

Flasher

series

SIGNUM SYSTEMS CORPORATION

Command-Line Flasher for ARM, XScale and Cortex

User Manual

SIGNUM
S Y S T E M S

COPYRIGHT NOTICE

Copyright (c) 2008 by Signum Systems Corporation. All rights are reserved worldwide. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Signum Systems.

DISCLAIMER

Signum Systems makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Also, Signum Systems reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Signum Systems to notify any person or organization of such revision or changes.

WARRANTY

Signum Systems warrants to the original purchaser that this product is free of defects in material and workmanship and performs to applicable published Signum Systems specifications for a period of SIX MONTHS from the date of shipment. If defective, the product must be returned to Signum Systems, prepaid, within the warranty period, and it will be repaired or replaced (at our option) at no charge. Equipment or parts which have been subject to misuse, abuse, alteration, neglect, accident, unauthorized installation or repair are not covered by warranty. This warranty is in lieu of any other warranty expressed or implied. **IN NO EVENT SHALL SIGNUM SYSTEMS BE LIABLE FOR CONSEQUENTIAL DAMAGES OF ANY KIND.** It is up to the purchaser to determine the reliability and suitability of this product for his particular application.

SIGNUM
S Y S T E M S
1211 FLYNN RD., UNIT #104
CAMARILLO, CA 93012, U.S.A
PHONE 805 • 383 • 3682
WWW.SIGNUM.COM

Table of Contents

Introduction	1
License	1
Installation	2
Installation Directory Content	2
First Time Run	3
Usage	3
Emulation Parameter File	6
Macro/Initialization Files	6
Programming without Target RAM	7
Error Codes	8
Board and CPU Customization	10
Programming Multiple Devices	10
Supported Flash Devices	11
CFI Programmer	12
STR9 Programmer	12
LPC Programmer	13
NAND Programmer	19
RAM Programmer	20
Custom Flash Devices	21

Flasher is a command-line oriented flash programming tool for the Signum JTAGjet emulator intended to aid the automation of your production process.

Introduction

This document describes the Flasher package designed to program flash devices from the command-line window (DOS prompt) or from within your application. Since no popup messages are displayed during the programming process, Flasher lends itself to use as a component of complex, fully automated production tool packages.

License

Flasher is designed for the use with the Signum JTAGjet emulator. This flasher-emulator tandem combination requires a license. If your emulator does not have it, an external license file must be obtained from Signum Systems. Once received, the Jxxxxx-Flasher.lic file needs to be placed in the directory where SigFlashCmd.exe resides. (The xxxxx part of the actual file name is the serial number of your JTAGjet emulator). Alternatively, the license file can be placed in the C:\Signum\Licenses folder where it becomes available to other applications, such as Chameleon or Flasher.

Installation

To install the Flasher package, execute the setup_flasher.exe program. The default installation folder is C:\Signum\Flasher.

Note: Spaces in file and folder names may interfere with DOS command-line programming and thus should be avoided.

If system-level JTAGjet USB driver has not been installed on your computer earlier, the system will prompt you for the appropriate driver when the emulator is connected to the USB port for the first time. Point the installation program to the Drivers\USB subfolder of the installation folder (in our example, C:\Signum\Flasher\Drivers\USB.)

Installation Directory Content

Table 1 details the contents of the Flasher installation directory.

D E L I V E R A B L E	D E S C R I P T I O N
SigFlashCmd.exe	Command line executable program. Also called from within t.bat.
.\bin*.*	Flasher executable components.
.\docs\docs.htm	Main documentation accessible through a shortcut after installation.
.\docs\flasher_um.pdf	This document.
.\docs*.*	Additional documentation (accessible via docs.htm file).
.\boards*.par, *.ini	CPU and target specific parameter and board initialization files.
p*.bat	Programming scripts for selected evaluation boards.
.\Drivers\USB*.*	JTAGjet USB driver.
.\tests*.*	Test scripts and log files from test runs.
.\testdata*.*	Test data used by test scripts.
t.bat	Test execution script. Example: t tinnov1510.

TABLE 1 The contents of installation directory.

First Time Run

Make sure that your emulator is connected to a USB port. A dialog box will prompt you to select the JTAGjet emulator to be used to program the flash. The information, saved in the SigFlashCmd.ini file, will be used in subsequent sessions. Connecting to another emulator is possible by using the -connect switch or by deleting the SigFlashCmd.ini file. The -connect option allows you to specify the emulator serial number. The -connect option overrides the counterpart setting in the SigFlashCmd.ini file.

Usage

SigFlashCmd.exe displays usage information if executed with the -help option (or without any options):

```
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
Usage:
  SigFlashCmd [options]

Available options:

-help                - Display help and all options if used with -flash.
-v                  - Display Flasher's version - can be used with -flash.
-connect            - Specify connection via a dialog box.
-connect <sn>       - Connect to the JTAGjet with serial number <sn>.

-cpu <cputype>      - The CPU core type: ARM7TDMI, ARM926EJ-S, C166 etc.
                    The <cputype>.par file will be used if it exists.
-nocpu              - Ignore any CPU connection errors and continue.
-emu <file> ...     - The emulation parameter file. CPU dependent.
-emu <n>=<v> ...    - Set emulation parameter <n> to <v>. CPU dependent.

-init <file> ...    - The initialization script file. Board dependent.
-term <file> ...    - The termination script file. Board dependent.

-flash <type>@<addr>[,<opt> ...] - The flash type (CFI, STR9, NAND etc.),
```

base address, and comma-separated flash options.

`-ram <size>@<addr>` - The RAM range accessible to the programmer.

`-dump` - Dump the flash geometry details.

`-erase chip` - Erase the chip.

`-erase <range>` - Erase all the sectors in the range.

`-erase` - Erase all programmed ranges before programming.

`-blankcheck chip` - Blank check the entire chip.

`-blankcheck <range>` - Blank check the range.

`-blankcheck` - Blank check only the erased or programmed ranges.

`-format <format>` - The format of the flash image file. The following file formats are accepted:

- `bin` - Plain binary file (default).
- `hex` - Intel HEX format.
- `srec` - Motorola S-record format.
- `Msbin` - Windows CE binary image data format.

`-offset [<addr>]` - The address offset for data from a file. If `<addr>` is missing, use the `<addr>` from the `-flash` option.

`-program <file>` - Program the flash with an image file.

`-range` - Ignore all data outside flash.

`-range <range>` - Ignore all data outside a given range.

`-dumpblk` - Display all code ranges (after applying `-range`).

`-verify` - Verify the flash image after programming.

`-verify <file>` - Verify the flash using a file without programming.

`-extra <cmd>` - Execute an extra command. Flash type dependent. More than one `-extra` options can be given.

`-extra` - List all extra commands for the flash in use.

`-read <range>` - Read and display the data from the given range.

`-save <fname>` - Save the read data to a binary file w/o display.

`-q` - Quiet mode. Display only the results and errors.

`-time` - Display the execution time for each step.

```
<range>: <size>@<addr> - <size> in bytes; <addr> is physical address.
    Sn|Bn          - A single sector or block. (S1, B0 are first.)
    Sn..Sm|Bn..Bm - Range of sectors or blocks. (Sn <= Sm, Bn <= Bm.)
    <size>@Sn|Bn - <size> bytes starting at sector Sn or block Bn.
```

If the `-help` and `-flash` options are used at the same time (as in “`-help -flash CFI`”), the help information includes a list of the flash programmer options for the specified programmer.

All addresses and sizes can be specified as one of the following:

- Decimal number (e.g., 1024)
- Hexadecimal number (e.g., 0x400)
- Hexadecimal or decimal number with K or M appended (e.g., 512K or 4M)

For testing purposes, the `-program` and `-verify` options accept their parameters in the `<size>@<addr>` format instead of a file name (`<fname>`). This results in using the testing pattern 0,1,2,3,...,0xFF, 1,2,3,0xFF,0, 2,3, ... of the specified size, which is useful for validating the configuration and the efficiency of operations of the flash device.

Examples

Note: The backslash character (\) in the examples below symbolizes line continuation—the entire command should be typed on one line. Please refer to the `t.bat` file for more usage examples of SigFlashCmd usage.

QUERY THE FLASH CAPABILITIES

```
SigFlashCmd -emu OMAP1710.par -flash cfi@0 -dump
```

PROGRAM A BIN FILE INTO THE CFI FLASH AT ADDRESS 0

```
SigFlashCmd -emu omap1710.par -ram 1K@0x20000000 \
    -flash cfi@0 -erase -program test.bin -verify
```

VERIFY A BIN FILE INTO THE CFI FLASH AT ADDRESS 0

```
SigFlashCmd -emu omap1710.par -ram 1K@0x20000000 \
    -flash cfi@0 -verify test.bin
```

PROGRAMMING A TEST PATTERN INTO THE FLASH WITHOUT VERIFICATION

```
SigFlashCmd -emu omap1710.par -ram 1K@0x20000000 \  
-flash cfi@0 -erase -program 1M@0x10000
```

Emulation Parameter File

The core and CPU dependent emulation parameters are stored in the .par file used in conjunction with the -emu option. These parameters are a subset of the parameter list displayed by Chameleon’s emu command. Refer to the .par files provided with the package for guidelines in determining which of these parameters are essential to the operation of the programmer. If found in the programmer package, the .par file designated for your target can be used unmodified.

You can use multiple -emu file options and set individual emulation parameters using the “-emu name=value” syntax.

Macro/Initialization Files

The .mac or .ini files, used in conjunction with the -init and -term options allow you to utilize the following subset of Chameleon Debugger’s macro commands.

C O M M A N D	D E S C R I P T I O N
S8/SB/S16/SW/S32/SD <addr> = <value>	Set Byte, 16 bit Word, 32 bit Double word.
S8/SB/S16/SW/S32/SD <addr> = <value>	OR value with memory at <addr>
S8/SB/S16/SW/S32/SD <addr> &= <value>	AND value with memory at <addr>
S8/SB/S16/SW/S32/SD <addr> ^= <value>	XOR value with memory at <addr>
D8/DB/D16/DW/D32/DD <addr>	Reads Byte, 16 bit Word, 32 bit Double word. Displays them when ECHO is on.
ECHO ON OFF	ECHO control. OFF by default.
PAUSE <ms>	Pause the execution of commands for <ms> milliseconds. Use to allow the PLL to stabilize.
EMU <name> = <value>	Emulation parameters (the same as in Chameleon).

COMMAND	DESCRIPTION
RESET [/HALT]	Reset the CPU and, optionally, stop the CPU if it did not stop automatically after a reset.
STOP	Stop the CPU.
MEM SPACE	Display the current memory space (in the echo mode only).
MEM SPACE DEFAULT	Select the default (index 0) memory space.
MEM SPACE <space-index>	Select the memory space defined by an index. Eg., 0x1F - CP15 register access; 0x1E - CP14 access.

TABLE 2 Chameleon macro commands that can be use with the `-init` and `-term` options.

Note: Do not use Chameleon's startup macro as-is. Chances are that for flash programming purposes, most Chameleon macros can be simplified.

Programming without Target RAM

Most flash devices can be programmed without using the RAM buffer on the target. If the `-ram` parameter is not specified, all programming actions are executed by accessing the flash or the flash controller directly through the JTAG. While the speed of the erase operation is the same as when running the code in the target's RAM, all the other memory operations may slow down significantly. The RAM-less method is typically only suitable for small boot-loaders or in case of emergency board recovery. Therefore, specifying RAM on the target and configuring it is strongly recommended. Most programmers require no more than 2K of RAM. The best results are achieved with internal static RAM. Use an initialization macro in conjunction with the `-init` option to enable and configure access to RAM that is not accessible all the time.

It is also possible to setup TCM memory as RAM for programmer. It requires writes to CP15 registers from an INI file (by using MEM SPACE command). See the `.\boards\dm644x_495.ini` file for an example of TCM configuration sequence.

Some programmers for on-chip flash devices, such as the LPC programmer, do not require RAM to be specified and will automatically use the available on-chip RAM for programming purposes.

Error Codes

Table 3 shows Flasher error codes. These codes can be used by the IF ERRORLEVEL N command in DOS batch files. A detailed description of each error is displayed in the DOS window. A SigFlashCmd.exe run terminates with one of the two following messages appearing as the last line in the DOS window:

```
** Exiting with error code NN  
- or -  
OK - Done
```

ERROR CODE	DESCRIPTION	WHAT TO DO
NON-PROGRAMMING ERRORS		
0	All required actions executed successfully.	No action required.
1	Usage problem: bad command line options.	Correct the command line options.
2	Emulator connection problem: the emulator specified with the –connect option or in the SigFlashCmd.ini file not found.	Make sure that the emulator is connected and not used by a different application. Use the –connect option to list the available emulators and to connect to a new emulator. Make sure that your emulator has a license for an external programmer.
3	Emulator license error.	Make sure that the Jxxxxx-Flasher.lic file is in the same folder as SigFlashCmd.exe.

ERROR CODE	DESCRIPTION	WHAT TO DO
4	Initial CPU connection problem: no power, cannot stop the CPU, etc.	Verify the JTAG connection and check power on the target. Make sure that the file specified in the -emu option defines the JTAG geometry and sets emulation parameters correctly. Power cycle the board and restart the process. Try to connect to the CPU from Chameleon Debugger.
5	Initialization macro execution problem. (The -init option.)	Verify the command syntax. Check the macro using Chameleon Debugger. Note, however, that the HALT option cannot be used with the “reset” macro command.
6	Termination macro execution problem. (The -term option.)	Ditto
7-10	Reserved	

PROGRAMMING ERRORS

11	Incorrect flash related parameters, such as a nonexistent flash programmer or unknown file format.	Details will be displayed in the window. Make sure that the command line options are set correctly.
12	Internal error — one of flash related DLLs missing.	Make sure that SigFlashCmd is started in the folder where it is located — check the “Start in” field in the shortcut to the program.
13	Flash initialization problem, such as “cannot detect CFI flash at the specified address.”	Make sure that the CPU is properly initialized.
14	Cannot open a file when programming and verifying.	Make sure that the file exists and the -format option specifies its format correctly.
15	Failure while erasing.	Make sure that the CPU is initialized and that flash write enable is set.

ERROR CODE	DESCRIPTION	WHAT TO DO
16	Failure while blank-checking.	Make sure, that the chip or sectors are erased. Make sure that the CPU is initialized properly.
17	Failure while programming	Same as for erasing. (See code 15).
18	Failure while verifying the code.	Make sure that verification is performed on the file used to program the flash.

TABLE 3 SigFlashCmd error codes.

Board and CPU Customization

Programming flash devices on custom-made processors and boards involves the following steps.

1. Adjust the command line parameters: modify memory addresses and specify the correct flash type.
2. Appropriately modify the emulation parameter file specified in the `-emu` option. If Chameleon Debugger is connected to the CPU, some of the parameters returned by the `emu` command may need to be stored in this file. To verify that SigFlashCmd detects your flash device, invoke the program without the `-program` and `-erase` options.
3. Adjust the startup and termination macros used with the `-init` and `-term` options, respectively. The initialization macro should enable RAM on the target, set up the PLL, kill the watchdogs and enable write access to the flash.
4. When testing your macros, use the “echo on” directive to receive screen feedback. Also, utilize the `DB`, `DW` and `DD` macro commands to validate your settings.

Programming Multiple Devices

The `-connect <SN>` option causes SigFlashCmd to connect to the CPU via the JTAGjet emulator with the specified serial number. This makes it possible to

program several flash devices in parallel. By running two or more instances of SigFlashCmd, each in its own DOS prompt window, you are able to program multiple devices simultaneously. If the specified emulator cannot be found, SigFlashCmd exits.

Also, it is possible to create a master batch file that with the use of the DOS `start` command executes several instances of SigFlashCmd concurrently. This would run each SigFlashCmd in its own window. Unfortunately, the window closes immediately after SigFlashCmd terminates with or without error. It is possible neither to verify that a particular `start` command has finished running, nor to find out what the resulting error code is. However, with the help of advanced DOS batch programming techniques and by starting multiple batch files from within one master batch file, you can create a batch program that starts programming multiple devices simultaneously and checks error codes of each individual run.

Supported Flash Devices

Table 4 lists the internal and external flash devices supported by Flasher.

DEVICE	DESCRIPTION
CFI	External CFI flash with any ARM, XScale or Cortex processor.
STR9	Internal flash for ST Micro STR9x processor (including security bits).
LPC	LPC2xxx (ARM7-based) and LPC17xx (Cortex-M3 based).
NAND	NAND with selected CPUs and NAND controllers.
RAM	Pseudo-flash in RAM. It allows to write to RAM using this tool.

TABLE 4 Supported devices.

The type of a flash device must be specified using the `-flash` option. The `-help` and `-flash` options display all flash-dependent options for specified flash programmer. These options are described in the following sections.

CFI Programmer

CFI Programmer Options

The figure below shows a fragment of the CFI programmer help information displayed by executing the command: `SigFlashCmd -help -flash CFI`.

The CFI programmer options:

```
dcc          - Force DCC based programming (to save time and memory).
buf          - Force RAM buffer based programming (to save memory).
noblk       - Do not use the programming block buffer.
blk         - Use the programming block buffer for non-Spansion/AMD devices.
nounlock    - Do not unlock sectors on Intel devices.
lock        - Lock sectors after programming Intel devices.
```

TODO: Describe CFI options here.

STR9 Programmer

STR9 Programmer Options

The figure below shows a fragment of the STR9 programmer help information displayed by executing the command: `SigFlashCmd -help -flash STR9`:

The STR9 programmer extra operations (use with the `-extra` option):

Reading:

```
?          - Display all settings (except ?mfg).
?idcode    - Display IDCODE as a 32-bit number.
?security  - Display the security bit (0 or 1).
?usercode  - Display USERCODE as a 32-bit number.
?config    - Display CONFIG register as a 64-bit number (with bit names).
?otp       - Display 256 bits of OTP as a hex number.
?mfg       - Display the MFG register as a 32-bit number.
```

Writing:

```
usercode=0x11223344 - Program the USERCODE register with a 32-bit number.
```

```

config=0xFF00F00... - Program the CONFIG register with up to 64 bits.
security=1          - Set the security bit. (Use erase=all to erase it).
erase=config        - Erase the CONFIG register (all 0's).
erase=usercode      - Erase the USERCODE register (all 1's).
erase=all           - Erase all. (The only way to erase the security
bit.)
Writing OTP - *** THESE OPERATIONS CANNOT BE UNDONE - USE WITH CAUTION ***
otp=0xFFFF11FF... - Program the OTP bytes (use FF to skip the byte).
otplock=1          - Program the OTPLOCK bit.
CONFIG register's bit names:
MSP0..MSP7         - Bits 0..7   - The Main Sector Protect bits.
SSP0..SSP3         - Bits 32..35 - The Secondary Sector Protect bits.
CSX                - Bit 48     - The CSx Mapping.
LVD_TH             - Bit 49     - The threshold of the LVD.
LVD_RST_SEL        - Bit 50     - The LVD reset input source.
LVD_WNG_SEL        - Bit 51     - The LVD warning signal input source.
OTP_LOCK           - Bit 63     - The OTP lock bit (cannot be set).
Security:
When the chip is secure, execute the following command to unlock it:

    SigFlashCmd -emu str9.par -flash str9 -extra erase=all

```

TODO: Describe STR9 options here.

LPC Programmer

This programmer is invoked when the `-flash LPC` option is used. The programmer supports programming of internal flash in all LPC2xxx and LPC17xx devices.

LPC Programmer Specific Files

The following configuration and test files can be used as sample programmer invocations (Table 5):


```

OK - Flash programming finished ...
OK - Exiting ...
OK - Done

C:\Signum\lib\Debug>sigflashcmd -cpu Cortex-M3 -flash LPC -dump
SigFlashCmd Version 1.16 (C) Signum Systems Corp. 2006-2009
sigflashcmd -cpu Cortex-M3 -flash LPC -dump
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
LPC: Detected CPU LPC1768 (ID=0x00033F35)
LPC: Using a fixed CPU frequency of 4000KHz.
Geometry:
  bank #0 addr=0x00000000 size=64KB (16*4KB)
  bank #0 addr=0x00010000 size=448KB (14*32KB)
OK - Flash programming finished ...
OK - Exiting ...
OK - Done

C:\Signum\Flasher>

```

The screen capture shows two program runs, one for the LPC2138 and one for the LPC1768 microcontrollers. The program reports the CPU names, IDs and flash sector sizes.

Another way to connect to Cortex-M3 based LPC17xx using the SWD protocol. In order to do so, you must use the ARM-SWD probe and apply the following additional command line options:

```

-emu CoresightSWJ=SWD -emu JtagHeader=ARM-SWD
- or -
-emu M3swd.par

```

The latter method utilizes the `./boards/m3swd.par` file that contains the two emulation settings used in the former method.

LPC Programmer Options

The figure below shows a fragment of the LPC programmer help information about the available options and a list of supported processors obtained by executing the command: `SigFlashCmd -help -flash LPC`.

LPC programmer options:

f<khz> - Specify the external crystal frequency in KHz. (Default: 14700.)
p1 - Handle the User Program Signature in the vector table (default).
p0 - Do not handle the User Program Signature in the vector table.
LPC<cpu> - Select the CPU by name. (Default: select by ID).

Supported CPUs:

LPC1751, LPC1752, LPC1754, LPC1756, LPC1758
LPC1764, LPC1765, LPC1766, LPC1768

LPC2101, LPC2102, LPC2103, LPC2104, LPC2105, LPC2106, LPC2109
LPC2114, LPC2119
LPC2124, LPC2129
LPC2131, LPC2132, LPC2134, LPC2136, LPC2138
LPC2141, LPC2142, LPC2144, LPC2146, LPC2148
LPC2194
LPC2212, LPC2214
LPC2292, LPC2294
LPC2361, LPC2362, LPC2364, LPC2365, LPC2366, LPC2367, LPC2368
LPC2377, LPC2378
LPC2387, LPC2388
LPC2458, LPC2468, LPC2478

Individual LPC programmer options are discussed below.

Flash Base Address and RAM Usage

The LPC programmer only allows 0x0 as the flash base address. It terminates with an error if a different base address is specified in the `-flash LPC@0x100` option.

The LPC programmer always uses internal ram at 0x40000000 (0x10000000 for LPC17xx), and thus it silently ignores the `-ram` option.

Specifying the Device Manually

After connecting to the CPU successfully, the LPC programmer reads the CPU ID and looks for a device with a matching ID in the programmer's internal database. If

the device cannot be not found, the programmer displays an error message and lists the names of supported CPUs.

Different revisions of the LPC2xxx/LPC17xx CPU may have different IDs. If an ID has not been found in the database, it still may be possible to program the CPU. Provided you have the necessary CPU information, you may be able to force the use of the desired CPU by specifying its name with the `-flash` option, like in `f-flash LPC,LPCnnnn`, where `nnnn` is the LPC device number, such as 2101 or 7168. Note the comma (,) separator between the programmer name (LPC) and the CPU name (LPCnnnn).

Crystal Frequency

The LPC programmer calls on-chip IAP (In-Application Programming) routines located in the LPC ROM. During erasing or programming, these routines require the actual CPU frequency to be passes to them to ensure proper timing constraints imposed on the flash programming process taking place in ROM.

You should specify the `f<freq>` option as follows:

```
-flash LPC,f12000
```

The default frequency is 14700 KHz.

Experiments show that selecting frequencies higher than that that of the actual crystal is not detrimental, otherwise than slowing the programming process to a certain extent.. However, due to the possibility of other side effects related to incorrect speed, the option should always specify the actual crystal frequency.

Some LPC devices — LPC28xx, LPC17xx for example — have internal an clock oscillator with a fixed frequency. Such devices make the programmer ignore the specified external crystal frequency and use an internal oscillator during programming. This may generate messages like this:

```
LPC: WARNING: CPU frequency 12000KHz specified by the f option is
ignored.
LPC: Using fixed CPU frequency 4000KHz.
```

User Program Signature

When booting, all LPC devices start to execute the code from internal ROM. After performing a certain amount of house-keeping, the ROM code verifies the integrity of the flash by calculating the sum of the first eight 32-bit words in the flash at 0x0. If this sum is 0x0, then the flash contains valid code and control can be transferred to the user's code in flash memory.

When the programmer attempts to program vectors, it calculates the 32-bit sum and stores it at the unused vector location 0x14 (for LPC2xxx) or 0x1C (for LPC17xx). This is signaled by the following messages:

```
LPC: User Program Signature in file: 0x18171615 (calculated: 0x878E959D)  
LPC: New User Program Signature set: 0x878E959D
```

You can skip the calculation of the checksum by specifying the `p0` option as follows:

```
-flash LPC,p0
```

The original value from the file is written to the flash when passes the programmer's verification. Otherwise, the following warning message appears:

```
LPC: User Program Signature in file: 0x18171615 (calculated: 0x878E959D)  
LPC: WARNING: New User Program Signature not set (,p0 option used)
```

The verification process does not compare the calculated checksum with the value stored in the file (which would result in a verification error), which is signaled by the following message:

```
LPC: User Program Signature skipped while verifying ...
```

If your file contains a good checksum — as in a complete flash image saved to a file — and you want the checksum to be verified, use the `-flash LPC,p0` option.

Programming Speed

To display programming times and performance data of a flasher run, use the -time option. Table 6 is an example of programming times for a 500 KB file.

CPU + JTAG	CPU CLOCK	ERASE	PROGRAM	VERIFY OR BLANK- CHECK
LPC2138 Adaptive	12 MHz external	0.43 sec	10.91 sec	4.88 sec
LPC2168 Adaptive	48 MHz from 4 MHz OSC	0.42 sec	5.97 sec	2.28 sec
LPC1768 JTAG 10Mhz	4 MHz internal	0.13 sec	8.95 sec	2.16 sec
LPC1768 JTAG 30Mhz	4 MHz internal	0.13 sec	8.17 sec	1.88 sec
LPC1768 SWD 10Mhz	4 MHz internal	0.13 sec	12.09 sec	1.73 sec
LPC1768 SWD 30Mhz	4 MHz internal	0.13 sec	9.96 sec	1.51 sec

TABLE 6 Timing example for a 500KB image file.

As Table 6 indicates, the erase time for the LPC17xx depends mainly on the CPU type. A faster JTAG clock does not affect it, but it does slightly improve the programming and verification speed.

Programming in the SWD mode is a little slower, while the verify-or-blankcheck operation is faster, than it is in the JTAG mode,.

It is expected that future versions of the LPC programmer, especially in the SWD mode, will work even faster,

NAND Programmer

The NAND programmer supports a limited number of NAND controllers—iMX31, DM644x and DM35x. The documentation is available to qualifying customers.

RAM Programmer

The RAM programmer can be used to write to or read data from memory (including a single location), and can be used for verifying the correctness of RAM access.

The programmer treats the entire 32-bit memory address space as one large flash. It allows you to erase a range (filling memory with 0xFF), blank check, program and verify the flash. Be sure to set the `-offset` (or range for erase and blankcheck) options correctly to avoid writing to non-existent memory. Consider the following example:

```
C:\test>sigflashcmd -cpu Cortex-M3 -flash RAM -program 2k@0x1000_0000 -verify
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
sigflashcmd -cpu Cortex-M3 -flash RAM -program 2k@0x1000_0000 -verify
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
Programming 0x10000000 .. 0x100007FF (2KB) ...
Verifying 0x10000000 .. 0x100007FF (2KB) ...
OK - Flash programming finished ...
OK - Exiting ...
OK - Done
C:\test>sigflashcmd -cpu Cortex-M3 -flash RAM -read 16@0x1000_0000
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
sigflashcmd -cpu Cortex-M3 -flash RAM -read 16@0x1000_0000
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
0x10000000 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
OK - Flash programming finished ...
OK - Exiting ...
OK - Done
```

In the preceding example, the first programmer call programs (writes) the 2K of RAM at 0x1000_0000 with a test pattern and verifies the result. The second call reads the initial 16 bytes of the same RAM and displays them.

Custom Flash Devices

Flasher does not provide for customization of flash-dependent modules. The existing flash-dependent SigFlashProg.dll modules handle the majority of external flash devices currently on the market. If you have special needs, please contact Signum Systems. Our contact information can be found at www.signum.com.



UM-B-FlashCmd 6.2.09.11.14 780