Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'flashrom.8' command

$ man flashrom.8

FLASHROM(8) System Manager's Manual FLASHROM(8)

NAME

flashrom - detect, read, write, verify and erase flash chips

SYNOPSIS

flashrom [-h|-R|-L|-z|

-p <programmername>[:<parameters>] [-c <chipname>]

(--flash-name|--flash-size|

[-E|-r <file>|-w <file>|-v <file>]

[(-l <file>|--ifd| --fmap|--fmap-file <file>) [-i <im?

age>]]

[-n] [-N] [-f])]

[-V[V[V]]] [-o <logfile>]

DESCRIPTION

flashrom is a utility for detecting, reading, writing, verifying and erasing flash chips. It's often used to flash BIOS/EFI/core? boot/firmware images in-system using a supported mainboard. However, it also supports various external PCI/USB/parallel-port/serial-port based devices which can program flash chips, including some network cards (NICs), SATA/IDE controller cards, graphics cards, the Bus Pirate de? vice, various FTDI FT2232/FT4232H/FT232H based USB devices, and more. It supports a wide range of DIP32, PLCC32, DIP8, SO8/SOIC8, TSOP32, TSOP40, TSOP48, and BGA chips, which use various protocols such as LPC, FWH, parallel flash, or SPI.

OPTIONS

IMPORTANT: Please note that the command line interface for flashrom will change before flashrom 1.0. Do not use flashrom in scripts or other automated tools without checking that your flashrom version won't interpret options in a different way.

You can specify one of -h, -R, -L, -z, -E, -r, -w, -v or no operation.

If no operation is specified, flashrom will only probe for flash chips.

It is recommended that if you try flashrom the first time on a system, you run it in probe-only mode and check the output. Also you are ad? vised to make a backup of your current ROM contents with -r before you try to write a new image. All operations involving any chip access (probe/read/write/...) require the -p/--programmer option to be used (please see below).

-r, --read <file>

Read flash ROM contents and save them into the given <file>. If the file already exists, it will be overwritten.

-w, --write <file>

Write <file> into flash ROM. This will first automatically erase the chip, then write to it.

In the process the chip is also read several times. First an in-memory backup is made for disaster recovery and to be able to skip regions that are already equal to the image file. This copy is updated along with the write operation. In case of erase er? rors it is even re-read completely. After writing has finished and if verification is enabled, the whole flash chip is read out and compared with the input image.

-n, --noverify

Skip the automatic verification of flash ROM contents after writing. Using this option is not recommended, you should only use it if you know what you are doing and if you feel that the time for verification takes too long.

Typical usage is: flashrom -p prog -n -w <file>

This option is only useful in combination with --write.

-N, --noverify-all

Skip not included regions during automatic verification after writing (cf. -l and -i). You should only use this option if you are sure that communication with the flash chip is reliable (e.g. when using the internal programmer). Even if flashrom is instructed not to touch parts of the flash chip, their contents could be damaged (e.g. due to misunderstood erase commands). This option is required to flash an Intel system with locked ME flash region using the internal programmer. It may be enabled by default in this case in the future.

-v, --verify <file>

Verify the flash ROM contents against the given <file>.

-E, --erase

Erase the flash ROM chip.

-V, --verbose

More verbose output. This option can be supplied multiple times (max. 3 times, i.e. -VVV) for even more debug output.

-c, --chip <chipname>

Probe only for the specified flash ROM chip. This option takes the chip name as printed by flashrom -L without the vendor name as parameter. Please note that the chip name is case sensitive.

-f, --force

Force one or more of the following actions:

* Force chip read and pretend the chip is there.

* Force chip access even if the chip is bigger than the maximum supported size for the flash bus.

* Force erase even if erase is known bad.

* Force write even if write is known bad.

-l, --layout <file>

Read ROM layout from <file>.

flashrom supports ROM layouts. This allows you to flash certain parts of the flash chip only. A ROM layout file contains multi? ple lines with the following syntax:

startaddr:endaddr imagename

startaddr and endaddr are hexadecimal addresses within the ROM file and do not refer to any physical address. Please note that using a 0x prefix for those hexadecimal numbers is not neces? sary, but you can't specify decimal/octal numbers. imagename is an arbitrary name for the region/image from startaddr to en? daddr (both addresses included).

Example:

00000000:00008fff gfxrom

00009000:0003ffff normal

00040000:0007ffff fallback

If you only want to update the image named normal in a ROM based on the layout above, run

flashrom -p prog --layout rom.layout --image normal -w some.rom

To update only the images named normal and fallback, run:

flashrom -p prog -l rom.layout -i normal -i fallback -w some.rom

Overlapping sections are not supported.

--fmap Read layout from fmap in flash chip.

flashrom supports the fmap binary format which is commonly used by coreboot for partitioning a flash chip. The on-chip fmap will be read and used to generate the layout.

If you only want to update the COREBOOT region defined in the fmap, run

flashrom -p prog --fmap --image COREBOOT -w some.rom

--fmap-file <file>

Read layout from a <file> containing binary fmap (e.g. coreboot roms).

flashrom supports the fmap binary format which is commonly used by coreboot for partitioning a flash chip. The fmap in the spec? ified file will be read and used to generate the layout.

If you only want to update the COREBOOT region defined in the

binary fmap file, run

flashrom -p prog --fmap-file some.rom --image COREBOOT -w some.rom

--ifd Read ROM layout from Intel Firmware Descriptor.

flashrom supports ROM layouts given by an Intel Firmware De? scriptor (IFD). The on-chip descriptor will be read and used to generate the layout. If you need to change the layout, you have to update the IFD only first.

The following ROM images may be present in an IFD:

fd the IFD itself

bios the host firmware aka. BIOS

me Intel Management Engine firmware

gbe gigabit ethernet firmware

pd platform specific data

-i, --image <imagename>

Only flash region/image <imagename> from flash layout.

--flash-name

Prints out the detected flash chips name.

--flash-size

Prints out the detected flash chips size.

-L, --list-supported

List the flash chips, chipsets, mainboards, and external pro? grammers (including PCI, USB, parallel port, and serial port based devices) supported by flashrom.

There are many unlisted boards which will work out of the box, without special support in flashrom. Please let us know if you can verify that other boards work or do not work out of the box.

IMPORTANT: For verification you have to test an ERASE and/or WRITE operation, so make sure you only do that if you have proper means to recover from failure!

-z, --list-supported-wiki

Same as --list-supported, but outputs the supported hardware in MediaWiki syntax, so that it can be easily pasted into the sup?

ported hardware wiki page ?https://flashrom.org/Supported_hard?
ware?. Please note that MediaWiki output is not compiled in by
default.

-p, --programmer <name>[:parameter[,parameter[,parameter]]]

Specify the programmer device. This is mandatory for all opera?
tions involving any chip access (probe/read/write/...). Cur?
rently supported are:

* internal (for in-system flashing in the mainboard)

* dummy (virtual programmer for testing flashrom)

* nic3com (for flash ROMs on 3COM network cards)

* nicrealtek (for flash ROMs on Realtek and SMC 1211 network
cards)

* nicnatsemi (for flash ROMs on National Semiconductor DP838*
network cards)

* nicintel (for parallel flash ROMs on Intel 10/100Mbit network
cards)

* gfxnvidia (for flash ROMs on NVIDIA graphics cards)

* drkaiser (for flash ROMs on Dr. Kaiser PC-Waechter PCI cards)

* satasii (for flash ROMs on Silicon Image SATA/IDE controllers)

* satamv (for flash ROMs on Marvell SATA controllers)

* atahpt (for flash ROMs on Highpoint ATA/RAID controllers)

* atavia (for flash ROMs on VIA VT6421A SATA controllers)

* atapromise (for flash ROMs on Promise PDC2026x ATA/RAID con?
trollers)

* it8212 (for flash ROMs on ITE IT8212F ATA/RAID controller)

* ft2232_spi (for SPI flash ROMs attached to an
FT2232/FT4232H/FT232H family based USB SPI programmer).

* serprog (for flash ROMs attached to a programmer speaking ser?
prog, including some Arduino-based devices).

* buspirate_spi (for SPI flash ROMs attached to a Bus Pirate)

* dediprog (for SPI flash ROMs attached to a Dediprog SF100)

* rayer_spi (for SPI flash ROMs attached to a parallel port by
one of various cable types)

* pony_spi (for SPI flash ROMs attached to a SI-Prog serial port bitbanging adapter)

* nicintel_spi (for SPI flash ROMs on Intel Gigabit network cards)

* ogp_spi (for SPI flash ROMs on Open Graphics Project graphics card)

* linux_mtd (for SPI flash ROMs accessible via /dev/mtdX on Linux)

* linux_spi (for SPI flash ROMs accessible via /dev/spidevX.Y on Linux)

* usbblaster_spi (for SPI flash ROMs attached to an Altera USB-Blaster compatible cable)

* nicintel_eeprom (for SPI EEPROMs on Intel Gigabit network cards)

* mstarddc_spi (for SPI flash ROMs accessible through DDC in MSTAR-equipped displays)

* pickit2_spi (for SPI flash ROMs accessible via Microchip PICkit2)

* ch341a_spi (for SPI flash ROMs attached to WCH CH341A)

* digilent_spi (for SPI flash ROMs attached to iCEblink40 devel? opment boards)

* jlink_spi (for SPI flash ROMs attached to SEGGER J-Link and compatible devices)

* ni845x_spi (for SPI flash ROMs attached to National Instru? ments USB-8451 or USB-8452)

* stlinkv3_spi (for SPI flash ROMs attached to STMicroelectron? ics STLINK V3 devices)

Some programmers have optional or mandatory parameters which are described in detail in the PROGRAMMER-SPECIFIC INFORMATION sec? tion. Support for some programmers can be disabled at compile time. flashrom -h lists all supported programmers.

-h, --help

Show a help text and exit.

-o, --output <logfile>

Save the full debug log to <logfile>. If the file already ex? ists, it will be overwritten. This is the recommended way to gather logs from flashrom because they will be verbose even if the on-screen messages are not verbose and don't require output redirection.

-R, --version

Show version information and exit.

PROGRAMMER-SPECIFIC INFORMATION

Some programmer drivers accept further parameters to set programmer-specific parameters. These parameters are separated from the programmer name by a colon. While some programmers take arguments at fixed posi? tions, other programmers use a key/value interface in which the key and value is separated by an equal sign and different pairs are separated by a comma or a colon.

internal programmer

Board Enables

Some mainboards require to run mainboard specific code to enable flash erase and write support (and probe support on old systems with parallel flash). The mainboard brand and model (if it re? quires specific code) is usually autodetected using one of the following mechanisms: If your system is running coreboot, the mainboard type is determined from the coreboot table. Other? wise, the mainboard is detected by examining the onboard PCI de? vices and possibly DMI info. If PCI and DMI do not contain in? formation to uniquely identify the mainboard (which is the ex? ception), or if you want to override the detected mainboard model, you can specify the mainboard using the

flashrom -p internal:mainboard=<vendor>:<board> syntax.

See the 'Known boards' or 'Known laptops' section in the output of 'flashrom -L' for a list of boards which require the specifi? cation of the board name, if no coreboot table is found.

Some of these board-specific flash enabling functions (called

board enables) in flashrom have not yet been tested. If your mainboard is detected needing an untested board enable function, a warning message is printed and the board enable is not exe? cuted, because a wrong board enable function might cause the system to behave erratically, as board enable functions touch the low-level internals of a mainboard. Not executing a board enable function (if one is needed) might cause detection or erasing failure. If your board protects only part of the flash (commonly the top end, called boot block), flashrom might en? counter an error only after erasing the unprotected part, so running without the board-enable function might be dangerous for erase and write (which includes erase).

The suggested procedure for a mainboard with untested board spe? cific code is to first try to probe the ROM (just invoke flashrom and check that it detects your flash chip type) without running the board enable code (i.e. without any parameters). If it finds your chip, fine. Otherwise, retry probing your chip with the board-enable code running, using

```
flashrom -p internal:boardenable=force
```

If your chip is still not detected, the board enable code seems to be broken or the flash chip unsupported. Otherwise, make a backup of your current ROM contents (using -r) and store it to a medium outside of your computer, like a USB drive or a network share. If you needed to run the board enable code already for probing, use it for reading too. If reading succeeds and the contens of the read file look legit you can try to write the new image. You should enable the board enable code in any case now, as it has been written because it is known that writing/erasing without the board enable is going to fail. In any case (success or failure), please report to the flashrom mailing list, see be? low.

Coreboot

On systems running coreboot, flashrom checks whether the desired

image matches your mainboard. This needs some special board ID to be present in the image. If flashrom detects that the image you want to write and the current board do not match, it will refuse to write the image unless you specify

flashrom -p internal:boardmismatch=force

ITE IT87 Super I/O

If your mainboard is manufactured by GIGABYTE and supports Dual? BIOS it is very likely that it uses an ITE IT87 series Super I/O to switch between the two flash chips. Only one of them can be accessed at a time and you can manually select which one to use with the

flashrom -p internal:dualbiosindex=chip

syntax where chip is the index of the chip to use (0 = main, 1 = backup). You can check which one is currently selected by leav? ing out the chip parameter.

If your mainboard uses an ITE IT87 series Super I/O for LPC<->SPI flash bus translation, flashrom should autodetect that configuration. If you want to set the I/O base port of the IT87 series SPI controller manually instead of using the value pro? vided by the BIOS, use the

flashrom -p internal:it87spiport=portnum

syntax where portnum is the I/O port number (must be a multiple of 8). In the unlikely case flashrom doesn't detect an active IT87 LPC<->SPI bridge, please send a bug report so we can diag? nose the problem.

AMD chipsets

Beginning with the SB700 chipset there is an integrated micro? controller (IMC) based on the 8051 embedded in every AMD south? bridge. Its firmware resides in the same flash chip as the host's which makes writing to the flash risky if the IMC is ac? tive. Flashrom tries to temporarily disable the IMC but even then changing the contents of the flash can have unwanted ef? fects: when the IMC continues (at the latest after a reboot) it

will continue executing code from the flash. If the code was re?
moved or changed in an unfortunate way it is unpredictable what
the IMC will do. Therefore, if flashrom detects an active IMC it
will disable write support unless the user forces it with the

flashrom -p internal:amd_imc_force=yes

syntax. The user is responsible for supplying a suitable image
or leaving out the IMC region with the help of a layout file.
This limitation might be removed in the future when we under?
stand the details better and have received enough feedback from
users. Please report the outcome if you had to use this option
to write a chip.
An optional spispeed parameter specifies the frequency of the
SPI bus where applicable (i.e. SB600 or later with an SPI flash
chip directly attached to the chipset). Syntax is

flashrom -p internal:spispeed=frequency

where frequency can be '16.5 MHz', '22 MHz', '33 MHz', '66 MHz',
'100 MHZ', or '800 kHz'. Support of individual frequencies de?
pends on the generation of the chipset:
* SB6xx, SB7xx, SP5xxx: from 16.5 MHz up to and including 33 MHz
* SB8xx, SB9xx, Hudson: from 16.5 MHz up to and including 66 MHz
* Yangtze (with SPI 100 engine as found in Kabini and Tamesh):
all of them
The default is to use 16.5 MHz and disable Fast Reads.

Intel chipsets

If you have an Intel chipset with an ICH8 or later southbridge
with SPI flash attached, and if a valid descriptor was written
to it (e.g. by the vendor), the chipset provides an alternative
way to access the flash chip(s) named Hardware Sequencing. It
is much simpler than the normal access method (called Software
Sequencing), but does not allow the software to choose the SPI
commands to be sent. You can use the

flashrom -p internal:ich_spi_mode=value

syntax where value can be auto, swseq or hwseq. By default (or

when setting ich_spi_mode=auto) the module tries to use swseq and only activates hwseq if need be (e.g. if important opcodes are inaccessible due to lockdown; or if more than one flash chip is attached). The other options (swseq, hwseq) select the re? spective mode (if possible).

ICH8 and later southbridges may also have locked address ranges of different kinds if a valid descriptor was written to it. The flash address space is then partitioned in multiple so called "Flash Regions" containing the host firmware, the ME firmware and so on respectively. The flash descriptor can also specify up to 5 so called "Protected Regions", which are freely chosen ad? dress ranges independent from the aforementioned "Flash Re? gions". All of them can be write and/or read protected individu? ally.

If you have an Intel chipset with an ICH2 or later southbridge and if you want to set specific IDSEL values for a non-default flash chip or an embedded controller (EC), you can use the

flashrom -p internal:fwh_idsel=value

syntax where value is the 48-bit hexadecimal raw value to be written in the IDSEL registers of the Intel southbridge. The up? per 32 bits use one hex digit each per 512 kB range between 0xffc00000 and 0xffffffff, and the lower 16 bits use one hex digit each per 1024 kB range between 0xff400000 and 0xff7fffff. The rightmost hex digit corresponds with the lowest address range. All address ranges have a corresponding sister range 4 MB below with identical IDSEL settings. The default value for ICH7 is given in the example below.

Example: flashrom -p internal:fwh_idsel=0x001122334567

Laptops

Using flashrom on older laptops that don't boot from the SPI bus is dangerous and may easily make your hardware unusable (see also the BUGS section). The embedded controller (EC) in some ma? chines may interact badly with flashing. More information is in

the wiki ?https://flashrom.org/Laptops?. Problems occur when the flash chip is shared between BIOS and EC firmware, and the latter does not expect flashrom to access the chip. While flashrom tries to change the contents of that memory the EC might need to fetch new instructions or data from it and could stop working correctly. Probing for and reading from the chip may also irritate your EC and cause fan failure, backlight fail? ure, sudden poweroff, and other nasty effects. flashrom will at? tempt to detect if it is running on such a laptop and limit probing to SPI buses. If you want to probe the LPC bus anyway at your own risk, use

flashrom -p internal:laptop=force_I_want_a_brick

We will not help you if you force flashing on a laptop because this is a really dumb idea.

You have been warned.

Currently we rely on the chassis type encoded in the DMI/SMBIOS data to detect laptops. Some vendors did not implement those bits correctly or set them to generic and/or dummy values. flashrom will then issue a warning and restrict buses like above. In this case you can use

flashrom -p internal:laptop=this_is_not_a_laptop

to tell flashrom (at your own risk) that it is not running on a laptop.

dummy programmer

The dummy programmer operates on a buffer in memory only. It provides a safe and fast way to test various aspects of flashrom and is mainly used in development and while debugging. It is able to emulate some chips to a certain degree (basic iden? tify/read/erase/write operations work).

An optional parameter specifies the bus types it should support. For that you have to use the

flashrom -p dummy:bus=[type[+type[+type]]]

syntax where type can be parallel, lpc, fwh, spi in any order.

If you specify bus without type, all buses will be disabled. If you do not specify bus, all buses will be enabled.

Example: flashrom -p dummy:bus=lpc+fwh

The dummy programmer supports flash chip emulation for automated self-tests without hardware access. If you want to emulate a flash chip, use the

flashrom -p dummy:emulate=chip

syntax where chip is one of the following chips (please specify only the chip name, not the vendor):

* ST M25P10.RES SPI flash chip (128 kB, RES, page write)
* SST SST25VF040.REMS SPI flash chip (512 kB, REMS, byte write)
* SST SST25VF032B SPI flash chip (4096 kB, RDID, AAI write)
* Macronix MX25L6436 SPI flash chip (8192 kB, RDID, SFDP)

Example: flashrom -p dummy:emulate=SST25VF040.REMS

Persistent images

If you use flash chip emulation, flash image persistence is available as well by using the

flashrom -p dummy:emulate=chip,image=image.rom

syntax where image.rom is the file where the simulated chip con? tents are read on flashrom startup and where the chip contents on flashrom shutdown are written to.

Example: flashrom -p dummy:emulate=M25P10.RES,image=dummy.bin

SPI write chunk size

If you use SPI flash chip emulation for a chip which supports SPI page write with the default opcode, you can set the maximum allowed write chunk size with the

flashrom -p dummy:emulate=chip,spi_write_256_chunksize=size

syntax where size is the number of bytes (min. 1, max. 256).

Example:

flashrom -p dummy:emulate=M25P10.RES,spi_write_256_chunksize=5

SPI blacklist

To simulate a programmer which refuses to send certain SPI com? mands to the flash chip, you can specify a blacklist of SPI com?

mands with the

flashrom -p dummy:spi_blacklist=commandlist

syntax where commandlist is a list of two-digit hexadecimal rep?

resentations of SPI commands. If commandlist is e.g. 0302,

flashrom will behave as if the SPI controller refuses to run

command 0x03 (READ) and command 0x02 (WRITE). commandlist may

be up to 512 characters (256 commands) long. Implementation

note: flashrom will detect an error during command execution.

SPI ignorelist

To simulate a flash chip which ignores (doesn't support) certain

SPI commands, you can specify an ignorelist of SPI commands with

the

flashrom -p dummy:spi_ignorelist=commandlist

syntax where commandlist is a list of two-digit hexadecimal rep?

resentations of SPI commands. If commandlist is e.g. 0302, the

emulated flash chip will ignore command 0x03 (READ) and command

0x02 (WRITE). commandlist may be up to 512 characters (256 com?

mands) long. Implementation note: flashrom won't detect an er?

ror during command execution.

SPI status register

You can specify the initial content of the chip's status regis?

ter with the

flashrom -p dummy:spi_status=content

syntax where content is an 8-bit hexadecimal value.

nic3com, nicrealtek, nicnatsemi, nicintel, nicintel_eeprom, nicintel_spi,

gfxnvidia, ogp_spi, drkaiser, satasii, satamv, atahpt, atavia , at?

apromise and it8212 programmers

These programmers have an option to specify the PCI address of

the card your want to use, which must be specified if more than

one card supported by the selected programmer is installed in

your system. The syntax is

flashrom -p xxxx:pci=bb:dd.f,

where xxxx is the name of the programmer, bb is the PCI bus num?

ber, dd is the PCI device number, and f is the PCI function num?

ber of the desired device.

Example: flashrom -p nic3com:pci=05:04.0

atavia programmer

Due to the mysterious address handling of the VIA VT6421A con?

troller the user can specify an offset with the

flashrom -p atavia:offset=addr

syntax where addr will be interpreted as usual (leading 0x (0)

for hexadecimal (octal) values, or else decimal). For more in?

formation please see its wiki page

?https://flashrom.org/VT6421A?.

atapromise programmer

This programmer is currently limited to 32 kB, regardless of the

actual size of the flash chip. This stems from the fact that, on

the tested device (a Promise Ultra100), not all of the chip's

address lines were actually connected. You may use this program?

mer to flash firmware updates, since these are only 16 kB in

size (padding to 32 kB is required).

nicintel_eeprom programmer

This is the first programmer module in flashrom that does not

provide access to NOR flash chips but EEPROMs mounted on gigabit

Ethernet cards based on Intel's 82580 NIC. Because EEPROMs nor?

mally do not announce their size nor allow themselves to be

identified, the controller relies on correct size values written

to predefined addresses within the chip. Flashrom follows this

scheme but assumes the minimum size of 16 kB (128 kb) if an un?

programmed EEPROM/card is detected. Intel specifies following

EEPROMs to be compatible: Atmel AT25128, AT25256, Micron (ST)

M95128, M95256 and OnSemi (Catalyst) CAT25CS128.

ft2232_spi programmer

This module supports various programmers based on FTDI

FT2232/FT4232H/FT232H chips including the DLP Design DLP-

USB1232H, openbiosprog-spi, Amontec JTAGkey/JTAGkey-

tiny/JTAGkey-2, Dangerous Prototypes Bus Blaster, Olimex ARM-USB-TINY/-H, Olimex ARM-USB-OCD/-H, OpenMoko Neo1973 Debug board (V2+), TIAO/DIYGADGET USB Multi-Protocol Adapter (TUMPA), TUMPA Lite, GOEPEL PicoTAP, Google Servo v1/v2 and Tin Can Tools Fly? swatter/Flyswatter 2.

An optional parameter specifies the controller type, channel/in? terface/port and GPIO-based chip select it should support. For that you have to use the

flashrom -p ft2232_spi:type=model,port=interface,csgpiol=gpio

syntax where model can be 2232H, 4232H, 232H, jtagkey, bus? blaster, openmoko, arm-usb-tiny, arm-usb-tiny-h, arm-usb-ocd, arm-usb-ocd-h, tumpa, tumpalite, picotap, google-servo, google-servo-v2 or google-servo-v2-legacy interface can be A, B, C, or D and csgpiol can be a number between 0 and 3, denoting GPIOL0-GPIOL3 correspondingly. The default model is 4232H the default interface is A and GPIO is not used by default.

If there is more than one ft2232_spi-compatible device con? nected, you can select which one should be used by specifying its serial number with the

flashrom -p ft2232_spi:serial=number

syntax where number is the serial number of the device (which can be found for example in the output of lsusb -v).

All models supported by the ft2232_spi driver can configure the SPI clock rate by setting a divisor. The expressible divisors are all even numbers between 2 and 2^17 (=131072) resulting in SPI clock frequencies of 6 MHz down to about 92 Hz for 12 MHz inputs. The default divisor is set to 2, but you can use another one by specifying the optional divisor parameter with the

flashrom -p ft2232_spi:divisor=div

syntax.

serprog programmer

This module supports all programmers speaking the serprog proto? col. This includes some Arduino-based devices as well as various

programmers by Urja Rannikko, Juhana Helovuo, Stefan Tauner, Chi Zhang and many others.

A mandatory parameter specifies either a serial device (and baud rate) or an IP/port combination for communicating with the pro? grammer. The device/baud combination has to start with dev= and separate the optional baud rate with a colon. For example

flashrom -p serprog:dev=/dev/ttyS0:115200

If no baud rate is given the default values by the operating system/hardware will be used. For IP connections you have to use the

flashrom -p serprog:ip=ipaddr:port

syntax. In case the device supports it, you can set the SPI clock frequency with the optional spispeed parameter. The fre? quency is parsed as hertz, unless an M, or k suffix is given, then megahertz or kilohertz are used respectively. Example that sets the frequency to 2 MHz:

flashrom -p serprog:dev=/dev/device:baud,spispeed=2M

More information about serprog is available in serprog-proto? col.txt in the source distribution.

buspirate_spi programmer

A required dev parameter specifies the Bus Pirate device node and an optional spispeed parameter specifies the frequency of the SPI bus. The parameter delimiter is a comma. Syntax is

flashrom -p buspirate_spi:dev=/dev/device,spispeed=frequency

where frequency can be 30k, 125k, 250k, 1M, 2M, 2.6M, 4M or 8M (in Hz). The default is the maximum frequency of 8 MHz.

The baud rate for communication between the host and the Bus Pi? rate can be specified with the optional serialspeed parameter. Syntax is

flashrom -p buspirate_spi:serialspeed=baud

where baud can be 115200, 230400, 250000 or 2000000 (2M). The default is 2M baud for Bus Pirate hardware version 3.0 and greater, and 115200 otherwise.

An optional pullups parameter specifies the use of the Bus Pi? rate internal pull-up resistors. This may be needed if you are working with a flash ROM chip that you have physically removed from the board. Syntax is

flashrom -p buspirate_spi:pullups=state

where state can be on or off. More information about the Bus Pirate pull-up resistors and their purpose is available in a guide by dangerousprototypes ?http://dangerousproto? types.com/docs/Practical_guide_to_Bus_Pirate_pull-up_resistors?. Only the external supply voltage (Vpu) is supported as of this writing.

pickit2_spi programmer

An optional voltage parameter specifies the voltage the PICkit2 should use. The default unit is Volt if no unit is specified. You can use mV, millivolt, V or Volt as unit specifier. Syntax is

flashrom -p pickit2_spi:voltage=value

where value can be 0V, 1.8V, 2.5V, 3.5V or the equivalent in mV. An optional spispeed parameter specifies the frequency of the SPI bus. Syntax is

flashrom -p pickit2_spi:spispeed=frequency

where frequency can be 250k, 333k, 500k or 1M (in Hz). The de? fault is a frequency of 1 MHz.

dediprog programmer

An optional voltage parameter specifies the voltage the Dediprog should use. The default unit is Volt if no unit is specified. You can use mV, milliVolt, V or Volt as unit specifier. Syntax is

flashrom -p dediprog:voltage=value

where value can be 0V, 1.8V, 2.5V, 3.5V or the equivalent in mV. An optional device parameter specifies which of multiple con? nected Dediprog devices should be used. Please be aware that the order depends on libusb's usb_get_busses() function and that

the numbering starts at 0. Usage example to select the second device:

flashrom -p dediprog:device=1

An optional spispeed parameter specifies the frequency of the SPI bus. The firmware on the device needs to be 5.0.0 or newer. Syntax is

flashrom -p dediprog:spispeed=frequency

where frequency can be 375k, 750k, 1.5M, 2.18M, 3M, 8M, 12M or 24M (in Hz). The default is a frequency of 12 MHz.

An optional target parameter specifies which target chip should be used. Syntax is

flashrom -p dediprog:target=value

where value can be 1 or 2 to select target chip 1 or 2 respec? tively. The default is target chip 1.

rayer_spi programmer

The default I/O base address used for the parallel port is 0x378 and you can use the optional iobase parameter to specify an al? ternate base I/O address with the

flashrom -p rayer_spi:iobase=baseaddr

syntax where baseaddr is base I/O port address of the parallel port, which must be a multiple of four. Make sure to not forget the "0x" prefix for hexadecimal port addresses.

The default cable type is the RayeR cable. You can use the op? tional type parameter to specify the cable type with the

flashrom -p rayer_spi:type=model

syntax where model can be rayer for the RayeR cable, byte? blastermv for the Altera ByteBlasterMV, stk200 for the Atmel STK200/300, wiggler for the Macraigor Wiggler, xilinx for the Xilinx Parallel Cable III (DLC 5), or spi_tt for SPI Tiny Tools-compatible hardware.

More information about the RayeR hardware is available at RayeR's website ?http://rayer.g6.cz/elektro/spipgm.htm?. The Altera ByteBlasterMV datasheet can be obtained from Altera

?http://www.altera.co.jp/literature/ds/dsbytemv.pdf?. For more information about the Macraigor Wiggler see their company homepage ?http://www.macraigor.com/wiggler.htm?. The schematic of the Xilinx DLC 5 was published in a Xilinx user guide ?http://www.xilinx.com/support/documentation/user_guides/xtp029.pdf?.

pony_spi programmer

The serial port (like /dev/ttyS0, /dev/ttyUSB0 on Linux or COM3 on windows) is specified using the mandatory dev parameter. The adapter type is selectable between SI-Prog (used for SPI devices with PonyProg 2000) or a custom made serial bitbanging programmer named "serbang". The optional type parameter accepts the values "si_prog" (default) or "serbang".

Information about the SI-Prog adapter can be found at its website ?http://www.lancos.com/siprogsch.html?.

An example call to flashrom is

flashrom -p pony_spi:dev=/dev/ttyS0,type=serbang

Please note that while USB-to-serial adapters work under certain circumstances, this slows down operation considerably.

ogp_spi programmer

The flash ROM chip to access must be specified with the rom parameter.

flashrom -p ogp_spi:rom=name

Where name is either cprom or s3 for the configuration ROM and bprom or bios for the BIOS ROM. If more than one card supported by the ogp_spi programmer is installed in your system, you have to specify the PCI address of the card you want to use with the pci= parameter as explained in the nic3com et al. section above.

linux_mtd programmer

You may specify the MTD device to use with the

flashrom -p linux_mtd:dev=/dev/mtdX

syntax where /dev/mtdX is the Linux device node for your MTD device. If left unspecified the first MTD device found (e.g. /dev/mtd0) will be used by default.

Please note that the linux_mtd driver only works on Linux.

linux_spi programmer

You have to specify the SPI controller to use with the

flashrom -p linux_spi:dev=/dev/spidevX.Y

syntax where /dev/spidevX.Y is the Linux device node for your SPI controller.

In case the device supports it, you can set the SPI clock frequency with the optional spispeed parameter. The frequency is parsed as kilohertz. Example that sets the frequency to 8 MHz:

flashrom -p linux_spi:dev=/dev/spidevX.Y,spispeed=8000

Please note that the linux_spi driver only works on Linux.

mstarddc_spi programmer

The Display Data Channel (DDC) is an I2C bus present on VGA and DVI connectors, that allows exchanging information between a computer and attached displays. Its most common uses are getting display capabilities through EDID (at I2C address 0x50) and sending commands to the display using the DDC/CI protocol (at address 0x37). On displays driven by MSTAR SoCs, it is also possible to access the SoC firmware flash (connected to the Soc through another SPI bus) using an In-System Programming (ISP) port, usually at address 0x49. This flashrom module allows the latter via Linux's I2C driver.

IMPORTANT: Before using this programmer, the display MUST be in standby mode, and only connected to the computer that will run flashrom using a VGA cable, to an inactive VGA output. It absolutely MUST NOT be used as a display during the procedure!

You have to specify the DDC/I2C controller and I2C address to use with the

flashrom -p mstarddc_spi:dev=/dev/i2c-X:YY

syntax where /dev/i2c-X is the Linux device node for your I2C controller connected to the display's DDC channel, and YY is the (hexadecimal) address of the MSTAR ISP port (address 0x49 is usually used). Example that uses I2C controller /dev/i2c-1 and

address 0x49:

flashrom -p mstarddc_spi:dev=/dev/i2c-1:49

It is also possible to inhibit the reset command that is normally sent to the display once the flashrom operation is completed using the optional noreset parameter. A value of 1 prevents flashrom from sending the reset command. Example that does not reset the display at the end of the operation:

flashrom -p mstarddc_spi:dev=/dev/i2c-1:49,noreset=1

Please note that sending the reset command is also inhibited if an error occurred during the operation. To send the reset command afterwards, you can simply run flashrom once more, in chip probe mode (not specifying an operation), without the noreset parameter, once the flash read/write operation you intended to perform has completed successfully.

Please also note that the mstarddc_spi driver only works on Linux.

ch341a_spi programmer

The WCH CH341A programmer does not support any parameters currently.

SPI frequency is fixed at 2 MHz, and CS0 is used as per the device.

ni845x_spi programmer

An optional voltage parameter could be used to specify the IO voltage. This parameter is available for the NI USB-8452 device.

The default unit is Volt if no unit is specified. You can use mV, milliVolt, V or Volt as unit specifier. Syntax is

flashrom -p ni845x_spi:voltage=value

where value can be 1.2V, 1.5V, 1.8V, 2.5V, 3.3V or the equivalent in mV.

In the case if none of the programmer's supported IO voltage is within the supported voltage range of the detected flash chip the flashrom will abort the operation (to prevent damaging the flash chip). You can override this behaviour by passing "yes" to the ignore_io_voltage_limits parameter (for e.g. if you are using an external voltage translator circuit). Syntax is

flashrom -p ni845x_spi:ignore_io_voltage_limits=yes

You can use the serial parameter to explicitly specify which connected NI USB-845x device should be used. You should use your device's 7 digit hexadecimal serial number. Usage example to select the device with 1230A12 serial number:

flashrom -p ni845x_spi:serial=1230A12

An optional spispeed parameter specifies the frequency of the SPI bus. Syntax is

flashrom -p ni845x_spi:spispeed=frequency

where frequency should a number corresponding to the desired frequency in kHz. The maximum frequency is 12 MHz (12000 kHz) for the USB-8451 and 50 MHz (50000 kHz) for the USB-8452. The default is a frequency of 1 MHz (1000 kHz).

An optional cs parameter specifies which target chip select line should be used. Syntax is

flashrom -p ni845x_spi:csnumber=value

where value should be between 0 and 7 By default the CS0 is used.

digilent_spi programmer

An optional spispeed parameter specifies the frequency of the SPI bus. Syntax is

flashrom -p digilent_spi:spispeed=frequency

where frequency can be 62.5k, 125k, 250k, 500k, 1M, 2M or 4M (in Hz). The default is a frequency of 4 MHz.

jlink_spi programmer

This module supports SEGGER J-Link and compatible devices.

The MOSI signal of the flash chip must be attached to TDI pin of the programmer, MISO to TDO and SCK to TCK. The chip select (CS) signal of the flash chip can be attached to different pins of the programmer which can be selected with the

flashrom -p jlink_spi:cs=pin

syntax where pin can be either TRST or RESET. The default pin for chip select is RESET. Note that, when using RESET, it is

normal that the indicator LED blinks orange or red.

Additionally, the VTref pin of the programmer must be attached to the logic level of the flash chip. The programmer measures the voltage on this pin and generates the reference voltage for its input comparators and adapts its output voltages to it.

Pinout for devices with 20-pin JTAG connector:

```
   +-------+
   |  1  2 |     1: VTref     2:
   |  3  4 |     3: TRST      4: GND
   |  5  6 |     5: TDI       6: GND
  +-+  7  8 |     7:           8: GND
  |    9 10 |     9: TCK       10: GND
  |   11 12 |    11:           12: GND
  +-+ 13 14 |    13: TDO       14:
    | 15 16 |    15: RESET     16:
    | 17 18 |    17:           18:
    | 19 20 |    19: PWR_5V    20:
    +-------+
```

If there is more than one compatible device connected, you can select which one should be used by specifying its serial number with the

```
flashrom -p jlink_spi:serial=number
```

syntax where number is the serial number of the device (which can be found for example in the output of lsusb -v).

The SPI speed can be selected by using the

```
flashrom -p jlink_spi:spispeed=frequency
```

syntax where frequency is the SPI clock frequency in kHz. The maximum speed depends on the device in use.

stlinkv3_spi programmer

This module supports SPI flash programming through the STMicroelectronics STLINK V3 programmer/debugger's SPI bridge interface

```
flashrom -p stlinkv3_spi
```

If there is more than one compatible device connected, you can select which one should be used by specifying its serial number with the

flashrom -p stlinkv3_spi:serial=number

syntax where number is the serial number of the device (which can be found for example in the output of lsusb -v).

The SPI speed can be selected by using the

flashrom -p stlinkv3_spi:spispeed=frequency

syntax where frequency is the SPI clock frequency in kHz. If the passed frequency is not supported by the adapter the nearest lower supported frequency will be used.

EXAMPLES

To back up and update your BIOS, run

flashrom -p internal -r backup.rom -o backuplog.txt

flashrom -p internal -w newbios.rom -o writelog.txt

Please make sure to copy backup.rom to some external media before you try to write. That makes offline recovery easier.

If writing fails and flashrom complains about the chip being in an unknown state, you can try to restore the backup by running

flashrom -p internal -w backup.rom -o restorelog.txt

If you encounter any problems, please contact us and supply backuplog.txt, writelog.txt and restorelog.txt. See section BUGS for contact info.

EXIT STATUS

flashrom exits with 0 on success, 1 on most failures but with 3 if a call to mmap() fails.

REQUIREMENTS

flashrom needs different access permissions for different programmers.

internal needs raw memory access, PCI configuration space access, raw I/O port access (x86) and MSR access (x86).

atavia needs PCI configuration space access.

nic3com, nicrealtek and nicnatsemi need PCI configuration space read access and raw I/O port access.

atahpt needs PCI configuration space access and raw I/O port access.

gfxnvidia, drkaiser and it8212 need PCI configuration space access and raw memory access.

rayer_spi needs raw I/O port access.

satasii, nicintel, nicintel_eeprom and nicintel_spi need PCI configuration space read access and raw memory access.

satamv and atapromise need PCI configuration space read access, raw I/O port access and raw memory access.

serprog needs TCP access to the network or userspace access to a serial port.

buspirate_spi needs userspace access to a serial port.

ft2232_spi, usbblaster_spi and pickit2_spi need access to the respective USB device via libusb API version 0.1.

ch341a_spi and dediprog need access to the respective USB device via libusb API version 1.0.

dummy needs no access permissions at all.

internal, nic3com, nicrealtek, nicnatsemi, gfxnvidia, drkaiser, satasii, satamv, atahpt, atavia and atapromise have to be run as superuser/root, and need additional raw access permission.

serprog, buspirate_spi, dediprog, usbblaster_spi, ft2232_spi, pickit2_spi, ch341a_spi and digilent_spi can be run as normal user on most operating systems if appropriate device permissions are set.

ogp needs PCI configuration space read access and raw memory access.

On OpenBSD, you can obtain raw access permission by setting securelevel=-1 in /etc/rc.securelevel and rebooting, or rebooting into single user mode.

BUGS

Please report any bugs to the flashrom mailing list ?flashrom@flashrom.org?.

We recommend to subscribe first at https://flashrom.org/mailman/listinfo/flashrom.

Many of the developers communicate via the #flashrom IRC channel on chat.freenode.net. If you don't have an IRC client, you can use the

freenode webchat ?http://webchat.freenode.net/?channels=flashrom?. You are welcome to join and ask questions, send us bug and success reports there too. Please provide a way to contact you later (e.g. a mail address) and be patient if there is no immediate reaction. Also, we provide a pastebin service ?https://paste.flashrom.org? that is very useful when you want to share logs etc. without spamming the channel.

Laptops

Using flashrom on older laptops is dangerous and may easily make your hardware unusable. flashrom will attempt to detect if it is running on a susceptible laptop and restrict flash-chip probing for safety reasons. Please see the detailed discussion of this topic and associated flashrom options in the Laptops paragraph in the internal programmer subsection of the PROGRAMMER-SPECIFIC INFORMATION section and the information in our wiki ?https://flashrom.org/Laptops?.

One-time programmable (OTP) memory and unique IDs

Some flash chips contain OTP memory often denoted as "security registers". They usually have a capacity in the range of some bytes to a few hundred bytes and can be used to give devices unique IDs etc. flashrom is not able to read or write these memories and may therefore not be able to duplicate a chip completely. For chip types known to include OTP memories a warning is printed when they are detected.

Similar to OTP memories are unique, factory programmed, unforgeable IDs. They are not modifiable by the user at all.

LICENSE

flashrom is covered by the GNU General Public License (GPL), version 2. Some files are additionally available under any later version of the GPL.

COPYRIGHT

Please see the individual files.

AUTHORS

Andrew Morgan

Carl-Daniel Hailfinger

Claus Gindhart

David Borg

David Hendricks

Dominik Geyer

Edward O'Callaghan

Eric Biederman

Giampiero Giancipoli

Helge Wagner

Idwer Vollering

Joe Bao

Joerg Fischer

Joshua Roys

Ky?sti M?lkki

Luc Verhaegen

Li-Ta Lo

Mark Marshall

Markus Boas

Mattias Mattsson

Michael Karcher

Nikolay Petukhov

Patrick Georgi

Peter Lemenkov

Peter Stuge

Reinder E.N. de Haan

Ronald G. Minnich

Ronald Hoogenboom

Sean Nelson

Stefan Reinauer

Stefan Tauner

Stefan Wildemann

Stephan Guilloux

Steven James

Urja Rannikko

Uwe Hermann

Wang Qingpei

Yinghai Lu

some others, please see the flashrom svn changelog for details.

All still active authors can be reached via the mailing list ?flashrom@flashrom.org?.

This manual page was written by Uwe Hermann ?uwe@hermann-uwe.de?, Carl-Daniel Hailfinger, Stefan Tauner and others. It is licensed under the terms of the GNU GPL (version 2 or later).

FLASHROM(8)