THIS SPEC IS OBSOLETE

Spec No: 001-98506

Spec Title: AN98506 - CYPRESS FFS WITH TWO BLOCK DRIVERS

Replaced by: NONE

Cypress FFS with Two Block Drivers

AN98506 discusses options of Cypress FFS such as a single file system partition on a single flash device, multiple flash devices with a single block driver, as well as multiple partitions (logical disks) with a single block driver, or even two separate physical flash disks (block devices), where each disk contains one or more flash devices.

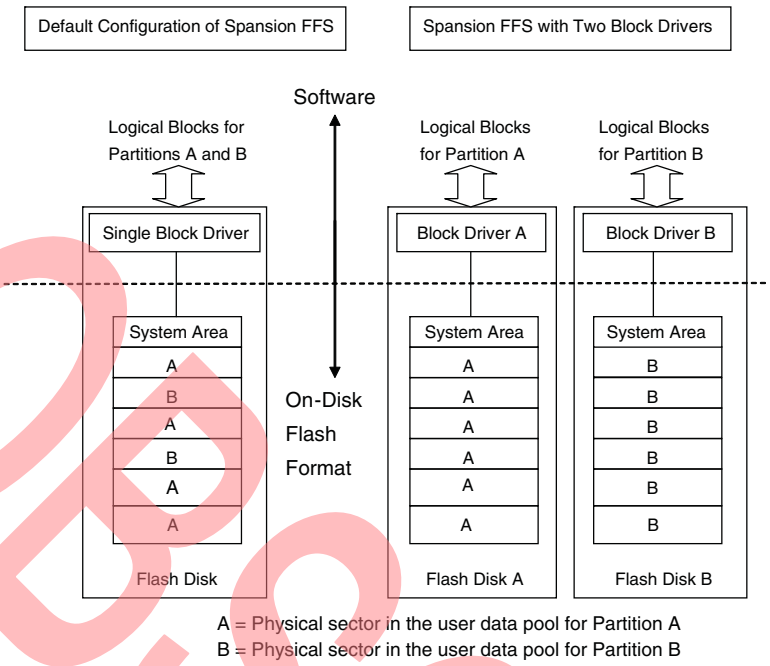
1 Introduction

The typical application of Cypress FFS is for a single file system partition on a single flash device. Cypress FFS also supports multiple flash devices with a single block driver, as well as multiple partitions (logical disks) with a single block driver. For some embedded applications, it may be valuable to store data on two separate physical flash disks (block devices), where each disk contains one or more flash devices.

2 One Block Driver vs. Two Block Drivers

For most platforms using Cypress FFS, a single block driver is sufficient. A single block driver can manage multiple flash devices as long as they are identical (same family and same density). When managing multiple logical disks with a single block driver the user data for each of the logical disks can be stored anywhere in the physical flash space. In other words, the block driver maps the sectors for multiple logical disks to arbitrary physical locations within a single pool of available space, where the mapping data is stored in a single system area.

When using separate physical flash disks, the system and user space for a given physical disk will be limited to that physical disk — thus the system area and user data for physical disk A are always disjoint from the formatted region for physical disk B. Also, each block device driver can be configured for a different type of flash. Separate block devices could be different parallel NOR flash families or one could be SPI and another could be NAND. If more than one flash device is used within a given disk, then all devices for that disk must be identical (same family and same density). Each block device will appear as a different drive letter at the file system interface. Applications can access the drives in parallel, but when closing the file system, both block devices have to be closed at the same time. This implementation assumes that each block driver manages space on independent physical flash devices. If the system must have two disk partitions on one flash device, then a single block device should be defined as in the previous paragraph.



3 Code Changes for Two Block Drivers

In order to support two block drivers, several updates are needed to the standard Cypress FFS release package. Change the following defines in SpansionFFS\FileSystem\custionfs.h from 1 to 2:

```
#define IONFS_DEVICE_NUM (2)
#define IONFS_VOLUME_NUM (2)
```

Note that setting the IONFS_DEVICE_NUM to a value greater than 1 will enable the following code in ion_pim_spansion.h.

```
#define IONFS_SPAN_DEVA0// device A is drive number 0
#if ( IONFS_DEVICE_NUM > 1 )
#define IONFS_SPAN_DEVB1// device B is drive number 1
#endif
```

If IONFS_DEVICE_NUM is set to 2, then nothing more needs to be done. If it's set to 3 or more, then duplicate the code that is wrapped by IONFS_SPAN_DEVB. Also, a new #define would be needed, for example IONFS_SPAN_DEVC, corresponding to the third device.

```
#if ( IONFS_DEVICE_NUM > 2 )
#define IONFS_SPAN_DEVC2// device C is drive number 2
#endif
```

Duplicate the applicable line for dev_setup in SpansionFFS\FileSystem\pim\ion_pim.c

```
static pim_setup_t dev_setup[] = {
   #if ( IONFS_DEVS & IONFS_DEV_ORNAND)
   {_TC("ornand"), pim_setup_span },
   #endif
   #if ( IONFS_DEVS & (IONFS_DEV_OneNAND | IONFS_DEV_NAND) )
   {_TC("nand"), pim_setup_ion},
   #endif
   #if ( IONFS_DEVS & IONFS_DEV_NOR )
   {_TC("nor"), pim_setup_span},
   {_TC("nor"), pim_setup_span}, /*2BDSOL******** duplicated this line for 2 BDs*******/
```

```
#endif
```

Update ten function calls in SpansionFFS\FileSystem\pim\spansion\ion_pim_spansion.c with the comment:

```
/*2BDSOL********RENAME TO POINT TO NEW BD:*******/
```

to point to the second block driver, for example:

```
case IONFS_SPAN_DEVB:
   /*2BDSOL********RENAME TO POINT TO NEW BD:*******/
   if((status = DEV_B_FTL_Format()) != FTL_ERR_PASS)
```

The above code is enabled by changing #define IONSFS_DEVICE_NUM to more than 1. Your build environment should already define __LLD (or __SLLD or __NLLD). Also add DEV_B__LLD (or DEV_B__SLLD or DEV_B__NLLD). If IONFS_DEVICE_NUM is set to 3 or more, then add new cases to the applicable functions in ion_pim_spansion.c, and create the necessary defines in the build environment (i.e. DEV_C__LLD).

4 Duplicate the Block Driver and Low-Level Driver

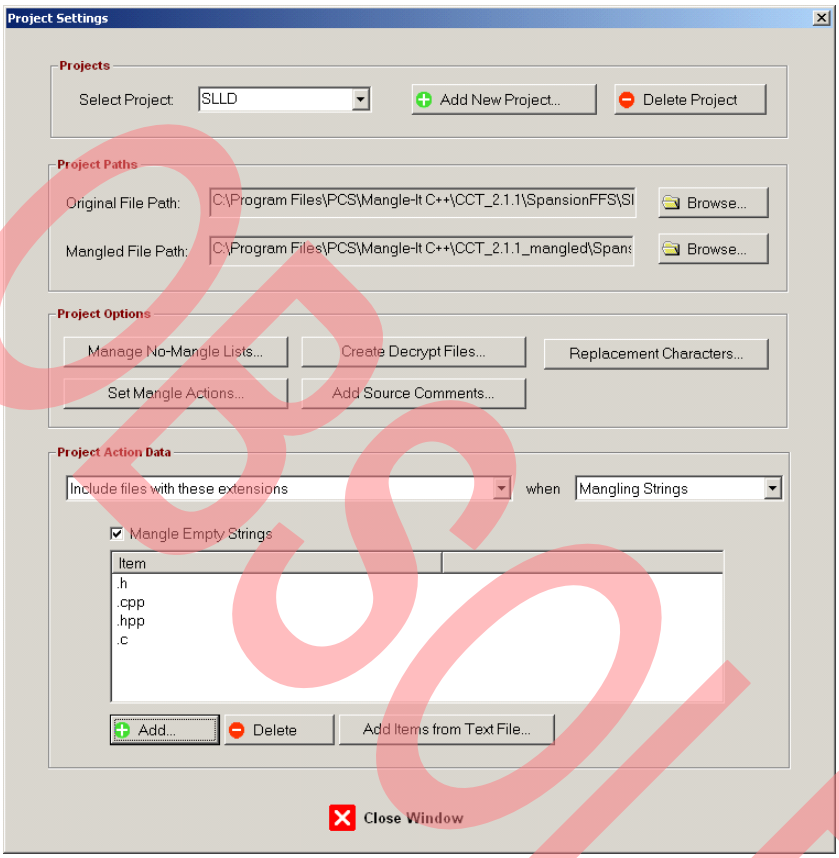
A second copy of the block driver (FTL) and the LLD (or SLLD or NLLD) must be generated with separate public APIs and variables. The Mangle-It obfuscator can be used to clone the existing block driver and LLD. A trial version of Mangle-It can be downloaded for free from http://www.pcsentinelsoftware.com/purchase.htm.

by clicking on “Purchase MangleIt C++ Code Obfuscator!” and then “Mangle-It C++ Source Code Obfuscator [#300178792]” and finally “Download Demo”.

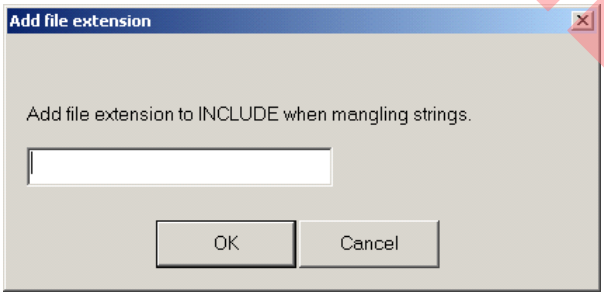
This tool can operate in Debug Mode with the GUI interface to prefix _debug_symbol_ to the variable names, function names, constants, and other symbols. It also prefixes _debug_file_ to the file names. The full version of Mangle-It enables a command line interface and Release Mode obfuscation, but these features are not needed.

1. Create a new Mangle-It project for the FTL.
   a. From the “Projects” menu, select “Create New Project Wizard”. Enter a unique name for your project to mangle the FTL tree and click “Next”. For the “Original File Path”, select the FTL tree, for example: C:\Program Files\ Spansion\SpansionFFS_2.1.1\CCT\SpansionFFS\FTL
   b. Create a new directory for the new copy of FTL, for example: C:\Program Files\ Spansion\SpansionFFS_2.1.1\CCT\SpansionFFS\DEV_B_FTL
   c. For the “Mangled File Path” select this new directory and click “Next”, “Next”, “Next”, “Next”, “Next”, and “Finish”.
2. Configure the Mangle-It project to obfuscate .c files.

a. Under the “Projects” menu, select “Project Settings”, and in the “Project Action Data” section, select “Mangling Strings” from the drop-down list on the right, and then select “Include files with these extensions” on the left.



b. Click on the “Add” button and enter “.c”, then click “OK”.



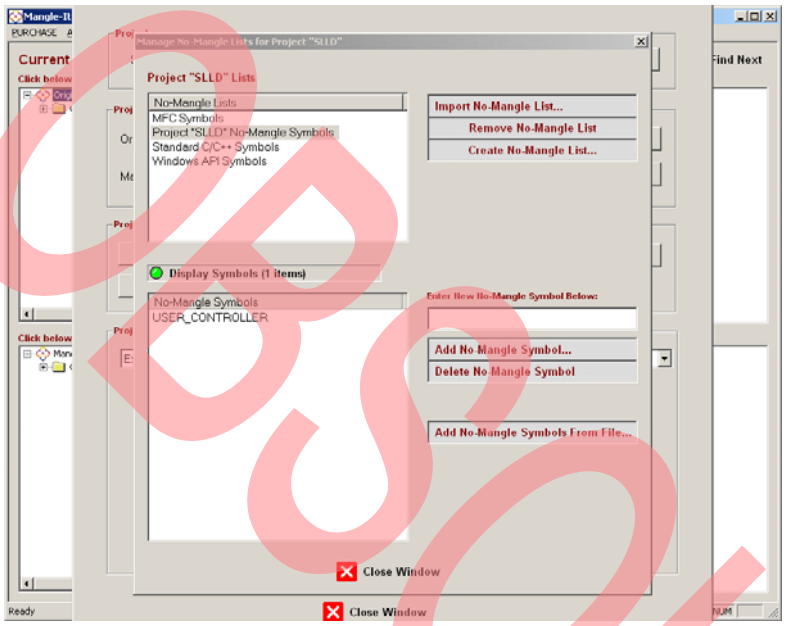
c. Also add the .c file extension for “Mangling Filenames” and “Mangling Symbols”.

3. Configure the Mangle-It project to not obfuscate certain symbols

a. Click on “Manage No-Mangle Lists”. Mangle-It includes lists of standard symbols that should not be obfuscated. Select the list for your project and click on “Display Symbols”. Add the following for your Mangle-It project list:

decry
os_type
table_storage

HSSIMU
hssim_Read_B
hssim_Write_B
hssim_Erase
numBlocks
blockSize
USER_CONTROLLER



Note this list was created using the SLLD. For LLD and NLLD, the no-mangle list may need to include additional symbols.

b. Create the obfuscated copy of FTL code by selecting “Mangle Project project files” from the “Actions” menu, where project is your project name, surrounded by quotes.

4. Create a new Mangle-It project for the LLD.

   a. Similarly for the LLD (or SLLD or NLLD), return to Step 1. and create a new Mangle-It project and a new directory, for example: C:\Program Files\Spansion\SpansionFFS_2.1.1\CCT\SpansionFFS\DEV_B_LLD
   In order to switch between the FTL and LLD Mangle-It projects, open the “Project Settings” window and click on the drop-down list for the “Select Project” field.

5. Replace the default obfuscation symbol prefix.

   a. All mangled filenames should be updated from “_debug_file_” to “DEV_B_”. Use global search and replace with a text editor, like Crimson Editor, to replace “_debug_file_” with “DEV_B_” and “_debug_symbol_” with “DEV_B_”.

6. Compile Cypress FFS

   a. If the obfuscated code does not compile properly, check the error messages. if additional symbols are used both inside and outside of the group of files being obfuscated, return to Step 3. and add them to the No-Mangle List.

   b. If IONFS_DEVICE_NUM is set to 3, then copy both the obfuscated FTL and LLD layers, replacing DEV_B with DEV_C. If IONFS_DEVICE_NUM is set to more than 3, make additional copies and update as needed.

5 File System Initialization Sequence

When calling the Cypress FFS public APIs to initialize the file system, there is added complexity with two block drivers. Each volume must be initialized (formatted and mounted) separately. During this initialization, supply both the volume ID number “/A/” or “/B/” and the device ID number (0 or 1).

Subsequent calls to the FFS will use “/A/” or “/B/” and Cypress FFS will manage the device ID automatically. The partition number is a DOS partition number that is a subset of the device ID.

The ionFS_zero_init() and ionFS_init() should be called only once, at the beginning of the sequence. For each volume, ionFS_mount() is called next, in case the volume was previously formatted. This avoids unnecessary formatting and possible loss of data. If the mount works, we can move forward. If the mount fails, the initialization code must format the block driver, format the file system, and attempt to mount the file system again. ionFS_get_sectors() must be called before ionFS_format() in order to supply the format size. The device must be formatted before ionFS_get_sectors() can determine the number of sectors. If ionFS_get_sectors() does not detect a formatted device, it will attempt a block driver format.

Below is a sample sequence of calls to initialize and format two devices.

```
int32_t sec_cnt,rtn;

ionFS_zero_init();
rtn = ionFS_init();
if ( 0 > rtn ) ionFS_exit(0);

rtn = ionFS_mount( _TC("/a/"), 0, 0, 0 );
if ( 0 > rtn )
{
if (IONFS_ENOFMT == ionFS_get_errno() )
{
rtn = pim_ioctl_span(0, eIOCTL_format, NULL);
if( 0 > rtn )
break;

sec_cnt = ionFS_get_sectors( 0 );
if ( 0 > sec_cnt )
break;

rtn = ionFS_format( _TC("/a/"), 0, 0, 0, sec_cnt, _TC("FAT16"), 0 );
if ( 0 > rtn )
break;

rtn = ionFS_mount( _TC("/a/"), 0, 0, 0 );
if ( 0 > rtn )
break;
}
else
{
break;
}
}
rtn = ionFS_mount( _TC("/b/"), 1, 0, 0 );
if ( 0 > rtn )
{
if ( IONFS_ENOFMT == ionFS_get_errno() )
{
rtn = pim_ioctl_span(1, eIOCTL_format, NULL);
if( 0 > rtn )
break;

sec_cnt = ionFS_get_sectors( 1 );
if ( 0 > sec_cnt )
```

```
break;

rtn = ionFS_format( _TC("/b/"), 1, 0, 0, sec_cnt, _TC("FAT16"), 0 );
if ( 0 > rtn )
break;

rtn = ionFS_mount( _TC("/b/"), 1, 0, 0 );
if ( 0 > rtn )
break;
}
else
{
break;
}
}
```

Refer to the Cypress FS Porting Guide, section 4.7.2, titled “Setup and Initialization of Devices and 2 Block Driver”, starting on page 30, for more information on initializing two block drivers.

6 Conclusion

The ability to access drives independently offers potential system performance benefits, as well as the ability to tailor memory usage to a specific device's partition size without affecting the other. It also allows flexibility and modularization in the product design, for example, allowing different devices to be targeting for developing at times in the product development cycle. Cypress FFS could be configured to support multiple drives and as long as at least one of them is present and mounts, the file system can successfully initialize. Any combination of supported drives could be used in the final product(s).

Document History Page

Document Title: AN98506 - Cypress FFS with Two Block Drivers Document Number: 001-98506				
Rev.	**ECN No.**	**Orig. of Change**	**Submission Date**	**Description of Change**
**	–	–	12/12/2012	Initial version
*A	4979087	MSWI	10/21/2015	Updated in Cypress template
*B	5872105	AESATMP8	09/04/2017	Updated logo and Copyright.
*C	6439004	SHIO	01/11/2019	Obsolete document

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Video | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709