
# Flash File Systems

## Meeting Today's and Tomorrow's Demands

**Tim Stoutamore, Principal Engineer**
**Blunk Microsystems**


# TargetFFS

- A successful flash file system for bare flash chips
- Used in BMW dashboard navigation systems, JPL satellites
- Shipped in over 60 million cell phones by over 12 different manufacturers, including Samsung





# First there was NOR

- Erase (up to 3 sec) sets every bit in a large (ex. 64KB) block
- Program allows you to clear individual bits
- "Wear Fatigue": must program/erase all blocks evenly
- Directly addressed: supports Execute-in-Place

# Next came NAND

- Fast erase sets every bit in a medium-sized (ex. 8KB) block
- Program allows you to clear individual bits, within partial programming limit (typ. 3-4 per page)
- "Wear Fatigue": must program/erase all blocks evenly
- Page Accessed: no Execute-in-Place
- Bad Blocks: both as shipped and failures during operation
- Bit Errors: requires error detection and correction algorithms
- Extra Bytes: 16 out-of-band bytes for every 512 data bytes

# Multi-Level Cell (MLC) NOR

- More dense: lower price per bit
- A single cell holds one of four voltage levels. Comparators determine whether that cell represents 11, 10, 01, 00. Each cell represents two bits.
- Can't rely on clearing a single bit without affecting adjacent bits. If you are writing the pattern 01 to a cell and power is lost in the transition zone representing 10, you will have cleared the wrong bit.

# Multi-Bit Cell (MBC) NOR

- More dense: lower price per bit
- A single cell holds two bits.
- Some MBC devices limit how many times a word can be partial programmed: can't use bit map algorithms that may repeatedly update the same word.

# Multi-Level Cell (MLC) NAND

- More dense: lower price per bit
- A single cell holds one of four voltage levels. Comparators determine whether that cell represents 11, 10, 01, 00. Each cell represents two bits.
- No partial programming allowed
- Pages must be written in numerical order
- More bit errors: requires ECC for 4-bit errors per 512 bytes.

# 90nm NOR

- Divided into 1024 byte pages
- No limit on partial programming, but if you do any partial programming you can only use half the page (512 bytes)


# File System Requirements

Need a program that will:

- Behave like a traditional file system
- Use flash memory as backing store
- Not violate the requirements/restrictions of flash memory
- Ex. Can't store control information in a fixed location






# The Power-Fail Guarantee

Directory structures, closed files, and files open for reading are never at risk. Only data written since the last synchronizing operation can be lost.





BLUNK

# How It's Done

After writing the control information to flash, don't modify any portion of flash memory that copy of control information 'thinks' contains valid data until after a new copy of the control information has been written.

# Recycle Sequence

Copy data on block(s) to be erased to another block

Write a new copy of control information

Erase the selected block(s)

# Atomic Updates

- NOR SBC, M18: clear flag in bit array
- NAND, NOR MBC: write control info with CRC


# Result

TargetFFS is widely used and relied upon to be power-fail safe. We perform our own extensive automated testing and customers routinely subject TargetFFS to automated power-fail testing.

In its first deployment, TargetFFS-NAND beat out flash file system products from Microsoft and WindRiver, being the only system that passed the customer's automated power-fail testing.






# UNIX-Like Features

Because TargetFFS isn't limited to conforming to FAT data structures, it is not limited to the FAT restrictions and has a more advanced UNIX-like API:

- File links
- Long file names
- UTF-8 file name support
- Quotas and reservations
- No Microsoft FAT licensing concerns

# Access Protections

Supports the “self”, “group”, and “other” file access protections, allowing applications to restrict some operations to privileged tasks.

TargetFFS calls FsGetId(), implemented by the application, to get the running task’s user and group IDs. These can be saved in a task-specific RTOS register.

# Per-Task CWD

The current working directory (CWD) is specified by two 32-bit variables. One function is called to save new CWD state variables when 'chdir()' is executed. Another function is called to read the CWD state variables when resolving relative paths. If the operating system accesses these variables in a task specific way (i.e. task registers), then each task has its own CWD.


# Seek Acceleration

```
int lseekmark(int fildes, int disable_update);
int fseekmark(FILE *stream, int
disable_update);
```

Bookmarks a file seek offset. Closest starting point is used when searching for a new file offset.





# Background Garbage Collection

Recycle operations, which convert dirty sectors to free sectors, may be performed in the background by calling vclean() from the idle task.


# Reserved Memory

Allows a number of flash sectors to be reserved, producing an early volume full indication.

The file system will immediately exchange reserved free sectors for dirty application sectors.

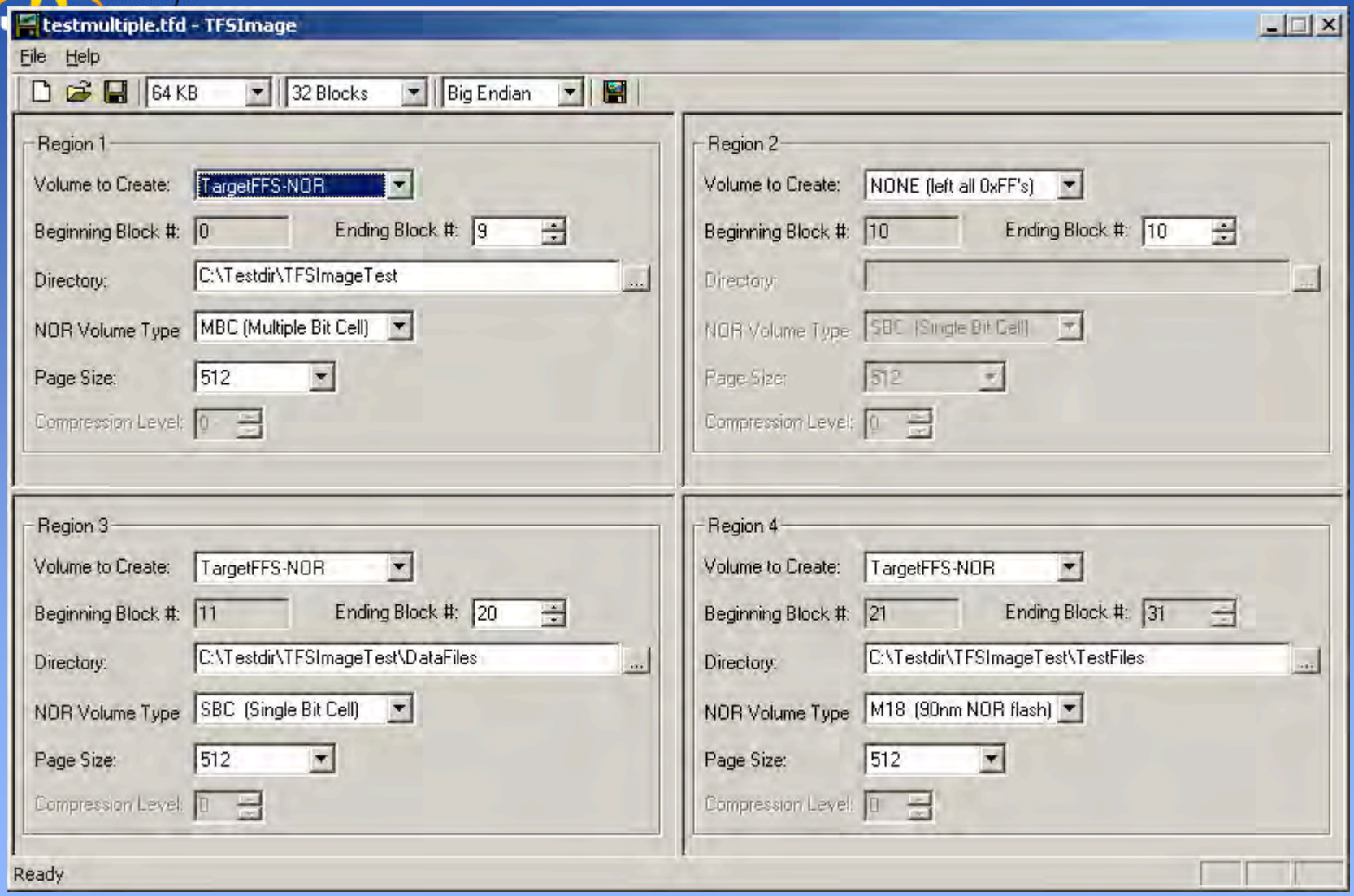
When combined with background recycling, ensures there is always a pool of free sectors available, boosting file system responsiveness for user-interface applications, even when the volume is full or nearly full.





BLUNK

# Image Tool(tm)




# Image Tool(tm) Features

Generates binary images for production programming.

Creates images for NOR (SBC, MBC/MLC, 90nm) and NAND[1] devices.

Supports TargetFFS, TargetFAT/TargetFTL, and TargetZFS volumes.

Two forms: GUI for development, CLI for production.

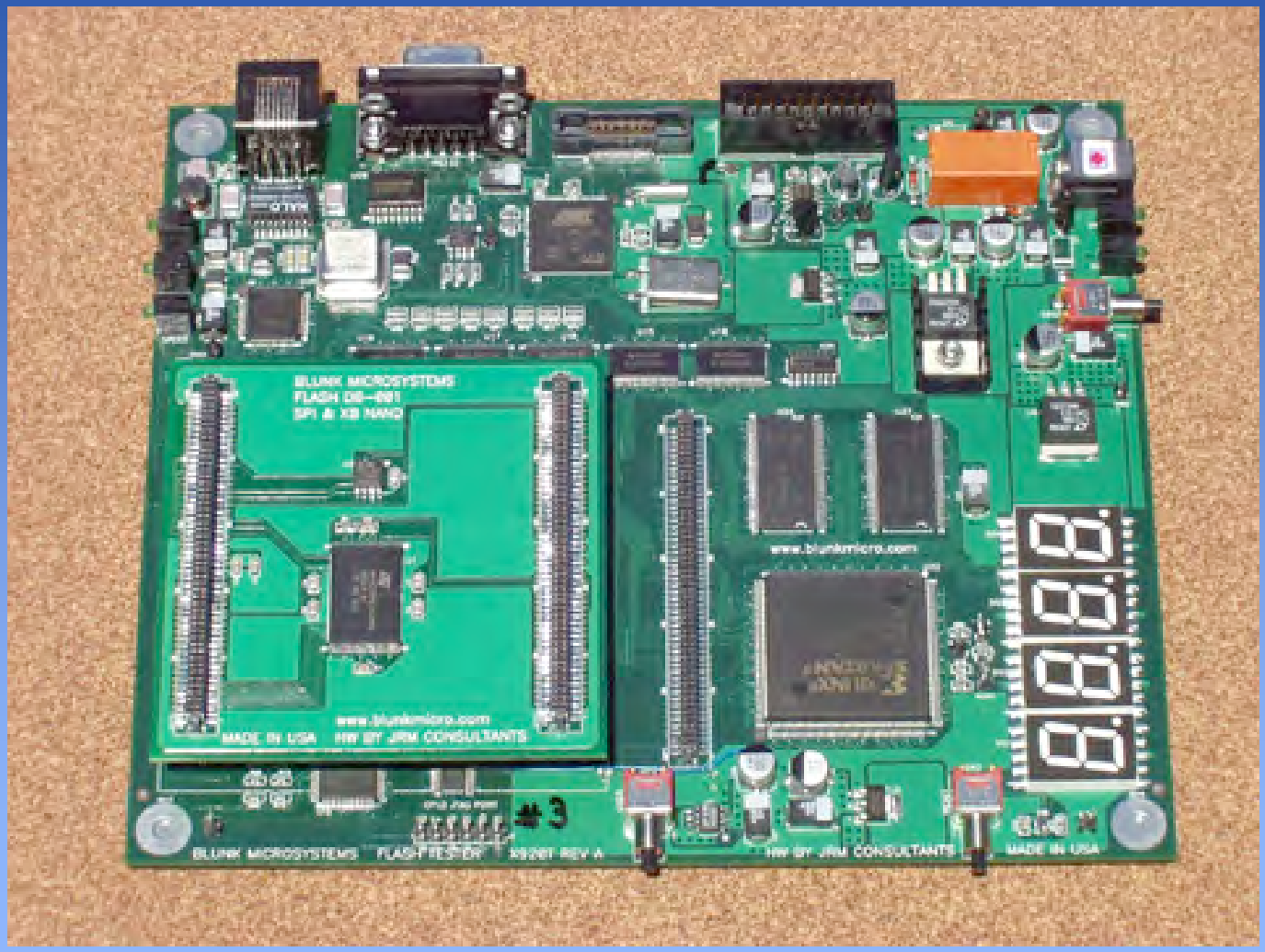
[1] Using the "Skip bad-block" programming method.





BLUNK

# Flash Test Board



# Flash Test Board Features

'Universal' flash interface via daughter card interface. Enables rapid support of new flash chips and architectures.

FPGA for optional hardware ECC assist for NAND devices.

Microprocessor controlled power interruption circuit (relay controlled by a G.P. output) for power-fail recovery demonstration and testing.

Complete software development environment, including compiler, editor, linker, assembler, and debugger, with support for fast downloads and run control via Ethernet.

Includes embedded Web, Telnet, and FTP servers for remotely initiating and monitoring tests, querying status, etc.

# Full Spectrum of Embedded File Systems

- TargetFFS-NOR: File System for NOR Flash
- TargetFFS-NAND: File System for NAND Flash
- TargetFAT: DOS/Windows Compatible File System
- TargetFTL-NOR: Flash Translation Layer for NOR Flash
- TargetFTL-NAND: Flash Translation Layer for NAND Flash
- TargetRFS: RAM File System (malloc()/free())
- TargetZFS: Compressed Read-only File System


# Current Developments

Common Internet File System (CIFS), also known as Samba, for PC file sharing without FAT.

Supporting new flash device architectures and new constraints (ex. recently supported new device and got design wins before part officially announced).

VHDL hardware ECC support, using flash test board as demo platform.

Continuing to respond to customer requests for features, drivers, tools, etc.