UG649 April 1, 2014

LibXil Memory File System (MFS) (v2.0)

Overview

The LibXil MFS 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.

MFS Functions

This section provides a linked summary and descriptions of MFS functions.

MFS Function Summary

The following list is a linked summary of the supported MFS functions. Descriptions of the functions are provided after the summary table. You can click on a function in the summary list to go to the description.

MFS Function Descriptions

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

Parameters	*`numbytes`* is the number of bytes of memory available for the file system. *`address`* is the starting(base) address of the file system memory. *`init_type`* is `MFSINIT_NEW`, `MFSINIT_IMAGE`, or `MFSINIT_ROM_IMAGE`.
Description	Initialize the memory file system. This function must be called before any file system operation. Use `mfs_init_genimage` instead of this function if the filesystem is being initialized with an image generated by `mfsgen`. The status/mode parameter determines certain filesystem properties: • `MFSINIT_NEW` creates a new, empty file system for read/write. • `MFSINIT_IMAGE` initializes a filesystem whose data has been previously loaded into memory at the base address. • `MFSINIT_ROM_IMAGE` initializes a Read-Only filesystem whose data has been previously loaded into memory at the base address.
Includes	`xilmfs.h`

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

Parameters	*`numbytes`* is the number of bytes of memory in the image generated by the mfsgen tool. This is equal to the size of the memory available for the file system, plus 4. *`address`* is the starting(base) address of the image. *`init_type`* is either `MFSINIT_IMAGE` or `MFSINIT_ROM_IMAGE`
Description	Initialize the memory file system with an image generated by mfsgen. This function must be called before any file system operation. The status/mode parameter determines certain filesystem properties: • `MFSINIT_IMAGE` initializes a filesystem whose data has been previously loaded into memory at the base address. • `MFSINIT_ROM_IMAGE` initializes a Read-Only filesystem whose data has been previously loaded into memory at the base address.
Includes	`xilmfs.h`

`int` **`mfs_change_dir`**`(char` *`*newdir`*`)`

Parameters	*`newdir`* is the chdir destination.
Returns	1 on success. 0 on failure.
Description	If *`newdir`* exists, make it the current directory of MFS. Current directory is not modified in case of failure.
Includes	`xilmfs.h`

`int` **`mfs_create_dir`**`(char *newdir)`

Parameters	*`newdir`* is the directory name to be created.
Returns	Index of new directory in the file system on success. 0 on failure.
Description	Create a new empty directory called *`newdir`* inside the current directory.
Includes	`xilmfs.h`

`int` **`mfs_delete_dir`**`(char *dirname)`

Parameters	*`dirname`* is the directory to be deleted.
Returns	Index of new directory in the file system on success. 0 on failure.
Description	Delete the directory dirname, if it exists and is empty.
Includes	`xilmfs.h`

`int` **`mfs_get_current_dir_name`**`(char *dirname)`

Parameters	*`dirname`* is the current directory name.
Returns	1 on success. 0 on failure.
Description	Return the name of the current directory in a preallocated buffer, *`dirname,`* of at least 16 chars. It does not return the absolute path name of the current directory, but just the name of the current directory.
Includes	`xilmfs.h`

`int` **`mfs_delete_file`**`(char *filename)`

Parameters	*`filename`* is the file to be deleted.
Returns	1 on success. 0 on failure.
Description	Delete *`filename`* from the directory.
Includes	`xilmfs.h`

Caution! This function does not completely free up the directory space used by the file. Repeated calls to create and delete files can cause the filesystem to run out of space.

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

Parameters	*`from_file`* is the original filename. *`to_file`* is the new file name.
Returns	1 on success. 0 on failure.
Description	Rename *`from_file`* to *`to_file`*. Rename works for directories as well as files. Function fails if *`to_file`* already exists.
Includes	`xilmfs.h`

`int` **`mfs_exists_file`**`(char *`*`filename`*`)`

Parameters	*`filename`* is the file or directory to be checked for existence.
Returns	`0` if *`filename`* does not exist. `1` if *`filename`* is a file. `2` if *`filename`* is a directory.
Description	Check if the file/directory is present in current directory.
Includes	`xilmfs.h`

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

Parameters	*`num_blocks_used`* is the number of blocks used. *`num_blocks_free`* is the number of free blocks.
Returns	1 on success. 0 on failure.
Description	Get the number of used blocks and the number of free blocks in the file system through pointers.
Includes	`xilmfs.h`

`int` **`mfs_dir_open`**`(char *`*`dirname`*`)`

Parameters	*`dirname`* is the directory to be opened for reading.
Returns	The index of dirname in the array of open files on success. `-1` on failure.
Description	Open directory dirname for reading. Reading a directory is done using `mfs_dir_read( )`.
Includes	`xilmfs.h`

`int mfs_dir_close(int fd)`

Parameters	*fd* is file descriptor return by open.
Returns	`1` on success. `0` on failure.
Description	Close the dir pointed by *fd*. The file system regains the fd and uses it for new files.
Includes	`xilmfs.h`

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

Parameters	*fd* is the file descriptor return by open; passed to this function by caller. *filename* is the pointer to file name at the current position in the directory in MFS; this value is filled in by this function. *filesize* is the pointer to a value filled in by this function: Size in bytes of filename, if it is a regular file; Number of directory entries if filename is a directory. *filetype* is the pointer to a value filled in by this function: `MFS_BLOCK_TYPE_FILE` if *filename* is a regular file. `MFS_BLOCK_TYPE_DIR` if *filename* is a directory.
Returns	`1` on success. `0` on failure.
Description	Read the current directory entry and advance the internal pointer to the next directory entry. *filename*, *filetype*, and *filesize* are pointers to values stored in the current directory entry.
Includes	`xilmfs.h`

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

Parameters	*filename* is the file to be opened. *mode* is Read/Write or Create.
Returns	The index of filename in the array of open files on success. -1 on failure.
Description	Open file filename with given mode. The function should be used for files and not directories: • `MODE_READ`, no error checking is done (if file or directory). • `MODE_CREATE` creates a file and not a directory. • `MODE_WRITE` fails if the specified file is a `DIR`.
Includes	`xilmfs.h`

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

Parameters	*fd* is the file descriptor return by open. *buf* is the destination buffer for the read. *buflen* is the length of the buffer.
Returns	Number of bytes read on success. 0 on failure.
Description	Read *buflen* number bytes and place it in *buf*. *fd* should be a valid index in "open files" array, pointing to a file, not a directory. *buf* should be a pre-allocated buffer of size *buflen* or more. If fewer than *buflen* chars are available then only that many chars are read.
Includes	`xilmfs.h`

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

Parameters	*fd* is the file descriptor return by open. *buf* is the source buffer from where data is read. *buflen* is the length of the buffer.
Returns	1 on success. 0 on failure.
Description	Write *buflen* number of bytes from *buf* to the file. *fd* should be a valid index in open_files array. *buf* should be a pre-allocated buffer of size buflen or more. ***Caution!*** Writing to locations other than the end of the file is not supported. Using `mfs_file_lseek( )` go to some other location in the file then calling `mfs_file_write( )` is not supported
Includes	`xilmfs.h`

int **mfs_file_close**(int *fd*)

Parameters	*fd* is the file descriptor return by open.
Returns	1 on success. 0 on failure.
Description	Close the file pointed by *fd*. The file system regains the *fd* and uses it for new files.
Includes	`xilmfs.h`

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

Parameters	`fd` is the file descriptor return by open. `offset` is the number of bytes to seek. `whence` is the file system dependent mode: • `MFS_SEEK_END`, then `offset` can be either 0 or negative, otherwise `offset` is non-negative. • `MFS_SEEK_CURR`, then `offset` is calculated from the current location. • `MFS_SEEK_SET`, then `offset` is calculated from the start of the file.
Returns	Returns `offset` from the beginning of the file to the current location on success. `-1` on failure: the current location is not modified.
Description	Seek to a given `offset` within the file at location `fd` in open_files array. ***Caution!*** It is an error to seek before beginning of file or after the end of file. ***Caution!*** Writing to locations other than the end of the file is not supported. Using the `mfs_file_lseek( )` function or going to some other location in the file then calling `mfs_file_write( )` is not supported.
Includes	`xilmfs.h`

Utility Functions

The following subsections provide a summary and the 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`.

Utility Function Summary

The following list is a linked summary of the supported MFS Utility functions. Descriptions of the functions are provided after the summary table. You can click on a function in the summary list to go to the description.

int mfs_ls(void)
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)

Utility Function Descriptions

`int` **`mfs_ls`**`(void)`

Parameters	None.
Returns	`1` on success. `0` on failure.
Description	List contents of current directory on `STDOUT`.
Includes	`xilmfs.h`

`int` **`mfs_ls_r`**`(int` *`recurse`*`)`

Parameters	*`recurse`* controls the amount of recursion: • `0` lists the contents of the current directory and stop. • > `0` lists the contents of the current directory and any subdirectories up to a depth of *`recurse`*. • = `-1` completes recursive directory listing with no limit on recursion depth.
Returns	1 on success. 0 on failure.
Description	List contents of current directory on `STDOUT`.
Includes	`xilmfs.h`

`int` **`mfs_cat`**`(char*` *`filename`*`)`

Parameters	*`filename`* is the file to be displayed.
Returns	`1` on success. `0` on failure.
Description	Print the file to `STDOUT`.
Includes	`xilmfs.h`

`int` **`mfs_copy_stdin_to_file`**`(char *`*`filename`*`)`

Parameters	*`filename`* is the destination file.
Returns	1 on success. 0 on failure.
Description	Copy from `STDIN` to named file. An end-of-file (EOF) character should be sent from `STDIN` to allow the function to return 1.
Includes	`xilmfs.h`

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

Parameters	*`from_file`* is the source file. *`to_file`* is the destination file.
Returns	`1` on success. `0` on failure.
Description	Copy *`from_file`* to *`to_file`*. Copy fails if *`to_file`* already exists or either from or to location cannot be opened.
Includes	`xilmfs.h`

Additional Utilities

The `mfsgen` program is provided along with the MFS library. You can use mfsgen to create an MFS memory image on a host system that can be subsequently downloaded to the embedded system memory. The mfsgen links to LibXil MFS and is compiled to run on the host machine rather than the target MicroBlaze™ or Cortex A9 processor system. Conceptually, this is similar to the familiar zip or tar programs.

An entire directory hierarchy on the host system can be copied to a local MFS file image using mfsgen. This file image can then be downloaded on to the memory of the embedded system for creating a pre-loaded file system.

Test programs are included to illustrate this process. For more information, see the `readme.txt` file in the `utils` sub-directory.

Usage: **mfsgen -{c** *`filelist`*| **t** | **x} vsb** *`num_blocks`* **f** *`mfs_filename`*

Specify exactly one of `c`, `t`, or `x` modes

`c`: creates an mfs file system image using the list of files specified on the command line (directories specified in this list are traversed recursively).

`t`: lists the files in the mfs file system image

`x`: extracts the mfs file system from image to host file system

`v`: is verbose mode

`s`: switches endianness

`b`: lists the number of blocks (*`num_blocks`*) which should be more than 2

- If the `b` option is specified, the *`num_blocks`* value should be specified
- If the `b` option is omitted, the default value of *`num_blocks`* is 5000
- The `b` option is meaningful only when used in conjunction with the c option

f: specify the host file name (*`mfs_filename`*) where the mfs file system image is stored

- If the `f` option is specified, the mfs filename should be specified
- If the `f` option is omitted, the default file name is `filesystem.mfs`

Libgen Customization

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 = 2.0
  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 attributes used by Libgen.

Table 1: **Attributes for Including Memory File System**

Attributes	Description
`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.
`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," page 7 require that you have defined `stdin` or `stdout`. Setting the `need_utils` to true causes stdio.h to be included. ***Caution!*** The underlying software and hardware platforms must support `stdin` and `stdout` peripherals for these utility functions to compile and link correctly.