NAME

stm32flash − flashing utility for STM32 through UART or I2C

SYNOPSIS

stm32flash [**−cfhjkouvCR**] [**−a** *bus_address*] [**−b** *baud_rate*] [**−m** *serial_mode*] [**−r** *filename*] [**−w** *filename*] [**−e** *num*] [**−n** *count*] [**−g** *address*] [**−s** *start_page*] [**−S** *address*[:*length*]] [**−F** *RX_length*[:*TX_length*]] [**−i** *GPIO_string*] [*tty_device* | *i2c_device*]

DESCRIPTION

stm32flash reads or writes the flash memory of STM32.

It requires the STM32 to embed a bootloader compliant with ST application note AN3155 or AN4221. **stm32flash** uses the serial port *tty_device* or the i2c port *i2c_device* to interact with the bootloader of STM32.

OPTIONS

−a *bus_address*

Specify address on bus for *i2c_device*. This option is mandatory for I2C interface.

−b *baud_rate*

Specify baud rate speed of *tty_device*. Please notice that the ST bootloader can automatically detect the baud rate, as explained in chapter 2 of AN3155. This option could be required together with option **−c** or if following interaction with bootloader is expected. Default is *57600*.

−m *mode*

Specify the format of UART data. *mode* is a three characters long string where each character specifies, in this strict order, character size, parity and stop bits. The only values currently used are *8e1* for standard STM32 bootloader and *8n1* for standard STM32W bootloader. Default is *8e1*.

−r *filename*

Specify to read the STM32 flash and write its content in *filename* in ra w binary format (see below **FORMAT CONVERSION**).

−w *filename*

Specify to write the STM32 flash with the content of *filename*. File format can be either ra w binary or intel hex (see below **FORMAT CONVERSION**). The file format is automatically detected. To by−pass format detection and force binary mode (e.g. to write an intel hex content in STM32 flash), use **−f** option.

−u Specify to disable write−protection from STM32 flash. The STM32 will be reset after this operation.

−j Enable the flash read−protection.

−k Disable the flash read−protection.

−o Erase only.

−e *num* Specify to erase *num* pages before writing the flash. Default is to erase the necessary pages based on the file content size and/or parameters provided with the -S option, or the whole flash if this

cannot be determined. With **−e 0** the flash would not be erased.

−v Specify to verify flash content after write operation.

−n *count*
Specify to retry failed writes up to *count* times. Default is 10 times.

−g *address*
Specify address to start execution from (0 = flash start).

−s *start_page*
Specify flash page offset (0 = flash start).

−S *address***[:***length***]**
Specify start address and optionally length for read/write/erase/crc operations.

−F *RX_length***[:***TX_length***]**
Specify the maximum frame size for the current interface. Due to STM32 bootloader protocol, host will never handle frames bigger than 256 byte in RX or 258 byte in TX. Due to current code, lowest limit in RX is 20 byte (to read a complete reply of command GET). Minimum limit in TX is 5 byte, required by protocol.

−f Force binary parser while reading file with **−w**.

−h Show help.

−c Specify to resume the existing UART connection and don't send initial INIT sequence to detect baud rate. Baud rate must be kept the same as the existing connection. This is useful if the reset fails.

−i *GPIO_string*
Specify the GPIO sequences on the host to force STM32 to enter and exit bootloader mode. GPIO can either be real GPIO connected from host to STM32 beside the UART connection, or UART's modem signals used as GPIO. (See below **BOOTLOADER GPIO SEQUENCE** for the format of *GPIO_string* and further explanation).

−C Specify to compute CRC on memory content. By default the CRC is computed on the whole flash content. Use**−S** to pro vide different memory address range.

−R Specify to reset the device at exit. This option is ignored if either**−g**, **−j**, **−k** or **−u** is also specified.

BOOTLOADER GPIO SEQUENCE

This feature is currently available on Linux host only.

As explained in ST application note AN2606, after reset the STM32 will execute either the application program in user flash or the bootloader, depending on the level applied at specific pins of STM32 during reset.

STM32 bootloader is automatically activated by configuring the pins BOOT0="high" and BOOT1="low"

and then by applying a reset. Application program in user flash is activated by configuring the pin BOOT0="low" (the level on BOOT1 is ignored) and then by applying a reset.

When GPIO from host computer are connected to either configuration and reset pins of STM32, **stm32flash** can control the host GPIO to reset STM32 and to force execution of bootloader or execution of application program.

The sequence of GPIO values to entry to and exit from bootloader mode is provided with command line option **−i** *GPIO_string*.

The format of*GPIO_string* is:

GPIO_string = [entry sequence][:[exit sequence]]
sequence = [[−]signal]&|,[sequence]

In the above sequences, negative numbers correspond to GPIO at "low" level; numbers without sign correspond to GPIO at "high" level. The value "n" can either be the GPIO number on the host system or the string "rts", "dtr" or "brk". The strings "rts" and "dtr" drive the corresponding UART's modem lines RTS and DTR as GPIO. The string "brk" forces the UART to send a BREAK sequence on TX line; after BREAK the UART is returned in normal "non−break" mode. Note: the string "−brk" has no effect and is ignored.

The ',' delimiter adds 100 ms of delay between signal toggles, whereas the '&' delimiter adds no delay. An empty signal, thus repeated ',' delimiters, can be used to insert larger delays in multiples of 100 ms. E.g. "rts,,,,−dtr" will set RTS, then wait 400 ms, then reset DTR. "rts&−dtr" will set RTS and reset DTR without delay. You can use ',' delimiters alone to simply add a delay between opening port and starting to flash.

Note that since version 0.6, an exit sequence will always be executed if specified, regardless of the -R option, to ensure the signals are reset. If -R is specified, but no exit sequence, a software-triggered reset will be performed.

As an example, let's suppose the following connection between host and STM32:

- host GPIO_3 connected to reset pin of STM32;
- host GPIO_4 connected to STM32 pin BOOT0;
- host GPIO_5 connected to STM32 pin BOOT1.

In this case, the sequence to enter in bootloader mode is: first put GPIO_4="high" and GPIO_5="low"; then send reset pulse by GPIO_3="low" followed by GPIO_3="high". The corresponding string for *GPIO_string* is "4,−5,−3,3".

To exit from bootloader and run the application program, the sequence is: put GPIO_4="low"; then send reset pulse. The corresponding string for*GPIO_string* is "−4,−3,3".

The complete command line flag is "−i 4,−5,−3,3:−4,−3,3".

STM32W uses pad PA5 to select boot mode; if during reset PA5 is "low" then STM32W will enter in bootloader mode; if PA5 is "high" it will execute the program in flash.

As an example, suppose GPIO_3 is connected to PA5 and GPIO_2 to STM32W's reset. The command:

stm32flash −i '−3&−2,2:3&−2,,,2' /dev/ttyS0

provides:

- entry sequence: GPIO_3=low, GPIO_2=low, 100 ms delay, GPIO_2=high
- exit sequence: GPIO_3=high, GPIO_2=low, 300 ms delay, GPIO_2=high

Another example, introducing a delay after port opening. The command:

stm32flash −i ',,,,,:rts&−dtr,,,2' /dev/ttyS00,

provides:

- entry sequence: delay 500 ms
- exit sequence: RTS=high, DTR=low, 300 ms delay, GPIO_2=high

EXAMPLES

Get device information:

```
stm32flash /dev/ttyS0
```

Write with verify and then start execution:

```
stm32flash −w filename −v −g 0x0 /dev/ttyS0
```

Read flash to file:

```
stm32flash −r filename /dev/ttyS0
```

Start execution:

```
stm32flash −g 0x0 /dev/ttyS0
```

Specify:

- entry sequence: RTS=low, DTR=low, DTR=high
- exit sequence: RTS=high, DTR=low, DTR=high

```
stm32flash −i −rts,−dtr,dtr:rts,−dtr,dtr /dev/ttyS0
```

FORMAT CONVERSION

Flash images provided by ST or created with ST tools are often in file format Motorola S−Record. Conversion between raw binary, intel hex and Motorola S−Record can be done through software package SRecord.

AUTHORS

The original software package **stm32flash** is written by *Geoffrey McRae* and is since 2012 maintained by *Tormod Volden.* See AUTHORS file in source code for more contributors.

Man page and extension to STM32W and I2C are written by *Antonio Borneo.*

Please report any bugs at the project homepage http://stm32flash.sourceforge.net .

SEE ALSO

srec_cat(1), **srec_intel**(5), **srec_motorola**(5).

The communication protocol used by ST bootloader is documented in following ST application notes, depending on communication port. The current version of **stm32flash** only supports *UART* and *I2C* ports.

- AN3154: CAN protocol used in the STM32 bootloader
  http://www.st.com/web/en/resource/technical/document/application_note/CD00264321.pdf

- AN3155: USART protocol used in the STM32(TM) bootloader
  http://www.st.com/web/en/resource/technical/document/application_note/CD00264342.pdf

- AN4221: I2C protocol used in the STM32 bootloader
  http://www.st.com/web/en/resource/technical/document/application_note/DM00072315.pdf

- AN4286: SPI protocol used in the STM32 bootloader
  http://www.st.com/web/en/resource/technical/document/application_note/DM00081379.pdf

Boot mode selection for STM32 is documented in ST application note AN2606, available from the ST website:
http://www.st.com/web/en/resource/technical/document/application_note/CD00167594.pdf

LICENSE

stm32flash is distributed under GNU GENERAL PUBLIC LICENSE Version 2. Copy of the license is available within the source code in the file *gpl−2.0.txt*.