

Figure 5. Root Directory Entry Packet

Data Region

A Data Region can be thought of as the storage location of all file and directory data; this Data Region constitutes the bulk of a partition. The sizes of files and subdirectories can be increased arbitrarily (as long as there are free clusters available) by simply adding more links to a file's chain in the FAT. However, files are allocated in units of clusters; therefore, if a 1KB file resides in a 32KB cluster, the remainder of the 31KB is wasted.

Addresses, clusters, sectors per cluster and the locations in which data regions start can be calculated from values stored in the boot sector using the following formulas:

Start of Reserved Sector = 0

FAT Region Start = Number of Reserved Sector + Start of Reserved Sector

Root Directory Start = FAT Region Start + (Number of FATs * Sector per FAT)

Data Region Start = Root Directory Start + ((Root Entries Count * 32) / Bytes Per Sector)

Note: The sizes of regions may vary depending on volume capacity. To calculate these sizes, refer to the *FAT16 File System* page at http://www.maverick-os.dk/FileSystemFormats/FAT16_FileSystem.html.

The following two examples show how to calculate the start of each region, in which the volume always starts with Sector 0 (because the volume is not partitioned).

Example 1. Calculate for a 2GB SD card formatted with the FAT16 file system given the following data for the SD card:

- SectPerFat = 239
- SectPerClus = 64
- ReservedSec = 2
- RootDirEntries = 512

Solve for the four region addresses, as follows:

BeginAddr = 0
FAT1Addr = ReservedSec+BeginAddr

```
                = 2 + 0
                = 2
FAT2Addr  = SectPerFat + FAT1Addr
                = 239 + 2
                = 241
RootDirAddr = (2*SectPerFat) + FAT1Addr
                = (2 * 239) + 2
                = 480
DataAddr = RootDirAddr + ((RootDirEntries*32)/512)
                = 480 + ((512 * 32) / 512)
                = 512
```

Example 2. Calculate for a 2GB SD card formatted with the FAT32 file system, given the following information for the SD card:

- SectPerFat = 239
- SectPerClus = 64
- UnusedSector = 1
- ReservedSec = 2
- RootDirEntries = 512

Solve for the four region addresses, as follows:

```
BeginAddr = 0(unless if partitioned, it must be locate using MBR)
FAT1Addr = UnusedSector+ReservedSec+BeginAddr
                = 1+2 + 0
                = 3
FAT2Addr = SectPerFat + FAT1Addr
                = 239 + 3
                = 242
RootDirAddr = Cluster #2 ; start of Data Region

DataAddr = Cluster #3
```

Note: The SD card values used in the above examples were derived from Disk Investigator, which we used to read the SD card. Disk Investigator can be downloaded from http://download.cnet.com/Disk-Investigator/3000-2248_4-10255339.html.

Endianness

Zilog's ZNEO Z16F Series MCUs operate in *Big-Endian mode*, while other devices operate in *Little-Endian mode*. In this application, addresses are converted to *Little-Endian mode* prior to reading and writing data to/from the SD card.

The following passage from a white paper titled *Endianness*, published by INTEL at http://www.intel.com/design/intarch/papers/endian.pdf, states Intel's definition of endianness:

Endianness is the format to how multi-byte data is stored in computer memory. It describes the location of the most significant byte (MSB) and least significant byte (LSB) of an address in memory. Endianness is dictated by the CPU architecture implementation of the system. The operating system does not dictate the endian model implemented, but rather the endian model of the CPU architecture dictates how the operating system is implemented.

Representing these two storage formats are two types of Endianness-architecture, Big-Endian and Little-Endian. There are benefits to both of these endian architectures. Big-Endian stores the MSB at the lowest memory address. Little-Endian stores the LSB at the lowest memory address. The lowest memory address of multi-byte data is considered the starting address of the data.

In *Big-Endian mode*, the Most Significant Byte (MSB) is stored at the lowest address; conversely, in *Little-Endian mode*, the Least Significant Byte (LSB) is stored at the lowest address.

Hardware Implementation

The tools used to develop this application are:

- ZDSII – ZNEO version 5.0.1
- ZNEO Z16F Series Development Kit (Z16F2800100ZCOG)
- ZNEO SD Card Daughter Board
- SD Cards:
  - 2GB SDSC
  - 4GB SDHC
- A host PC

The ZNEO Z16F Series Development Board is shown in Figure 6, and the SD Card Daughter Board is shown in Figure 7.

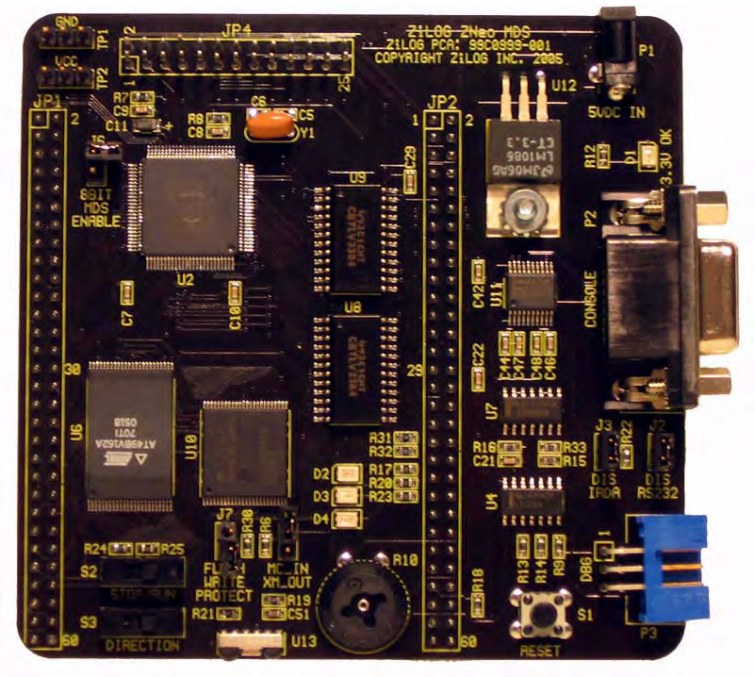

Figure 6. ZNEO Z16F Series Development Board

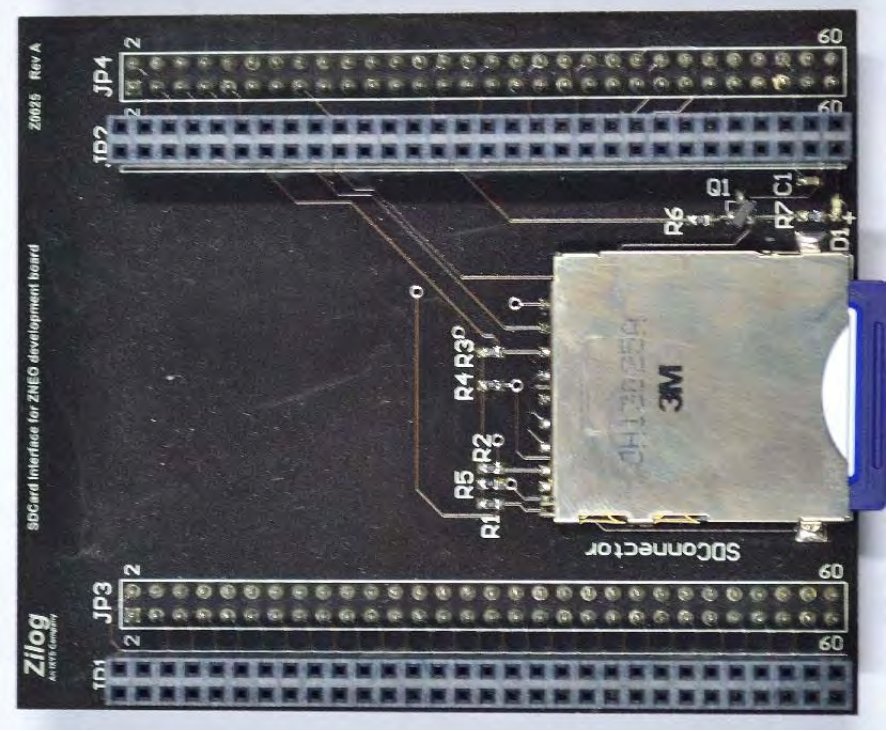

Figure 7. SD Card Daughter Board

Figure 8 shows the SD card, serial and power connections to the ZNEO Z16F Series Development Board.

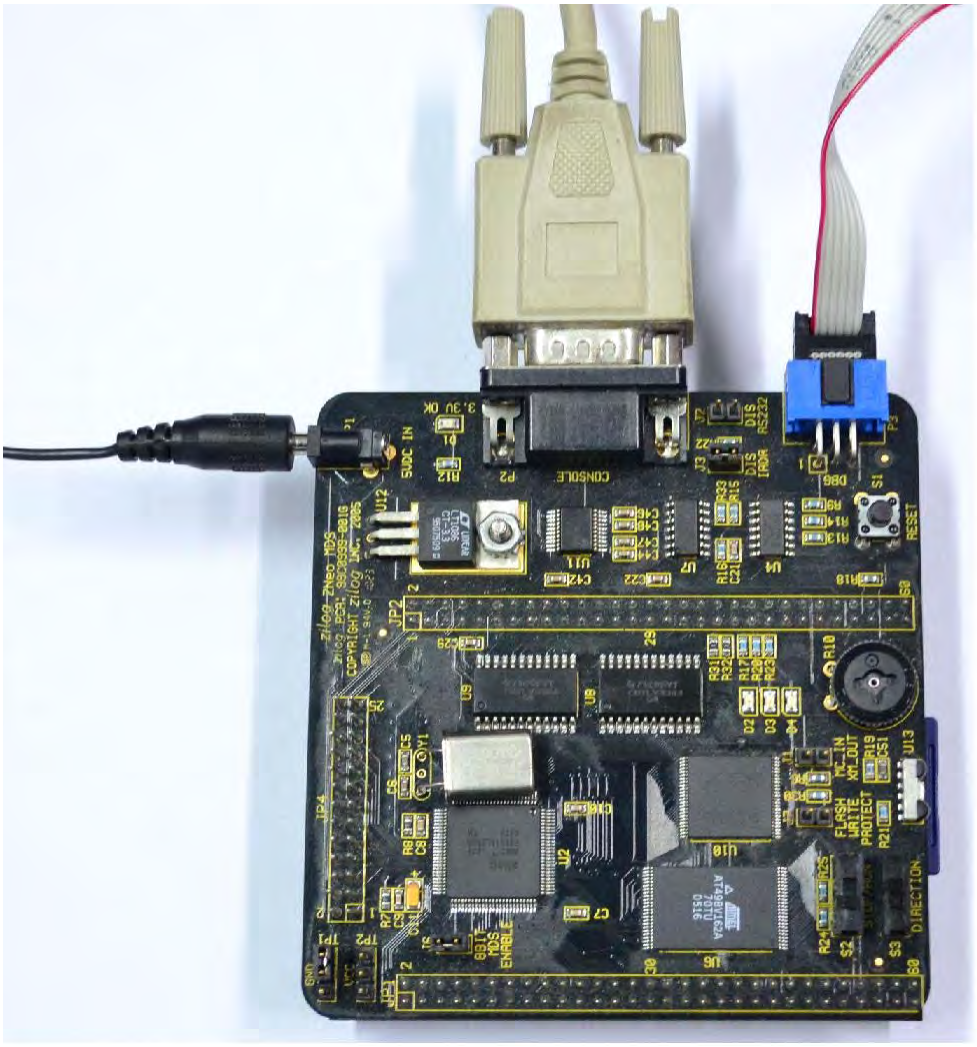

Figure 8. ZNEO Z16F Series Development Board with Hardware Connected

Enhanced Serial Peripheral Interface

All variants of the ZNEO family offer at least one Enhanced Serial Peripheral Interface (ESPI) port. Any of these ESPI ports can be used for SD card communication because each can be configured as a master. By manipulating the CPOL and CPHA bits, the relationship between the data and the clock phase is altered.

The ESPI allows the data exchange between the ZNEO Z16F Series MCU and other peripheral devices, such as an SD card. The ESPI is a full-duplex, synchronous, character-oriented channel which supports a four-wire interface. For a master, the valid options are *transmit only* or *transmit/receive*. For a slave, all options are valid. When a slave is operating in *Receive Only mode*, it will transmit characters as all 1s.

Universal Asynchronous Receiver/Transmitter

The ZNEO CPU contains two fully-featured UARTs with LIN protocol support. UART communication is full-duplex and capable of handling asynchronous data transfers. These UARTs support 8-bit and 9-bit data modes, selectable parity and an efficient bus transceiver driver enable signal for controlling a multitransceiver bus. UART0 is used to enable communication between the ZNEO CPU and the PC running HyperTerminal.

Software Implementation

The software required to run the SD card application includes the functions and routines that comprise the ESPI and UART blocks, the file system, and a number of SD commands.

ESPI Functions

To configure the ESPI interface on the ZNEO Z16F Series MCU, the design must consider the settings of the following four functions:

- ESPI enable/disable
- GPIO setting
- ESPI clock rate
- ESPI data size

Each of these functions is described in the following paragraphs.

ESPI Enable/Disable

To ensure a successful start, the ESPI should be disabled via the ESPICTL Register before configuring the GPIO pin settings and the clock rate settings, and must be reenabled after all configurations are performed.

GPIO Setting

The ESPI signals CLK, CS, MOSI, MISO must be configured via pin selection and the GPIO Register (PCAF) before configuring the SPI interface, as shown in the following code segment:

```
PCAF = (SPI_SS | SPI_SCK | SPI_MOSI | SPI_MISO); //GPIO Settings
```

ESPI Clock Rate

The ESPI clock rate can be set via the ESPIBR Baud Rate Generator, as shown in the following code segment:

```
void spi_set_baud(unsigned int divisor)
 {
  ESPICTL &= 0xBE;
  // Temporarily disable the SPI module
```

```
 ESPIBRH = divisor >> 8;
 ESPIBRL = divisor;
 ESPICTL |= 0x41;
 // Reenable SPI
}
```

ESPI Data Size

The Enhanced SPI data size is set to 8 bits for data transmission.

ESPI Routines

The ESPI-related routines used in this application include:

void spi_init(void); Initializes the ESPI interface by configuring the ESPIMODE, ESPIBRH and ESPICTL registers.

void spi_send_byte(unsigned char cspi_input); Sends 8-bit data to the SD card interface.

unsigned char spi_rcv_byte(void); Receives 8-bit data from the SD card interface.

For the user to enter names for files and directories and to view card statistics and other items listed in the menu, HyperTerminal must be configured to communicate with the LIN-UART block via the communication port (RS-232). The LIN-UART must also be configured to interface with HyperTerminal, as shown in the following code segment:

```
init_uart(_UART0, _DEFFREQ, _DEFBAUD); - this line set the UART
Frequency and Baud Rate

select_port(_UART0); - select UART 0
```

SD Commands

This application uses a number of SD commands, each of which is briefly described below.

CMD0. Reset the card to an idle state.

CMD8. When in an idle state, request the card to send the contents of its current operational condition as a response on the MISO line. Any negative response indicates that the SD card cannot be initialized correctly.

CMD55/ACMD41. Ask the card to start initialization, response is zero if initialization complete otherwise 0.

CMD58. Ask the SD card to send OCR, if CMD8 and CMD55/ACMD41 is successful, CMD58 response include HCS which indicate if the SD card is High Capacity or not.

CMD16

Set the block length (in bytes) for all the following block commands, both read and write. In the sample program, the data length is set to 512 bytes.

CMD17. Read a block of data that its size is determined by the CMD16 command.

CMD24. Write a block of data that its size is determined by the CMD16 command.

CMD32. Set the Start of the Block to be erase

CMD33. Set the Last Block to be erase

CMD38. Ask the card to start Block Erase specified by Start Block and Last Block.

The ZNEO Z16F Series MCU sends a series of commands to initialize the SD card, checks the status of the command response, then sends a series of commands to write/read data to and from the SD card.

SD Routines

This application uses a number of SD routines, each of which is briefly described below.

void sd_initialize_media(void); This routine is used to initialize the SD card by sending the CMD0, CMD8, CMD55/ ACMD41 and CMD58 commands in sequence using the following string:

```
void spi_send_byte(unsigned char cspi_input);
```

If the response to each command is successful, a nonzero value will result; a zero indicates failure.

int sd_write_media(unsigned long BlockAddress, unsigned char *buffer); This routine is used to send a block of data, based on the number identifying the block, to the SD card. To send the CMD24 command followed by the data and a checksum, use the following string:

```
int sd_read_block(sd_status *flag, unsigned long addr, unsigned
char *data)
```

If the response to the command is successful, a nonzero value will result; a zero indicates failure.

int sd_read_media(unsigned long BlockAddress, unsigned char *buffer); This routine is used to receive a block of data, based on the number identifying the block, from the SD card.

To send the CMD17 command to the SD card first, use the following string:

```
int sd_read_block(sd_status *flag, unsigned long addr, unsigned
char *data)
```

If the response to the command is successful, repeat the following command to read the data back from the SD card:

```
unsigned char spi_rcv_byte(void)
```

The return value indicates the status of the command and data response. A nonzero value indicates success; a zero indicates failure.

File System Routines

The file system manages how data is written to the SD card, following the DOS format. Managing the file system involves the following routines:

struct dir_Structure* FILE (unsigned char flag, unsigned char *fileName); This routine is called all below routines and functions, it is the center of the file system.

unsigned char read_File (unsigned char flag, unsigned char *fileName); This routine reads and views data from the file.

unsigned char read_Directory (unsigned char flag, unsigned char *fileName); This routine reads and views directory files.

void write_File (unsigned char *fileName); This routine creates a file with a given file name.

void create_Directory (unsigned char *fileName); This routine creates a directory with a given directory name.

void delete_File (unsigned char *fileName); This routine deletes a file with a given file name.

void delete_Directory (unsigned char *dirName); This routine deletes a directory with a given directory name.

void memory_Statistics (void); This routine displays card statistics, including its total capacity and amount of free memory via HyperTerminal.

unsigned char format_SD_Card (void); This routine formats the SD card to its default file system; only the volume name can be modified.

Test Procedure

Observe the following instructions to assemble, configure and test the SD card application.

Switch and Jumper Settings

This first section of the procedure addresses the settings for each switch and jumper on the ZNEO Z16F Series Development Board.

1. Remove the SD card, if any, from its slot in the SD Card Daughter Board.
2. Remove the shunts, if any, from jumpers J1, J2 and J7.
3. Use shunts to short (i.e, close the connections on) jumpers J3 and J6.
4. If LED D1 on the SD Card Daughter Board is illuminated (ON), slide Switch S3 to its opposite position to turn this LED off, as indicated in Figure 9.

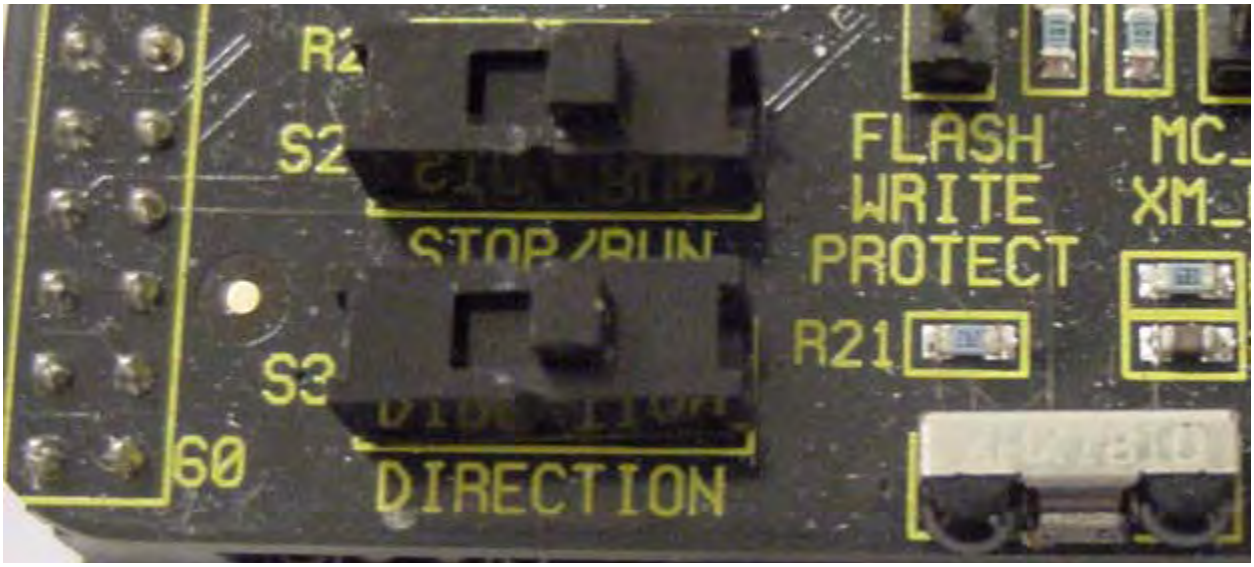

Figure 9. Correctly Positioned S2 and S3 Switches

Notes: When viewing the ZNEO Z16F Series Development Board with the UART (P2) and debug (P3) connectors on your right, the position of the S3 switch will be to the right (see Figure 9 to determine the correct setting). To verify that Switch S3 is set correctly, and without an SD card inserted, LED D1 on the SD Card Daughter Board should not illuminate.

To avoid potential confusion, bear in mind that the LED D1 power indicator on the ZNEO Z16F Series Development Board will be illuminated when power has been applied. Conversely, LED D1 on the SD Card Daughter Board will be illuminated either when an SD card is active in its slot or when the position of the Switch S3 is set to the left.

5. Ensure that Switch S2 is set to the same position as S3.

Install ZDSII for ZNEO

This section of the procedure guides you through the process of installing ZDSII – ZNEO on your host PC.

1. Download ZDSII – ZNEO v5.0.1 and install it on your host PC.
2. Download the zipped AN0320-SC01 file from its location on the Zilog website at http://www.zilog.com/docs/appnotes/AN0320-SC01.zip. Save this file and unzip it to an appropriate location on your host PC.
3. Launch ZDSII. In the ZDSII menu bar, navigate via the **File** menu to **Open Project**; the **Open** dialog box will display. Browse to the AN0320-SC01 folder that you just downloaded, look inside this folder for the `.zdsproj` file, and click **Open**.

4. After the project is open, select **Settings** from the **Project** menu. In the **Code Generation** panel, ensure that the **Limit Optimization** checkbox is selected and that **Memory Model** is set to **Small**, as shown in Figure 10.

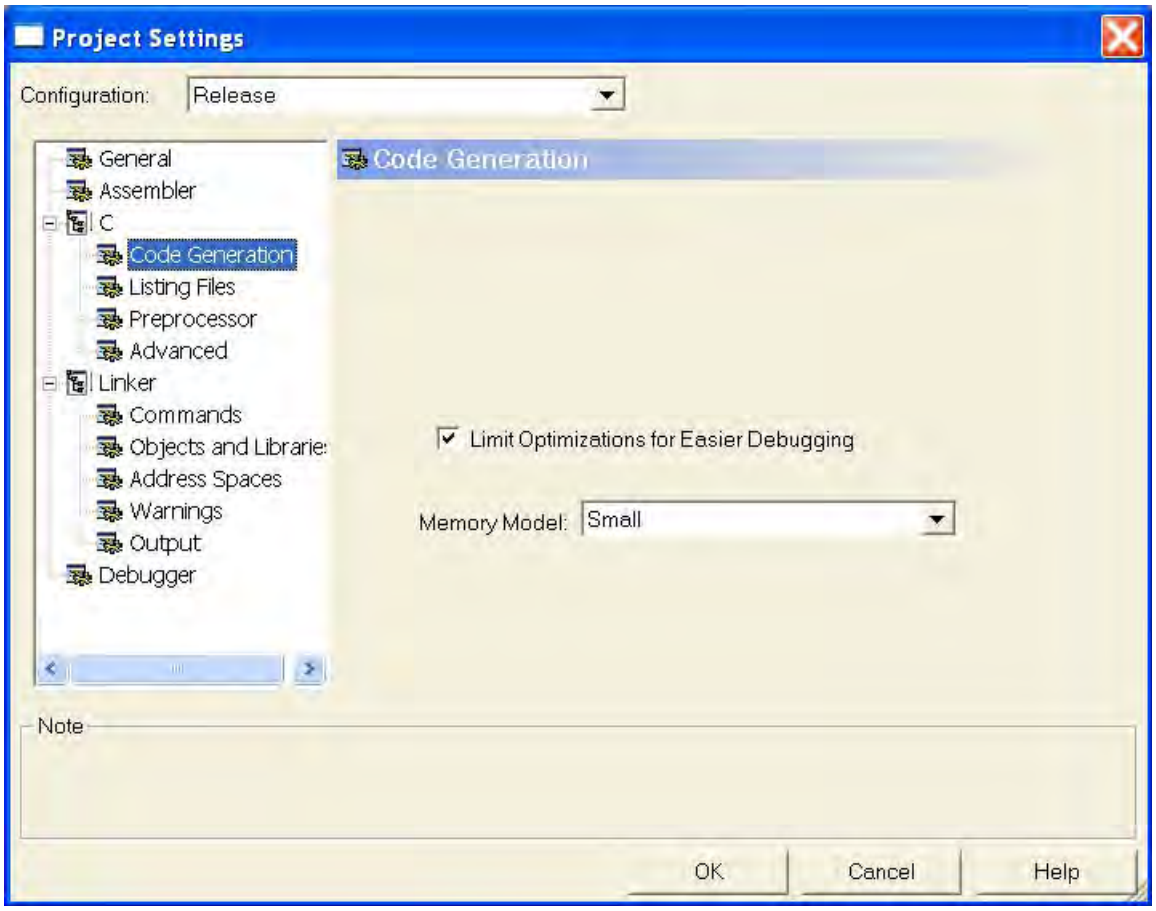

Figure 10. Code Generation Panel Settings

5. In the **Debugger** panel of the **Settings** window, select the **Z16F2811AL** checkbox. Next, above the Debug Tool pane, click **Setup** to launch the Configure Target dialog box. In this dialog, ensure that the Clock Source is set as **External** and that the Frequency is set to **20.00000 MHz**. In the External Bus Interface pane, select **8-Bit**, as shown in Figure 11.

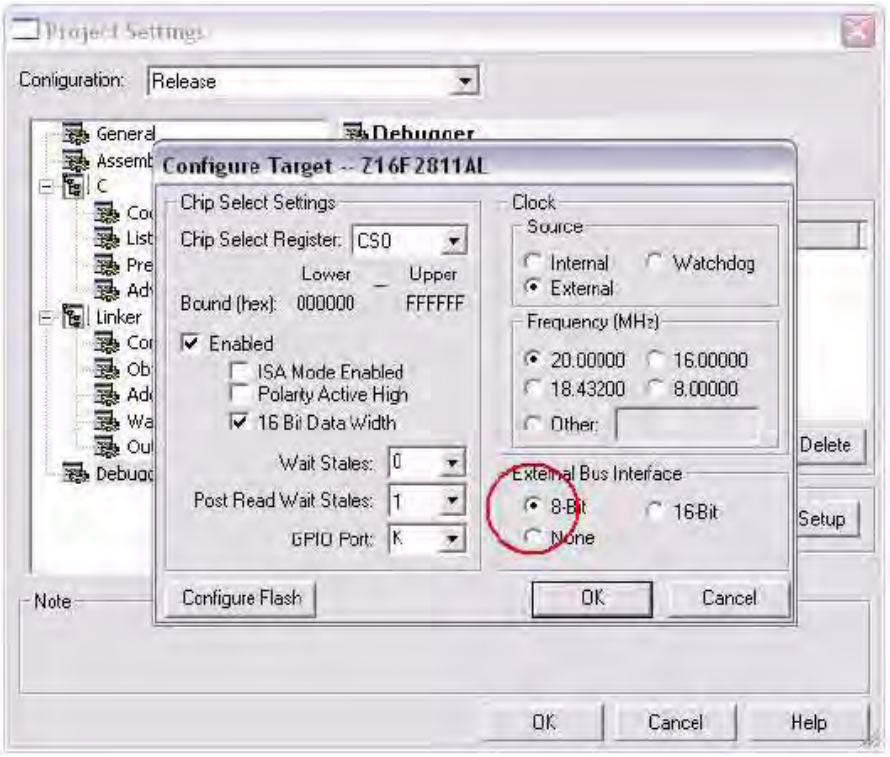

Figure 11. External Bus Interface

6. Click **Configure Flash** to open the Target Flash Settings dialog box. In this dialog, select the **Internal Flash** checkbox and the **Automatically Detect Device** checkbox, as shown in Figure 12.

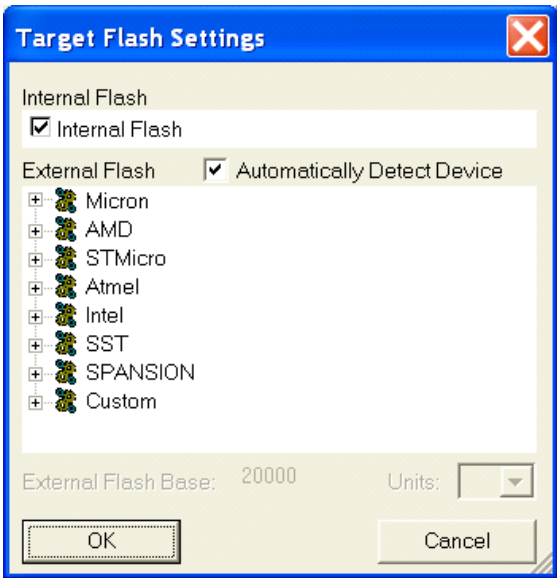

Figure 12. Target Flash Settings

7. In the Object and Libraries panel of the Project Settings window, in the C Startup Module pane, select **Standard**. In the **Use Default Libraries** pane, select the **C Run-time Library** checkbox. From the **Floating Point Library:** drop-down menu, select **Real**. See Figure 13.

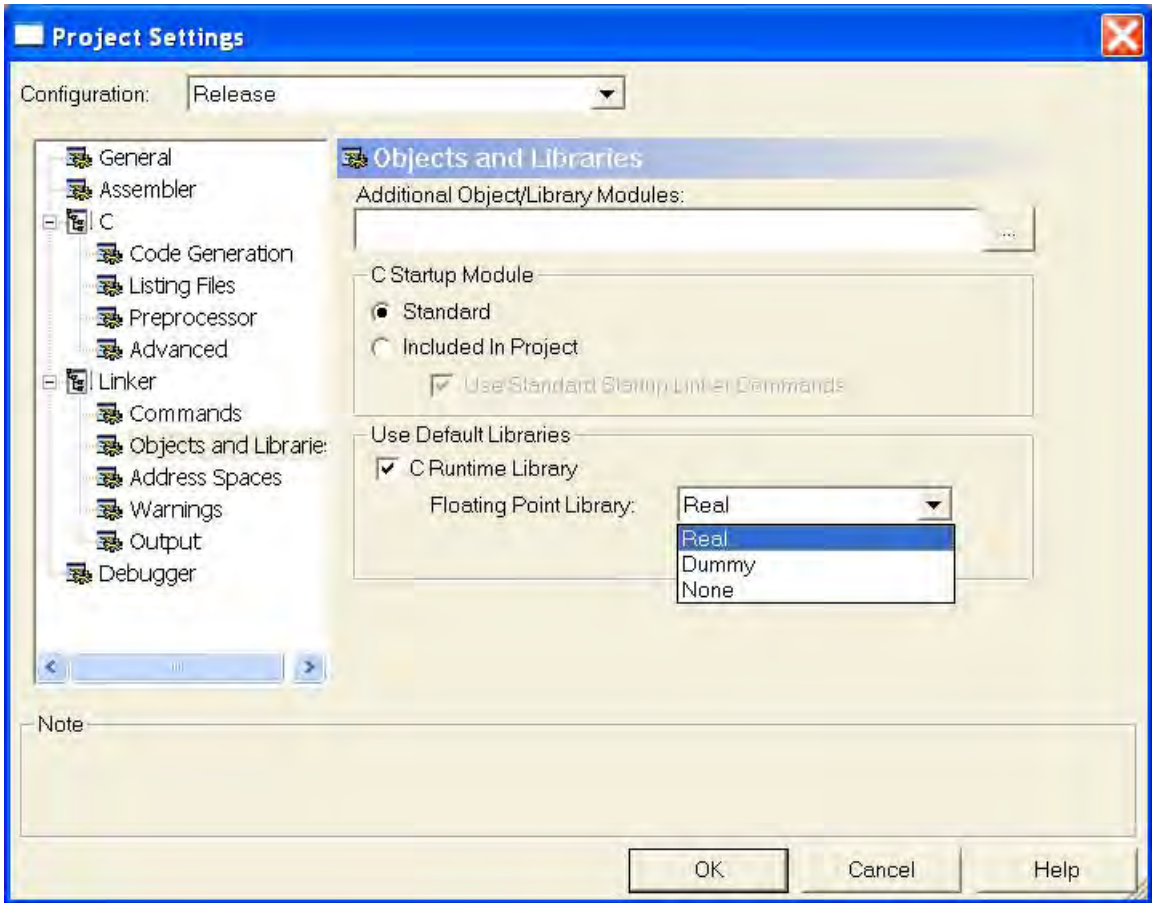

Figure 13. Objects and Libraries Settings

8. In the ZDSII workspace area, click **External Dependencies**, then double-click the `SIO.h` file. Ensure that `_DEFBAUD` is set to 57600.
9. From the **Build** menu, choose **Rebuild All** to build the code and load it into the ZNEO Z16F Series MCU.

Observe the following instructions to configure HyperTerminal.[1]

1. To launch HyperTerminal, navigate via the PC's **Start** menu to **All Programs** → **Accessories** → **Communications** → **HyperTerminal**. Configure HyperTerminal to reflect the settings shown in Figure 14.

1. This SD card application was tested using HyperTerminal running on a Windows XP SP3 system.

Figure 14. HyperTerminal Properties

2. Use a serial cable to connect the ZNEO Z16F Series Development Board to the serial port of the host PC.
3. Insert an SD card into its slot on the ZNEO SD Card Daughter Board and connect this Daughter Board to the Development Board.

4. Power up the ZNEO Z16F Series Development Board and wait a moment for the initialization screen to appear in the HyperTerminal window, as shown in Figure 15.

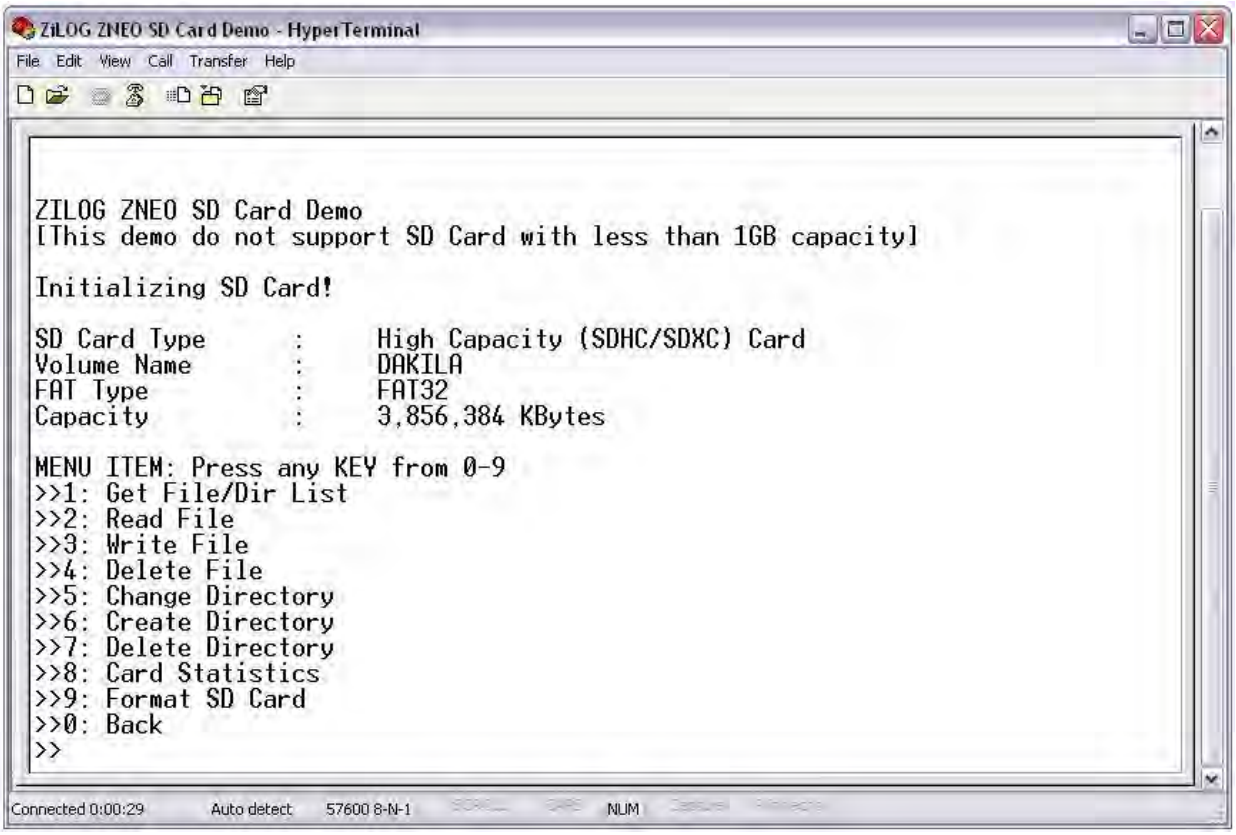

Figure 15. SD Card Initialization Screen in HyperTerminal

5. To view the list of files contained in the root directory (or any folder), press the 1 key on your PC's keyboard. A list of files and directories will appear in the HyperTerminal window; see Figure 16.

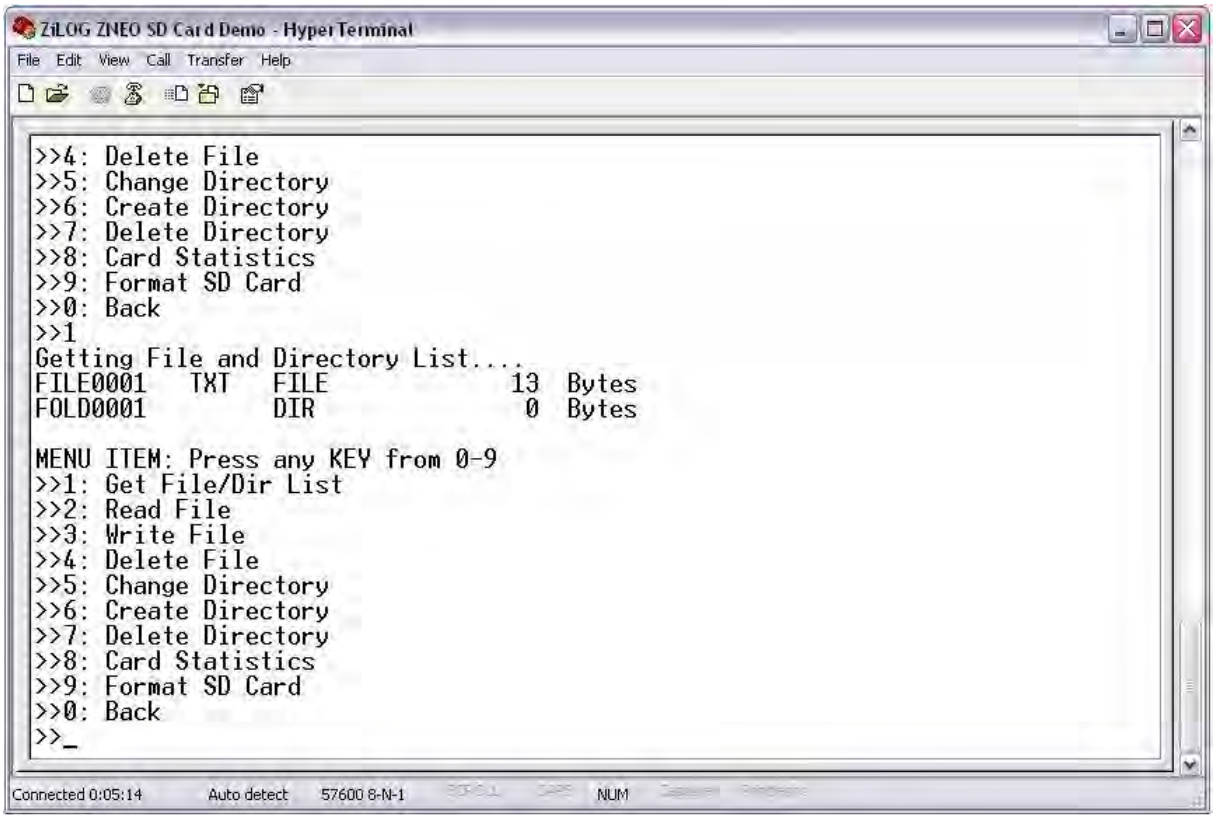

Figure 16. Getting a File and Directory List

6. To read a specific file, enter a 2 on the keyboard. At the **Enter File or Directory Name** prompt that appears, enter the name of the file you want to read. The name and extension of the file must be separated by a period (.); e.g., `sample.txt`. See Figure 17.

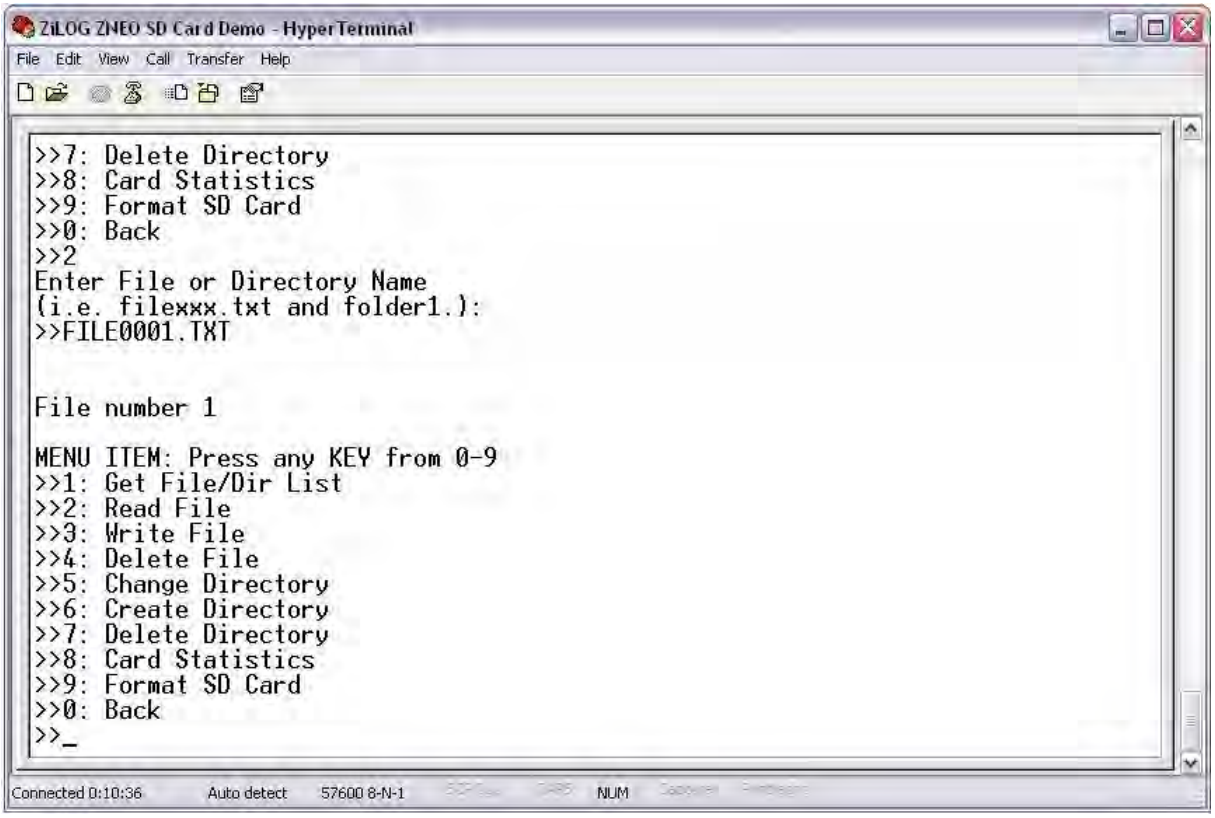

Figure 17. Reading a File

7. To create or append a file, enter a 3 on the keyboard. At the **Enter File or Directory Name** prompt, enter the name of the file (including its extension and following the format described in Step 6) and press the Enter key. A screen similar to Figure 18 will appear.

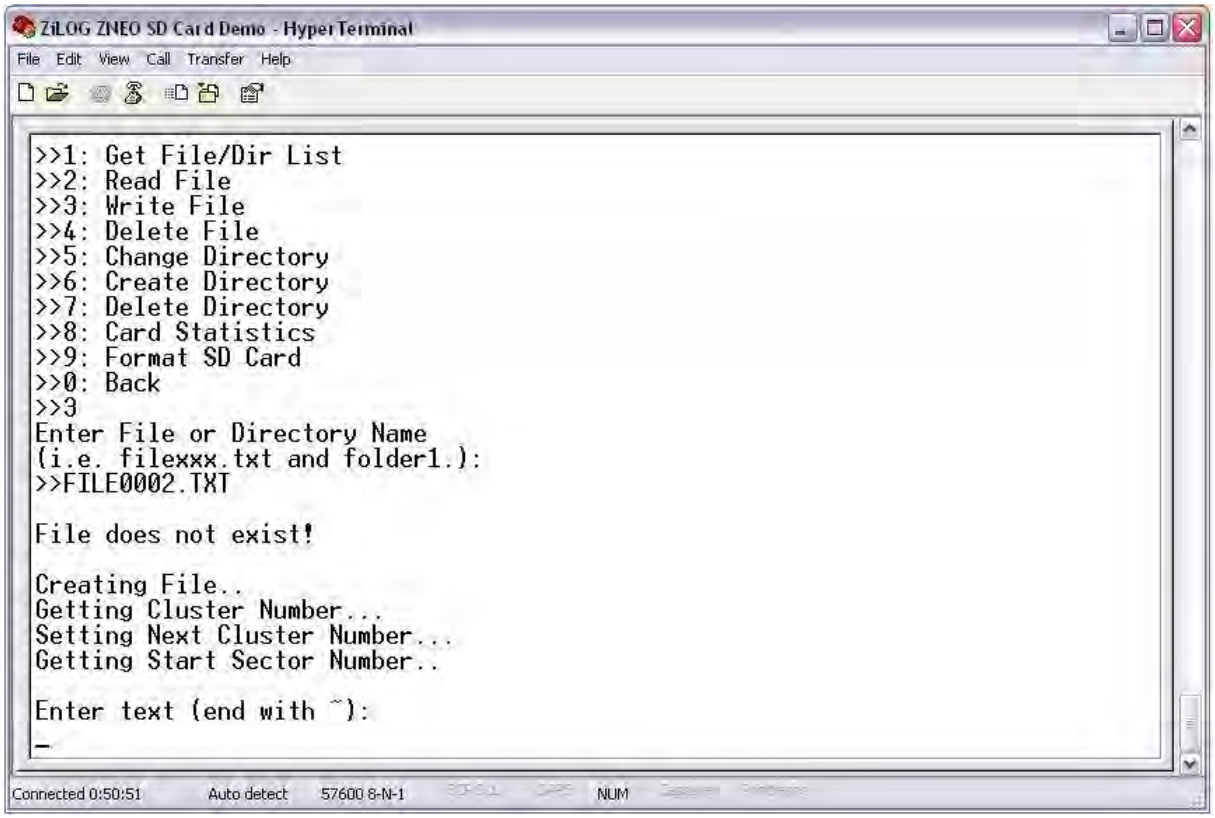

Figure 18. Creating or Appending a File

8. Enter a text string to create or append to a file, and end this string with the tilde character (~). Press the Enter key to view a result similar to the screen shown in Figure 19.

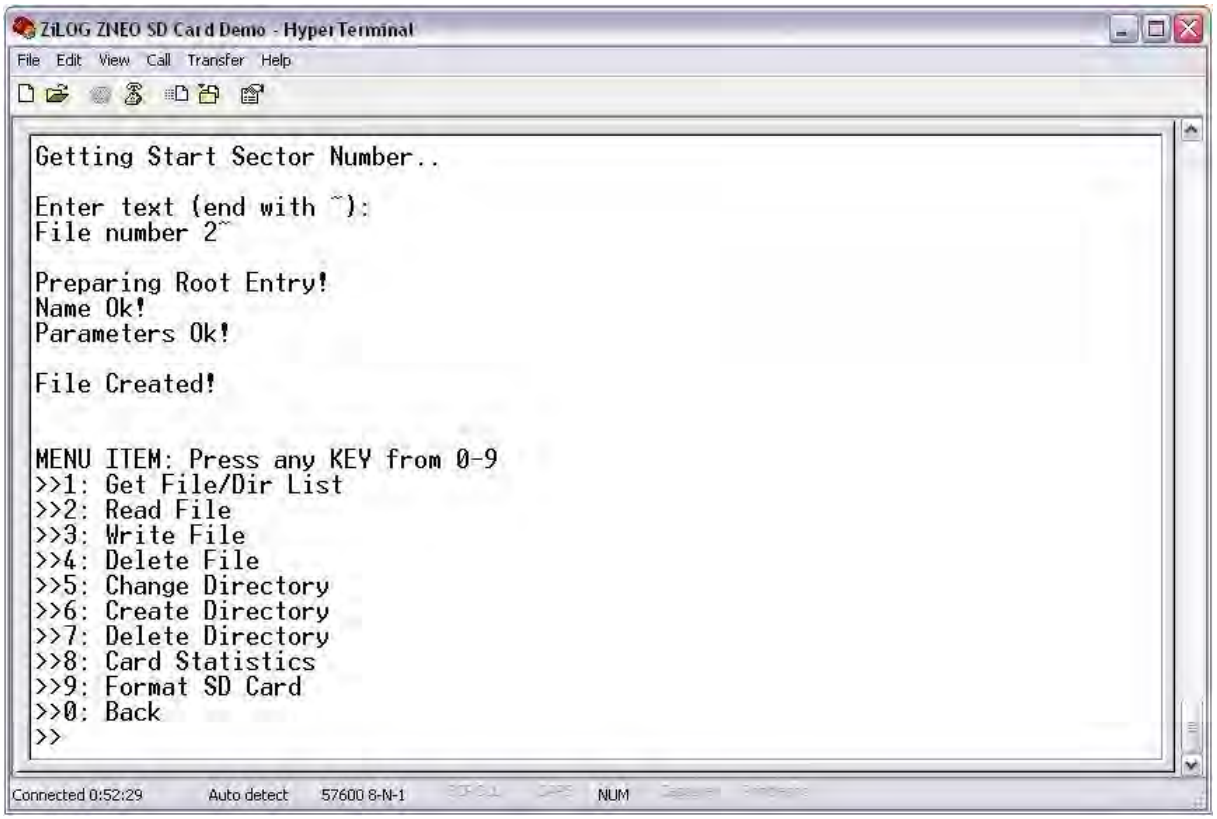

Figure 19. Appending a Text String

9. To delete a file, enter a 4 on your keyboard. At the **Enter File or Directory Name** prompt, enter the name of the file you want to delete and press Enter key. Press the Enter key to view a result similar to the screen shown in Figure 20. Be sure to wait for the confirmation that your file was deleted, as shown in Figure 21.

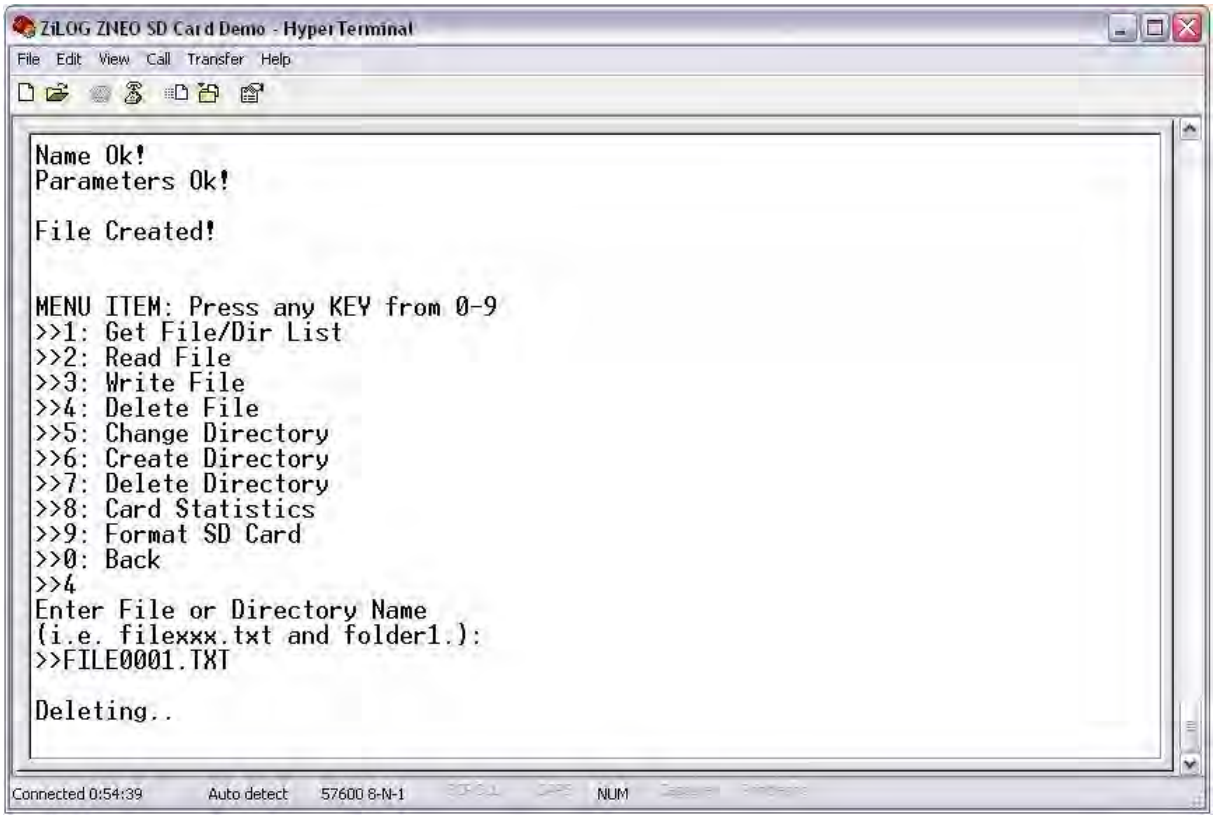

Figure 20. Deleting a File, #1 of 2

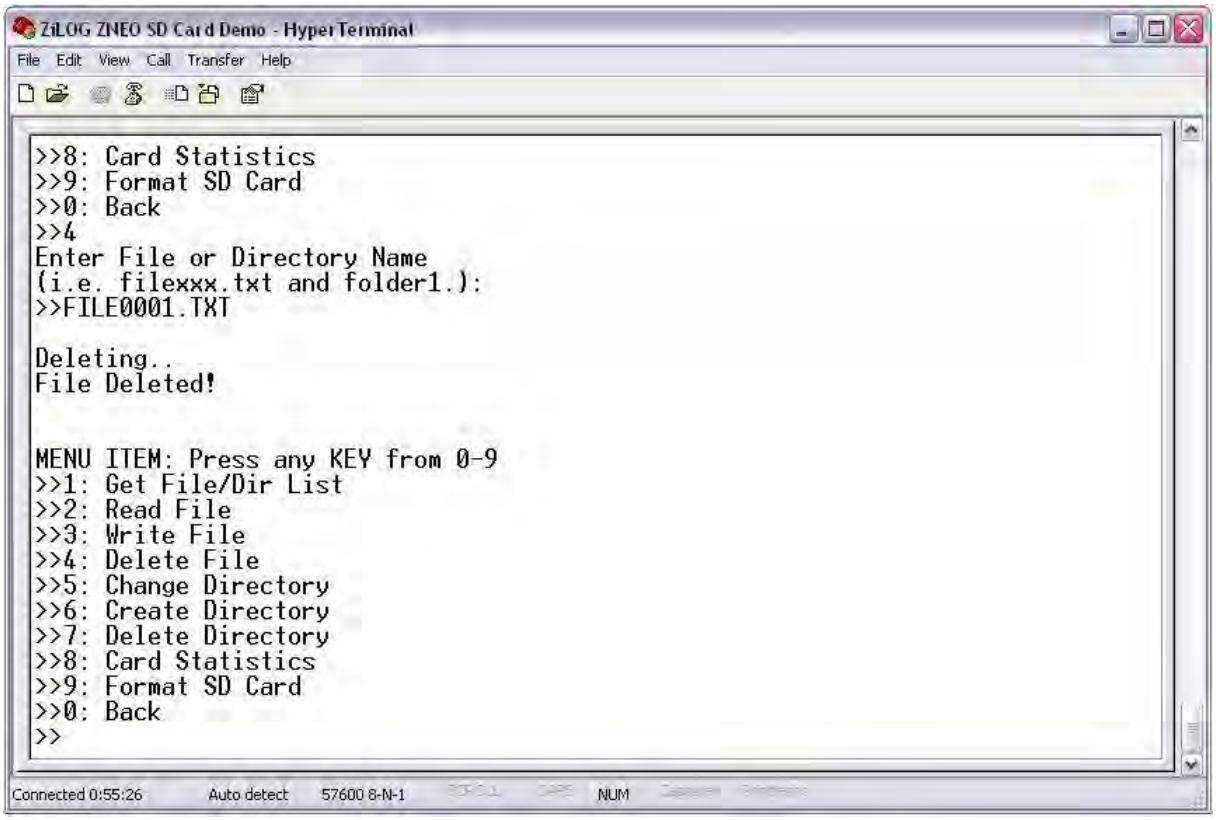

Figure 21. Deleting a File, #2 of 2

10. To view card statistics, enter an 8 on your keyboard. Total Capacity and Free Memory data will appear in the HyperTerminal window, as shown in Figure 22.

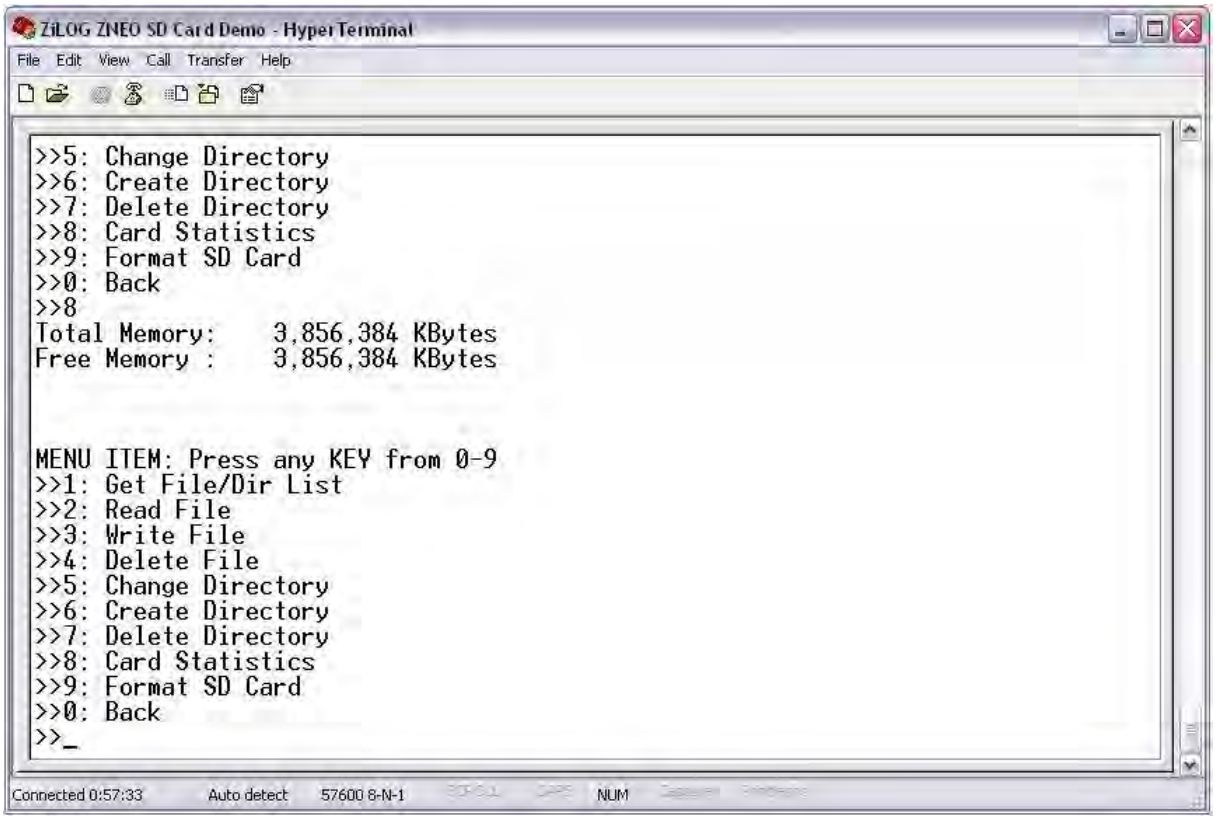

Figure 22. Viewing Card Statistics

11. To format an SD card, enter 9 on your keyboard. At the **Format SD Card?** prompt, enter a 1. When the open prompt appears, enter the volume name and press the Enter key.
12. At the **Enter Volume Name:** prompt, enter the name of the newly-formatted volume, and press Enter. When formatting is complete, the system responds as shown in Figure 23.

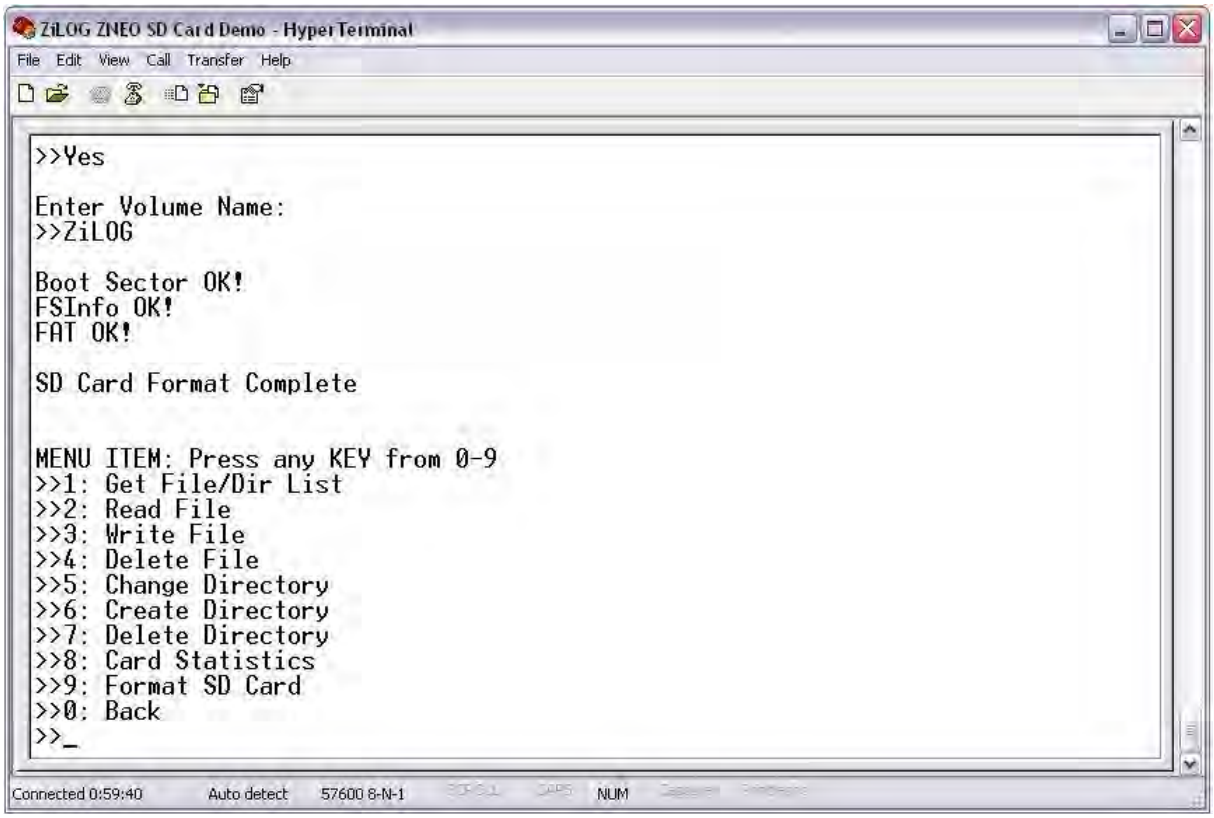

Figure 23. Formatting an SD Card

13. To create a directory, enter a 6 on the keyboard. At the **Enter File or Directory Name** prompt, enter the name of the directory, end this name with a period (see the `FOLDER01` example shown in Figure 24), and press the Enter key. The result is shown in Figure 25.

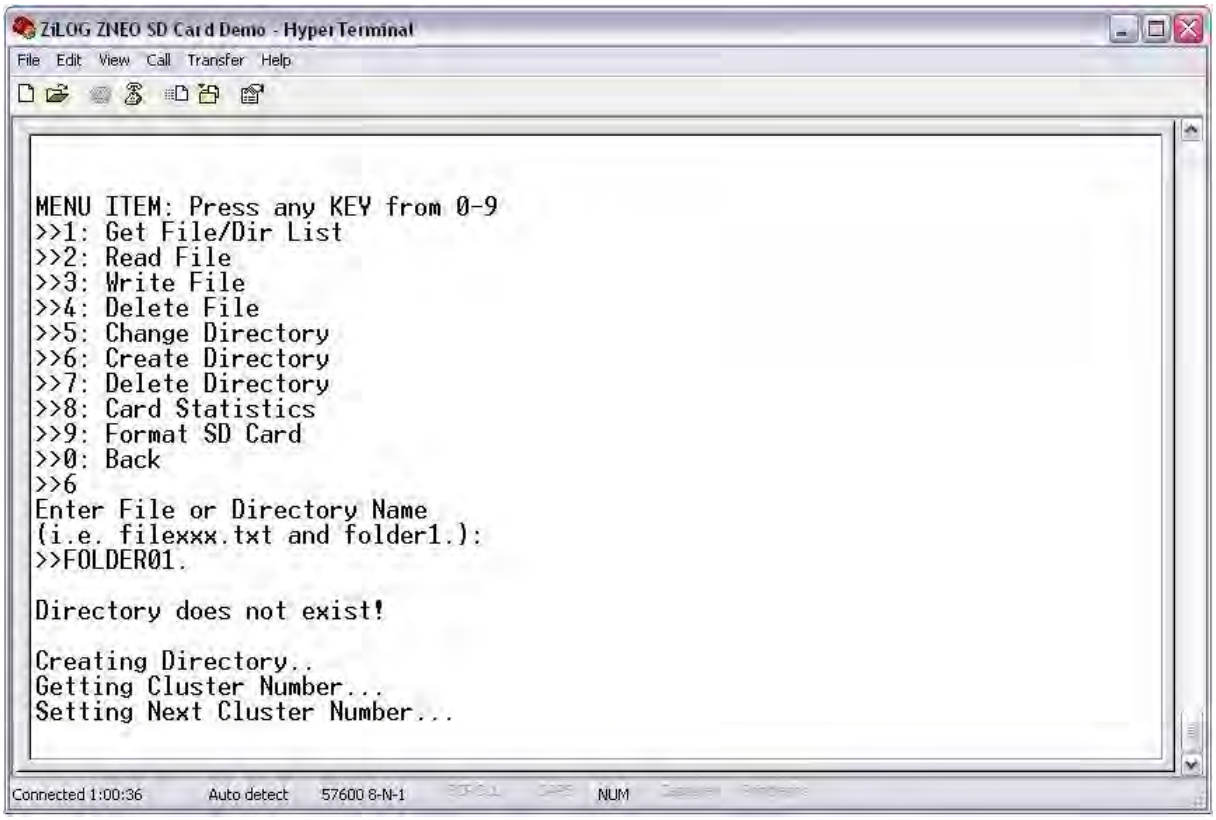

Figure 24. Creating a Directory, #1 of 2

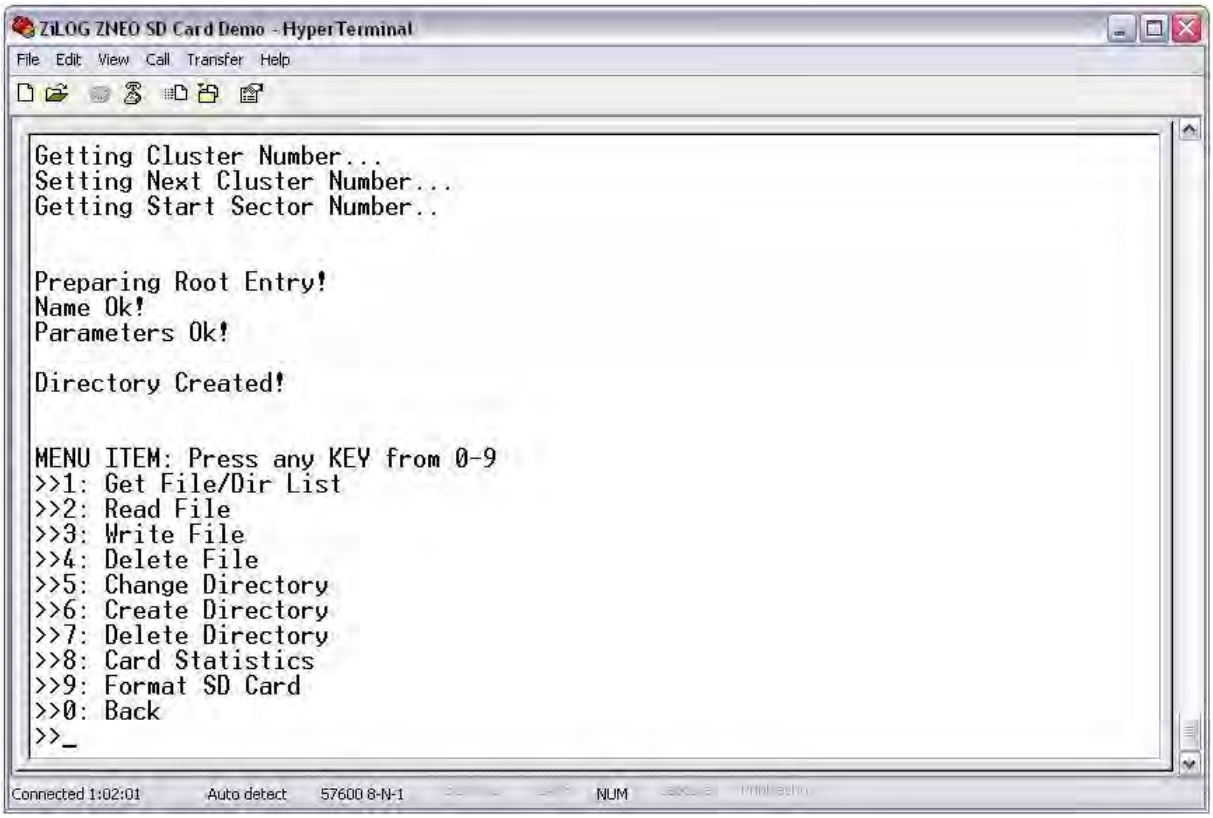

Figure 25. Creating a Directory, #2 of 2

14. To delete a directory, enter a 7 on your keyboard. At the **Enter File or Directory Name** prompt, enter the name of the directory, end this name with a period, and press the Enter key to view a result similar to the `FOLDER01` example shown in Figure 26. Be sure to wait for the confirmation that your file was deleted.

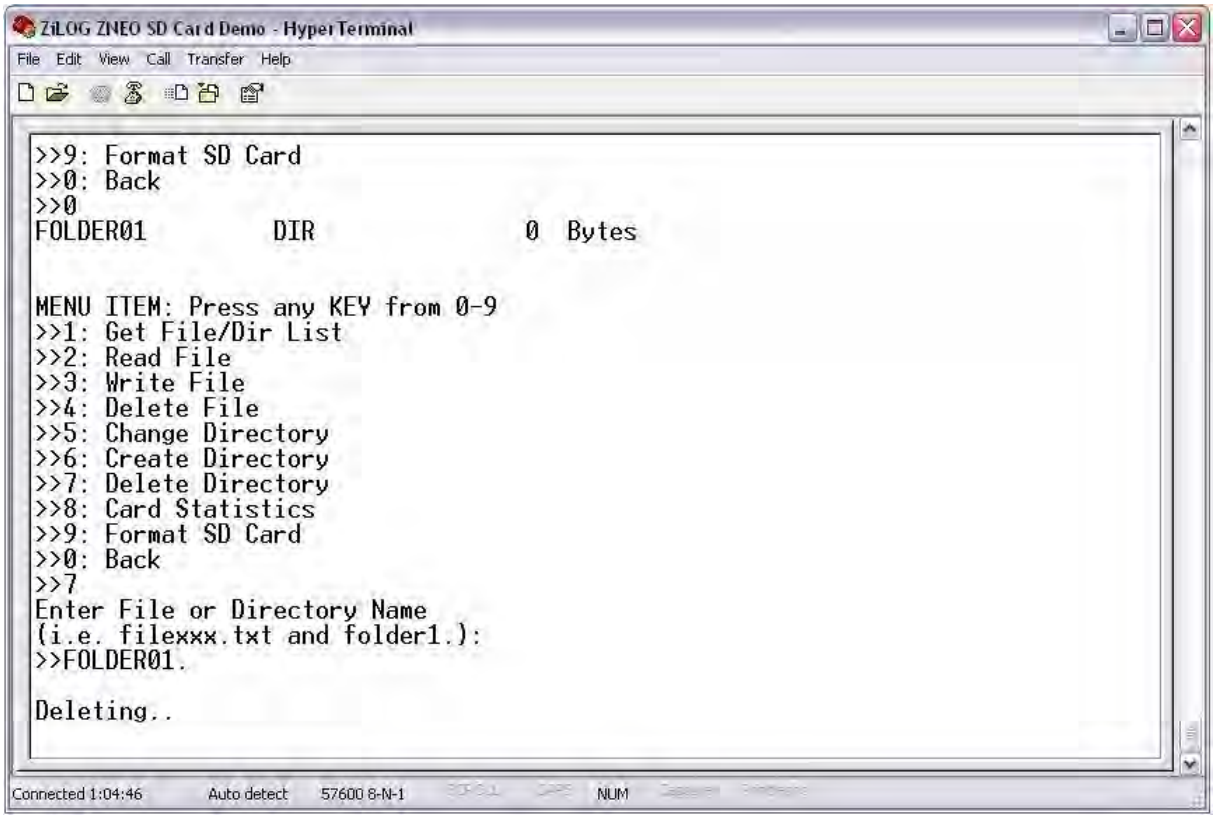

Figure 26. Deleting a Directory, #1 of 2

15. To change directories, enter a 5 on your keyboard. At the **Enter File or Directory Name** prompt, enter the name of the directory and press the Enter key to view a result similar to the example shown in Figure 27.

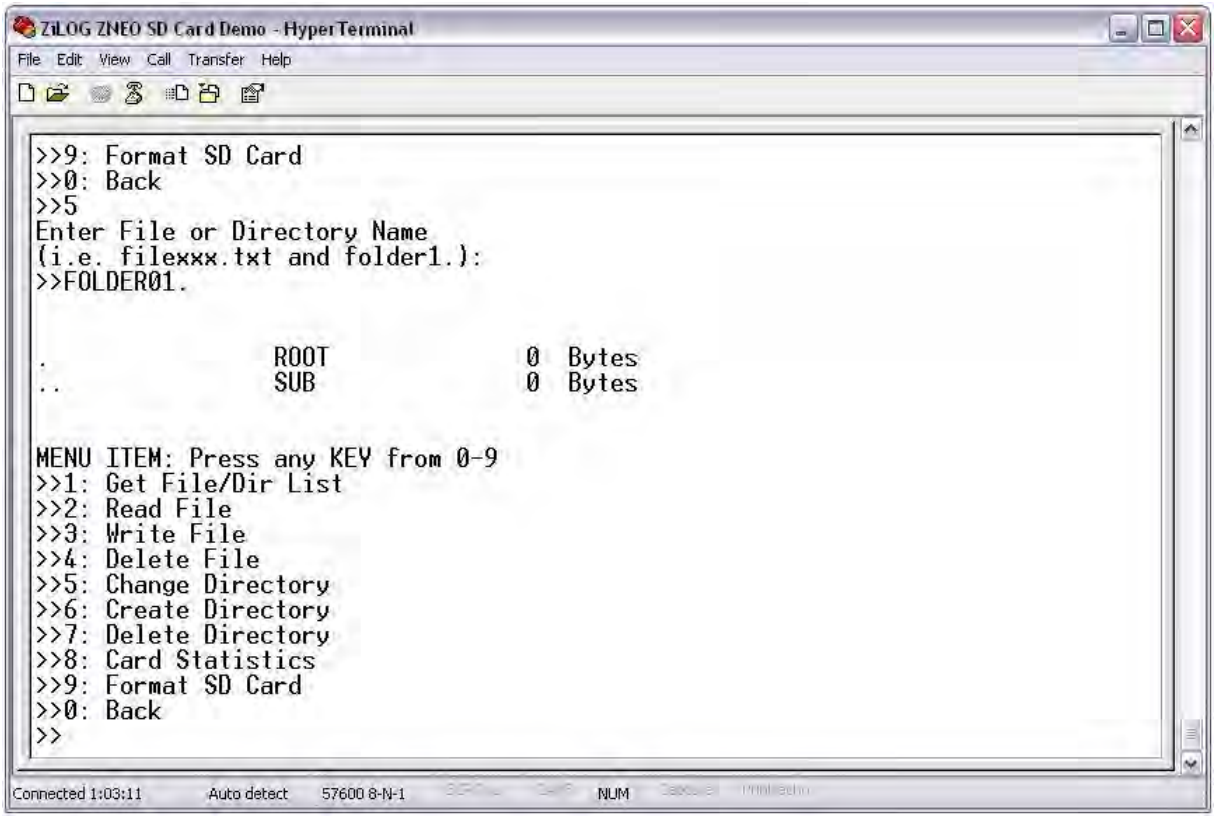

Figure 27. Changing a Directory

16. To return to the previous directory, enter a 5 on your keyboard. At the **Enter File or Directory Name** prompt, enter the name of the directory and press the Enter key to view a result similar to the screen shown in Figure 28.

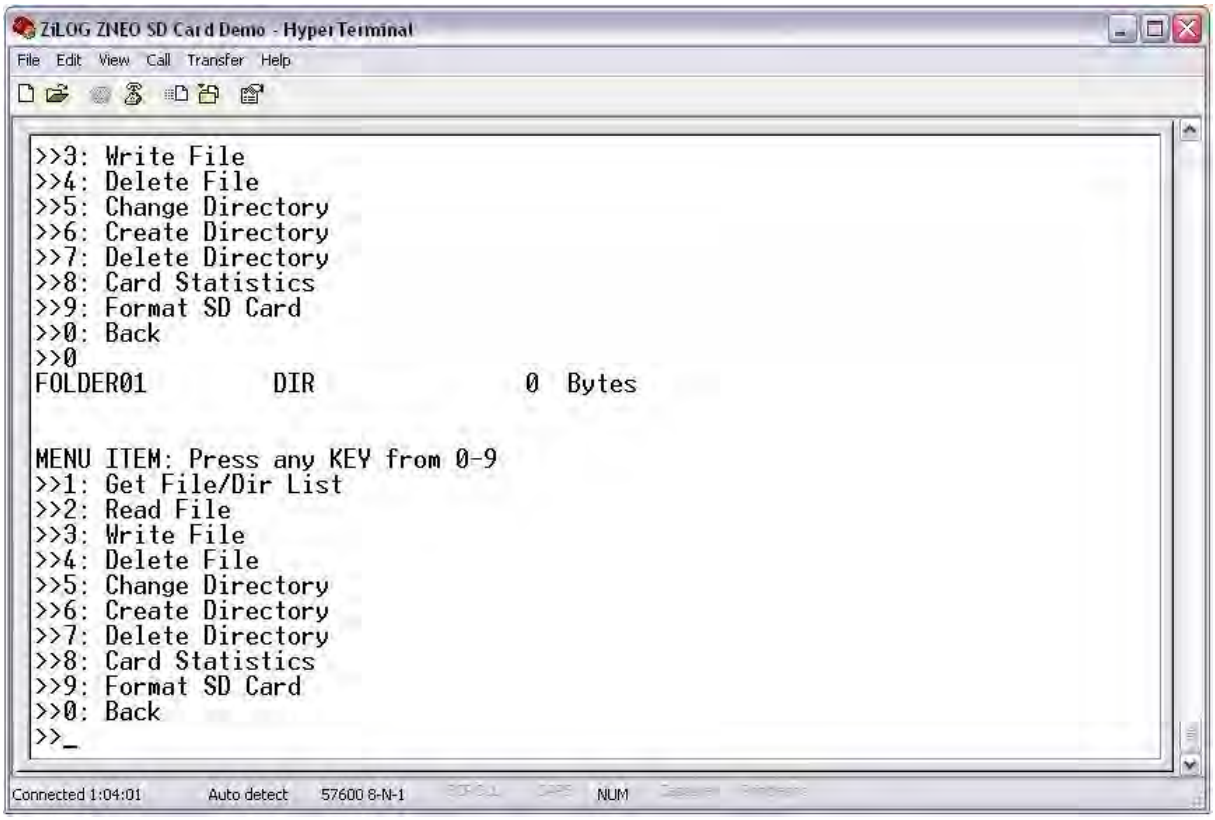

Figure 28. Returning to the Previous Directory

Results

This application was tested using a 2GB Sandisk/KingCom SDSC card and a 4GB PQI/ Transcend/TEAM SDHC card on both Windows XP SP3 and Windows 7 systems. Upon testing, the results are:

- The card could be removed while running without problems
- The card could be read by a standard PC as a removable disk regardless of whether it was formatted to FAT16 or to FAT32
- The card could be reinserted into the PC's SD card reader slot with no problems; the file system resumed operations
- A new SD card could be inserted without errors
- Newly created files and directories could be read by the PC without errors
- Directories were created up to 5 levels (128 levels is the maximum)

Summary

The application developed for this Application Note document takes advantage of ZNEO Z16F Series MCU's advanced ESPI features to provide a fast, low-overhead solution for interfacing its SPI block with an SD card quickly and inexpensively. Using SPI protocols allows for a flexible interface between SD card storage media and the ZNEO CPU. The software is modular and easy to customize for any application that requires a secondary storage solution. In effect, this application demonstrates the ability to provide cost-effective removable storage using the ZNEO Z16F Series MCU.

Technical Information

FAT File System (Microsoft TechNet website)

How FAT Works (Microsoft TechNet website)

SD Specifications, Part 1: Physical Layer Simplified Specification Version 3.01 (published by the SD Card Association)

Microsoft Extensible Firmware Initiative FAT32 File System Specification (published by the University of Washington)

FAT16 File System Specifications (website)

FAT16 File System Disk - FAT16 Subdirectory Management (website)

References

The following documents describe the functional specifications of Zilog's Z16F ZNEO Series of MCUs.

- ZNEO CPU Core User Manual (UM0188)
- ZNEO Z16F Series Product Specification (PS0220)
- ZNEO Z16F Series Development Kit User Manual (UM0202)

Appendix A. Schematic Diagram

Figure 29 is a schematic diagram showing how the SD card is connected with the ZNEO MCU.

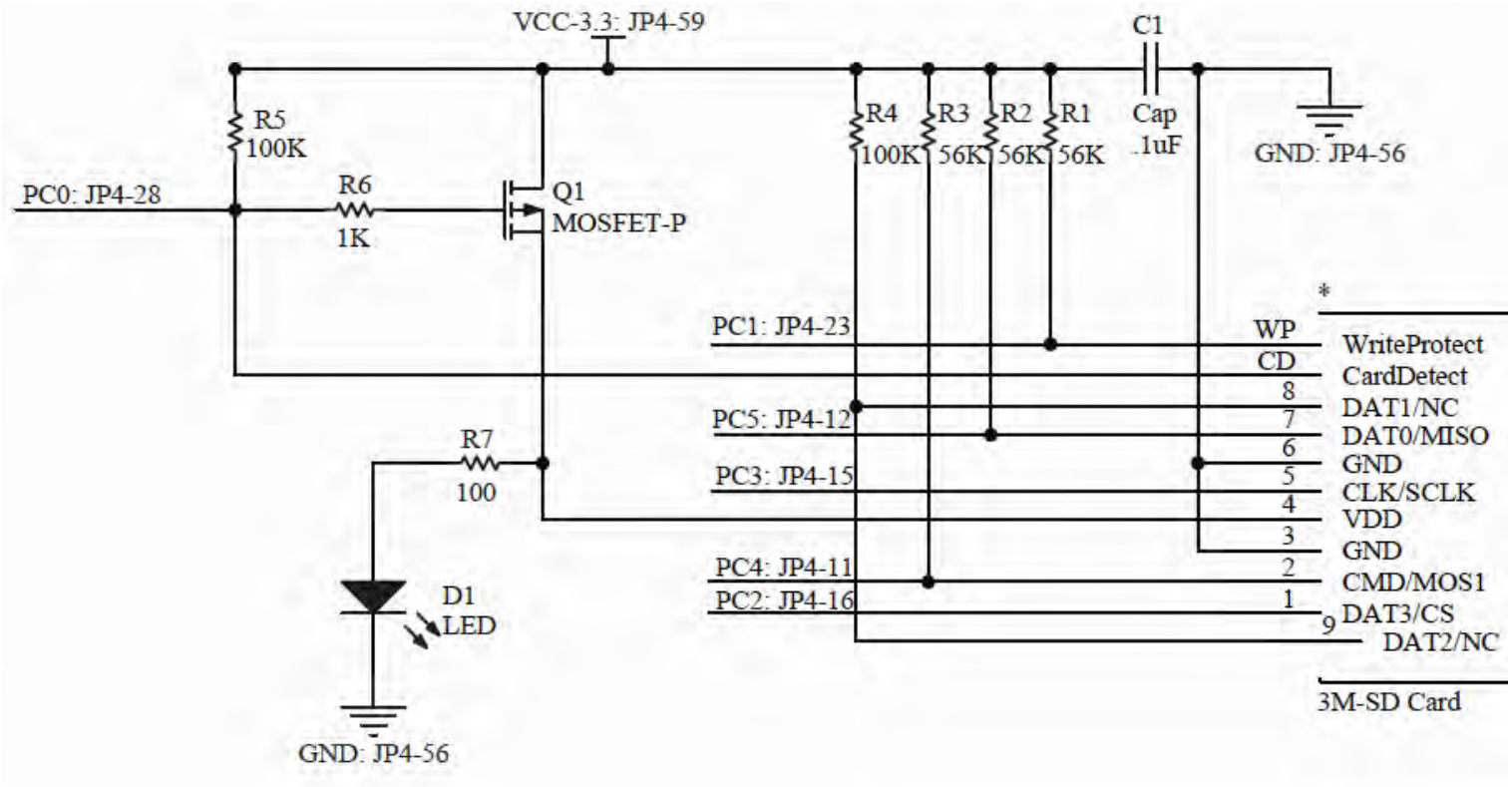

Figure 29. SD card to ZNEO MCU Interface

Appendix B. Flow Charts

The schematic diagram in Figure 30 shows the main function for implementing an SD card using ZNEO Z16F Series MCUs.

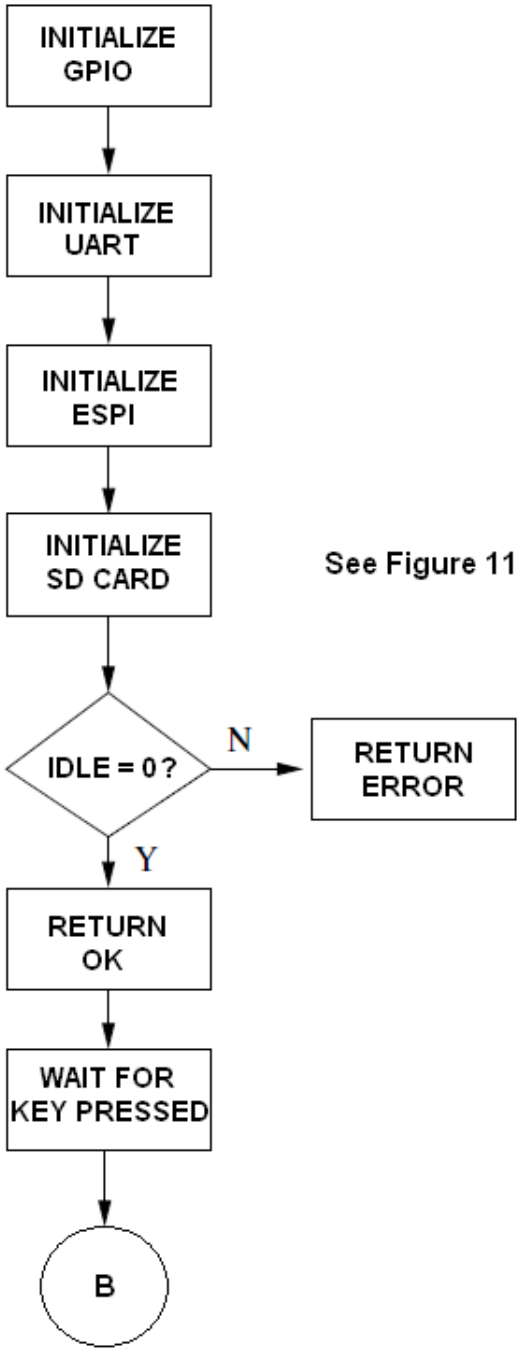

Figure 30. Flow of the Main Function

The schematic diagram in Figure 31 shows the flow of the initialization sequence.

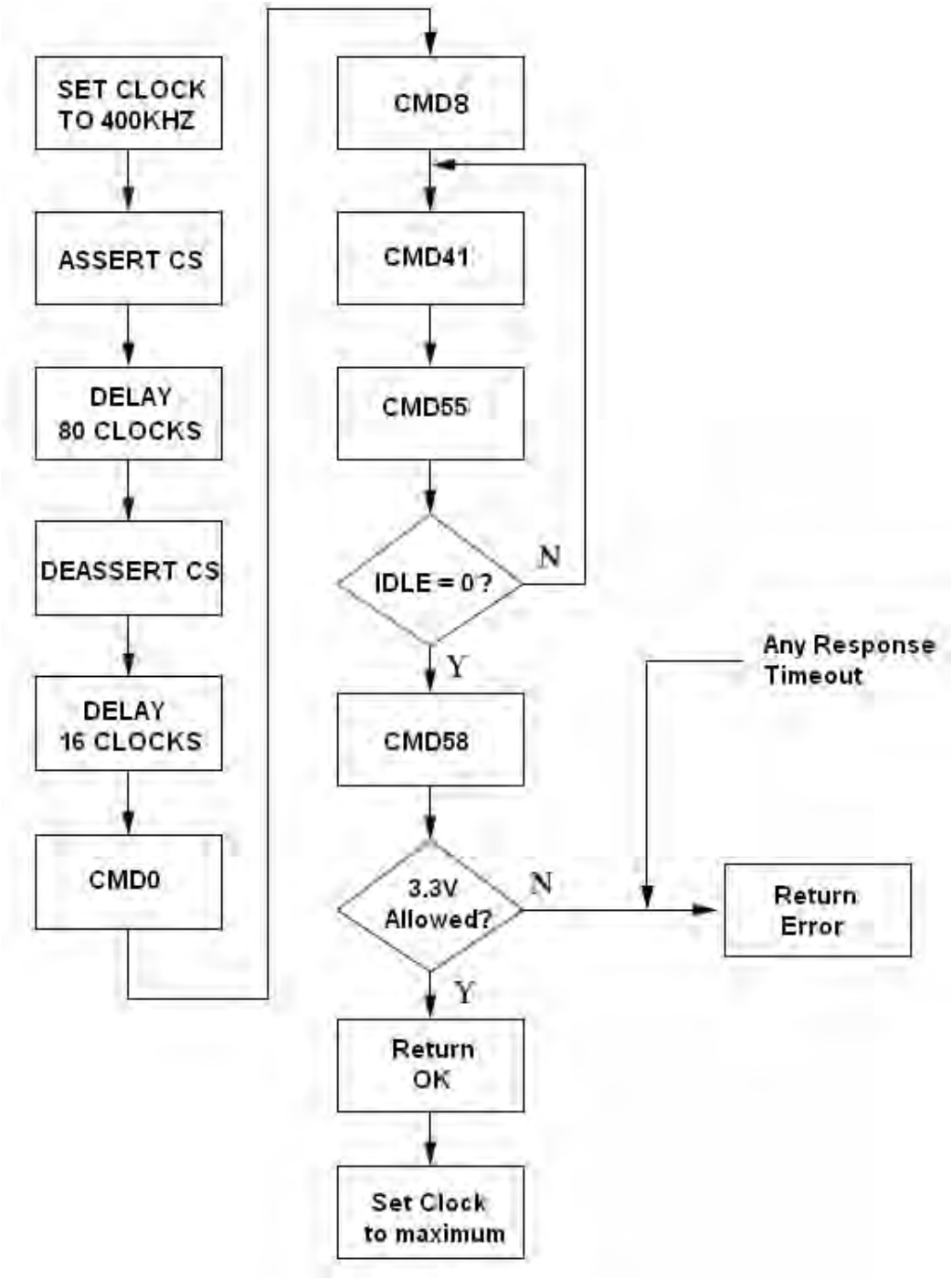

Figure 31. Flow of the Initialization Sequence

The schematic diagram in Figure 32 shows the flow of the SD card's input/output operations.

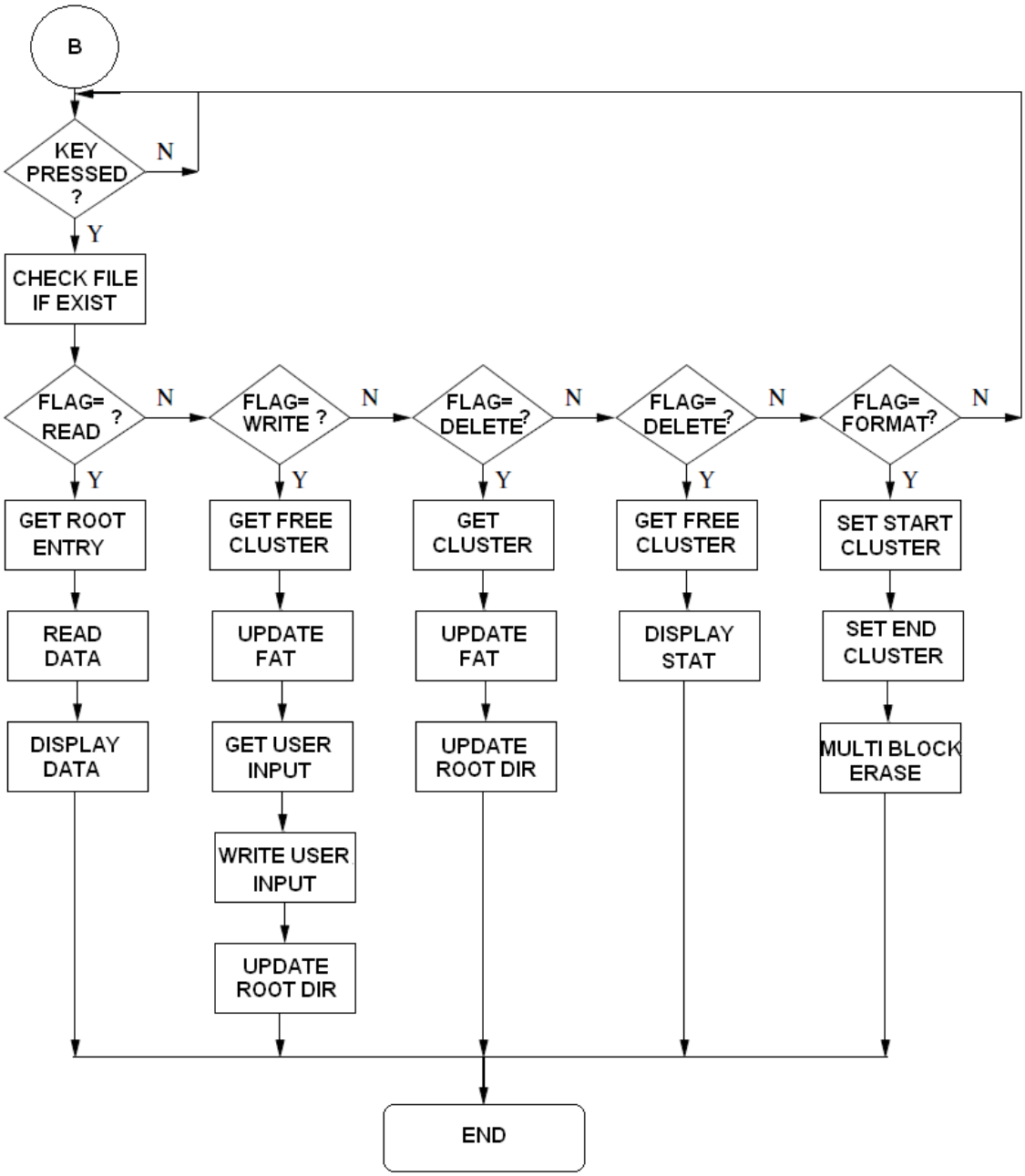

Figure 32. Flow of the SD Card I/O Operation

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.