# Flashmon 2.0 Module User Guide

Pierre Olivier, Jalil Boukhobza
UMR 3192 Lab-STICC
Université Européenne de Bretagne, Université de Bretagne Occidentale
20 Avenue Le Gorgeu - CS 93837, 29238 Brest cedex 3
pierre.olivier@univ-brest.fr, jalil.boukhobza@univ-brest.fr

May 6, 2013

Flashmon is a Linux kernel module that monitors NAND flash memory access events using Kprobes and Jprobes. Flashmon monitors accesses for **bare flash chips** soldered on embedded boards, managed by the Linux Memory Technology Device (MTD) subsystem. *Flashmon will not work for devices that are not managed by MTD: SD/MMC cards, USB pen drives, SSD drives, etc.*

The monitored events are: flash pages read / write requests, and flash block erase operations. Those operations are traced at the MTD subsystem level to stay independent from the file system and the device driver. Traced functions are :

- nand_read ;
- nand_write ;
- nand_erase ;

This document provides information regarding the compilation, installation and usage of Flashmon.

## 1 Compiling Flashmon

The core functions of flashmon are located in the file `flashmon_core.c`. The file `flashmon_log.c` is dedicated to the flash events logging system. The file `flashmon_finder.c` contains fucntions which aim to dynamically detect the kernel functions performing flash accesses and put probes on them. Compilation can be achieved with the help of the Makefiles present in this archive.

Flashmon can be (cross-)compiled as a standalone module, or integrated in a kernel source tree for compilation as a module or built-in function (through the linux *menuconfig* menu).

## 1.1 Compiling Flashmon as a stand alone module

### 1.1.1 Cross-compilation

As flash is extensively used as secondary storage in embedded system, Flashmon is likely to be cross-compiled for a given target system. The following variables of the `Makefile` provided can be edited:

- `KERN_DIR` must point to the root directory of the kernel sources ;
- `ARCH` is the target architecture ;
- `CROSS_COMPILE` is the prefix for the cross compilation toolchain.

Once these variable are correctly specified a `make` command in Flashmon directory should compile the module, which will be `flashmon.ko`.

### 1.1.2 x86/amd64 compilation

Our desktop computers rarely embed bare flash chips. Nevertheless for developement / debug purposes Flashmon[1] will also works with the Nandsim flash simulator which emulates in RAM / on disk a mtd device. The `Makefile.x86` file can be used to compile Flashmon for a desktop computer: `make -f Makefile.x86`.

## 1.2 Integrating flashmon in the kernel source tree

Flashmon is not integrated in the kernel mainline. Nevertheless, the script `patch-ker.sh` can be used to integrate flashmon sources in the kernel source tree. Flashmon can then be activated as a module or built-in function through the linux *menuconfig*, during kernel configuration.

To use this script :

```
usage:  ./patch-ker.sh  c/l kernelpath
 if c/l is c, then copy. If l then link
```

The script takes a letter as first parameter, `c` is to copy flashmon sources directly in the kernel sources, `l` is to link the flashmon sources in the current folder to the kernel sources. The second parameter for the script is the path of the root of the kernel source tree.

Next, when launching the linux *menuconfig*, one can observe the flashmon entry in **Device drivers → Memory Technology Device (MTD) support → NAND Device Support**. Flashmon entry is named **Support for Flashmon NAND Flash Operations Tracer**. It can be switch to module (<M>) or built-in function (<*>). The default size of Flashmon log can also be configured here (see the following sections for more information about the log).

[1] Sources: `http://lxr.free-electrons.com/source/drivers/mtd/nand/nandsim.c`,
Tutorial: `http://mytechrants.wordpress.com/2010/01/20/ubiubifs-on-nandsim/`

Enabling Flashmon as a built-in function is interesting to trace the flash operations performed during the boot process.

## 2 Starting Flashmon

Please note that the system in which Flashmon is loaded must be operated by a kernel compiled with the kprobes and kallsyms options activated. Flashmon is inserted from the command line as follows:

`insmod flashmon.ko TRACED_PART=X LOG_MODE=Y PROG_PID=Z.`

When inserting the module various options can be specified:

- `TRACED_PART` allows to specify the index of the MTD partition to trace. For example, launched with `TRACED_PART=5` Flashmon will trace only the accesses to /dev/mtd5. If `TRACED_PART` is -1 or is not specified, the whole flash chip is monitored ;
- `LOG_MODE` activates the temporal event log of Flashmon (described in the next sections). Logged events are maintained in RAM, and this option allows specifying the maximum number of events kept in the log. As for today the size of one log entry is 28 bytes. When `LOG_MODE=0` the log is disabled. If not specified the size of the log is arbitrarily define to 1024 ;
- `PROG_PID` allows specifying a user space program which will be notified (with a signal) each time a flash event occurs. One must specifies here the PID for this user space process ;
- `LOG_TASK` allows to specify if flashmon should log the current task name each time a flash event is traced. By default this feature is enabled. The parameter can be set to 0 to disable it. It should considerably reduce Flashmon memory footprint.

If flashmon is integrated in the kernel source tree (see above), you can use `modprobe` to insert flashmon. The `modinfo flashmon` command can also give some information about the module and its parameters:

```
pierre@as3:~/Bureau/flashmon-code/trunk$ modinfo flashmon.ko
filename:       flashmon.ko
description:    Trace informations about flash page reads / writes, and flash block erase opera
author:         Pierre Olivier <pierre.olivier@univ-brest.fr>
license:        GPL
srcversion:     88E08D101170558026DFCF2
depends:        mtd
vermagic:       2.6.32-46-generic-pae SMP mod_unload modversions 586TSC
parm:           PROG_PID:Userspace PID to notify (int)
parm:           LOG_MODE:Log mode 1=on 0=off (int)
parm:           TRACED_PART:Traced partition index, -1=all (int)
parm:           LOG_TASK:Insert for each event in the log the name of the current task at the t
```

# 3 Flashmon outputs & controls

Flashmon provides two outputs: 1) An "architectural-based" output that reflects the state of each flash memory block of the monitored device, the `/proc/flashmon` file, and 2) an "event-based" temporal log, `/proc/flashmon_log`.

## 3.1 Architectural output: /proc/flashmon

`/proc/flashmon` text file contains a number of lines equals to the number of flash blocks in the monitored device. Each line has the following pattern :

```
<number of reads> <number of writes> <number of erase operations>
```

These are the counts of read / write / erase operations performed on the corresponding block *since the module is loaded*. The line 0 corresponds to the flash block 0, the line 1 to the flash block 1, etc. Please note that the page read and write requests are merged/summed at a block level. Each time reading this file is requested, it is generated based on flashmon internal data.

## 3.2 Temporal log output: /proc/flashmon_log

When the `LOG_MODE` parameter is specified and superior to 0, flash events monitored by the module are logged in the `/proc/flashmon_log` file. If `LOG_MODE=0` this file does not exist. The maximum size of the log is esqual to the value of `LOG_MODE`: it can be tuned to control Flashmon RAM usage. The log file has a csv format, each line of has the following pattern:

```
<timestamp for the event>;<event type>;<event address>;<task name>
```

One event is a block erase, a page read or a page write. The unit of the time is the second. It is obtained with the `getnstimeofday` kernel function, giving a nanosecond precision. It corresponds to the time elapsed since Flashmon was inserted. The event type is a character: 'E' for a block erase, 'R' for a page read and 'W' for a page write. The event address is the number of the targetted page for R/W operations, and the number of the targetted bloc for an erase operation. Page 0 and block 0 are the first page and the first block of the flash chip, respectively. The address is the number of the page / block inside the whole flash chip, independently of the fact that flashmon monitors the whole chip or just a partition. The `<task name>` field is a string containing the name of the task currently executed when the event is traced. As Flashmon memory consumption can grow fast when logging the task name, one can explicitly specify not to log the task name when flashmon is launched with the `LOG_TASK` parameter.

The data structures behind the log file are implemented as a circular log buffer, so when the maximum count of events is reached, Flashmon begins to overwrite the oldest events with the new ones.

### 3.3 Flashmon control

Monitoring flash events can be controlled by writing commands strings in the `/proc/flashmon` and `/proc/flashmon_log` files. 3 commands are avaible: start, stop and reset.

- `reset`: When written to `/proc/flashmon`, reset the event counters to 0 and clear the log. Clear only the log when written to `/proc/flashmon_log` ;
- `stop`: When written to `/proc/flashmon`, stop the monitoring: The next flash accesses will not be logged, and accesses counters will not be incremented until the monitoring is re-enabled. When written to `/proc/flashmon_log`, stop the monitoring only for the log ;
- `start`: Restart the monitoring for both architectural view and log when written to When written to `/proc/flashmon`. Restart only the log when written to When written to `/proc/flashmon_log`.

For example, the following command will reset the flash event counters to 0 and also reset the log :

```
# echo reset > /proc/flashmon
```

To clear only the log:

```
# echo reset > /proc/flashmon_log
```

By default, Flashmon is loaded with monitoring enabled.

## 4 Notification for user-space process

If a process PID is specified when inserting the module, every time a flash event is monitored the module will send a signal to the PID indicated. It means that you first have to start the process to notify, get his PID, and then launch Flashmon with this information as a parameter of the module. The type of the signal is 14 (SIGALRM). To write a C application which will recognise and respond to the signal, one must write a signal handler, a function performing whatever you want every time the signal is received. Then, in the main function, link the reception of the signal SIGALRM to the handler :

```
void my_signal_handler( int sig_num )
{
/* Things done each time the signal is received */
}

int main( int argc, char ** argv )
{
...
/* Link the signal reception to the handler: */
signal( SIGALRM , my_signal_handler );
...
}
```

# 5 Tools

In the `Tools` directory are a set of tools that can ease the use of Flashmon. Please read the `README` file in this directory for more information about these tools.

# 6 References (French)

1. J.Boukhobza, I. Khetib, P.Olivier, Flashmon: un outil de trace pour les accès la mémoire flash NAND, Proceedings of the 1st EWiLi (Embed With Linux) Workshop, Saint Malo, France, 2011.