AN2260
Rev. 2, 11/2004

Fast I/O Library for the MSC8101ADS Using Ethernet Communication

By Madalin Stoica

This application note describes a fast I/O library for the Freescale MSC8101ADS board developed using Metrowerks® CodeWarrior® for StarCore™. This fast I/O library gives file access to applications running on the MSC8101ADS board during testing and debugging. It is in effect a virtual file system that allows the MSC8101ADS to access a PC file system through its Ethernet port as if it were its own. The MSC8101ADS accesses files and manages the communications with the PC through its Fast Ethernet link.

Like any other standard C language compiler, Metrowerks CodeWarrior provides an I/O library that supports working with files. Because the MSC8101ADS does not work with files in real applications, the I/O library is used only for testing and debugging purposes. This standard library works via the JTAG interface so that no I/O port is locked, and the user can run programs working both with files and any other MSC8101ADS I/O port. Although this standard I/O library is very flexible in its combination of a JTAG port with parallel communication, the I/O operations are slow. Working with huge files becomes difficult, and much time is consumed on loading and storing data rather than on the computation procedures of interest. Therefore, a really fast I/O library such as the one discussed in this application note is highly beneficial during testing and debugging. Using the fast I/O library instead of the standard C library speeds up the reading/writing processes in projects more than 1500 times. This document discusses the features and advantages of the fast I/O library, as well as the required hardware set-up.

CONTENTS

The fast I/O library has repeatedly demonstrated its efficiency during the testing of vocoders designed for StarCore DSPs. Testing that required more than 72 hours using the standard C language library completed in less than three minutes using the fast I/O library. One of the biggest advantage of the fast I/O library is ease of use. C programmers who are familiar with the standard I/O library can easily switch to the fast I/O library for the following reasons:

- Its interface is very similar to that of the standard I/O library.
- Function names are similar.
- The behavior of all functions is identical to that of the corresponding standard functions.

The similarities in function names and the perfect matching of the behaviors are very important in translating a standard application into a new one based on an advanced library. Using the Ethernet-oriented fast I/O library as a starting-point, you can develop a similar fast I/O library based on an ATM or RS-232 connection.

1 Architecture

The fast I/O library is based on a client-server model in which an application operating on an MSC8101ADS as the client works with a Win32 application operating on a PC-based server that can access the PC file system directly. The client application working on the MSC8101ADS board uses the Ethernet capabilities of the board to send requests to the server. It sends the name of the file, the operation type (read/write), and the data enclosed in an Ethernet packet. The server awaits the client request via the Ethernet link, performs all requested I/O operations, and sends the results to the client using the same Ethernet card. The server builds an Ethernet packet containing all requested data and returns it to the client.

Both the client and server applications work at a very low level from the Ethernet link point of view. From the server side, a TCP/IP-oriented application would be simpler to write and develop, but it would require using an IP stack for the MSC8101ADS, a process that consumes a lot of development time. We used data link layer programming for developing the library. **Figure 1** shows the architecture of the fast I/O library:

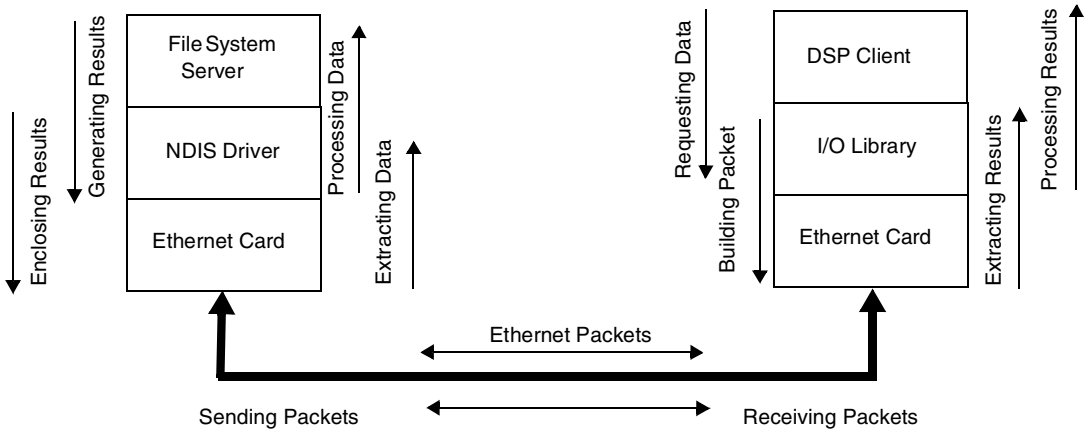

Figure 1. Application Modules

Using the fast I/O library imposes the following restrictions on the MSC8101ADS board:

- The MSC8101ADS Ethernet controller cannot be used for other purposes while the fast I/O library is in use. If you want to develop an application that uses the Ethernet link (for example, a TCP/IP application), you cannot use the fast I/O library. The fast I/O library has a module that requires full control of the entire Ethernet communications link.

- The board-integrated codec cannot be used with the fast I/O library. The design of the MSC8101ADS board does not permit the use of the Ethernet controller and a codec in the same time.
- To increase communication speed, the client does not wait until the server answers its request. If a timeout elapses (a 400 ms timeout is defined, but the value is programmable), the client considers its request to be lost and retransmits it to the server. To implement the timing function, the first hardware timer of the board is blocked, so an application that uses hardware timers must overwrite the timer interrupt routine in order to use the fast I/O library.
- Not all standard I/O functions are supported. Only binary file accessing is allowed. This should not be an issue because any other standard function can be rewritten as a combination of the supported functions.

2 Server Interface

The fast I/O library server is a Win32 application working on the Windows NT or Windows 2000 operating systems. **Figure 2** shows the program display.

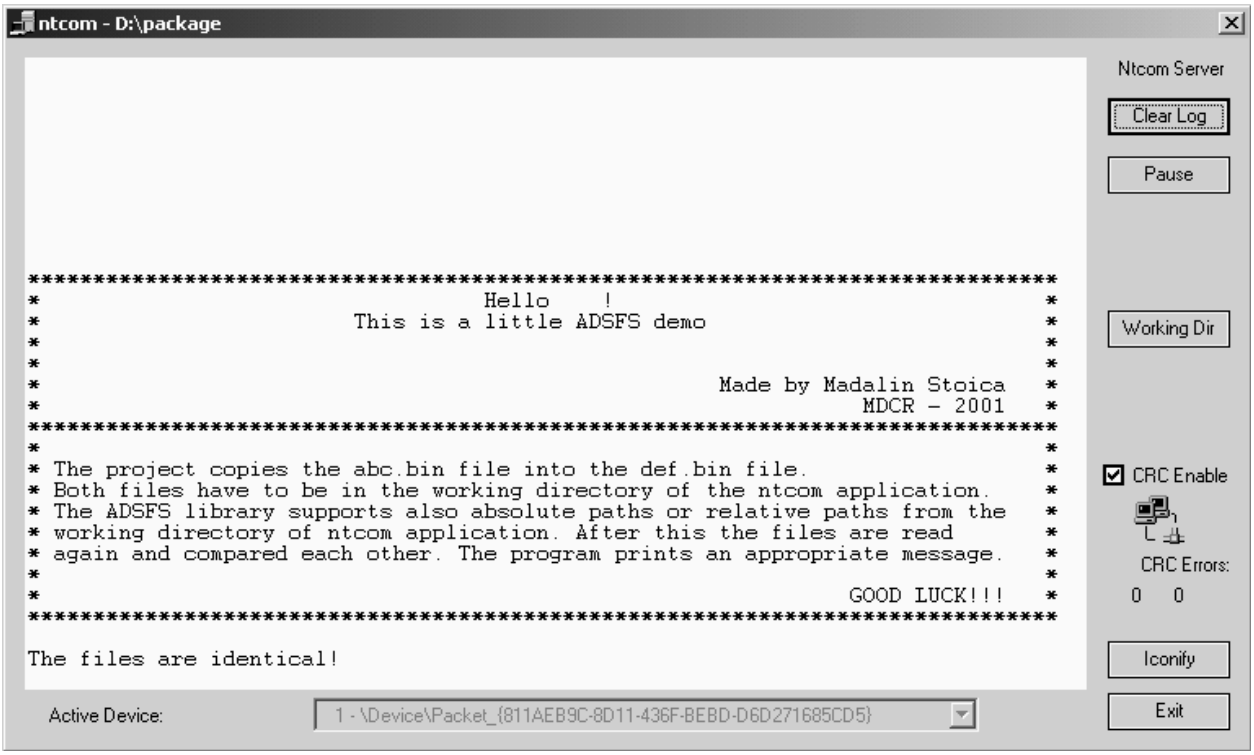

Figure 2. The ntcom Server Interface Display

The server application displays all messages (works like a *stdout*) and communicates with the client to configure all communications parameters. Key features of the interface are as follows:

- *Clear Log*. This button clears the white screen and the `screen.txt` file that contains all messages, including those displayed on the screen.
- *Pause*. This button enables the user to pause the communications and read the last messages printed on the screen. Because the server cannot notify the client to stop, this operation simply blocks the communications thread of the server so that the client does not receive any answers to its requests that

would cause it to retransmit all unanswered packets. The client treats this pause as a communications problem, so the green LED on the board blinks during this operation to indicate that no answer is received.

- *Working Dir*. This button allows the user to change the working directory of the application so that all paths used are relative to this directory. The working directory is displayed on the title bar of the application. A corresponding registry entry is made so that the application can start from the last working directory.
- *CRC Enable*. This check button activates or deactivates the CRC on the Ethernet packets. No correction is made on packets with a CRC error. All these packets are ignored and retransmitted. When CRC detection is activated, the server offers information on the number of packets with CRC errors. The number on the left is the number of packets received by the PC that contain errors. The number on the right is the number of packets received by the MSC8101ADS board that contain errors.
- *Iconify*. This button minimizes the application so that only the icon in the system tray remains active. Double-clicking this icon redisplays the application window.
- The icon displayed on the right is the traffic monitor for the Ethernet link, and it changes its face according to traffic characteristics.

At the bottom of the application window is a drop-down list that displays all available Ethernet cards (all cards for which the NDIS driver is activated). When a card is selected, the list becomes unavailable so that the working card cannot be changed during the communications process. If you have only one card (the usual setup), it is automatically selected and the drop-down list is disabled.

3 System Requirements

The fast I/O library requires the following resources:

- *MSC8101ADS board*. Only the client works on this board, and the ethernet capabilities of the board are used for communication.
- *PC running Windows 2000 or Windows NT*. The server part of the application works on a PC, and the fast I/O library makes the PC file system available to the MSC8101ADS board.
- *Dedicated 100 Mbps Ethernet card.* This card must be installed on the PC to provide communication support to the ADS board running the client. We recommend that you have a dedicated network card even if your PC has one for communicating within your LAN. You can have only one Ethernet card, but while your link with ADS is up, you must uninstall all other Windows drivers and clients working on the card (see **Section 4**, *Installation*, for details).
- *Ethernet crossover cable*. This cable is required for the physical link between the host PC and the client MSC8101ADS. Because there is a direct connection between the two Ethernet controllers, you should have a 100 Mbps Ethernet crossover cable.

In addition to all of these physical devices, there are software components that are discussed in the next section.

4 Installation

The software package distributed with the fast I/O library contains five subdirectories, as follows:

- `doc`. Contains the application note with details on the fast I/O library.
- `lib`. Contains the library (`adsfs.elb`) and the header file (`adsfs.h`) with the library interface.

- `tests`. Contains the application example, a CodeWarrior project (`test.mcp`) as the client, and the server-side application (`ntcom.exe`).
- `WinNTDriver`. Contains the NDIS driver for Windows NT.
- `Win2KDriver`. Contains the NDIS driver for Windows 2000.

Also included are `readme.txt` files with details on how to use the fast I/O library in MSC8101ADS projects. To install the fast I/O library software package, perform the following steps:

1. Plug the dedicated Ethernet card into the host PC. Install the specific driver as your PC operating system requires.
2. Ensure that you have the `packet.sys` and `oemsetup.inf` files specific to your operating system.

   The distribution packet contains two directories, `WinNTDriver` and `Win2KDriver`, that include the necessary files.
3. Install the NDIS packet driver as follows:

   For Windows NT:

   a. Open the control panel under **SETTINGS** on the Start menu.
   b. Run the **NETWORK** program.
   c. Select the **PROTOCOLS** tab and click on **ADD**.
   d. Click on **HAVE DISK** and point to the directory containing the driver for Windows NT.
   e. Close the network program and reboot your system.

      NDIS 3.0 should be installed on your computer.

   For Windows 2000:

   a. Open the control panel under **SETTINGS** on the Start menu.
   b. Open the **NETWORK** and **DIAL-UP CONNECTION**.
   c. Double click on the dedicated Ethernet card.
   d. Click on the **PROPERTIES** button.

      A list displaying available clients, services, and protocols appears. Deselect all entries in the list. Now, you cannot use this card for other purposes (for example, to communicate with your LAN). This is why we recommend that you have a dedicated Ethernet card for communicating with the MSC8101ADS and another Ethernet card for the usual Windows-specific communications.
   e. Click on the **INSTALL** tab.
   f. Select **PROTOCOLS** and click on **ADD**.
   g. Click on **HAVE DISK**, point to the directory containing the Win2K driver, and select the `packet.inf` file).
   h. Click on **OPEN** and then on **OK**.
   i. Select **NDIS FOR WIN2K PACKET PROTOCOL** and click on **OK**.

      On the Properties panel associated with the dedicated card, the new NDIS 5.0 driver appears.
   j. Reboot the system to activate the new driver.

Note: All of these steps should be performed only once.

WARNING: NDIS 3.0 works on Windows NT machines, and NDIS 5.0 works on Win2K. Although NDIS 5.0 is a newer version of the software and is an upgrade, Windows NT

does not support all features of the NDIS 5.0 driver. Install the proper driver for each operating system in order to avoid system crashes.

Now that NDIS driver is installed on your computer, you must set it up properly for using the fast I/O library as follows:

- Activate the NDIS driver and deactivate all other clients, services, or protocols for the dedicated Ethernet card.
- Deactivate the NDIS driver for the LAN-dedicated card.

If you have only one Ethernet card, you cannot use it for the LAN and the fast I/O library at the same time. To access the LAN, you must activate the appropriate clients and protocols and deactivate the NDIS driver and *vice versa* to communicate with the MSC8101ADS board. These changes must be made each time you switch the card destination.

To enable the fast I/O library and disable the LAN on Windows NT, perform the following steps:

1. In the control panel, select **NETWORK** and then select **BINDINGS**.
2. Show Bindings for **ALL ADAPTERS**.
3. Open the relevant adapter options by clicking on the plus sign at the left of the name.
4. For the MSC8101ADS board-dedicated card:
   a. Enable **NDIS 3.0 PACKET DRIVER**.
   b. Disable all other options for this adapter.
5. For the LAN Ethernet card:
   a. Disable **NDIS 3.0 PACKET DRIVER**.
   b. Enable **TCP/IP** and **WINS CLIENT**.
6. Click on **OK**.
7. Reboot the system to activate the new settings.

To enable the fast I/O library and disable the LAN on Windows 2000, perform the following steps:

1. In the control panel, double click on the dedicated Ethernet card.
2. Select only the **NDIS FOR WIN2K PACKET** protocol.
3. Click on **OK**. It is not necessary to reboot your system.

When set-up is complete, you are ready to build your own StarCore application based on the fast I/O library. **Figure 3** shows the set-up scheme. The `readme.txt` file in the distribution packet provides details on MSC8101ADS configuration.

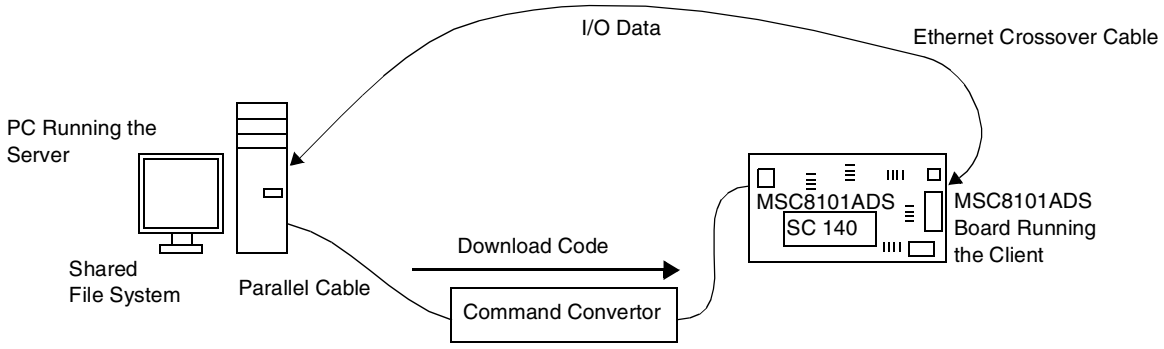

Figure 3. Fast I/O Library Hardware Diagram

5 API Interface

The API interface of the fast I/O library is represented in the `adsfs.h` source file, which contains the headers of all functions. For programming convenience, all functions have the same name as the corresponding standard I/O functions, but they are prefixed with `MDCR_MSC8101ADS_`. As the `adsfs.h` file shows, simply defining the MDCR_MSC8101ADS_FS symbol causes an application to use the fast I/O library. If this symbol in not defined, all I/O functions are linked to the standard functions. One of the main advantages of the new I/O library is that a simple symbol definition transforms the application from a classical one into an advanced one and *vice versa*.

A key difference between the fast I/O library and the standard I/O library is the FILE type. Also, the fast I/O library includes some new type definitions and constants. In `adsfs.h`, MDCR_MSC8101ADS_FILE_T is a new type for identifying files. According to this new type definition, the EOF and NULL constants are redefined as MDCR_MSC8101ADS_FILE_EOF and MDCR_MDSC8101ADS_FILE_NULL, respectively.

5.1 Library Functions

This section describes all the functions of the fast I/O library, presenting all parameters and the return value.

5.1.1 adsfs_init()

Description: Initializes the fast I/O library. This function must be called before any other library function. It sets up all the resources of the MSC8101ADS board. When this function is called, the following activities occur:

- The MSC8101ADS Ethernet controller is initialized and waits for a 100 Mbps link.
- A MAC address is attached to the board (the default value is 0x0, 0x0, 0x4D, 0x44, 0x43, 0x52).
- The MSC8101 communications processor module (CPM) comes up with all the buffers needed for transmission allocated. All peripherals needed by an Ethernet-based communications system (for example, the Ethernet transceiver) are ready to send and receive data.

Function Prototype `void adsfs_init(void);`

Return Value None

Parameters None

5.1.2 adsfs_SetSrc()

Description: Replaces the default MSC8101ADS MAC address with a new one received as parameter. While it is rare that another Ethernet card would have the same MAC address as the default MAC address of the board, it can happen. Therefore, this function changes the default MSC8101ADS MAC address to prevent duplication of the MAC address. If you use this function, you should call it before calling the initialization function. Otherwise, the Ethernet transceiver is initialized with the default MAC address, but all packets have the new address as their source address, resulting communication problem. The transceiver rejects all response packets due to a mismatch of MAC addresses.

Code Example
```
void adsfs_SetSrc(unsigned char Src_AD0, unsigned char Src_AD1,
unsigned char Src_AD2, unsigned char Src_AD3, unsigned char
Src_AD4, unsigned char Src_AD5);
```

Return Value None

Parameters The new MAC address the user wants to associate with the MSC8101ADS board.

5.1.3 adsfs_SetDst()

Description: Identifies the PC MAC address for the MSC8101ADS and thereby shortens the initialization time required for setting up communications. This function is typically used in conjunction with the adsfs_SetSrc() function. If the default values are used, the MSC8101ADS board can communicate with the host PC to get the MAC address. This operation completes in less that one second, so you should use the default initialization values by calling only the adsfs_init() function.

Function Prototype
```
void adsfs_SetDst(unsigned char Dst_AD0, unsigned char Dst_AD1, unsigned
char Dst_AD2, unsigned char Dst_AD3, unsigned char Dst_AD4, unsigned char
Dst_AD5);
```

Return Value None

Parameters The MAC address of the PC Ethernet card.

5.1.4 MDCR_MSC8101ADS_fopen()

Description: Opens a file. The fast I/O library allows you to use files only in binary mode, so only the first character in the mode string is considered. The server opens files with the corresponding *rb*, *wb,* or *ab*, respectively.

Function Prototype
```
MDCR_MSC8101ADS_FILE_T MDCR_MSC8101ADS_fopen(const char *filename, const
char *mode);
```

Return Value A file identifier for the open file. If any error occurs, the function returns the MDCR_MSC8101ADS_NULL constant.

Parameters *filename* A string containing the name of the file to open.

mode A string with the type of access permitted. The modes used are identical with the modes provided by the standard I/O library.

5.1.5 MDCR_MSC8101ADS_fclose()

Description: Closes a file.

Function Prototype
```
int MDCR_MSC8101ADS_fclose(MDCR_MSC8101ADS_FILE_T stream);
```

Return Value 0 if the stream is successfully closed and *MDCR_MSC8101ADS_EOF* if an error occurs.

Parameters *stream* The identifier of the file to close.

5.1.6 MDCR_MSC8101ADS_fread()

Description: Reads data from a stream. up to *count* items of *size* bytes from the input *stream* and stores them in *buffer*.

Function Prototype
```
size_t MDCR_MSC8101ADS_fread(void *buffer, size_t size, size_t count,
MDCR_MSC8101ADS_FILE_T stream);
```

Return Value The number of full items actually read, which may be less than *count* if an error occurs or if the end of the file is encountered before reaching *count*.

Parameters *buffer* Storage location for data.

size Item size in bytes.

count Maximum number of items to be read.

stream The identifier of the file to read.

5.1.7 MDCR_MSC8101ADS_fwrite()

Description: Writes data to a stream up to *count* items, of *size* length each, from *buffer* to the output *stream*.

Function Prototype
```
size_t MDCR_MSC8101ADS_fwrite(const void *buffer, size_t size, size_t
count, MDCR_MSC8101ADS_FILE_T stream);
```

Return value

Return Value The number of full items actually written, which may be less than *count* if an error occurs.

Parameters

Parameters *buffer* Pointer to data to be written.

size Item size in bytes.

count Maximum number of items to be written.

stream The identifier of the file to write to.

5.1.8 MDCR_MSC8101ADS_printf()

Description: Prints formatted output to the standard output stream. While there is no standard output stream, the console window of the server application is considered as stdout. Because you often want a history with all output messages, the data written in the virtual output stream is also sent to a special file called `screen.txt`. This file is created in the working directory of the server application. Browsing this file, you can see old messages. The server application (native windows executable) opens the `screen.txt` file with the append option enabled, so all the history can be found there.

Note: The `screen.txt` file should be considered as a reserved word for the library point. Do not used this name for any other file in your applications. While this file is always opened with the append option, it can become huge. To avoid this problem, delete this file from time to time, erasing historical information that is no longer needed. This operation can easily be performed by clicking on the **CLEAR LOG** button of the `ntcom.exe` application.

Function Prototype
```
int MDCR_MSC8101ADS_printf(const char * format, ...);
```

Return Value The number of characters printed.

Parameters *format* Pointer to data to be written.

argument Optional arguments.

5.1.9 MDCR_MSC8101ADS_fprintf()

Description: Formats and prints a series of characters and values to the output *stream.* Use this function for printing data to any file. Sending data to the reserved `screen.txt` file is similar to using the MDCR_MSC8101ADS_printf() function.

Function Prototype

```
int MDCR_MSC8101ADS_fprintf(MDCR_MSC8101ADS_FILE_T stream, const char *
format, ...);
```

Return Value The number of bytes written.

Parameters *stream* The identifier of the file to write.

format Format-control string.

argument Optional arguments.

5.1.10 MDCR_MSC8101ADS_fgets()

Description: Reads a string from the input *stream* argument and stores it in *string*. This function reads characters from the current stream position to and including the first newline character, to the end of the stream, or until the number of characters read is equal to $n - 1$, whichever comes first. The result stored in *string* is appended with a null character. The newline character, if read, is included in the string.

Function Prototype

```
char * MDCR_MSC8101ADS_fgets(char * line, int length,
MDCR_MSC8101ADS_FILE_T stream);
```

Return Value *string* pointer.

Parameters *line* Storage location for data.

length Maximum number of characters to read.

stream The identifier of the file to read.

6 Application Example

This section presents a short example application that copies the `abc.bin` file into the `def.bin` and then reads and compares both files. An appropriate message is displayed in the server window and written into the `screen.txt` file. The C code of the main() function is presented in **Section 7**, *Source Files*. A CodeWarrior project containing this simple demo is included in the distribution packet. The project was developed using Metrowerks CodeWarrior, Release 1.1.

We begin by calling the adsfs_init() function to initialize the communication. We include this function call in an `#ifdef` so that it is activated only when we want to use the fast I/O library. Also, this approach prevents calls to the MSC8101ADS hardware-specific function when the simulator is used because the fast I/O library works only on the MSC8101ADS board and not on the simulator. The fast I/O library becomes active only when MDCR_MSC8101ADS_FS is defined.

To test our application, we make a CodeWarrior project and change the main() function to the one presented in **Section 7**, *Source Files*. Since all library functions are included in the `adsfs.elb` file, we include this file in the project by selecting the **FILE MAPPINGS** option in the **TARGET PROJECT SETTINGS** and including `.elb` files as **IMPORTER TYPE**.

After the C source code is compiled and downloaded on the MSC8101ADS, the green LED of the board blinks to indicate that there are communication errors and the MSC8101ADS has not received responses to the packets sent. This is a normal behavior while the server-side `ntcom.exe` application is not started on the PC. When the application starts, the *ntcom* program identifies all Ethernet cards on the PC that support the NDIS driver. If it finds only one card, the application automatically starts using it. Otherwise, a list of all cards is displayed, and the user must select the working Ethernet card.

WARNING: Be careful to select the correct Ethernet card because otherwise the system may crash. In order to avoid crashes, you should deactivate the NDIS driver for all unnecessary cards so that only the card with a link to the MSC8101ADS board has an active NDIS driver. Because the NDIS driver works properly alone, deactivate all other clients, services, or protocols on the library-dedicated Ethernet card.

When the *ntcom* program starts, the leds no longer blink. If the PC running the server application is very busy, it may not have enough time to process all the client requests. If this occurs, the green led of the board toggles from time to time, indicating that some requests have not received answers and should be retransmitted.

When communications start, the server and the client send each other their MAC address. Because this “hand shaking” operation occurs each time the client is restarted, it is not necessary to restart the server application. Also, it does not matter whether the server or the client application starts first.

If the server application is stopped or killed, you must restart the client as well as the server application. If the server is restarted before the client finishes its job, error message may appear on the server console to indicate that the client has sent requests for files that are no longer valid. If the system is working correctly, the server screen and the `screen.txt` file appear as follows:

Example 1. Server Application Screen

```
*****************************************************************************
*                                  Hello    !                                *
*                        This is a little ADSFS demo                        *
*                                                                           *
*                                                                           *
*                                                  Made by Madalin Stoica   *
*                                                                MDCR - 2001*
*****************************************************************************
*                                                                           *
* The project copy the abc.bin file into the def.bin file.                  *
* Both files have to be in the working directory of the ntcom application.  *
* The ADSFS library supports also absolute paths or relative paths from the *
* working directory of ntcom application. After this the files are read again*
* and compared. The program prints an appropriate message.                  *
*                                                                           *
*                                                          . GOOD LUCK!!!   *
*****************************************************************************

The files are identical!
```

If the last message is `The files are NOT identical!`, there are problems with the MSC8101ADS configuration. You need to recheck your set-up. The fast I/O library functions are the basis for all I/O operations. Additional standard I/O functions can easily be written based on the functions included in the library.

7 Source Files

Example 2. C Source of the adsfs.h File

```
#ifndef __ADSFS_H
#define __ADSFS_H

#include <stddef.h>

#ifdef MDCR_MSC8101ADS_FS
#define MDCR_MSC8101ADS_FILE_T signed long
#define MDCR_MSC8101ADS_FILE_NULL  -1
#define MDCR_MSC8101ADS_FILE_EOF   -1
#define MDCR_MSC8101ADS_printf MDCR_MSC8101ADS_printf
#define MDCR_MSC8101ADS_fprintf MDCR_MSC8101ADS_fprintf
#define MDCR_MSC8101ADS_fopen MDCR_MSC8101ADS_fopen
#define MDCR_MSC8101ADS_fclose MDCR_MSC8101ADS_fclose
#define MDCR_MSC8101ADS_fwrite MDCR_MSC8101ADS_fwrite
#define MDCR_MSC8101ADS_fread MDCR_MSC8101ADS_fread
#define MDCR_MSC8101ADS_fgets MDCR_MSC8101ADS_fgets

void adsfs_init(void);
/* init the Eth Card on ADS*/
void adsfs_SetSrc(unsigned char Src_AD0, unsigned char Src_AD1, unsigned char Src_AD2, unsigned char Src_AD3,
unsigned char Src_AD4, unsigned char Src_AD5);
/* Set the MAC address on the ADS */
void adsfs_SetDst(unsigned char Dst_AD0, unsigned char Dst_AD1, unsigned char Dst_AD2, unsigned char Dst_AD3,
unsigned char Dst_AD4, unsigned char Dst_AD5);
/*this function receives the ethernet address of the destination - the ADS knows the address of the
PC part */

MDCR_MSC8101ADS_FILE_T MDCR_MSC8101ADS_fopen(const char *filename, const char *mode);
int MDCR_MSC8101ADS_fclose(MDCR_MSC8101ADS_FILE_T stream);
size_t MDCR_MSC8101ADS_fwrite(const void *buffer, size_t size, size_t count, MDCR_MSC8101ADS_FILE_T
stream );
size_t MDCR_MSC8101ADS_fread(void *buffer, size_t size, size_t count, MDCR_MSC8101ADS_FILE_T stream
);
int MDCR_MSC8101ADS_printf(const char * format, ...);
int MDCR_MSC8101ADS_fprintf(MDCR_MSC8101ADS_FILE_T stream, const char * format, ...);
char * MDCR_MSC8101ADS_fgets(char * line, int length, MDCR_MSC8101ADS_FILE_T file);

#else
#include <stdio.h>

#define MDCR_MSC8101ADS_FILE_T FILE *
#define MDCR_MSC8101ADS_FILE_NULL  NULL
#define MDCR_MSC8101ADS_FILE_EOF   EOF
#define MDCR_MSC8101ADS_printf printf
#define MDCR_MSC8101ADS_fprintf fprintf
#define MDCR_MSC8101ADS_fopen fopen
#define MDCR_MSC8101ADS_fclose fclose
#define MDCR_MSC8101ADS_fwrite fwrite
#define MDCR_MSC8101ADS_fread fread
#define MDCR_MSC8101ADS_fgets fgets
#endif

#endif  /* __ADSFS_H */
```

Example 3. C Source Code for the main.c File in the Demo

```
#include "stdio.h"
#include "adsfs.h"

#define TBUFSZ 8000

void main(void)
{
   unsigned char buffer[TBUFSZ];
   unsigned char buffer1[TBUFSZ];
   MDCR_MSC8101ADS_FILE_T rxid;
   MDCR_MSC8101ADS_FILE_T txid;
   unsigned long len, len1;

   adsfs_init();
 MDCR_MSC8101ADS_printf("\n*************************************************************************\n");
 MDCR_MSC8101ADS_printf("*                              Hello    !                               *\n");
 MDCR_MSC8101ADS_printf("*                    This is a little ADSFS demo                       *\n");
 MDCR_MSC8101ADS_printf("*                                                                      *\n");
 MDCR_MSC8101ADS_printf("*                                                                      *\n");
 MDCR_MSC8101ADS_printf("*                                              Made by Madalin Stoica  *\n");
```

```
 MDCR_MSC8101ADS_printf("*                                                                    MDCR - 2001   *\n");
 MDCR_MSC8101ADS_printf("*************************************************************************************\n");
 MDCR_MSC8101ADS_printf("                                                                                     *\n");
 MDCR_MSC8101ADS_printf("* The project copies the abc.bin file into the def.bin file.                          *\n");
 MDCR_MSC8101ADS_printf("* Both files have to be in the working directory of the ntcom application.           *\n");
 MDCR_MSC8101ADS_printf("* The ADSFS library supports also absolute paths or relative paths from the          *\n");
 MDCR_MSC8101ADS_printf("* working directory of ntcom application. After this the files are read             *\n");
 MDCR_MSC8101ADS_printf("* again and compared each other. The program prints an appropriate message.         *\n");
 MDCR_MSC8101ADS_printf("*                                                                                    *\n");
 MDCR_MSC8101ADS_printf("*                                                                      GOOD LUCK!!!   *\n");
 MDCR_MSC8101ADS_printf("*************************************************************************************\n\n");

   /*this part open the abc.bin file and copy it into the def.bin file. If the def.bin file exists it */
   /* is overwritten, otherwise is created. */


  rxid = MDCR_MSC8101ADS_fopen("abc.bin", "rb");
  if (rxid == MDCR_MSC8101ADS_FILE_NULL)
  {
    MDCR_MSC8101ADS_printf("Cannot open the abc.bin file.\n");
    exit(1);
  }

  txid = MDCR_MSC8101ADS_fopen("def.bin", "wb");
  if (txid == MDCR_MSC8101ADS_FILE_NULL)
  {
    MDCR_MSC8101ADS_printf("Cannot open the def.bin file.\n");
    exit(1);
  }

  for (len = TBUFSZ; len == TBUFSZ;)
  {
    len =MDCR_MSC8101ADS_fread(buffer, sizeof(char ), TBUFSZ, rxid);
    len =MDCR_MSC8101ADS_fwrite(buffer, sizeof(char), len, txid);
  }

  MDCR_MSC8101ADS_fclose(rxid);
  MDCR_MSC8101ADS_fclose(txid);

  rxid = MDCR_MSC8101ADS_fopen("abc.bin", "rb");
  if (rxid == MDCR_MSC8101ADS_FILE_NULL)
  {
    MDCR_MSC8101ADS_printf("Cannot open the abc.bin file.\n");
    exit(1);
  }

  txid = MDCR_MSC8101ADS_fopen("def.bin", "rb");
  if (txid == MDCR_MSC8101ADS_FILE_NULL)
  {
    MDCR_MSC8101ADS_printf("Cannot open the def.bin file.\n");
    exit(1);
  }

  for (len = TBUFSZ, len1 = TBUFSZ; len == TBUFSZ && len1 == TBUFSZ;)
  {
    len = MDCR_MSC8101ADS_fread(buffer, sizeof(char ), TBUFSZ, rxid);
    len1 = MDCR_MSC8101ADS_fread(buffer1, sizeof(char ), TBUFSZ, txid);
    if (memcmp(buffer,buffer1,len))
    {
       MDCR_MSC8101ADS_printf("The files are NOT identical!!!\n");
       MDCR_MSC8101ADS_fclose(rxid);
       MDCR_MSC8101ADS_fclose(txid);
       return;
    }
  }

  MDCR_MSC8101ADS_printf("The files are identical!\n");
  MDCR_MSC8101ADS_fclose(rxid);
  MDCR_MSC8101ADS_fclose(txid);

}
```

8 Related Reading

[1] *MSC8101 Reference Manual* (MSC8101RM).

[2] *SC140 DSP Core Reference Manual* (MSC140DSPCORERM).

[3] *Metrowerks Enterprise C Compiler Manual*.

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2260
Rev. 2
11/2004

freescale™
semiconductor