liteFS-NAND™

FLASH FILE SYSTEM
FOR SPANSION S34ML/S34MS NAND FLASH MEMORY

HIGHLIGHTS

- **Reliable, Re-entrant Embedded File System**
- **POSIX and Standard C API**
- **Guaranteed Recovery from Unexpected Powerloss**
- **Use of NAND Flash Memory is Invisible to Applications**
- **Dynamic Creation and Deletion of Files, Directories, and Links**
- **Performs Complete Wear-Leveling and Bad Block Management**
- **Supports Multiple Volumes of Unlimited Size**
- **Licensed as Royalty-Free Source Code**
- **Portable to Virtually any RTOS and Toolchain**
- **Includes Sample Applications and Test Program**

Blunk Microsystems has been providing leading edge embedded software since 1995, focusing on high-performance systems using 32-bit processors.

BLUNK Microsystems

liteFS-NAND is a high performance UNIX-like embedded file system with a full POSIX and ANSI C compliant application program interface. Supports dynamic creation and deletion of files, directories, and links with read and write capability. Not a static ROM-image file system.

Guaranteed file system integrity across unexpected shutdowns. Only data written since the last synchronizing operation (fsync(), fflush(), etc.) can be lost. Closed files, directory structures, and files open for reading are never at risk.

Complete wear leveling, both erase and read wear. Erase wear-leveling prevents early device failure by spreading erase cycles evenly across all erasable blocks. Read-wear leveling avoids bit errors due to repetitively reading the same flash location. When a read-count threshold is reached, the affected data is copied to a new location, refreshing its voltage margins.

Bit errors are corrected when pages are read from flash. When the number of corrected bits exceeds a threshold, the affected data is automatically copied to another block. This is another protection against data loss due to read-wear causing bit errors that exceed the ECC strength.

Supports background garbage collection, to reclaim dirty flash sectors during idle periods or by a low priority task. Otherwise, block erasures are performed just-in-time.

Manages bad blocks, both initial bad blocks and those that fail during operation. Recovery is performed without user intervention. Once detected, bad blocks are neither programmed nor erased.

Supports the POSIX “self”, “group”, and “other” file access protections, allowing some accesses to be restricted to privileged tasks.

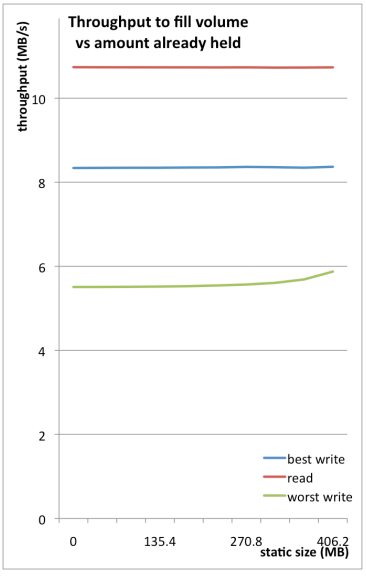
Caches both user data and file system metadata for a configurable fixed RAM footprint that performs well with all volume sizes, up to and including Terabytes.

Optional task-based buffering provides constant streaming rates in spite of intermittent halts in device throughput due to program or erase delays, by either reading-ahead or writing-behind. Enabled by an open() flag, this buffering is otherwise transparent to the application.

Optimized drivers take advantage of S34ML/S34MS special features, such as cache operations, special read for copy back, and multi-plane mode. SW ECC included. Available Verilog for HW ECC.

Other features: Per-task Current Working Directory support. Use with RTOS or in polled mode. >= 242 character file names. Full 100% ANSI C source code. One-year of technical support.

On ARM9 Test System:



Blunk Microsystems liteFS-NAND™

www.blunkmicro.com

AVAILABLE COMPONENTS

TargetXFS

High performance UNIX-like flash-friendly file system. Configurable fixed RAM footprint. Performs well with all volume sizes, up to and including Terabytes.

TargetFTL-NDM

Flash Translation Layer for NAND memory. Performs wear-leveling and virtual-to-physical page mapping using advanced 2nd generation cached algorithm. Configurable fixed RAM footprint.

TargetNDM

Smart bad block manager, supports multiple partitions per device and bulk pre-programming using the "Skip Bad Block" method.

TargetFTL-NOR

Flash Translation Layer for NOR memory. Performs wear-leveling. Adaptively selects virtual-to-physical page mapping algorithm based on volume size. Configurable fixed RAM footprint.

TargetFAT

High performance, widely used, robust DOS-compatible file system with support for NAND, NOR, and SPI flash and removable cards.

Blunk's products include the industry's most extensive line of embedded file systems, an IPv6 Ready TCP/IP stack, embedded web server, RTOS, board support packages, and an IDE with advanced kernel awareness.

Blunk Microsystems, LLC
6576 Leyland Park Drive
San Jose, CA 95120-4558
Tel: (408) 323-1758
sales@blunkmicro.com

Streaming Read Speed	10.7 MB/s avg
Best-Case Streaming Write Speed[1]	8.4 MB/s avg
Worst-Case Streaming Write Speed[2]	5.6 MB/s avg
Mount/Initialization Time	0.86 sec avg
Expected Utilization[3]	86.8%

1. Newly formatted or cleaned volume
2. Dirty sectors, no garbage collection
3. Volume with >= 128 erasable blocks

Application Program Interface includes:

```
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);
int fsync(int fid);
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);
void sync(void);
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

liteFS-NAND may only be distributed as fully linked executable code. $5K license limits distribution to 10,000 units and use to one site or three programmers. $5K upgrade removes unit, site, and seat limitations.