Xilinx Standalone Library Documentation

XilMFS Library v2.3

UG649 (2017.1) April 5, 2017

XILINX

Table of Contents

Appendix A: Additional Resources and Legal Notices

XILINX

Chapter 1

Overview

The XilMFS library provides the capability to manage program memory in the form of file handles. You can create directories and have files within each directory. The file system can be accessed from the high-level C language through function calls specific to the file system.

XILINX

Chapter 2

XilMFS Library API

Overview

This chapter provides a linked summary and detailed descriptions of the XilMSF library APIs.

Functions

- void mfs_init_fs (int numbytes, char *address, int init_type)
- void mfs_init_genimage (int numbytes, char *address, int init_type)
- int mfs_change_dir (const char *newdir)
- int mfs_delete_file (char *filename)
- int mfs_create_dir (char *newdir)
- int mfs_delete_dir (char *newdir)
- int mfs_rename_file (char *from_file, char *to_file)
- int mfs_exists_file (char *filename)
- int mfs_get_current_dir_name (char *dirname)
- int mfs_get_usage (int *num_blocks_used, int *num_blocks_free)
- int mfs_dir_open (const char *dirname)
- int mfs_dir_close (int fd)
- int mfs_dir_read (int fd, char **filename, int *filesize, int *filetype)
- int mfs_file_open (const char *filename, int mode)
- int mfs_file_read (int fd, char *buf, int buflen)
- int mfs_file_write (int fd, const char *buf, int buflen)
- int mfs_file_close (int fd)
- long mfs_file_lseek (int fd, long offset, int whence)
- int mfs_ls ()
- int mfs_ls_r (int recurse)
- int mfs_cat (char *filename)
- int mfs_copy_stdin_to_file (char *filename)
- int mfs_file_copy (char *from_file, char *to_file)

Function Documentation

void mfs_init_fs (int *numbytes,* char * *address,* int *init_type*)

Initialize the file system.
This function must be called before any file system operations. Use `mfs_init_genimage()` instead of this function for initializing with file images generated by `mfsgen`.

Parameters

numbytes	Number of bytes allocated or reserved for this file system.
address	Starting address of the memory block. address must be word aligned (4 byte boundary).
init_type	• MFSINIT_NEW creates a new, empty file system for read/write • MFSINIT_IMAGE initializes a file system whose data has been previously loaded into memory at the base address. • MFSINIT_ROM_IMAGE initializes a Read-Only file system whose data has been previously loaded into memory at the base address.

void mfs_init_genimage (int *numbytes,* char * *address,* int *init_type*)

Initialize the file system with a file image generated by `mfsgen`.
This function must be called before any file system operations. Use `mfs_init_fs()` instead of this function for other initialization.

Parameters

numbytes	Number of bytes allocated or reserved for this file sy stem.
address	Starting address of the memory block. address must be word aligned (4 byte boundary).
init_type	• MFSINIT_IMAGE initializes a file system whose data has been previously loaded into memory at the base address. • MFSINIT_ROM_IMAGE initializes a Read-Only file system whose data has been previously loaded into memory at the base address.

int mfs_change_dir (const char * *newdir*)

Modify `global mfs_current_dir` to index of `newdir` if it exists.
`mfs_current_dir` is not modified otherwise.

Parameters

newdir	is the name of the new directory

Returns

1 for success and 0 for failure

int mfs_delete_file (char * *filename*)

Delete a file from directory.

WARNING: *This function does not completely free up the directory space used by the file. Repeated calls to create and delete files can cause the file system to run out of space.*

Parameters

filename	Name of the file to be deleted. Delete the data blocks corresponding to the file and then delete the file entry from its directory.

Returns

1 on success, 0 on failure

Note

Delete will not work on a directory unless the directory is empty.

int mfs_create_dir (char * *newdir*)

Create a new empty directory called newdir inside the current directory.

Parameters

newdir	is the name of the directory

Returns

index of new directory in the file system on success. 0 on failure

int mfs_delete_dir (char * *newdir*)

Delete the directory named newdir if it exists, and is empty.

Parameters

newdir	is the name of the directory

Returns

Index of new directory in the file system on success. 0 on failure.

int mfs_rename_file (char * *from_file*, char * *to_file*)

Rename `from_file` to `to_file`.
Rename works for directories as well as files. Function fails if to_file already exists. works for dirs as well as files cannot rename to something that already exists

Parameters

from_file	
to_file	

Returns

1 on success, 0 on failure

int mfs_exists_file (char * *filename*)

check if a file exists

Parameters

filename	is the name of the file

Returns

0 if filename is not a file in the current directory
1 if filename is a file in the current directory
2 if filename is a directory in the current directory

int mfs_get_current_dir_name (char * *dirname*)

get the name of the current directory

Parameters

dirname	= pre_allocated buffer of at least MFS_MAX_FILENAME_SIZE+1 chars The directory name is copied to this buffer

Returns

1 if success, 0 if failure

int mfs_get_usage (int * *num_blocks_used,* int * *num_blocks_free*)

get the number of used blocks and the number of free blocks in the file system through pointers

Parameters

num_blocks_used	
num_blocks_free	the return value is 1 (for success) and 0 for failure to obtain the numbers

int mfs_dir_open (const char * *dirname*)

open a directory for reading each subsequent call to mfs_dir_read() returns one directory entry until end of directory

Parameters

dirname	is the name of the directory to open

Returns

index of dir in array mfs_open_files or -1

int mfs_dir_close (int *fd*)

close a directory - same as closing a file

Parameters

fd	is the descriptor of the directory to close

Returns

1 on success, 0 otherwise

int mfs_dir_read (int *fd*, char ** *filename*, int * *filesize*, int * *filetype*)

read values from the next valid directory entry The last 3 parameters are output values

Parameters

fd	is the file descriptor for an open directory file
filename	is a pointer to the filename within the MFS itself
filesize	is the size in bytes for a regular file or the number of entries in a directory
filetype	is MFS_BLOCK_TYPE_FILE or MFS_BLOCK_TYPE_DIR

Returns

1 for success and 0 for failure or end of dir

int mfs_file_open (const char * *filename*, int *mode*)

open a file

Parameters

filename	is the name of the file to open
mode	is MFS_MODE_READ or MFS_MODE_WRITE or MFS_MODE_CREATE this function should be used for FILEs and not DIRs no error checking (is this FILE and not DIR?) is done for MFS_MODE_READ MFS_MODE_CREATE automatically creates a FILE and not a DIR MFS_MODE_WRITE fails if the specified file is a DIR

Returns

index of file in array mfs_open_files or -1

int mfs_file_read (int *fd*, char * *buf*, int *buflen*)

read characters to a file

Parameters

fd	is a descriptor for the file from which the characters are read
buf	is a pre allocated buffer that will contain the read characters
buflen	is the number of characters from buf to be read fd should be a valid index in mfs_open_files array Works only if fd points to a file and not a dir buf should be a pointer to a pre-allocated buffer of size buflen or more buflen chars are read and placed in buf if fewer than buflen chars are available then only that many chars are read

Returns

num bytes read or 0 for error=no bytes read

int mfs_file_write (int *fd,* const char * *buf,* int *buflen*)

write characters to a file

Parameters

fd	is a descriptor for the file to which the characters are written
buf	is a buffer containing the characters to be written out
buflen	is the number of characters from buf to be written out fd should be a valid index in mfs_open_files array buf should be a pointer to a pre-allocated buffer of size buflen or more buflen chars are read from buf and written to 1 or more blocks of the file

Returns

1 for success or 0 for error=unable to write to file

int mfs_file_close (int *fd*)

close an open file and recover the file table entry in mfs_open_files corresponding to the fd if the fd is not valid, return 0 fd is not valid if the index in mfs_open_files is out of range, or if the corresponding entry is not an open file

Parameters

fd	is the file descriptor for the file to be closed

Returns

1 on success, 0 otherwise

long mfs_file_lseek (int *fd,* long *offset,* int *whence*)

seek to a given offset within the file

Parameters

fd	should be a valid file descriptor for an open file
whence	is one of MFS_SEEK_SET, MFS_SEEK_CUR or MFS_SEEK_END
offset	is the offset from the beginning, end or current position as specified by the whence parameter if MFS_SEEK_END is specified, the offset can be either 0 or negative otherwise offset should be positive or 0 it is an error to seek before beginning of file or after the end of file

Returns

-1 on failure, value of the offset from the beginning of the file, on success

int mfs_ls ( )

list contents of current directory

Returns

1 on success and 0 on failure

int mfs_ls_r (int *recurse*)

recursive directory listing list the contents of current directory if any of the entries in the current directory is itself a directory, immediately enter that directory and call mfs_ls_r() once again

Parameters

recurse	If parameter recurse is non zero continue recursing else stop recursing recurse=0 lists just the current directory recurse = -1 allows unlimited recursion recurse = n stops recursing at a depth of n

Returns

1 on success and 0 on failure

int mfs_cat (char * *filename*)

print the file to stdout

Parameters

filename	- file to print

Returns

1 on success, 0 on failure

int mfs_copy_stdin_to_file (char * *filename*)

copy from stdin to named file

Parameters

filename	- file to print

Returns

1 on success, 0 on failure

int mfs_file_copy (char * *from_file*, char * *to_file*)

copy from_file to to_file to_file is created new copy fails if to_file exists already copy fails is from_file or to_file cannot be opened

Parameters

from_file	
to_file	

Returns

1 on success, 0 on failure

Chapter 3

Utility Functions

This chapter provides a summary and detailed descriptions of the utility functions that can be used along with the MFS.
These functions are defined in `mfs_filesys_util.c` and are declared in `xilmfs.h`. /**

Chapter 4

Library Parameters in MSS File

A memory file system can be integrated with a system using the following snippet in the Microprocessor Software Specification (MSS) file.

```
BEGIN LIBRARY
parameter LIBRARY_NAME = xilmfs
parameter LIBRARY_VER = 2.3
parameter numbytes= 50000
parameter base_address = 0xffe00000
parameter init_type = MFSINIT_NEW
parameter need_utils = false END
```

The memory file system must be instantiated with the name xilmfs. The following table lists the libgen customization parameters.

Parameter	Description
LIBRARY_NAME	Specifies the library name. Default is `xilmfs`
LIBRARY_VER	Specifies the library version. Default is 2.3
numbytes	Number of bytes allocated for file system.
base_address	Starting address for file system memory.
init_type	Options are: MFSINIT_NEW (default) creates a new, empty file system. MFSINIT_ROM_IMAGE creates a file system based on a pre-loaded memory image loaded in memory of size numbytes at starting address base_address. This memory is considered read-only and modification of the file system is not allowed. MFS_INIT_IMAGE is similar to the previous option except that the file system can be modified, and the memory is readable and writable.

Parameter	Description
need_utils	true or false (default=false) If true, this causes `stdio.h` to be included from `mfs_config.h`. The functions described in Utility Functions require that you have defined `stdin` or `stdout`. Setting the need_utils to true causes stdio.h to be included.

WARNING: *The underlying software and hardware platforms must support stdin and stdout peripherals for these utility functions to compile and link correctly.*

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support .

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

Automotive Applications Disclaimer

AUTOMOTIVE PRODUCTS (IDENTIFIED AS ”XA” IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE (”SAFETY APPLICATION”) UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD (”SAFETY DESIGN”). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.