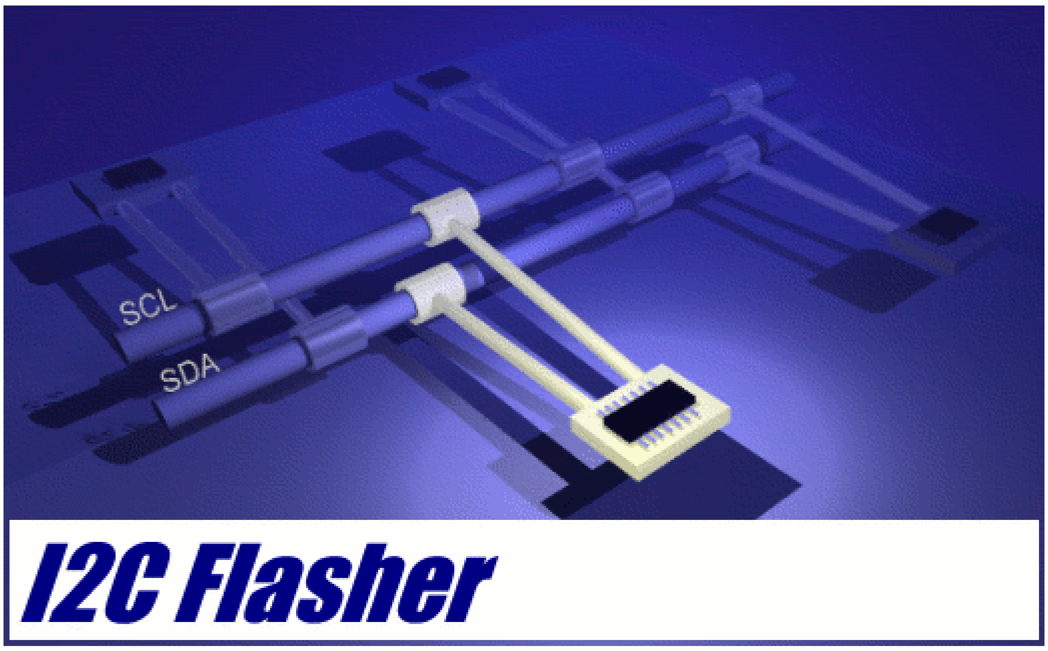
I2C Memory Device Flash Tool

I2C Flasher

User Manual



telos Systementwicklung GmbH
Kaiser-Wilhelm-Strasse 93, 20355 Hamburg / Germany
Phone: +49 (0)40 450 173 – 61
Mail: tsupport@telos.de
Web: http://www.telos.info/

Document:	I2C_Flasher_User_Manual.odt
Version:	2.7
Date:	2018-09-19

Table Of Contents

1 Disclaimer

This product is not designed for use in life support appliances, devices or systems where malfunction of this product can reasonably be expected to result in personal injury.

Customers using or selling this product for use in such applications do so at their own risk and agree to fully indemnify telos for any damages resulting from such improper use or sale.

2 Product Outline

This document is the user manual for the ***I2C Flasher*** application. It refers to the software version 1.1.

It focuses on the features that can be offered by this program. This application is basically a front-end for ***Tracii XL***. For details concerning the I2C adapter refer to the hardware documentation.

This application is part of the ***I2C Studio*** development environment and uses the I2C Framework provided. In order to use this application with ***Tracii XL*** a separate ***I2C Flasher*** license must be purchased. It supports the integrated dummy device for testing and evaluation purposes without a license. Because the dummy device is solely realized in software, neither a ***Tracii XL*** nor a real memory device is needed in this case.

3 Purpose

The application ***I2C Flasher*** can be used to write, read and verify the data of I2C memory devices.

4 Supported Memory Devices

The product supports a wide range of different serial I2C memory devices. By running the application with the command line option '`-l`' a complete list of supported devices is printed out on the console:

```
    I2cFlasher -l
```

EEPROMs differ mainly in the number of address bits used for addressing the internal memory (not to confuse with the I2C bus address), capacity, page size and maximum bus speed. All these parameters have to be considered for an optimal writing performance. The application uses an internal map that holds the essential parameters for each device supported. It needs only a special device id to set up all necessary writing parameters.

This version of ***I2C Flasher*** also supports NXP (formerly Philips) Semiconductors ***Ultimate One Chip Third Generation (UOCIII)*** solution, a single integrated chip with programmable flash memory designed for mid and low-end analog TV receivers. The ***UOCIII*** comes in many different versions specialized on miscellaneous types of appliances (e.g., LCD or CRT TV's), and equipped with flash memory of differing size. Each ***UOCIII*** version has its own identifier as depicted in Table 1 below. The full identifier has to be passed on the command line (with the "`-d`" switch) to select a certain ***UOCIII*** variant.

Further information about the various UOCIII devices and their capabilities is not in the scope of this manual. Please contact NXP Semiconductors for details and data sheets.

ID Number	Full identifier
1	UOCIII_N1B_128
2	UOCIII_N1D_128
3	UOCIII_N1D_256
4	UOCIII_N1F_128_MONO
5	UOCIII_N1F_128_AV_SWITCH_ONLY
6	UOCIII_N1F_128_AV_FULL_STEREO
7	UOCIII_N1F_256_AV_FULL_STEREO
8	UOCIII_LCD_N1A_128
9	UOCIII_LCD_N1B_128
10	UOCIII_LCD_N1B_256
11	UOCIII_LCD_N1C_128
12	UOCIII_LCD_N1C_256
13	LOCI_N1A
14	LOCI_N1B
15	LOCI_N1C
16	LOCII_N1A
17	UOC_TOP_STEREO_128K
18	UOC_TOP_AV_STEREO_TC_TXT_CC_64K
19	UOC_TOP_MONO_TXT_CC_64K
20	UOCV_N1A_FULL_STEREO_WITHOUT_DEINTERLACER
21	UOCV_N1A_FULL_STEREO_WITH_DEINTERLACER

Table 1: NXP UOCIII Versions

5 Installation

The application ***I2C Flasher*** can be used directly if the following requirements are fulfilled:

- ✗ A ***Tracii XL*** is connected to the computer (for evaluation purposes, the dummy device may also be used).
- ✗ The ***I2C Studio*** development environment is installed.
- ✗ For NXP ***UOCIII*** only: A working installation of the URT (Universal Register Tool) API released by NXP. Please ensure that you use a recent version of this library.

Then, the application ***I2C Flasher*** can be started directly from the command line.

6 Usage

The application is a command line application. The execution requires a number of command line arguments that depend on the job that has to be performed. Optional arguments can be entered additionally to change the default behavior of the application.

- ✗ Arguments need to be separated with one or more spaces.
- ✗ Beside the normal form of an argument, which starts with two hyphens (‘`--`’), there is always a shortcut available that starts with a single hyphen (‘`-`’).
- ✗ If an argument requires an additional parameter, it has to be supplied directly after the option (e.g. `-o` *`outputfile`* or `--out-file` *`outputfile`*) or separated with a single equal sign in case of a long option (e.g. `--out-file=`*`outputfile`*).

Wherever numeric values are expected as parameters to command line arguments, they can be given in decimal or hexadecimal format. If the prefix ‘`0x`’ is added, the value is interpreted as a hexadecimal value.

6.1 Command Overview

Table 2 below gives an overview about all command-line options supported by the application:

Option	Description
-h, --help	Print help and exit
-V, --version	Print version and exit
Device Options:	
-s, --serial=<[0x]num>	Serial number of Tracii XL
-d, --device-id=<id>	Device identifier string (option `'-l, --list-devices'` displays a list)
-j, --net-host= <<host>[:<port>][@<password>]>	Specify host name/IP address and password for the host connected to Tracii XL board in order to flash a memory device over TCP/IP network
-b, --list-boards	Show list of boards (default = off)
-l, --list-devices	Show list of supported memory devices or, combined with `'-d, --device-id'`, list memory device parameters (default = off)
-k, --list-clockvals	Show list of supported bus clock frequencies (default = off)
-t, --list-termvals	Show list of supported termination values (default = off)
-n, --scan-addresses	Scan I2C bus for slave addresses (default = off)
Bus Options:	
-a, --slave-address=<[0x]address>	I2C bus slave address (default = '0x50')
-x, --prefix-slave-address= <[0x]address>	I2C bus slave address for prefix message (default = `'0x50'`)
-m, --prefix-message= <'[0x]xx [0x]yy ..'>	Message prefix bytes as string of space-separated, hexadecimal number pairs send to slave with address `<prefix-slave-address>` before flash operation

-c, --clock-freq=<freq>	Bus clock frequency [Hz] (default = ”100000”)
-u, --set-termination=<num>	Set I2C bus termination [Ohm] (option `'-t, --list-termvals'` displays supported values, 0 = off)

File Options:	
-i, --in-file=<filename>	Input data file
-f, --file-format=<[bin|hex|ihex]>	Input file format: plain binary, ASCII Hex or Intel Hex (default = `'bin'`)
-o, --out-file=<filename>	Dump memory image to file
-g, --log-file=<filename>	Log file
Flash Options:	
-e, `--enable-testpin=<[0|1]>`	Enable waiting for input testpin(s) (default = `'-1'`)
-p, --testpin-status=<[low|high]>	Required status for enabled testpin(s) (default = `'high'`)
-r, --retries=<num>	Number of retries on error(default = `'0'`)
-v, --verify	Verify data (default = off)
`-w, --set-write-prot`	Set software write protection, support is device-dependent (default = off). Warning: Setting software write protection is irreversible!
`-y, --set-vesa-mode`	Assume VESA Enhanced EDID standard, maintains serial number counter and manufacturing date automatically (default = off)
`-z, --loop`	Enable write loop (default = off)

Table 2: Command Overview

6.2 Help

A complete list of command line arguments including a brief description can be obtained by using the argument

`-h` or `--help`

6.3 Version Information

The version number of the application is displayed by the option

    -V or --version

Do not confuse the "`-V, --version`" option with "`-v, --verify`" described in section 6.7.4 below.

6.4 Device Options

6.4.1 Serial Number

This option specifies the serial number of ***Tracii XL*** to be used by the application:

    -s <[0x]num> or --serial <[0x]num>

The serial number `<num>` can be specified in decimal or hexadecimal notation. See section 6.4.3 on how to obtain a list of boards connected to the computer.

6.4.2 Device ID

The memory device ID can be supplied with the following option:

    -d <id> or --device-id <id>

`<id>` is a string containing a valid device ID as listed by the "List Devices" command in section 6.4.5 below.

This device ID is a unique identifier for any memory device supported by the application. It is the key to some essential information needed by the application in order to use the right set of parameters for device access. Therefore, any access to a specific memory device won't work until a device ID is specified on the command line.

6.4.3 Network Support

The application supports flashing over TCP/IP networks. The following option redirects all traffic to a ***Tracii XL*** board connected to a remote host accessible via the network:

    -j <<host>[:<port>][@<password>]> or
    --net-host <<host>[:<port>][@<password>]>

The remote host may be specified either by its name or its IP address. The port number and password are optional and depend on the remote host configuration. If omitted default values will be used (port number 3000, empty password). The scheduler running on this host must be configured properly to accept network connections. See the ***I2C Studio*** manual on how to set up the remote host configuration.

6.4.4 List Boards

This option lists all boards currently connected to the computer and accessible by the I2C Scheduler:

`-b` or `--list-boards`

The command also shows additional information about the license status and service contract.

6.4.5 List Devices

This command prints a complete list of the IDs of all memory devices currently supported:

`-l` or `--list-devices`

See also section 6.4.2 for a further explanation of these IDs.

6.4.6 List Bus Clock Frequencies

A list of valid bus clock frequencies can be obtained from the master with the option:

`-k` or `--list-clockvals`

The list depends on whether the target memory device supports fast and/or high-speed mode or not. See also section 6.5.3 below on how to set the I2C bus clock speed.

6.4.7 List Termination Values

A list of valid bus termination values will be displayed by the option:

`-t` or `--list-termvals`

See also section 6.5.4 below on how to set the bus termination.

6.4.8 Scan I2C Bus

The I2C bus can be scanned for the presence of slaves and their bus addresses by the option:

`-n` or `--scan-addresses`

This command outputs a list of slave addresses of all devices currently connected to the I2C bus. These devices may are not memory devices.

Currently only 7-bit addresses are considered because I2C EEPROMs with 10-bit addresses are not yet available. The flasher application scans all addresses in the valid range from 0x08 up to 0x77. Addresses out of this range are reserved according to the I2C standard and are not considered.

6.5 Bus Options

6.5.1 Slave Address Selection

The I2C-bus slave address of the memory device can be set using the command line argument:

`-a <[0x]address>` or `--slave-address <[0x]address>`

By default, the application expects the target memory device at the bus address 0x50. Bus addresses are always in 7-bit address format; 10-bit addresses are not supported. Note, that some manufacturers use so-called 8-bit addresses that are obtained by shifting the seven address bits by one to the left and adding the read/write bit as LSB. This address scheme is not mentioned in the I2C specification and will therefore not be used here.

6.5.2 Prefix Message

It is possible to define an I2C message that is sent immediately before the target memory device is accessed, e.g. to change the active port of an I2C bus IO expander before starting the flash process. Therefore, the I2C slave address and a number of data bytes to be sent to that slave can be defined.

The address is defined using the option

`-x <[0x]address>` or
`--prefix-slave-address <[0x]address>`

The data bytes have to be given as a space-separated sequence of numerical hexadecimal values in the range `[0x00 .. 0xFF]` that follow the option:

`-m <"[0x]xx [0x]yy .."> ` or
`--prefix-message <"[0x]xx [0x]yy .."> `

Important Note:
For compatibility reasons, the prefix message is always sent in standard mode with 100 kHz bus clock even if the `-c, --clock-freq` option specifies a different clock speed. Prefix messages are intended only for sending a couple of bytes to a device different from the target device prior to the flashing process.

6.5.3 I2C Bus Clock Frequency

The flasher handles all bus clock frequencies supported by ***Tracii XL***. Currently, normal, fast and high-speed mode up to 2 Mbit/s is available. High-speed mode is optional and needs a separate license. See the ***Tracii XL*** documentation for further details.

The default bit rate used to access the I2C bus is 100000 bit/s or 100 kHz respectively. It can be modified using the following option:

`-c <frequency>` or `--clock-freq <frequency>`

The parameter value must be specified as the desired bus clock speed in [Hz], e.g. 100000 for normal speed (100 kHz). If the specified bus clock frequency exceeds the maximum frequency supported by the target device the application exits with an error.

If the specified frequency is above 400000 (maximum speed in fast mode is 400 kHz according to the I2C bus specification) the flasher application automatically switches to high-speed mode. If the device does not support this mode the application will exit with an error.

The clock frequency cannot be specified arbitrarily but has to take certain discrete values provided by the master. See section 6.4.6 above on how to get a list of valid clock frequencies.

Important Note:
Setting the bus clock speed doesn't affect optional prefix messages (see section 6.5.2 above) which are always sent in standard mode with 100 kHz bus clock for compatibility reasons. Prefix messages are intended only for sending a couple of bytes to a device different from the target device prior to the flashing process.

6.5.4 Bus Termination

If the application frequently reports I2C bus errors and/or data transfers apparently are slower than expected then the I2C bus is likely to be terminated not appropriately. The following option can be used to set the termination:

`-u <num>` or `--set-termination <num>`

The unit of `<num>` is Ohm.

The bus termination cannot be specified arbitrarily but must take certain discrete values provided by the master. If the termination value is not supported the application will exit with a bus error. See section 6.4.7 above on how to get a list of valid termination values.

6.6 File Options

6.6.1 Input File

This option specifies the name of the input file that contains the data to be written:

`-i <filename>` or `--in-file <filename>`

If the directory where the file is located differs from the current one the path has to be included. The input file format can be specified with the option described in section 6.6.1 below. This option also triggers the flashing process.

Important Note: The data in the memory device will be overwritten without further notification.

6.6.2 Input File Format

This option determines the input file format:

`-f <[bin|hex|ihex]>` or `--file-format <[bin|hex|ihex]>`

The formats supported are binary (the default), ASCII Hex or Intel Hex. In case of binary format the file will be read as a simple unformatted byte stream without any interpretation. If one of the latter two formats has been selected, the content of the file will be interpreted accordingly and converted to binary format. If the input file contains errors the interpretation fails and the flashing process will not be started.

6.6.3 Output File

This argument initiates a memory dump of the selected device:

`-o <filename>` or `--out-file <filename>`

If this option is passed, the memory of the selected device will be dumped in binary format to the specified file before the flashing process starts.

If the directory where the file is located differs from the current one the path has to be included.

6.6.4 Log File

This argument specifies the name of an optional log file in ASCII text format:

`-g <filename>` or `--log-file <filename>`

If the directory where the file is located differs from the current one the path has to be included.

The log file protocols the command line arguments and all warning or error messages issued by the application. A time stamp is added to each message. If the specified file already exists the log output will be <u>appended</u>. Otherwise, a new file will be created.

6.7 Flash Options

6.7.1 Input Testpin Control

The input testpins of the I2C board can be used to trigger the flashing process. By using the argument

`-e <[0|1]>` or `--enable-testpin=<[0|1]>`

the flasher is asked to wait for the specified testpin signal before the flash process is started. The parameter of this argument selects the testpin:

✗ `0`: First Testpin

✗ `1`: Second Testpin

If a testpin is selected, the application waits – by default – until the test signal's level is 'high' before it starts to access the target memory device. In addition, the signal level on which the application should trigger can be specified by the option below.

Input testpin control can also be applied in loop mode to trigger a single flashing cycle (see section 6.7.7 below). This option may be very useful for automation.

6.7.2 Input Testpin Status

By using the option

`-p <[low|high]` or `--testpin-status [low|high]>`

the application can be forced to wait for either a '`low`' or a '`high`' signal level. This option is only useful in conjunction with the '`-e, --enable-testpin`' option above. The association of a testpin status argument with a certain testpin depends on the order of their appearance on the command line.

6.7.3 Number of Retries

This option specifies the number of retries the application should perform if a write operation fails:

`-r <num>` or `--retries <num>`

It is only useful in conjunction with the "`-i, --in-file`" or "`-v, --verify`" options. The parameter `<num>` gives the number of *retries*, i.e. the number of write attempts will be at least one and at most `<num+1>`.

Write errors are reported by the scheduler and include I2C protocol errors, timeouts, bus arbitration losses and incomplete data transfers.

6.7.4 Verify Data

This flag controls the data verification:

`-v` or `--verify`

If provided, the memory of the device is read out after a successful write operation and compared to the data read from the input file. If both data sets differ, the application exits with an error.

The use of this flag is recommended.

Do not confuse the "`-v, --verify`" option with upper-case "`-V, --version`" described in section 6.3, "Version Information" above.

6.7.5 Set Write Protection

This option sets the Software Write Protection:

`-w` or `--set-write-prot`

More precisely, the *Permanent Software Write Protection* is activated. The memory device must support this feature. If used in conjunction with a write action, the write protection is set after the write process has finished successfully.

Do not confuse *Software* with *Hardware* Write Protection, which is realized through a certain voltage level on a designated input pin.

Warning: Use with caution. Setting Software Write Protection is **irreversible**! The whole memory or a part of it cannot be written anymore.

6.7.6 Set VESA Mode

This option turns on some special handling for VESA DDC1/DDC2-compliant memory devices:

`-y` or `--set-vesa-mode`

The input file must be compliant to the VESA Enhanced EDID standard. With this option the application maintains a serial number counter and manufacturing date automatically.

The serial counter is managed through a file named “`<input_filename>.asc`“ in the same directory as the input file. `<input_filename>` is the filename specified through the “`-i, --in-file`“ option. If it has an extender, it will be replaced by “`.asc`”, otherwise a new extender “`.asc`” will be appended. It contains the serial number in ASCII text format and is updated after each flash operation to realize counter increment. If it doesn’t exist the serial number is read from a backup file named “`<input_filename>.asc.bak`”. If this file is also missing the counter value will be extracted directly from the flash image in the input file, and a new counter file along with its backup containing the incremented serial number will be created after a successful flash operation.

The manufacturing date is also managed automatically. Prior to flashing, the current week and year is stored in the flash image in a standard-compliant form. This requires the system time of the host computer to be set correctly.

With this counting mechanism it is not necessary to provide an initial counter file because the flash image itself will be used to init the serial number counter. However, the counter can be initialized to an arbitrary number by providing an ASCII formatted file named “`<input_filename>.asc`“ that contains a single line with the desired initial number.

Note: The input image file will not be changed. So if the files containing the current serial number (“`<input_filename>.asc`“ and “`<input_filename>.asc.bak`”) are both deleted serial numbering starts again with the value obtained from the input image given on the command line.

6.7.7 Enable Write Loop

This option enables the write loop:

`-z` or `--loop`

The same data can be written to the memory device as many times the user wants to. This is useful in case of flashing an arbitrary number of memory devices of the same type with the same data, e.g. in an industrial production process.

In this mode the time-consuming opening of the board and initialization of the flashing process is done only once. After each write cycle the user is prompted for another cycle to start or to finish the application.

This option also works under input testpin control (see section 6.7.1 and 6.7.2 above). In this case the write cycle is triggered by the selected input testpin and signal level instead of manual keyboard input which enables control of the flashing process by an automated environment (e.g. via some micro controller's GPIO pins, a PLC etc.).

7 Error Codes

The following table shows the error codes that are returned by the application:

Status	Meaning
0x00	Success
Initialization errors	
0x01	General failure
0x02	Board not found
0x03	Wrong serial number
0x04	License error
0x05	No memory device specified or specified memory device not found
0x06	Input data file not found
0x07	File read error
0x08	Specified download speed not supported
0x09	Other parameter error
Runtime errors	
0x10	Bus error
0x11	Write error
0x12	Maximum number of retries exceeded
0x13	Verification error

Table 3: Error Codes

8 Flashing Time Measurements

The table below shows flash time measurements for some typical EEPROM memory devices. All measured times are average values obtained from 25 flashing cycles, addressing the whole memory range with the maximum bus clock speed supported by the device.

Manufacturer	Device	Capacity	Clock Speed	Avg. Flash Time
Microchip	24LC256	256K (32 Kbytes)	400 kHz	8.96 sec
STM	M24128	128K (16Kbytes)	400 kHz	6.08 sec
Samsung	S524LB0X B1	64K (8 Kbytes)	400 kHz	4.92 sec
Fairchild	FM24C256	256K (32 Kbytes)	400 kHz	8.68 sec
Microchip	24LC64	64K (8 Kbytes)	400 kHz	5.68 sec

Table 4: Measured flashing times for some typical EEPROM’s

The benchmarks were performed on a PC running under Win2K, equipped with an Intel Pentium III processor at 650 MHz, 512 MB SD-RAM 100 and USB 1.1

9 Examples

9.1 Writing and Verifying Memory

In order to write the contents of a file into a memory device whose I2C address is 0x50, the following parameters are required:

- ✗ The **input file**
- ✗ The **target device type**
- ✗ The **serial number** of the I2C adapter.

Example 1: Flashing an ordinary EEPROM

- ✗ The binary input file name is “romimage.bin”.
- ✗ The optional log file is “flasher.log”.
- ✗ The target memory device is an AT24C04 device.
- ✗ The hexadecimal serial number of ***Tracii XL*** is 0x1234.

The device can be written by

```
I2cFlasher –s 0x1234 –d AT24C04 –i romimage.bin
```

If the verify flag ‘`–v`’ is specified, the content of the target memory device is verified by reading out the memory and comparing it to the input file. The target memory device is verified directly after it has been written. This is the recommended operation mode:

```
I2cFlasher –s 0x1234 –d AT24C04 –i romimage.bin –v
```

The AT24C04 device is capable of I2C fast mode with data transfers up to 400000 bit/s. In order to use fast mode transfers to speed up flashing the clock frequency has to be specified on the command line:

```
I2cFlasher –s 0x1234 –d AT24C04 –c 400000
  -i romimage.bin –v
```

In addition, an I2C message can be defined that is sent to a certain slave before the target memory device is accessed, e.g. to change the active port of an I2C bus IO expander before starting the flash process. Therefore, the I2C slave address and a number of data bytes to be sent to that slave can be defined.

```
I2cFlasher –s 0x1234 –d AT24C04 –x 0x51
  -m “de ad be ef” –c 400000 –i romimage.bin –v
```

In this case, the four bytes “`0xDE`”, “`0xAD`”, “`0xBE`” and “`0xEF`” are sent to the slave with bus address 0x51 prior to the flashing process.

Important Note:
For compatibility reasons the prefix message is <u>always</u> sent in standard mode with 100 kHz bus clock even if the “`-c`” option in this case specifies a different clock

speed. Prefix messages are intended only for sending a couple of bytes to a device different from the target device prior to the flashing process.

Therefore, in the case above the first four bytes (the prefix message) are sent to the slave device with address 0x51 in standard mode with 100 kHz clock speed. Then the bus is switched to fast mode with 400 kHz and the data transfer to the target device starts.

If no input file or prefix message is specified a warning message will appear to inform the user that no data has been written to the target device.

By specifying a log file, all relevant information, warning and/or error messages will be copied to the file:

```
    I2cFlasher –s 0x1234 –d AT24C04 –i romimage.bin
      -g flasher.log -v
```

If the log file already exists the messages will be appended, otherwise a new file will be created.

In the case of I2C bus errors it may be useful to change the bus termination before flashing:

```
    I2cFlasher –s 0x1234 –d AT24C04 –u 4960
```

Setting the termination value may be seamlessly combined with a write operation. A list of all termination values supported is displayed by:

```
    I2cFlasher –s 0x1234 –t
```

The flashing process can also be triggered by the input testpins provided by ***Tracii XL***:

```
    I2cFlasher –s 0x1234 –e 1 –e 0 –p high –p low
      -i romimage.bin -v
```

This combination of options causes the application to delay the start of the flashing operation until input testpin 0 is low and testpin 1 is high *simultaneously*. The assignment of input pins and status flags simply depends on their order on the command line.

Example 2: Flashing an ordinary EEPROM over the network

✗ The binary input file name on the local file system is “romimage.bin”

✗ The target memory device is an AT24C04 device.

✗ The remote host is a machine named “traciixlhost”.

✗ The scheduler running on the remote host is listening on port 3001, using the password “Geheim”.

✗ The hexadecimal serial number of Tracii XL connected to this remote host is 0xabcd.

The target memory device connected to the ***Tracii XL*** at the remote host can be written and verified by

```
    I2cFlasher -j traciixlhost:3001@Geheim -s 0xabcd
      -d AT24C04 -i romimage.bin -v
```

Other options for logging, sending prefixes etc. may also be specified as shown in Example 1 above.

Example 3: Flashing a VESA DDR1/DDR2-compliant memory device

- ✗ The binary input file name is “vesaimage.bin”.
- ✗ The target memory is a Microchip 24LC21A
- ✗ The hexadecimal serial number of ***Tracii XL*** is 0x1234.

The device can be written by

```
    I2cFlasher -s 0x1234 -d MC24LC21A -i vesaimage.bin -y
```

Other options for verifying, logging, sending prefixes etc. may also be specified as shown in Example 1 above

If a file named “`vesaimage.asc`” exists in the same directory as the input image it will be used to read and store the incremental counter for the serial number. A backup of this important file named “`vesaimage.asc.bak`” will always be created after the write operation finished successfully. If the original file is missing, this backup will be used.

The typical way to start an automated flashing loop is to provide a VESA Enhanced EDID-compliant image that contains the initial serial number. Alternatively, an ASCII-formatted text file named “`vesaimage.asc`” containing the initial serial number may be created manually.

Example 4: Flashing an NXP UOCIII

- ✗ The input file name in Intel hex format is “hercules_p1.hex”.
- ✗ The target is an NXP ***UOCIII*** with 128 KBytes of flash memory with identifier `UOCIII_N1B_128`
- ✗ The hexadecimal serial number of ***Tracii XL*** is 0x1234.

The device can be written by

```
  I2cFlasher -s 0x1234 -d UOCIII_N1B_128 -i hercules_p1.hex
```

The input file format specifier `-f, --file-format` is not mandatory in this case, because the flasher application passes all actions performed during UOCIII flashing to a special library that is responsible for UOCIII flashing only.

The input file must be valid for UOCIII – contact NXP Semiconductors for details about its special structure and how to generate this file. The UOCIII device also requires a working installation of the URT (Universal Register Tool) API released by NXP.

9.2 Reading from Memory

In order to save the contents of a memory device with I2C address 0x50 in a file, the following parameters are required:

- ✗ The **output file**
- ✗ The **target device type**
- ✗ The **serial number** of the I2C adapter.

Example:

- ✗ The output file name is “memory.dmp”.
- ✗ The target memory is an AT24C04 device.
- ✗ The hexadecimal serial number of ***Tracii XL*** is 0x1234.

The memory dump can be written by

```
    I2cFlasher –s 0x1234 –d AT24C04 –o memory.dmp
```

9.3 Input File Formats

The application accepts three different input file formats:

- ✗ Binary
- ✗ Intel HEX
- ✗ ASCII HEX

The binary format is the default.

By using the command line argument

`-f <fileformat>` or `--file-format <fileformat>`

the file format can be selected explicitly.

The supported `<fileformat>` values are

- ✗ `bin` for binary files
- ✗ `hex` for ASCII Hex files
- ✗ `ihex` for Intel Hex files.

Example:

- ✗ The input file name is “romdata.ihx”.
- ✗ The input file is in Intel Hex format.
- ✗ The target memory is an AT24C04 device.
- ✗ The hexadecimal serial number of ***Tracii XL*** is 0x1234.

The contents of the input file will be decoded, converted to a binary image and written to the device by

```
    I2cFlasher –s 0x1234 –d AT24C04 –i romdata.ihx
      -f ihex –v
```

10 Abbreviations and Glossary

Application	The program that runs on the personal computer and uses the features of the connected I2C adapter(s)
DDC	Display Data Channel. A VESA standard for “Plug-and-Play” monitors. It knows a transmit-only (DDC1) and a bi-directional mode (DDC2).
I2C	Inter IC Bus. Two wire bus for inter IC communication
I2C Adapter	Alternative Term: ***Tracii XL***
I2C Bus	A set of connected I2C master devices and I2C slave devices.
I2C Device	A device which is connected to the I2C bus.
I2C Master	I2C device that controls the transfers on the I2C bus
I2C Receiver	I2C device that is addressed by an I2C message
I2C Slave	I2C device that is chosen by the I2C master for a transfer
I2C Transmitter	I2C device that is the source of an I2C message
Scheduler	Component of the I2C Framework. It runs as an OS service process and manages the I2C adapters that are connected to the system.
Tracii XL	I2C adapter for the USB bus. It provides a master module, a slave module and a tracer module.
URT	Universal Register Tool. A Tool released by NXP (formerly Philips) Semiconductors to manipulate and/or read out registers of ICs via I2C.
USB	Universal Serial Bus
VESA	Video Electronics Standard Association. A non-profit corporation to set and support industry-wide interface standards designed for the PC, workstation, and other computing environments.

11 Contact

Address:	telos Systementwicklung GmbH Kaiser-Wilhelm-Strasse 93 20355 Hamburg/Germany
Phone:	+49(0)40 450173-61
Fax:	+49(0)40 450173-99
URL:	http://www.telos.info/i2cstudio
Email (Support)	tsupport@telos.de