

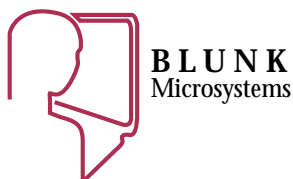
TargetFFS-NAND™

FLASH FILE SYSTEM

HIGHLIGHTS

- Reliable, Re-entrant Embedded File System
- Provides POSIX and Standard C APIs
- Guaranteed Integrity Across Unexpected Resets
- Use of NAND Flash Memory is Invisible to Applications
- Supports Dynamic Creation and Deletion of Files, Directories, and Links
- Performs Wear-Leveling, Bad Block Management, and ECC
- Supports Multiple Volumes of Unlimited Size
- Licensed as Royalty-Free Source Code
- Portable to Virtually any RTOS and Toolchain
- Includes Sample Applications

Blunk Microsystems provides system software, device drivers, and board support packages to the embedded systems market, both off-the-shelf products and custom work done under contract.



TargetFFS-NAND is a reliable, re-entrant embedded file system with POSIX and ANSI C compliant application program interfaces (APIs). Supports dynamic creation and deletion of files, directories, and links with read and write capability. Not a static ROM-image file system. Use of NAND flash memory for the backing store is invisible to the application layer.

File system integrity is guaranteed across unexpected shutdowns. Only data written since the last synchronizing operation (`fclose()`, `fflush()`, etc.) can be lost. Directory structures, closed files, and files open for reading are never at risk.

Implements wear-leveling to prolong the life of the flash memory. Erase cycles are spread evenly across all erasable blocks. A wear count is maintained starting with the first time a flash volume is formatted and the current wear count is available to applications via the `vstat()` call.

Performs bad block management. Both the bad blocks that exist in new devices and those that arise during operation are detected. Recovery is performed without user intervention. Once detected, bad blocks are neither programmed nor erased.

“Thin” driver layer performs only the basic operations that flash devices support: read page, write page, erase block, etc. The driver is stateless. Sample drivers for several NAND chips are provided and these are easily ported to new devices.

Supports concurrent use of multiple volumes. Flash volumes can be permanently installed at startup, or added and deleted as removable devices come and go. Volume size is unlimited. A single volume can be implemented using multiple NAND devices.

Supports the UNIX “self”, “group”, and “other” file access protections, allowing applications to restrict some operations to privileged tasks. The file system calls `FsGetId()`, implemented by the application, to get the running task’s user and group IDs.

Includes fast and efficient error correction code (ECC) routines that detect up to 8 bit errors and correct up to 4 bit errors. Supports hardware-based ECC if present.

Optimized for fast mounts. Typical mount time for a 64MB volume is approximately one second. Ongoing garbage collection ensures minimal use of RAM. Can coexist with TargetFFS-NOR™, Blunk Microsystems’ NOR flash file system.

Sample applications include a binary search test, a boot loader, and a command line shell. The boot loader recovers from power loss during an application update. It loads the most recent copy that has been successfully written to flash. The shell supports “cd”, “ls”, “mkdir”, “pwd”, etc. and may be extended with user commands.

Developed using TargetOS™, Blunk Microsystems’ real-time operating system, TargetFFS-NAND has been ported to several other real-time kernels including Nucleus, VxWorks, and Precise MQX. Can be used without a kernel. Works on big- and little-endian systems.

The TargetFFS-NAND source code is 100% ANSI C and is integrated with CodeWarrior. It has been used with ANSI C compilers from ARM, Diab Data, GNU, Software Development Systems, and WindRiver.

Royalty free. Includes full source code, user’s manual, sample applications, and one year of technical support. ►

CONTACT INFORMATION

- Visit our web site:
www.blunkmicro.com
- Customer Support:
(408) 323-9833
- Technical Support:
(408) 323-1758
- Fax:
(408) 323-1757
- E-mail:
sales@blunkmicro.com
- Address:
**6576 Leyland Park Drive
San Jose, CA 95120
USA**

OTHER COMPONENTS

TargetOS

Scalable, Deterministic, Priority-Based Preemptive Real-Time Kernel

TargetTCP

High Performance TCP/IP Stack with Standard Sockets API

TargetLAPB

LAPB Stack for Fast Data Transfer on Point-to-Point Networks

TargetWeb

HTTP 1.0 and 1.1 Compliant Embedded Web Server

TargetZFS

Compressed Data Read-Only Embedded File System

TargetFFS-NOR

Reliable Power-Fail-Safe Embedded NOR Flash File System

The TargetFFS-NAND Application Program Interface:

```
int access(const char *path, int amode);
int chdir(const char *path);
int chmod(const char *path, mode_t mode);
int chown(const char *path, uid_t owner, gid_t group);
void clearerr(FILE* stream);
int close(int fid);
int closedir(DIR *dirp);
int dup(int fid);
int dup2(int fid, int fid2);
int fclose(FILE *stream);
int fcntl(int fid, int cmd, ...);
FILE *fdopen(int fid, const char *mode);
int feof(FILE *stream);
int ferror(FILE *stream);
int fflush(FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
char *fgets(char *s, int n, FILE *stream);
int fileno(FILE *stream);
FILE *fopen(const char *filename, const char *mode);
int format(char *path);
int fprintf(FILE *stream, const char *format, ...);
int fputc(int c, FILE *stream);
int fputs(const char *string, FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
FILE *freopen(const char *filename, const char *mode, FILE *stream);
int fscanf(FILE *stream, const char *format, ...);
int fseek(FILE *stream, long offset, int mode);
int fsetpos(FILE *stream, const fpos_t *pos);
int fstat(int fid, struct stat *buf);
long ftell(FILE *stream);
int ftruncate(int fid, off_t length);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
int getc(FILE *stream);
int getchar(void);
char *getcwd(char *buf, size_t size);
char *gets(char *s);
int isatty(int fid);
int link(const char *existing, const char *new);
off_t lseek(int fid, off_t offset, int whence);
int mkdir(const char *path, mode_t mode);
int mount(char *path);
int open(const char *path, int oflag, ...);
DIR *opendir(const char *dirname);
void perror(const char *s);
int printf(const char *format, ...);
int putc(int c, FILE *stream);
int putchar(int c);
int puts(const char *s);
int read(int fid, void *buf, unsigned int nbyte);
struct dirent *readdir(DIR *dirp);
int remove(const char *filename);
int rename(const char *old, const char *new);
void rewind(FILE *stream);
void rewinddir(DIR *dirp);
int rmdir(const char *path);
int scanf(const char *format, ...);
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
int stat(const char *path, struct stat *buf);
int sync(int fid);
FILE *tmpfile(void);
char *tmpnam(char *s);
int truncate(const char *path, off_t length);
int ungetc(int c, FILE *stream);
int unlink(const char *path);
int unmount(char *path);
int utime(const char *path, const struct utimbuf *times);
int vfprintf(FILE *stream, const char *format, va_list arg);
int vprintf(const char *format, va_list arg);
int vstat(const char *path, union vstat *buf);
int write(int fid, const void *buf, unsigned int nbyte); ■
```

Licensing Terms

TargetFFS-NAND™ is royalty free. Purchasers are granted a non-exclusive license to use the provided source code at a single site. Licensees have the right to disseminate or resell the software in executable format only. The source code and derived object code may not be redistributed or resold.

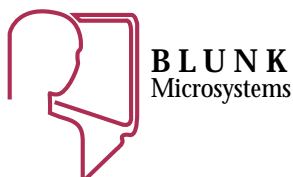
TargetFFS-NOR™

FLASH FILE SYSTEM

HIGHLIGHTS

- Reliable, Re-entrant Embedded File System
- Provides POSIX and Standard C APIs
- Guaranteed Integrity Across Unexpected Resets
- Use of NOR Flash Memory is Invisible to Applications
- Supports Dynamic Creation and Deletion of Files, Directories, and Links
- Performs Wear-Leveling
- Supports Multiple Volumes of Unlimited Size
- Licensed as Royalty-Free Source Code
- Portable to Virtually any RTOS and Toolchain
- Includes Sample Applications

Blunk Microsystems provides system software, device drivers, and board support packages to the embedded systems market, both off-the-shelf products and custom work done under contract.



TargetFFS-NOR is a reliable, re-entrant embedded file system with POSIX and ANSI C compliant application program interfaces (APIs). Supports dynamic creation and deletion of files, directories, and links with read and write capability. Not a static ROM-image file system. Use of NOR flash memory for the backing store is invisible to the application layer.

File system integrity is guaranteed across unexpected shutdowns. Only data written since the last synchronizing operation (`fclose()`, `fflush()`, etc.) can be lost. Directory structures, closed files, and files open for reading are never at risk.

Implements wear-leveling to prolong the life of the flash memory. Erase cycles are spread evenly across all erasable blocks. A wear count is maintained starting with the first time a flash volume is formatted and the current wear count is available to applications via the `vstat()` call.

Supports the large erasable block sizes typical of NOR flash memory, usually 64KB or larger. Memory is logically divided into 512 byte pages that are assigned to individual files as needed. Before a block is erased, the pages in use are copied to another block.

“Thin” driver layer performs only the basic operations that flash devices support: erase block, write byte, etc. The driver is stateless. Supports 8, 16, or 32-bit interfaces to the CPU. Wider interfaces and interleaved devices provide higher performance. Sample drivers for several NOR chips are provided and these are easily ported to new devices.

Supports concurrent use of multiple volumes. Flash volumes can be permanently installed at startup, or added and deleted as removable devices come and go. Volume size is unlimited. A single volume can be implemented using multiple NOR devices.

Flash memory can be shared between a boot program and a TargetFFS-NOR volume. Because flash memory is not accessible while being erased or programmed, the boot program can read files, change directories, etc., but cannot create files, directories, or links. Alternatively, the boot program can copy itself to RAM and jump there before making calls to TargetFFS-NOR.

Supports the UNIX “self”, “group”, and “other” file access protections, allowing applications to restrict some operations to privileged tasks. The file system calls `FsGetId()`, implemented by the application, to get the running task’s user and group IDs.

Optimized for fast mounts. Ongoing garbage collection ensures minimal use of RAM. Can coexist with TargetFFS-NAND™, Blunk Microsystems’ NAND flash file system.

Sample applications include a binary search test, a boot loader, and a command line shell. The boot loader recovers from power loss during an application update. It loads the most recent copy that has been successfully written to flash. The shell supports “cd”, “ls”, “mkdir”, “pwd”, etc. and may be extended with user commands.

Developed using TargetOS™, Blunk Microsystems’ real-time operating system, TargetFFS-NOR has been ported to several other real-time kernels including Nucleus, VxWorks, and Precise MQX. Can be used without a kernel. Works on big- and little-endian systems.

The TargetFFS-NOR source code is 100% ANSI C and is integrated with CodeWarrior. It has been used with ANSI C compilers from ARM, Diab Data, GNU, Software Development Systems, and WindRiver.

Royalty free. Includes full source code, user’s manual, sample applications, and one year of technical support. ►

CONTACT INFORMATION

- Visit our web site:
www.blunkmicro.com
- Customer Support:
(408) 323-9833
- Technical Support:
(408) 323-1758
- Fax:
(408) 323-1757
- E-mail:
sales@blunkmicro.com
- Address:
**6576 Leyland Park Drive
San Jose, CA 95120
USA**

OTHER COMPONENTS

TargetOS

Scalable, Deterministic, Priority-Based Preemptive Real-Time Kernel

TargetTCP

High Performance TCP/IP Stack with Standard Sockets API

TargetLAPB

LAPB Stack for Fast Data Transfer on Point-to-Point Networks

TargetWeb

HTTP 1.0 and 1.1 Compliant Embedded Web Server

TargetZFS

Compressed Data Read-Only Embedded File System

TargetFFS-NAND

Reliable Power-Fail-Safe Embedded NAND Flash File System

The TargetFFS-NOR Application Program Interface:

```
int access(const char *path, int amode);
int chdir(const char *path);
int chmod(const char *path, mode_t mode);
int chown(const char *path, uid_t owner, gid_t group);
void clearerr(FILE* stream);
int close(int fid);
int closedir(DIR *dirp);
int dup(int fid);
int dup2(int fid, int fid2);
int fclose(FILE *stream);
int fcntl(int fid, int cmd, ...);
FILE *fdopen(int fid, const char *mode);
int feof(FILE *stream);
int ferror(FILE *stream);
int fflush(FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
char *fgets(char *s, int n, FILE *stream);
int fileno(FILE *stream);
FILE *fopen(const char *filename, const char *mode);
int format(char *path);
int fprintf(FILE *stream, const char *format, ...);
int fputc(int c, FILE *stream);
int fputs(const char *string, FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
FILE *freopen(const char *filename, const char *mode, FILE *stream);
int fscanf(FILE *stream, const char *format, ...);
int fseek(FILE *stream, long offset, int mode);
int fsetpos(FILE *stream, const fpos_t *pos);
int fstat(int fid, struct stat *buf);
long ftell(FILE *stream);
int ftruncate(int fid, off_t length);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
int getc(FILE *stream);
int getchar(void);
char *getcwd(char *buf, size_t size);
char *gets(char *s);
int isatty(int fid);
int link(const char *existing, const char *new);
off_t lseek(int fid, off_t offset, int whence);
int mkdir(const char *path, mode_t mode);
int mount(char *path);
int open(const char *path, int oflag, ...);
DIR *opendir(const char *dirname);
void perror(const char *s);
int printf(const char *format, ...);
int putchar(int c);
int puts(const char *s);
int read(int fid, void *buf, unsigned int nbyte);
struct dirent *readdir(DIR *dirp);
int remove(const char *filename);
int rename(const char *old, const char *new);
void rewind(FILE *stream);
void rewinddir(DIR *dirp);
int rmdir(const char *path);
int scanf(const char *format, ...);
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
int stat(const char *path, struct stat *buf);
int sync(int fid);
FILE *tmpfile(void);
char *tmpnam(char *s);
int truncate(const char *path, off_t length);
int ungetc(int c, FILE *stream);
int unlink(const char *path);
int unmount(char *path);
int utime(const char *path, const struct utimbuf *times);
int vfprintf(FILE *stream, const char *format, va_list arg);
int vprintf(const char *format, va_list arg);
int vstat(const char *path, union vstat *buf);
int write(int fid, const void *buf, unsigned int nbyte); ■
```

Licensing Terms

TargetFFS-NOR™ is royalty free. Purchasers are granted a non-exclusive license to use the provided source code at a single site. Licensees have the right to disseminate or resell the software in executable format only. The source code and derived object code may not be redistributed or resold.