NAME

mxli - Marc's exquisite LPC ISP programmer, version 3.x

SYNOPSIS

mxli [**−?**] [**−a** *destAddress,..*] [**−b** *baud*] [**−c** *crystal/Hz*] [**−d** *serialDevice*] [**−e** *sectors*] [**−f** *flashSize*] [**−g**] [**−i**] [**−j** *startAddress*] [**−k**] [**−l** *desiredLockLevel*] [**−m** *ramSize*] [**−n**] [**−o** *outputFile*] [**−p**] [**−r** *byteCount*] [**−s** *flashBank*] [**−t** *bootupTimeMs*] [**−u** *ucName*] [**−v**] [**−w**] [**−x**] [**−y** *flashBank*] [**−A** *flashBankAddress,..*] [**−B** *blockSize,..*] [**−E**] [**−F** *sectorSizexcount,..*] [**−I** *ID,..*] [**−L** *allowLockLevel*] [**−M** *size@baseAddress,..*] [**−N** *ucName*] [**−P** *protocol*] [**−R** *size@address,..*] [**−S** *count@index*] [**−T** *serialTimeoutMs*] [**−−crpAddress** *address*] [**−−deviceDefinition**] [**−−deviceList**] [**−−uid**] [**−−version**] [*image.bin* [*image2.bin..*]]

DESCRIPTION

mxli is a GPLv3 licensed ISP FLASH programmer for NXP's LPC line of ARM microcontrollers. mxli uses the UART communication channel of the LPC's boot-ROM ISP handler. mxli is designed to use STDOUT for output of a FLASH images OR explicit command output, like a microcontrollers serial number. That way, the output can easily be processed by other programs (scripting). STDERR is used for error reporting or progress/info/debug output. mxli version 3 pays special attention to multiple FLASH banks present at families like (as example) LPC1800 and LPC4300. mxli handles code read protection (CRP), and as default it takes care that CRP is NOT applied (accidentally) to your controllers FLASH. mxli is fast, probably much faster that other open source UART ISP flash programs. However, (this) version 3 is slightly slower than mxli-2.x . mxli-3 is a re-write of mxli-2 with focus on comprehensive support for all NXP LPC microcontrollers and a more detailed database.

TERMS AND DEFINITIONS

In this manual, we use the term **LPC** as a nickname for the NXP LPC microcontroller used as a target for mxli.

mxli builds on top of the **UART ISP boot loader** available to all LPCs. This boot loader is the program running from boot ROM when the microcontroller is in **ISP mode**.

The devices typically have a **FLASH memory** for storing the firmware. The FLASH memory is organized in 1 or 2 **FLASH banks** (may be more in the future) denoted by capital letters **A**,**B**,**..**. The special symbol **Z** is used for a hypothetical bank (always) at address 0. For most single FLASH bank LPCs, **Z**=**A**. When booting from FLASH, one of the banks is typically mapped to address 0 of the ARM core. This means, that the addresses may be different when writing FLASH data from the addresses used upon execution of the FLASH image. FLASH bank **Z** is the typical execution view of addresses, while **A** and **B** are the typical FLASH programming views of addresses. If you don't specify a FLASH bank, then **A** is used.

Each of the FLASH banks consists of one or more **sector groups**, with each group consisting of a number of **sectors** of the same size. So a FLASH bank is made up of n1 sectors of size s1 (group 1) followed by n2 sectors of size s2 (group 2) and so on. mxli assumes, that all FLASH banks share the same grouping of sectors.

All LPC devices include **on-chip RAM**. Because the on-chip RAM is typically not contiguous, we call the contiguous sections thereof **RAM regions**. Quite a few devices have multiple on-chip RAM regions, but some of these RAM regions might not be ready for use in ISP mode. RAM regions not ready for use in ISP mode must be considered non-existent from mxli's point of view. From this available RAM regions, the RAM requirements of the UART ISP boot loader must be subtracted to get the RAM available for mxli's use.

An **image** (the executable file to flash into the LPC) provided to mxli will typically be one contiguous block (**image segment**) of instructions and data. In more complex scenarios, however, the image might be split up into several contiguous blocks (**image segments**) at neccessarily different destination addresses. One **hex-file** can contain multiple segments. **bin-files** cannot, but you can provide multiple bin-files on the command-line along with multiple destination addresses (option -a). An image consists of one or more bin-files and/or one or more hex-files.

CONCEPTS

There are a few basic concepts that need to be understood for using mxli successfully.

Number formats

mxli accepts - in most places - multiple number formats. You can write numbers a decimal (examples: 1, -20), as hexadicimal (0x20, -0x10) or decimal plus ISO prefixes (examples: 1ki, 3k, 416Mi).

Structured parameters and lists

Many options of this program accept a single numeric parameter or a comma-separated list of numeric parameters (Example FLASH bank addresses). Other options take pairs with special syntax, for example "64@0x100" is in fact the tuple (64,0x100), only with a special syntax. And finally there are options that allow lists of tuples. In this manual page, the parameters that allow lists are followed by a comma and 2 dots, like this: *adress***,...** These lists consist of one or more elements. Likewise, lists of tuples are are written as one tuple followed by a comma and 2 dots, like this: *count***@***address***,..** . In the rare case where an empty list is allowed, a single comma , denotes the empty list.

Multiple FLASH segments

mxli supports scattering multiple FLASH segments within the FLASH.

Multiple input/output files.

To support multiple FLASH segments and binary files, mxli supports multiple input/output files. STDIN/STDOUT redirection is limited to single segments.

OPTIONS

mxli has roughly 5 different kinds of options: (a) actions, (b) communication parameters, (c) overrides, (d) microcontroller device definitions and (e) CRP options.

Actions

mxli defaults to invoking the device's ISP handler by applying a RESET/BOOT sequence, querying the device ID(s) and exit. To the user, this appears to be nothing, but in fact it was already checking the serial communication and knowing the microcontroller. The exit code of mxli is 0 only if these actions succeeded. Otherwise it prints an error message to STDOUT.

−n prints to STDOUT the name of the microcontroller and some information about its memory sizes.

−r *count*

Reads *count* bytes from FLASH and outputs to STDOUT. The read is performed in the default FLASH bank, offset from option -a.

−w Reads STDIN and writes these bytes to FLASH. NOT IMPLEMENTED, YET.

−e *bank*
−e *sector*
−e *sectorFrom..sectorTo*

Erases the specified sectors. Specifying an explicit FLASH bank *bank* erases that FLASH bank as a whole. The other versions of the erase command allow erasure of a single sector *sector* or a consecutive range of sectors *sectorFrom***,** *sectorFrom***+1,..** *sectorTo***-1,** *sectorTo* . *sector*, *sectorFrom*, *sectorTo* may be prefixed by a *bank* to explicitely specify the FLASH bank of the sector. FLASH bank A is used by default. The erase command is somewhat special from other commands in that it doesn't consider any addresses provided by an image file. That means, erase does not select a FLASH bank from the address provided by the image file or the **-a** arguments. This is, because you don't have to specify an erase command for writing an image - mxli takes care to erase the affected sectors in advance. As an example, if you want to erase sectors 3 to 5 of FLASH bank B, you have to specify -e B3..B5 .

−s *bank*

Sets the active FLASH bank to *bank*. This is supported for multi-FLASH-bank controllers only. The ISP command used for such controllers (e.q. LPC4300) clashes with the checksum command available on other controllers (e.g. LPC800), so don’t use it if it’s not needed.

−p Probes baud rates for communication and outputs successful tries to STDOUT. The output can be used as command-line parameters for mxli, i.e. the output is one set of communication parameter command-line options per line of output with the first line being the highest baud rate.

−i Prints detailed human-readable member info to STDOUT.

−k Prints the boot code version number (like: 13.4) to STDOUT.

−x Executes the code in FLASH by applying a RESET sequence as a final action of the program.

−Q mxli does not perform IO but quits before performing communication. This switch is included for querying mxli’s database quickly and even without an LPC controller connected. Commands requiring IO are ignored without error. Please note, that ’mxli -iQ’ is not a very intelligent invocation of mxli, but maybe ’mxli --deviceList -iQ’ or ’mxli -iQu LPC4357’ are.

−−deviceList
Prints out all compiled in devices’ names.

−−deviceDefinition
Prints the selected device’s command-line definition. This is useful if you want to derive your own device definition (for an unsupported device) and you want to start from an existing one known to mxli.

−−uid Prints out the unique device serial number, if supported by the device.

Communication parameters

−b *baud*
Sets the communication baud rate to *baud***.**

−c *frequency*
The ISP boot loader needs to know what crystal frequency is used. *frequency* is measured in Hz here.

−d *device*
This option changes the communication device to *device***.** The default is **/dev/ttyUSB0**.

−t *bootupTimeMs*
This option sets the time mxli waits after de-asserting RESET before it starts to communicate with the LPC. This time depends on your target board. The default is 300 (ms).

−E This option enables echoing of requests by the LPC. This reduces transmission speed but can improve reliability of communication for some devices, because this implements a simple flow control mechanism. The default is OFF.

−T *timeoutMs*
This option sets the maximum time mxli waits for an expected character/byte to arrive. After that time, mxli considers the communication line broken and terminates with an error. Don’t set this to small values without a real need. Some operations of the UART ISP handler are delayed by design (like flash erase) and the LPC does not respond until completion of the operation. The default value is 500 (ms).

Overrides

Override parameters are used to change existing microcontroller configurations easily. Most LPC microcontroller family members are quite similar and differ mainly in FLASH memory and SRAM size(s). This is, where override parameters come in handy. Override parameters are supposed to be used in conjuction with wildcard microcontroller names. Override parameters are lowercase single character options.

−f *flashSize*
: This option changes the size of one FLASH memory bank to *flashSize***.**

−m *ramSize***,..**
: Changes the size(s) of the RAM regions available in the microcontroller.

−u *name*
: Forces the use of the microcontroller definition of the device named *name* instead of the probed device. *name* may be quoted on the command line for those device names containing spaces.

−y *flashBank*
: Changes the FLASH bank to use as destination. This has the effect of adding the base address of FLASH bank *flashBank* to all addresses provided by the image file. Specifying **Z** as FLASH bank treats your addresses as absolute addresses. If unspecified, this option defaults to **A**.

Microcontroller device definitions

The possibility to define a new microcontroller device's programming parameters on-the-fly has been present from the very first version of mxli. Since NXP's LPC families keep growing fast you almost inevitably end up with a device on your table that is not yet supported by mxli's compiled-in device table. You may have a new member of a known family or even a member of an entirely new LPC family. In the first case, you most probably will be able to flash that device by forcing some known member of the family and using override options for FLASH size and/or RAM size. In the second case, you probably have to define the device's memories and IDs from scratch. Provided you're not after the very last bit of performance while flashing the device, you don't need to specify the most accurate description of the new device. It is sufficient to specify a minimum description suitable to perform flashing at reasonable speed. To be more precise: it will most probably do the job to specify the first RAM region of the device and the first FLASH bank and its sector layout, the checksum location and the ISP protocol. You can easily put the new device definition into the variable MXLI_PARAMETERS or write a wrapper (shell-)script for mxli that puts the definition on mxli's command line.

−A *flashBankAddress***,..**
: The LPC families LPC1800 and LPC4300 (at least) have multiple FLASH banks, numbered A,B,C.. . All FLASH banks share the same sector layout - which is defined by the -F option. Each element of the list provided to this option defines the base address *flashBankAddress* of one of the available FLASH banks. The number of the elements and their position in the list define the number of available FLASH banks as well as their numbering. As an example, LPC4357 FLASH banks A and B are defined as follows: -A 0x1A000000,0x1B000000 . Another interesting example might be the LPC4370 (FLASH-less part) definition: -A, with the single comma indicating the zero-elements list. If omitted, this option defaults to -A 0 (one bank at address 0).

-B *blockSize***,..**
: Defines the allowed 'Copy RAM to FLASH' transfer size(s). This size defines the granularity of FLASH write operations. Typically 4 different sizes are allowed for a given LPC family, most frequently -B 256,512,1024,4096. If unspecified, this option defaults to -B 1024.

−F *sectorSize***x***count***,..**
: The LPC FLASHs consist of multiple groups of sectors. All sectors of a given group of *count* sectors share the same sector size *sectorSize*. Different groups use different sector sizes. Each group of sectors is defined by one element of the list provided to this option. The tuple operator is the

letter 'x' with the meaning of 'times'. As an example (LPC17xx Family FLASH definition): -F 4096x16,32kix14 means: 16 sectors of 4kiB followed by 14 sectors of 32kiB.

−I *ID*,.. Every LPC controller can be identified by its device IDs. Unfortunately, one or two device IDs are used, depending on the family. **mxli** uses its compiled-in database to (try to) determine from the first ID whether or not a second ID is supposed to be read. Of course, the entry provided with this option is considered by **mxli**, too. Each ID is defined by one element of the list of IDs provided to this options. However, there's more to IDs than exact matches. Some parts of the IDs have to be disregarded. So in fact, the values provided to this option are values and bit masks. The values can be written in hexadecimal 0x... or binary 0b.... and every upper-case 'X' means don't care (nibble or bit respectively). Furthermore leading Xes may be omitted, but NOT leading zeros. Decimal values are permitted, too, but these do not allow don't care bits. As an example the LPC4353 ID definition looks like this: -I0xA001C830,0xXXXXXX44 . Or else: -I0xA001C830,0x44

−M *size*@*baseAddress*,..
LPC microcontrollers can have up to 6 on-chip RAM regions (at least). Not every RAM region is powered up at ISP handler time. Please provide only those RAM regions, that are functional at that time. Every RAM region has a base address *baseAddress* and a size *size*. Each RAM region is defined by one element of the list provided to this option. The tuple operator is the @-sign with the meaning of 'at base address'. As an example (LPC810 RAM definition) -M 1024@0x10000000, means: one RAM region starting at 0x10000000 with a total size of 1kiB. The trailing comma is not a typo, nor is it required - it's as optional as commas are in 'C' initializer lists.

−N *name*
This option specifies the name of the LPC controller. There's no limitation on characters used.

−P UUENCODE
−P BINARY
This option selects the data transmission protocol used by the ISP handler. To my knowledge this is always **UUENCODE** with the single exceptions of the LPC800 family that uses the **BINARY** transmission protocol. Default: UUENCODE .

−R *size*@*address*,..
The ISP handler needs some RAM regions for its own purposes. This RAM - of course - cannot be used by **mxli** for temporary storage. Every such RAM region has a size *size* and a base address *address*. For a positive value of *size* the area used by the ISP handler is *address*,*address+1*,..*address+size-1* . If *size* is negative, then *address* must match one of the RAM regions defined by the -M option and the area used by the ISP handler is counted from the top of the RAM region downwards (like a stack usage, coincidently :) . Each RAM region used by the ISP handler is defined by one element of the list provided to this option. As an example, LPC213x ISP handlers required the following definition -R 0xE0@0x40000120,-0x120@0x40000000, meaning: ISP handler uses RAM from 0x40000120 (including) up to 0x40000200 (excluding) and the top 32+256 bytes of the same RAM region, which depends on the total RAM available. If unspecified, this option defaults to 1kiB from bottom of first RAM region and 1kiB from top of it.

−S *count*@*dstIndex*
This option defines that the first *count* entries of the vector table within a FLASH bank are used for calculating a checksum and that the destination place for storing the negated checksum is *dstIndex*. The sum is calculated from 32-bit accesses to the addresses 0 .. (*count*-1)*4 (including). However, if *dstIndex* lies inside the checked range, then it is excluded from the calculation. The ISP boot loader uses such a checksum as critereon for valid code in the FLASH. Typically the

ISP boot loader checks the first 8 vectors. On ARM7 LPCs the (negated) checksum is put into the unused vector entry index 5. On Cortex-M processors, vector entry index 7 is used for the (negated) checksum, typically. As an example the LPC800 checksum calculation is defined by -S8@7 while LPC2100 checksum calculation -S8@5 is used.

CRP options

CRP (code read protection) is a feature of LPC controllers to restrict read access to the FLASH memory in order to protect the intellectual property of the firmware creator or to change the possible use of some ISP-related controller pins. While it it unlikely, that someone accidentally ends up in an unwanted CRP level it is pretty likely that someone using CRP ends up in an unwanted level of CRP. mxli tries to minimize the chance of locking up an LPC controller unintentionally. Therefore mxli supports downgrading of CRP levels, which can be used to verify the firmware to work properly before enabling the final CRP level. Furthermore, mxli's default maximum CRP level is 0 - NO CRP - and mxli verifies every image for that. If you intend to use CRP you really have to consult the LPC's user manual about that. CRP is controlled by a single 4-byte word at (typically) address 0x2FC inside the FLASH bank. mxli checks for CRP in every FLASH bank it writes. Therefore, any image containing that address contains at least one image CRP level (most probably CRP level 0). The user may wish to use that level or the user may wish to set a level different from this. CRP level 3 and NO_ISP mode (CRP level 4 in mxli's numbering) effectively disable ISP entry by activating the /BOOT pin at reset. To enable these CRP levels, you need both options, -l and -L and the arguments provided must match each other and the image CRP level or mxli will refuse to apply the CRP level. An LPC put into CRP level 3 or 4 can only be re-programmed if the program inside the FLASH invokes the ISP handler or erases the FLASH.

−L *allowLevel*
This sets the maximum allowed CRP level to *allowLevel*. The default is 0 - CRP disabled.

WARNING: Do not provide an *allowLevel* greater than 2 **unless you are absolutely sure about what you're doing**.

−l *setLevel*
This requests to use CRP level *setLevel* instead of the CRP level of the image. The level must be allowed before. This option defaults to -1 (unset) and this translates into: if the image CRP level is 0 or withing the allowed range of CRP levels and CRP level <=2 then keep the CRP word.

−−crpAddress=*crpAddress*
This changes the address of the CRP word relative to the FLASH bank origin. This option is an override option and its purpose is mainly for testing mxli's CRP mechanisms. Default: 0x2FC. Beware: changing this value effectively disables mxli's CRP handling.

Other options

−? Prints a short help.

−a *address***,..**
This changes the FLASH-bank relative target address of the FLASH image. *address***,..** may be one address for one image segment or a list of addresses for multiple image files. This option applies to both bin-files and hex-files. For hex-files, one address applies to one hex-file, not one address to one segment of a hex-file. If unspecified, the address defaults to 0 for all image files and hence image segments.

−g Enables maximum debugging output.

−v Verbose: print progress to stderr.

−G *level*
Sets the debug level to values between -1 (silent), 0 (normal), 1 (progress: -v), 2 (info: -V) 3 (debug: -g).

−−version
Prints mxli’s version number as mxli-m.n with m an n as natural numbers (of 1 or 2 digits), like for example: mxli-3.0 .

USING mxli

mxli can only communicate with the LPC controller if that one is in ISP mode. In order to enter ISP mode the current program in FLASH must invoke the IAP call ’reinvokeISP’ or you must hand over control to the ISP code at boot-up. The ISP mode becomes active if either no valid FLASH image is present or if device specific pins are pulled low. For example a LPC812 with PIO0_12 (called /BOOT) pulled LOW at RESET enters ISP mode. Many development boards allow control of the signals /RESET and /BOOT over the serial port and probably you have to set jumpers for this to work. **mxli** works best with serial port control of /RESET and /BOOT. It uses DTR as /RESET signal and RTS as /BOOT. Some development boards don’t provide control over these signal or provide one of them only. These boards typically have buttons to enable /RESET or /BOOT. To enter ISP mode press and hold /BOOT then press and release /RESET and finally release /RESET.

It should be noted, that that code read protection can prevent entering ISP mode through /RESET and /BOOT.

ENVIRONMENT

mxli puts the contents of the environment variable **MXLI_PARAMETERS** in front of the command line parameters as if it was typed on the command line. This allows putting the user’s favourite options into login script. Typical examples are: the option -v (verbose), changing the baud rate (-b 230400) or communication device (-d /dev/ttyUSB1).

FILES

mxli does not use any configuration file(s) so far.

EXAMPLES

The first invokation of mxli I would suggest is:

```
$ mxli -n
```

This identifies the microcontroller connected, if everything works fine. If that doesn’t work then your RESET/BSL logic/settings don’t work or you have to choose a communication device other than the default /dev/ttyUSB0. Look at your board’s schematic or manual how to put the correct jumper settings or what button to press for entering ISP mode.

Once you got basic communication running, you can flash a known device like this (assuming, you have to use /dev/ttyUSB1):

```
$ mxli -d/dev/ttyUSB1 yourProgram.hex
```

Once this works, you might want to get better performance. Use the switch **−q** and raise the baud rate, if possible:

```
$ mxli -q -b 230400 -d /dev/ttyUSB1
```

OK. Now let’s read back the image. You have to specify the amount of data to read, let’s assume 4kiB. And let’s assume the communication device is the default one.

```
$ mxli -r 4ki > image.bin
```

Oh, yes, mxli writes binary images only. That could be improved...

How does a command line look like, that defines a rather complex microcontroller, let’s say LPC4357, and puts an image into flash bank selected by the hex-image’s (first) destination address and enables that flash bank ?

```
$ mxli -N LPC4357 -F8kix8,64kix7 -B1024,4096 -A0x1A000000,0x1B000000
-M32ki@0x10000000,40ki@0x10080000 -I0xA001C830,0xXXXXXX00 \
  -R0x200@0x1008000,-288@0x10000000 -S8@7 -yZ yourProgram.hex
```

Translation: Name of the device is LPC4357. Every FLASH bank consists of 8 sectors of 8kiB, followed by 7 sectors of size 64kiB. FLASH may be accessed in 1024byte or 4096byte units. There are two FLASH banks, the first at address 0x1A000000, the second at address 0x1B000000. LPC4357 has 32kiB RAM at address 0x10000000 and 40kiB RAM at address 0x10080000. The ISP handler uses the RAM area 0x10080000..0x100801FF and the top 256+32 bytes of this 40kiB RAM area. The device ID of LPC4357 is two IDs in reality, namely 0xA001C830 and the lower 8 bits of the second ID must be 0. LPC4357 valid code critereon is: sum of the first 8 vector table entries is zero. mxli uses table entry 7 for adjusting the checksum. Selection of (virtual) FLASH bank Z causes the hex file's addresses to be treated as absolute values, rather than FLASH-bank (A as default) relative values.

SUPPORTED DEVICES

mxli supports (by compiled-in tables) the following families of NXP's portfolio: LPC800, LPC1100, LPC1200, LPC1300, LPC1500, LPC1700, LPC2100, LPC2300, LPC4300. Other families should be supported by mxli's command-line device definition feature.

BUGS

The source code of mxli can easily be ported to closed source operating systems like M$ Widows or McO$.

AVAILABILITY

mxli is currently available for Linux / *nix Platforms for compilation with GNU C Compiler (gcc). mxli is written in gnu99 dialect (-std=gnu99) and compilation tested with gcc-4.7.3. mxli-3 can be ported to free-standing environments to implement hand-held UART ISP FLASH programmers. The latest version of mxli can be downloaded from www.windscooting.com/softy/mxli.html .

AUTHORS

Copyright Marc Prager (marc @ windscooting.com), 2011-2014

SEE ALSO

lpcprog of lpctools, lpc21isp