Technical Note

Working with the eZ80Acclaim!™ MCU-Based Flash File System

TN003901–0105

Introduction

This Technical Note discusses the usage of the file system that resides within the external Flash memory interfaced to the eZ80Acclaim!™ microcontroller. The ZiLOG TCP/IP Software Suite (ZTP v1.4.x) uses the RZK operating system which supports the eZ80Acclaim!™ MCU based embedded Flash file system. The APIs developed for this file system are used to develop user applications with minimum effort.

The file system discussed in this Technical Note is designed for embedded hardware environments such as the ZiLOG eZ80Acclaim!™ Development Module environment. The file system is implemented within 1MB of external Flash leaving sufficiently large memory to accommodate the user application data. Flash is used for long-term storage purposes, and RAM is used as an operating storage for storing and manipulating data.

This Technical Note lists a standard set of API functions such as `fcreate()`, `fopen()`, `fread()`, `fwrite()`, and `fclose()` that are used for file manipulation operations. For a detailed information about the use of these APIs, refer to the ZiLOG File System v1.0.0 Reference Manual (RM0039).

eZ80Acclaim!™ Based Flash File System

This section discusses the initialization of the eZ80Acclaim!™ MCU-based embedded Flash file system, the file system configuration, the file access and manipulation functions, the Flash memory interface, and the methods of working with files. The project settings required for using the Flash file system are also listed in this section.

Initializing the Flash File System

For proper functioning of the Flash file system, it is necessary to initialize the RAM volume or the Flash volume prior to storing files either in RAM or Flash. Initialization of the RAM volume or the Flash volume is achieved by invoking the `ZFSInit ()` API. Prior to calling any other ZiLOG File System API, it is mandatory for applications to call the `ZFSInit ()` API.

If the data storage area is a RAM volume, then the `ZFSInit ()` API formats the volume with the native file system. In case of Flash volumes, the `ZFSInit ()` API initially performs an integrity check of the file system data. If uncorrectable errors are found, the `ZFSInit ()` API reformats the Flash volume.

File System Configuration

The ZiLOG File System is configured to suit user application requirements. Users can control the amount of memory required by the file system by setting the total number of VOLUMES, BLOCKS, and SECTORS.

With reference to the eZ80Acclaim!™ MCU based Flash file system, a VOLUME is defined as a contiguous storage area for the file system, and is similar to a drive with a specific name. For Flash, a BLOCK represents a physically erasable area, and for RAM a BLOCK refers to one RAM volume. SECTORS are single units of storage at the physical disk level.

Additionally, the maximum number of files that can be open simultaneously is user-configurable. The maximum number of directories that can be created is also user-specified.

The following macros are used to set the parameters for the Flash file system.

ZFS_TOTAL_NUM_VOLUMES. Specifies the number of volumes in the file system.

ZFS_TOTAL_NUM_BLOCKS. Specifies the total number of blocks in the file system. The user is required to manually calculate the number of blocks in each volume, and set the value of this macro appropriately. While determining the number of blocks per volume, note that a RAM volume contains only one block, and a Flash volume contains a number of blocks equivalent to the number of physical erasable units in the file system.

ZFS_TOTAL_NUM_SECTORS. Specifies the total number of sectors in all the Flash volumes of the file system. The number of sectors must be configured appropriately by changing the parameters located in the `ZFScfg.c` file.

ZFS_MAX_FILE_OPEN_COUNT. Specifies the maximum number of files that can be open simultaneously.

ZFS_MAX_DIRS_SUPPORTED. Specifies the maximum number of directories in the entire file system.

Users must set the above macros available in the `ZFSCfg.h` file, which is located in the `..ZTP 1.4.0\RZK\eZ80F91\Inc\FS` directory depending on the requirement of the application.

Moreover, the details of the file system volume configurations must be stored in the `g_zfs_cfg` variable which is located in the `..ZTP 1.4.0\RZK\eZ80F91\Src\FS\ZFSCfg.c` directory. The parameters involved in the configuration of file system volumes are members of the `ZFS_CONFIG_t` structure. The name and description for each parameter is provided below.

vol_name. This structure member contains the volume name (a string of maximum length equal to16 bytes). The `RZK_CWD_PATH` macro contains the valid volume name.

vol_type. This structure member contains the type of the volume. At the time of publication of this document, ZFS includes support for volumes stored in either RAM or external Flash only.

vol_address. This structure member contains the starting address of the volume which is the address at which data storage begins. Users must modify this starting address depending on the target board address mapping.

vol_size. This structure member contains the size of the volume in bytes which is the memory available to store files.

Note: For complete details about the configuration of the file system parameters, refer to the ZiLOG File System v1.0.0 User Manual (UM0179).

File Access and Manipulation Function

The File Access and Manipulation function block includes the functions used to manage files after the file system is created and initialized. The functions `fopen ()`, `fread ()`, `fwrite ()`, `fclose ()`, `fseek ()`, and `feof ()` are used to manage files. These functions collectively constitute the file system API.

For a complete list of these APIs and their descriptions, please refer to the ZiLOG File System v1.0.0 Reference Manual (RM0039).

Flash Memory Interface

The Flash Memory interface allows permanent data storage, and enables users to continue working with the file system after cycling power to the storage device.

The ZTP v1.4.x release supports file storage only in the external Micron Flash memory located on the eZ80Acclaim!™ Development Board.

Working with Files

This Technical Note contains sample code to demonstrate the use of file system APIs. The `file_handling.c` file listed in the Source Code in C section on page 6 can be added to any ZTP v1.4.x project by following the procedure listed in the Using the Flash File System section on page 4 . The output of the file manipulation operation can be observed in the HyperTerminal application.

Project Settings for Using the Flash File System

To use the Flash file system, perform the following operations in the ZDSII environment.

1. Select the **Project → Settings** menu. In the **Project Settings** dialog box, click the **C** tab and select the **Preprocessor** category. Observe that the macro provided below appears in the **Preprocessor Definitions** field. By default, **Project Settings** contains all macros.

   ```
   RZK_FS_ENABLE,
   ERASE_FLASH=0
   RTC
   ```

   To completely erase the file system area located in the Flash memory, set the definition `for ERASE _FLASH` to 1. After the project is built, compiled, downloaded, and executed on the target board, all of the data present in the file system area is erased. Conse-

quently, the ERASE_FLASH definition must be changed to 0, and the project must be rebuilt, recompiled, and reexecuted to work with the files.

Note: The Flash file system is dependent on the Real-Time Clock (RTC). The date and time information is displayed provided the RTC is enabled.

2. All macro definitions and initialization parameters required to work with the Flash file system are available in ZFSCfg.h file. By default, all parameters are initialized appropriate to the eZ80F91 Development Board. If different hardware is used, then modifications must be incorporated in the ZFSZfg.h file.
3. Add the ZFSCfg.c file to the ZTPDemo_F91 project, and the filesystem.lib files to the **Object Binary Modules** field under the **Linker tab** of the **Project Settings** dialog box.
4. The RZK_CWD_PATH macro in the Zsysgen.h file must contain a valid directory name that is used as the current working directory for the threads created in RZKApplicationEntry. Moreover, the RZK_MAX_CWD_PATH_LEN macro must be set to the maximum string length for the current directory. By default, the directory name is set to "EXTF:/", and the maximum string length for the current directory is set to 128 bytes.

Using the Flash File System

To use the ZiLOG Flash File System APIs, perform the following steps:

1. Initialize the file system by invoking the ZFSInit() API in the main() function.
2. Open the main.c file, and add the extern void file_manipulation (void); function prototype declaration.
3. Add the file_manipulation(); function in the main function just above the return(OK); statement.
4. Add the ZFSCfg.c file to the ZTPDemo_F91 project (This file must be included in the project while performing an operation on the file system).
5. Add the file_handling.c file to the ZTPDemo_F91 project.
6. Ensure that the RZK_CWD_PATH macro located in the Zsysgen.h file features the same volume name as that used in the ZFSCfg.c file.
7. Launch the HyperTerminal application and set it to 57.6 Kbps Baud, 8-N-2, with no flow control.
8. Compile, build, download, and execute the project on the target board. The HyperTerminal application displays the details of file opening, writing, reading and closing operations.

Working with Files through the ZTP Shell

The ZTP shell includes commands for manipulation of files within the file system. These commands are similar to DOS commands, and allow the user to create new files or directo-

ries. For a complete list of file system related commands, refer to the *ZiLOG File System v1.0.0 Reference Manual (RM0039)*.

Uploading Files to the Flash File System through the FTP Server

ZTP v1.4.x implements the FTP client and server, and the TFTP client. Files located in the PC can be uploaded to the Flash memory on the eZ80Acclaim!™ target controller using an FTP client running on a Windows system. This feature is very useful while writing applications involving HTML pages. Web pages can be dynamically uploaded to the Flash memory and accessed, instead of statically linking web pages during compilation.

Summary

The ZiLOG File System is implemented on the Real-Time ZiLOG Kernel (RZK) which is a real-time preemptive multitasking kernel. The ZiLOG File System implements RZK for the Micron Flash devices, and supports all of the basic file and directory operations listed in the *ZiLOG File System v1.0.0 Reference Manual (RM0039)*. The ZiLOG File System kernel maintains and manages all of the data structures used in a project, thereby reducing the burden of the developer, and the project development time.

References

Further details about ZTP v1.4.x , RZK v1.1.x , and the ZiLOG File System can be found in the references listed in Table 1.

Table 1. List of References

Topic	Document Name
RZK v1.1	ZiLOG Real-Time Kernel (RZK) v1.1.0 Product Brief (PB0155)
	ZiLOG Real-Time Kernel (RZK) v1.1.0 Quick Start Guide (QS0048)
	ZiLOG Real-Time Kernel (RZK) v1.1.0 Reference Manual (RM0006)
	ZiLOG Standard Library API Reference Manual (RM0037)
	ZiLOG Real-Time Kernel (RZK) v1.1.0 User Manual (UM0075)
ZTP v1.4	ZiLOG TCP/IP Software Suite (ZTP) v1.4 Quick Start Guide (QS0049)
	ZiLOG TCP/IP Stack v1.4.0 API Reference Manual (RM0040)
	ZiLOG TCP/IP Software Suite (ZTP) v1.4 Reference Manual (RM0041)
ZiLOG File System	ZiLOG File System Quick Start Guide (QS0050)
	ZiLOG File System Reference Manual (RM0039)
	ZiLOG File System User Manual (UM0179)

Source Code in C

The `file_handling.c` file is listed in this section.

```
/*
*********************************************************************
* File : file_handling.c
* Description : This file contains the examples for using the file
* system APIs that demonstrate the method of reading and writing to the
* file during run-time.
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*********************************************************************
*/

#include <stdio.h>
#include "ZTypes.h"
#include "zfsapi.h"
#include "cfileapi.h"

void file_manipulation(void)
{
   UINT8 buffer[ 100 ] = {" ZiLOG Electronics Pvt Limited"};
                                 // This buffer contains data to write
   INT32 numBytesWritten ;
   INT32 numBytesRead ;
   UINT8 buf_read[ 100 ] ;

   struct FILE *file_handle ;

   INT32 status ;
   file_handle = fopen("EXTF:/file1.txt", "wb");
```

```
if(file_handle == NULL)
{
 printf("\n File open error and error number is : %d",
        ZFSGetErrNum());
}
else
{
 printf("\n File is opened in WRITE mode");
}

// To Write to file.

numBytesWritten = fwrite(&buffer[0], 1, 100, file_handle);

if(numBytesWritten <= 0)
{
 printf("\n Write error");
}
else
{
 printf("\n Written %ld bytes to the file",numBytesWritten);
}

// Close the File.

status = fclose(file_handle);

if(status != 0)
{
 printf("\n Unable to close the file");
}
else
{
 printf("\n Closed the file");
}

// To read from the file.

file_handle = fopen("EXTF:/file1.txt", "rb");

if(file_handle == NULL)
{
 printf("\n File open error and error number is : %d",
        ZFSGetErrNum());
}
```

```
   else
   {
    printf("\n File is opened in READ mode");
   }

   numBytesRead = fread(&buf_read[0], 1, 100, file_handle);
   if(numBytesRead <= 0)
   {
    printf("\n Read error");
   }
   else
   {
    printf("\n Read %ld bytes from the file\n", numBytesRead);
   }

   printf("The contents of the File are:-");
   printf(" %s", buf_read);


   // Close the File

   status = fclose(file_handle);
   if(status != 0)
   {
    printf("\n Unable to close the file");
   }
   else
   {
    printf("\n Closed the file");
   }

}
```

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer