eZ80® Family of Microprocessors

Zilog File System

User Manual

UM017914-1211

www.zilog.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists or to request copies of publications, visit 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concerning the devices, applications or technology described is intended to suggest possible uses and may be superseded.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.

eZ80 and eZ80Acclaim! are trademarks or registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.

Revision History

Each instance in the Revision History table below reflects a change to this document from its previous version. For more details, click the appropriate links in the table.

Date	Revision Level	Description	Page
Dec 2011	14	Updated for the RZK 2.4.0 release, which no longer supports the eZ80190 MPU; modified Manual Conventions and Directory Structure sections.	vii, viii, 2
Aug 2010	13	Updated Figure 1, Table 5 and the directory paths in the Zilog File System Architecture, Getting Started, Directory Structure, Zilog File System APIs, Zilog File System Configuration and Zilog File System Macro Configuration sections for the RZK v2.3.0 release.	vii, viii, 2, 4, 7–7,9
Sep 2008	12	Updated for RZK v2.2.0 release; updated Figure 1, Directory Structure, Zilog File System Configuration, Zilog File System Macro Configuration, Integrating a Flash Driver sections.	2, 7, 7, 12
Jul 2007	11	Globally updated document to adhere to style.	All
Jul 2007	10	Globally updated for RZK v2.1.0 release.	All
May 2007	09	Removed Appendix A – Executing Sample Applications in the ZDS II Environment and Appendix B – Executing Sample Applications in the IAR Environment.	n/a
Jul 2006	08	Globally updated for RZK v2.0.0 release.	All

Table of Contents

Introduction

This User Manual describes the Zilog File System for Zilog Real-Time Kernel (RZK) software for eZ80 CPU-based microprocessors and microcontrollers. The current Zilog File System release supports the eZ80Acclaim! family of devices, which includes the eZ80F91, eZ80F92 and eZ80F93 microcontrollers and the eZ80L92 microprocessor.

About This Manual

Zilog recommends that you read and understand the complete manual before using the product. This manual is used as a user guide for Zilog File System.

Intended Audience

This document is written for Zilog customers who have prior exposure to RTOS and writing real-time application code and experienced at working with microprocessors/microcontrollers and writing assembly code or compilers.

In addition to this manual, you should consider reading the following documentation:

- eZ80F91 MCU Product Specification (PS0192)
- eZ80F91 Development Kit User Manual (UM0142)
- eZ80 CPU User Manual (UM0077)
- eZ80Acclaim! Development Kits Quick Start Guide (QS0020)
- Zilog Developer Studio II – eZ80Acclaim! User Manual (UM0144)
- Zilog Real-Time Kernel Reference Manual (RM0006)
- Zilog Real-Time Kernel User Manual (UM0075)
- Zilog File System Reference Manual (RM0039)

Manual Organization

The Zilog File System User Manual is comprised of the following chapters.

Zilog File System Overview

This chapter provides an overview of the Zilog File System, how to get started, the Zilog File System use model, APIs and File and Directory Naming Conventions.

Zilog File System Configuration

This chapter provides a brief description of Zilog File System configuration.

Integrating a Flash Driver

This chapter provides details about how to write a new Flash driver and integrate it with the Zilog File System.

Abbreviations/Acronyms

The following abbreviations/acronyms are used in this document.

Abbreviations/ Acronyms	Expansion
ADC	Analog-to-Digital Converter
IJT	Interrupt Jump Table
IPC	Inter Process Communication
IVT	Interrupt Vector Table
LSB	Least-Significant Byte
lsb	Least-Significant Bit
MSB	Most-Significant Byte
msb	Most-Significant Bit

Manual Conventions

The following assumptions and conventions are adopted to provide clarity and ease of use:

Use of X.Y.Z and A.B.C

Throughout this document, `X.Y.Z` represents the RZK version number in *Major.Minor.Revision* format, and A.B.C represents the ZDS II – eZ80Acclaim! version number in *Major.Minor.Revision* format.

Use of <tool>

Throughout this document, `<tool>` refers to ZDS II.

Use of the Words *Set* and *Clear*

The words *set* and *clear* imply that a register bit or a condition contains the values *logical 1* and *logical 0*, respectively. When either of these terms is followed by a number, the word *logical* may not be included, but it is implied.

Courier New Typeface

Code lines and fragments, equations and various executable items are distinguished from general text by appearing in the Courier New typeface where applicable.

For example, `void AppThreadEntry (void)`.

Hexadecimal Values

Hexadecimal values are designated by a lowercase *h* and appear in the Courier New typeface. For example, STAT is set to `F8h`.

Use of Initial Uppercase Letters

The use of initial uppercase letters designates the names of states, modes and commands as well as settings and conditions in general text. A few examples are provided below:

- The receiver can force the SCL line to Low to force the transmitter into a Wait state
- A Start command triggers the processing of the initialization sequence
- In Transmit Mode, the byte is sent most significant bit first
- The Slave receiver leaves the data line High.
- The bus is considered busy after the Start condition.
- The Master can generate a Stop condition to abort the transfer.

Zilog File System Overview

The Zilog File System (ZFS) is implemented on the Zilog Real-Time Kernel (RZK) which is a real-time, preemptive and multitasking kernel. The Zilog File System implements a file system over RZK for Micron Flash devices and supports all basic file and directory operations. In addition, the Zilog File System can be configured in the Zilog Developer Studio integrated development environment (ZDSII IDE).

The features of the Zilog File System include:

- Implements a core that is independent of the underlying memory device.
- Supports easy configuration of volumes (such as `C:\` or `D:\` drives).
- Provides configuration parameters such as the maximum number of directories to be created and the maximum number of files to be opened at a time. These parameters, related to volume, optimize system operation and serve to consume less memory.
- Supports multiple volume access whether RAM memory, Flash memory or both memories are employed.
- Implements full-fledged directory operation support.
- Easy system configuration.
- Provides a way to port the Zilog File System core easily to another toolset.
- Supports all basic file and directory operations.
- Supports multiple access to a single file, however, it can be edited by only a single person at a time.
- Recovers data after a power failure and implements garbage collection for a Flash device to allow maximum usage of device memory to store files and directories.
- All APIs are multithread safe; that is, they are re-entrant file system APIs.
- Supports the use of period (‘.’) in filenames or directory names to distinguish between the filename and its extension.
- Supports media error handling; that is, recovery of lost data in Flash memory.
- Supports NOR Flash devices.

Zilog File System Architecture

For details about the architecture of the Zilog File System, refer to the Zilog File System Reference Manual (RM0039), which can be found on zilog.com and is also located in the following ZDSII filepath:

```
<ZDSII installed directory>\Program Files\Zilog\
ZDSII_eZ80Acclaim!_A.B.C\ZTP\ZTPX.Y.Z_Lib\RZK\Docs
```

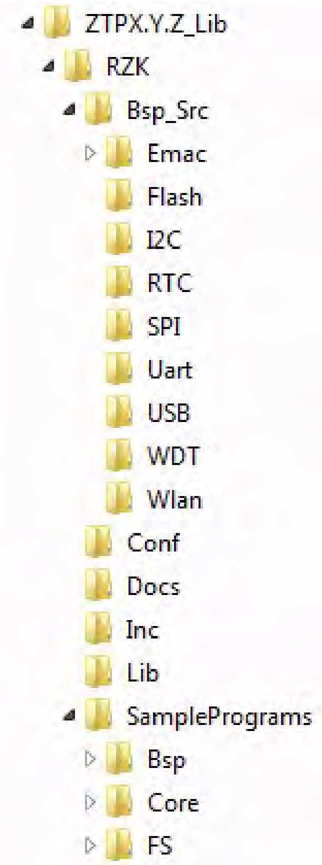
Getting Started

The Zilog File System development software can be installed on different platforms that run the Windows operating system. Refer to the release notes associated with the RZK release to determine the Windows platforms on which the Zilog File System can be installed. The Zilog File System installation files are part of the RZK release file. In this User Manual, only directories related to the Zilog File System are described. For more information about the directory structure of RZK, refer to the Zilog Real-Time Kernel User Manual (UM0075), which can be found on zilog.com and is also located in the following filepath:

```
<ZDSII installed directory>\Program Files\Zilog\
ZDSII_eZ80Acclaim!_A.B.C\ZTP\ZTPX.Y.Z_Lib\RZK\Docs
```

Directory Structure

Figure 1 displays the RZK directory structure.



```
◢ ZTPX.Y.Z_Lib
  ◢ RZK
    ◢ Bsp_Src
      ▷ Emac
        Flash
        I2C
        RTC
        SPI
        Uart
        USB
        WDT
        Wlan
      Conf
      Docs
      Inc
      Lib
    ◢ SamplePrograms
      ▷ Bsp
      ▷ Core
      ▷ FS
```

Figure 1. RZK Directory Structure

The Zilog File System directory contains a common directory for all target processors. Figure 1 displays the following four subdirectories of the Zilog File System:

ZTPX.Y.Z_Lib\RZK\Inc. This directory contains the header files that must be included in the user application.

ZTPX.Y.Z_Lib\RZK\Lib. This directory contains the `NOFS.obj` stub file (for the ZDS II development environment) that must be included if Zilog File System support is not required in the system, even though the system uses the Zilog File System APIs.

ZTPX.Y.Z_Lib\RZK\Conf. This directory contains the `ZFS_Conf.c` file that describes the configuration of the Zilog File System. This file must be included in the application project workspace to interoperate with the Zilog File System.

ZTPX.Y.Z_Lib\RZK\SamplePrograms\FS. This directory contains sample programs written for the Zilog File System. One such sample program is the `FSShell` program, which is an interactive shell application that showcases different operations performed by the Zilog File System. Only a few commands are provided in the `FSShell` application. For detailed information about the `FSShell` application shell commands, refer to the *readme* file present in the following directory:

```
<ZDS II Installed directory>\Program Files\Zilog\
ZDSII_eZ80Acclaim!_A.B.C\ZTP\ZTPX.Y.Z_Lib\RZK\SamplePrograms\FS\
FSShell
```

The above directory also contains all of the sample programs associated with the RZK release.

The name of the four subdirectories listed above is the same for every target eZ80 microprocessor or microcontroller.

Use Model

The Zilog File System is provided as a library and interfaces to the file system via well-known APIs. The Zilog File System provides you with these APIs; however, you must call the appropriate API to obtain service from the file system. The `ZFSInit` API must be called before performing any file- or directory-related operations on the volume.

`ZFSInit()` must be called in a thread body and not in the `main()` function.

The code segment below provides an example of Zilog File System calls, from the initialization of the file system to calling an API.

```
void AppThreadEntry(void)
{
  ZFS_STATUS_t status;
  int cnt;
  int ctr;
  PZFS_VOL_PARAMS_t pvol_params, ptmp_vol;
  ctr = ZFSGetVolumeCount();
  if(ctr <= 0)
  {
  // error in getting the volume count
  return ;
  }

  // allocate memory for volume parameters
  // (sizeof(ZFS_VOL_PARAMS_t) * number of volumes)
  // and store it in pvol_params
  printf("\nInitializing FileSystem, Please Wait...");
  status = ZFSInit(pvol_params);
  if(status != ZFSERR_SUCCESS)
{
  printf("FAILED : %d", status);
  ptmp_vol = pvol_params;
  for(cnt = 0 ; cnt < status ; cnt ++, ptmp_vol++ )
  {
  printf("\n\nVolume Name: %s", ptmp_vol->vol_name);
  printf("\nFormatting the volume: %s", ptmp_vol->vol_name);
  status = ZFSFormat((INT8*) &ptmp_vol->vol_name[0]);
  if(status != ZFSERR_SUCCESS)
  {
  printf("FAILED");
  return;
  }
  else
  printf("SUCCESS");
  }
  }
  else
  printf("DONE") ;
  // Now call any Zilog File System APIs
  // Create a directory
  status = ZFSMkdir("EXTF:/","Dir.0");
  if(status != ZFSERR_SUCCESS)
  printf("New Directory is created");
  else
  printf("Unable to create a directory: %d", status);
  :
  :
}
```

Zilog File System APIs

The Zilog File System provides a number of standard APIs that execute different actions. These APIs are briefly described in Table 1. For more detailed information about the Zilog

File System APIs, refer to the Zilog File System Reference Manual (RM0039), which can be found on zilog.com and is also located in the following filepath:

```
<ZDS II Installed directory>\Program
Files\Zilog\ZDSII_eZ80Acclaim!_A.B.C\ZTP\ZTPX.Y.Z_Lib\RZK\Docs
```

Table 1. Zilog File System Standard API

Function Name	Description
`ZFSChdir`	Change the current working directory.
`ZFSClose`	Close the opened file.
`ZFSDelete`	Delete an existing file.
`ZFSDeleteDir`	Delete an existing directory or subdirectories.
`ZFSFormat`	Format the media used in the Zilog File System.
`ZFSGetCwd`	Returns the current working directory.
`ZFSGetCwdLen`	Returns the number of bytes contained in CWD string.
`ZFSGetDirFileCount`	Returns the number of files and directories present in the given directory.
`ZFSGetErrNum`	Returns the error number if recent Zilog File System API execution contains an error.
`ZFSGetVolumeCount`	Returns the number of volumes present in the system.
`ZFSGetVolumeParams`	Returns the volume parameters such as free space, used space, volume name and volume size.
`ZFSInit`	Initializes the Zilog File System and returns the invalid volume(s) information for the required processes such as formatting of the volume etc.
`ZFSList`	Lists all files and directories present in given path.
`ZFSMkdir`	Creates a directory under the given path.
`ZFSOpen`	Opens a file for reading/writing/appending or create a new file.
`ZFSRead`	Reads data from an opened file.
`ZFSRename`	Renames a file.
`ZFSRenameDir`	Renames a directory.
`ZFSSeek`	Sets file read/write pointer to the specified location.
`ZFSShutdown`	Uninitializes the file system.
`ZFSWrite`	Writes data to a opened file.

Table 2 provides a list of C Run-Time standard library APIs that are supported by the Zilog File System.

Table 2. Zilog File System – Supported C Run-Time Standard Library APIs

Function Name	Description
fopen	Opens a file for reading/writing.
fclose	Closes an opened file.
fputc	Puts a character into the file.
fgetc	Returns a character from the file.
fputs	Stores a string into the file.
fgets	Gets a string from the file.
fread	Reads the specified number of bytes from the file.
fwrite	Writes the specified number of bytes into the file.
fseek	Alters the file pointer position.
ftell	Returns the file pointer position.
feof	Determines whether it is end of file or not.

File and Directory Naming Conventions

The following conventions are applicable to the naming of directories, files and volumes in the Zilog File System:

- Names must start with an alphabet or with an underscore (_)
- Names must be less than 16 bytes in length
- Names can contain a combination of alphabets, numbers, periods (.) and underscores (_)
- Names must not contain two successive periods
- Names must not contain any special characters

Examples

The following list presents valid file\directory\volume names.

- `a_b.c`
- `a_b.c`
- `_b.c`
- `__b.c.txt`

Conversely, the following list presents file\directory\volume names that are invalid.

- `1a.c`

- `.2`
- `file.c`

Zilog File System Configuration

The Zilog File System is configured according to your requirements. The Zilog File System configuration file is located in the following path:

```
<ZDS II Installed directory>\Program Files\Zilog\
ZDSII_eZ80Acclaim!_A.B.C\ZTP\ZTPX.Y.Z_Lib\RZK\Conf
```

For internal and external Flash volumes, the Zilog File System requires equal-sized blocks to store files and directories. If block sizes are unequal, the behavior of the Zilog File System is unknown. The driver must be capable of handling the logical block sizes for reading/writing/erasing operations. Table 3 lists the logical block sizes for the sample Flash drivers provided with the RZK release.

Table 3. Logical Block Sizes for Sample Flash Drivers

Flash Type	Block Size	Starting Address	Comments
eZ80F91 Internal Flash	32 KB	0x8000	First 32 KB may contain the boot-up code for RST vectors.
eZ80F92 Internal Flash	16 KB	0x4000	First 16 KB may contain the boot-up code for RST vectors.
eZ80F93 Internal Flash	16 KB	0x4000	First 16 KB may contain the boot-up code for RST vectors.
MT28F008B	128 KB	0x120000	Only 128 KB equal sized blocks are used.
AT49BV162A	64 KB	0x330000	Only 64 KB equal sized blocks are used (eZ80F91 Mini module configuration).
AM29LV160B	64 KB	0x310000	Only 64 KB equal sized blocks are used (eZ80F91 Mini module configuration).

Zilog File System Macro Configuration

The Zilog File System provides macros for the configuration of different volumes and the behavior of the File System. These macros, described in Table 4, are present in the `ZFS_Conf.c` file, which is located in the following filepath:

```
<ZDS II Installed directory>\Program Files\Zilog\
ZDSII_eZ80Acclaim!_A.B.C\ZTP\ZTPX.Y.Z_Lib\RZK\Conf
```

You must provide the correct values of the macros and the volume configuration; otherwise, the behavior of the Zilog File System is not defined.

Table 4. Zilog File System Macros

Macro	Default Value	Description
`ZFS_TOTAL_NUM_BLOCKS`	7	This macro contains the total number of blocks present in the system. For each RAM volume, add 1 block. For each Flash volume, add a relative number of blocks. A block is a physical erase block in Flash.
`ZFS_TOTAL_NUM_SECTORS`	`(0xE0000/ZFS_SEC_SIZE)`	This macro contains the total number of sectors present in the Zilog File System, excluding the sectors present in RAM volumes.
`ZFS_TOTAL_NUM_VOLUMES`	1	This macro contains the number of volumes present in the Zilog File System.
`ZFS_MAX_FILE_OPEN_COUNT`	20	This macro contains the number of maximum file open instances at a time. Therefore, at a given point of time, a maximum of 20 file open instances is allowed. This value is per system value and not per volume value.
`ZFS_MAX_DIRS_SUPPORTED`	50	This macro contains the maximum number of directories present in the system. This value is per system and not per volume. This value also includes the root directories of volumes configured.

Zilog File System Volume Configuration

The Zilog File System provides a structure (`ZFS_CONFIG_t`) to accommodate the different parameters of a volume. These structures are briefly described in Table 5.

Table 5. Description of Structure Members of ZFS_CONFIG_t

Member	Description	Values It Contains
vol_name	This member contains the volume name; it starts with a letter or an underscore (_) and contains only letters, a number or an underscore (_).	String of a maximum length of 16 bytes.
vol_type	Volume type. This member contains the type of the volume.	This member contains any of the following values: ZFS_RAM_DEV_TYPE for a volume that resides in RAM. ZFS_EXT_FLASH_DEV_TYPE for a volume that resides in either internal or external Flash.
vol_addr	This member contains the starting address of the volume.	Starting address of the volume.
vol_size	Size of the volume in bytes.	Size of the volume in bytes.
vol_blks	Number of blocks present in the volume.	For a RAM volume, contains 1. For a Flash volume, this value relates to the number of physical erasable blocks that are present within the volume memory range.
vol_secs	Number of sectors present in the volume.	Volume size ÷ ZFS_SEC_SIZE. The Zilog File System supports only 512 bytes sector size. (The ZFS_SEC_SIZE macro is defined to be 512).
pfn_drv_init	Driver init function for file system storage device.	For a RAM volume, contains RamDrv_Init. For internal/external Flash volumes, place the function named FS_<device>_Init.
pfn_drv_read	Driver read function for file system storage device.	For a RAM volume, contains RamDrv_Read. For internal/external Flash volumes, place the function named FS_<device>_Read.
pfn_drv_write	Driver write function for file system storage device.	For a RAM volume, contains RamDrv_Write. For internal/external Flash volumes, place the function named FS_<device>_Write.
pfn_drv_erase	Driver erase function for file system storage device.	For a RAM volume, contains RamDrv_Erase. For internal/external Flash volumes, place the function named FS_<device>_Erase.
pfn_drv_close	Driver close function for file system storage device.	For a RAM volume, contains RamDrv_Close. For internal/external Flash volumes, place the function named FS_<device>_Close.

An example of volume configuration for RAM and Flash is provided in the code segment that follows. Configuration of the volume must be stored into the `g_zfs_cfg` variable that is present in the `ZFS_Conf.c` file.

```
typedef struct
{
  INT8 vol_name[ ZFS_MAX_VOL_NAME_LEN + 1] ;
  UINT8 vol_type ;                    // ZFS_VOL_RAM,
ZFS_VOL_INTFLASH,
  // ZFS_VOL_EXTFLASH

  UINT8* vol_addr;                    // starting address of volume.
  UINT32 vol_size ;                   // in bytes
  UINT vol_blks ;                     // number of blocks present in
                                      // the volume. (for RAM it
                                      // will be 1, for Flash
                                      // related to the erasable

  UINT vol_secs ;                     // units number of sectors
                                      // function pointers for all
// driver entries and other routines that require the different
// search algorithm function pointers for all devices.

  DRV_INIT pfn_drv_init ;
  DRV_READ pfn_drv_read ;
  DRV_WRITE pfn_drv_write ;
  DRV_ERASE pfn_drv_erase ;
  DRV_CLOSE pfn_drv_close ;

  // function pointers for Zilog File System routines
} ZFS_CONFIG_t, *PZFS_CONFIG_t ;
```

Sample Zilog File System Configuration

The sample Zilog File System configuration contains two volumes:

- One volume resides in Flash (EXTF) that starts at address location `0x120000`, with 7 blocks and a volume size of `0xE0000`
- One volume resides in RAM (RAMF) that starts at address location `0xB80000`, with 1 block and a volume size of `0x80000`

System-wide, 20 file open instances can be present at a time, and 50 directories can be created throughout the system.

The following code segment presents an example Zilog File System configuration file, ZFS_Conf.c.

```
#define ZFS_TOTAL_NUM_BLOCKS (7 + 1)
#define ZFS_TOTAL_NUM_SECTORS (0xE0000/ZFS_SEC_SIZE)
#define ZFS_TOTAL_NUM_VOLUMES (1 + 1)
#define ZFS_MAX_FILE_OPEN_COUNT (20)
#define ZFS_MAX_DIRS_SUPPORTED (50)
#define ERASE_FLASH (0)
ZFS_CONFIG_t g_zfs_cfg[ ZFS_TOTAL_NUM_VOLUMES ] =
{
  {
  "EXTF",                           // vol name
  ZFS_EXT_FLASH_DEV_TYPE,           // vol type
  (UINT8*)0x120000,                 // vol_start_addr
  0xE0000,                          // vol_size
  7,                                // vol_blocks
  (0xE0000/ZFS_SEC_SIZE),           // number of sectors
    FS_MT28F008_Init,
    FS_MT28F008_Read,
    FS_MT28F008_Write,
    FS_MT28F008_Erase,
    FS_MT28F008_Close
  },
  {
  "RAMF",                           // vol name
  ZFS_RAM_DEV_TYPE,                 // vol type
  (UINT8*)0xB80000,                 // vol_start_addr
  0x80000,                          // vol_size
  1,                                // vol_blocks
  (0x80000/ZFS_SEC_SIZE),           // number of sectors

  RamDrv_Init,
  RamDrv_Read,
  RamDrv_Write,
  RamDrv_Erase,
  RamDrv_Close

}
} ;
```

Integrating a Flash Driver

This chapter briefly describes how to create a new driver for a Flash device other than those supported, and how to integrate this Flash driver with the Zilog File System library so that files and data can be stored in the Flash driver within the structure of the Zilog File System.

The Zilog File System's hardware abstraction module requires the Flash driver to be written with an appropriate prototype and requires the function to return particular values when the function succeeds or fails.

The Zilog File System requires the Flash driver to provide basic access routines for Flash that perform the reading or writing of a number of bytes to and from Flash memory, in addition to performing the erasure of the physical blocks of Flash. For more information about the functionality of the basic access routines of the Flash driver that must be integrated with the Zilog File System, refer to Flash Driver APIs section in the Zilog Real-Time Kernel Reference Manual (RM0006).

Depending upon the characteristics of the Flash device[1], access to the Flash device can be made sequentially and on a first-come/first-served basis. To achieve this type of sequence, the developer must use any one of the synchronization objects present in RZK (for example, a semaphore). These functions are referred to as Flash driver wrapper functions in the Zilog File System.

When the Flash driver wrapper functions and driver routines are ready, the Flash driver is integrated with the Zilog File System to store files and directories in Flash memory. The Zilog File System provides a way of integrating a custom Flash driver that can be used to store files and directories.

The `ZFS_Conf.c` file defines a global variable, `g_zfs_cfg`, which is of the `ZFS_CONFIG_t` structure type. You can change the member values of the structure to suit your requirements. For more information about Zilog File System configuration, see <CrossRef>Zilog File System Configuration on page 8. For more information about creating a project workspace for your sample application, refer to Zilog Real-Time Kernel User Manual (UM0075).

To provide an example, suppose a custom Flash driver, with the name `MYFLASH`, has routines such as `MYFLASH_Init`, `MYFLASH_Read`, `MYFLASH_Write`, `MYFLASH_Erase` and `MYFLASH_Close` and that the starting address is `0x100000`, with seven erasing blocks to be used for the storage of files and directories for the Zilog File System. Each block contains 64 KB of space of storage. The configuration block must appear like the code segment provided below:

[1] Some Flash devices, upon reading a byte, return the status byte if the Flash device is currently operating in write or erase modes.

```
#define ZFS_TOTAL_NUM_BLOCKS        ( 7 )
#define ZFS_TOTAL_NUM_SECTORS       ((7 * 0x10000)/ZFS_SEC_SIZE )
#define ZFS_TOTAL_NUM_VOLUMES       ( 1 )

ZFS_CONFIG_t g_zfs_cfg = {
  "EXTF",                           // vol name
  ZFS_EXT_FLASH_DEV_TYPE,           // vol type for Flash
                                    // device type
  (UINT8*)0x100000,                 // vol_start_addr = 0x100000
  (7 * 0x10000),                    // vol_size (7 * 64KB)
  7,                                // vol_blocks
  ((7 * 0x10000)/ZFS_SEC_SIZE),     // number of sectors
  MYFLASH_Init,
  MYFLASH_Read,
  MYFLASH_Write,
  MYFLASH_Erase,
  MYFLASH_Close
  } ;
```

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.