Flash Magic
GUI and Command Line Manual

Manual Revision 1.70

EMBEDDED SYSTEMS ACADEMY

For support contact support@esacademy.com

For the latest news on Flash Magic visit us at

www.esacademy.com/software/flashmagic

EMBEDDED SYSTEMS ACADEMY

Embedded Systems Academy provides training and consulting services, specializing in CAN, CANopen and Embedded Internetworking. For more information visit

www.esacademy.com

Contents

About This Manual

This manual follows some set conventions with the aim of making it easier to read. The following conventions are used:

0x	Hexadecimal (base 16) values are prefixed with “0x”.
italictext	Replace the text with the item it represents
[ ]	Items inside [and] are optional
a | b	a OR b may be used
…	One or more items may go here.

Chapter 1 - Introduction

NXP Semiconductors produce a range of Microcontrollers that feature both on-chip Flash memory and the ability to be reprogrammed using In-System Programming technology. Flash Magic is Windows software from the Embedded Systems Academy that allows easy access to all the ISP features provided by the devices. These features include:

- Erasing the Flash memory (individual blocks or the whole device)
- Programming the Flash memory
- Modifying the Boot Vector and Status Byte
- Reading Flash memory
- Performing a blank check on a section of Flash memory
- Reading the signature bytes
- Reading and writing the security bits
- Direct load of a new baud rate (high speed communications)
- Sending commands to place device in Bootloader mode

Flash Magic provides a clear and simple user interface to these features and more as described in the following sections.

Under Windows, only one application may have access the COM Port at any one time, preventing other applications from using the COM Port. Flash Magic only obtains access to the selected COM Port when ISP operations are being performed. This means that other applications that need to use the COM Port, such as debugging tools, may be used while Flash Magic is loaded.

Note that in this manual third party Compilers are listed alphabetically. No preferences are indicated or implied.

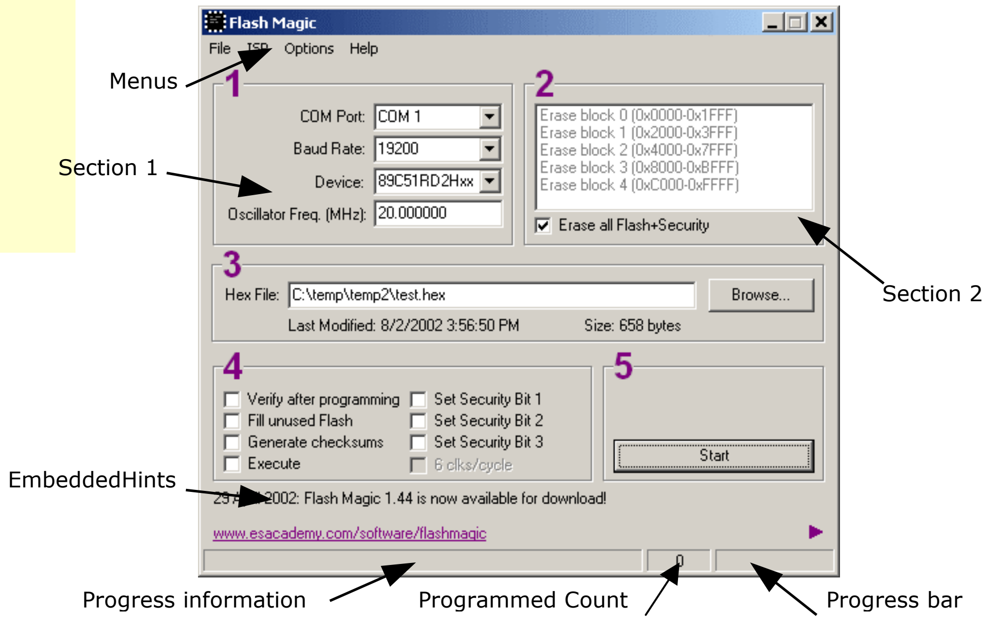
Chapter 2 - Minimum Requirements

- Windows NT/2000/XP/Vista
- Mouse
- COM Port or Ethernet interface
- 16Mb RAM
- 10Mb Disk Space

Chapter 3 - User Interface Tour

3.1 Main Window

The following is a screenshot of the main Flash Magic window. The apperance may differ slightly depending on the device selected.



The window is divided up into five sections. Work your way from section 1 to section 5 to program a device using the most common functions. Each section is described in detail in the following sections.

At the very bottom left of the window is an area where progress messages will be displayed and at the very bottom right is where the progress bar is displayed. In between the messages and the progress bar is a count of the number of times the currently selected hex file has been programmed since it was last modified or selected.

Just above the progress information EmbeddedHints are displayed. These are rotating Internet links that you can click on to go to a web page using your default browser. If you wish to quickly flick through all the hints then you can click on the fast forward button:

3.2 Menus

There are five menus, File, ISP, Options, Tools and Help.

The File menu provides access to loading and saving Hex Files, loading and saving settings files and exiting the application.

The ISP menu provides access to the less commonly used ISP features.

The Options menu allows access to the advanced options and includes an item to reset all options.

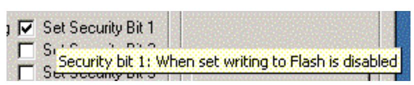
The Tools menu provides features that support the operation and use of Flash Magic.

The Help menu contains items that link directly to useful web pages and also open the Help About window showing the version number.

The Loading and Saving of Hex Files and the other ISP features are described in the following sections.

3.3 Tooltips

Throughout the Flash Magic user interface extensive use has been made of tooltips. These are small text boxes that appear when you place the pointer over something and keep it still for a second or two.



Note that tooltips do not appear for items that are disabled (grayed out).

3.4 Saving Options

The options in the main window and the Advanced Options window are automatically saved to the registry whenever Flash Magic is closed. This removes the need for an explicit save operation. When Flash Magic is restarted the main window and the Advanced Options window will appear as you left it, so you do not have to repeatedly make the same selections every time you start the application.

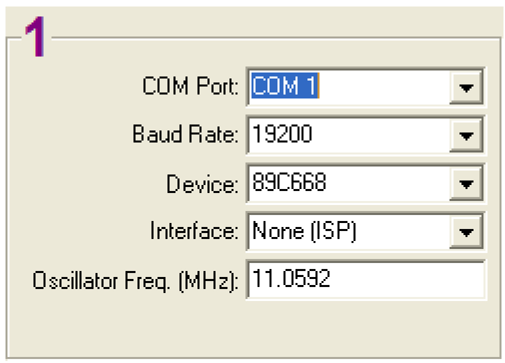
If you wish to reset the options to the original defaults then choose Reset from the Options menu.

Chapter 4 - Five Step Programming

For each step there is a corresponding section in the main window as described in the User Interface Tour.

4.1 Step 1 – Connection Settings

Before the device can be used the settings required to make a connection must be specified.



COM Port Settings

Select the desired COM port from the drop down list or type the desired COM port directly into the box. If you enter the COM port yourself then you must enter it in one of the following formats:

- COM n
- n

Any other format will generate an error. So if you want to use COM 5 (which is not present on the drop down list) you can directly type in either "COM 5" or "5".

Select the baud rate to connect at. Try a low speed first. The maximum speed that can be used depends on the crystal frequency on your hardware. You can try connecting at higher and higher speeds until connections fail. Then you have found the highest baud rate to connect at.
Alternatively, some devices support high speed communications. Please refer to the High Speed Communications section for information.

Select the device being used from the drop down list. Ensure you select the correct one as different devices have different feature sets and different methods of setting up the serial communications.

Select the interface being used, if any. An interface is a device that connects between your PC and the target hardware. If you simply have a serial cable or USB to serial cable connecting your COM port to the target hardware, then choose "None (ISP)". Choosing the correct interface will automatically configure Flash Magic for that interface, along with enabling and disabling the relevant features.

Enter the oscillator frequency used on the hardware. Do not round the frequency, instead enter it as precisely as possible. Some devices do not require the oscillator frequency to be entered, so this field will not be displayed.

Once the options are set ensure the device is running the on-chip Bootloader if you are using a manual ISP entry method.
Note that the connection settings affect all ISP features provided by Flash Magic.

Ethernet Settings

Selecting a device with "Ethernet" at the end of the name indicates that the Ethernet bootloader should be used. The Ethernet options will be displayed.

Enter an IP address for the bootloader. This must be an address on the same subnet as the PC running Flash Magic and it must not be already in use. The bootloader will be assigned this IP address while Flash Magic is accessing the device.

The IP address must have the format "a.b.c.d" where a, b, c and d are decimal values from zero to 255.

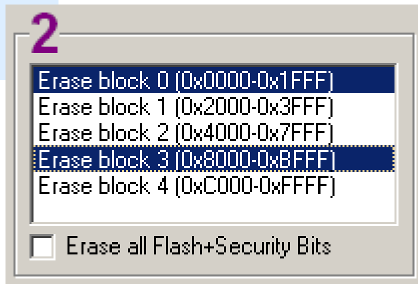
Enter the MAC address of the bootloader. Typically the MAC address is hard-coded in the bootloader.

The MAC address must have the format "aa-bb-cc-dd-ee-ff" where aa, bb, cc, dd, ee and ff are two digit hexadecimal values.

Note that the connection settings affect all ISP features provided by Flash Magic.

4.2 Step 2 – Erasing

This step is optional, however if you attempt to program the device without first erasing at least one Flash block, then Flash Magic will warn you and ask you if you are sure you want to program the device.



Select each Flash block that you wish to erase by clicking on its name. If you wish to erase all the Flash then check that option. If you check to erase a Flash block and all the Flash then the Flash block will not be individually erased. If you wish to erase only the Flash blocks used by the hex file you are going to select, then check that option.

For most devices erasing all the Flash also results in the Boot Vector and Status Byte being set to default values, which ensure that the Bootloader will be executed on reset, regardless of the state of the PSEN pin or other hardware requirements. Only when programming a Hex File has been completed will the Status Byte be set to 00H to allow the code to execute. This is a safeguard against accidentally attempting to execute when the Flash is erased. On some devices erasing all the Flash will also erase the security bits. This will be indicated by the text next to the Erase all Flash option. On some devices erasing all the Flash will also erase the speed setting of the device (the number of clocks per cycle) setting it back to the default. This will be indicated by the text next to the Erase all Flash option.

4.3 Step 3 – Selecting the Hex File

This step is optional. If you do not wish to program a Hex File then do not select one.

You can either enter a path name in the text box or click on the Browse button to select a Hex File by browsing to it.
Also you can choose Open… from the File menu.

Note that the Hex file is not loaded or cached in any way. This means that if the Hex File is modified, you do not have to reselect it in Flash Magic. Every time the Hex File is programmed it is first re-read from the location specified in the main window.

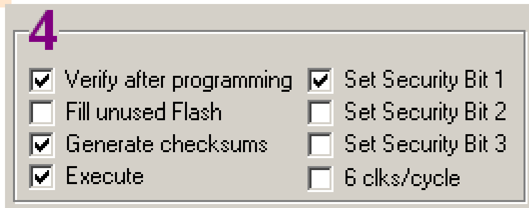
The date the Hex file was last modified is shown in this section. This information is updated whenever the hex file is modified. The hex file does not need to be reselected.

Clicking on more info or choosing Information… from the File menu will display additional information about the Hex file. The information includes the range of Flash memory used by the Hex file, the number of bytes of Flash memory used and the percentage of the currently selected device that will be filled by programming the Hex file.

If the device supports programming and execution from RAM, for example the ARM devices, then the hex file may contain records for the RAM. First the flash will be programmed followed by the RAM. Programs loaded into RAM via a hex file may be executed using such features as the Go option. See chapter 6 for more details.

4.4 Step 4 – Options

Flash Magic provides various options that may be used after the Hex File has been programmed.



This section is optional, however Verify After Programming, Fill Unused Flash and Gen Block Checksums may only be used if a Hex File is selected (and therefore being programmed), as they all need to know either the Hex File contents or memory locations used by the Hex File.

Checking the Verify After Programming option will result in the data contained in the Hex File being read back from Flash and compared with the Hex File after programming. This helps to ensure that the Hex File was correctly programmed. If the device does not support verifying then this item will be disabled.

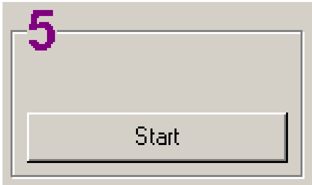
Checking the Fill Unused Flash option will result in every memory location not used by the Hex File being programmed with the value that sets all the bits to a programmed state. Once a location has been programmed with this feature it cannot be reprogrammed with any other value, preventing someone from programming the device with a small program to read out the contents of Flash or altering the application's operation.

Checking the Gen Block Checksums option will instruct Flash Magic to program the highest location in every Flash block used by the Hex File with a special "checksum adjuster value". This value ensures that when a checksum is calculated for the whole Flash Block it will equal 55H, providing the contents of the Flash block have not be altered or corrupted. Please refer to the Block Checksums section for more information.

Checking the Execute option will cause the downloaded firmware to be executed automatically after the programming is complete. Note that this will not work if using the Hardware Reset option or a device that does not support this feature.

4.5 Step 5 – Performing the Operations

Step 5 contains a Start button.



Clicking the Start button will result in all the selected operations in the main window taking place. They will be in order:

- Erasing Flash
- Programming the Hex File
- Verifying the Hex File
- Filling Unused Flash
- Generating Checksums
- Programming the clocks bit
- Programming the Security Bits
- Executing the firmware

Once started progress information and a progress bar will be displayed at the bottom of the main window. In addition the Start button will change to a cancel button. Click on the cancel button to cancel the operation.
Note that if you cancel during erasing all the Flash, it may take a few seconds before the operation is cancelled.

Once the operations have finished the progress information will briefly show the message "Finished…". The Programmed Count shown next to the progress bar will increment. This shows the total number of times the hex file has been programmed. Modifying the hex file or selecting another hex file will reset the count. Alternatively, right-clicking over the count provides a menu with the option to immediately reset the count.

Chapter 5 – Block Checksum Generation

5.1 Introduction

Often it is desirable for an application to be able to verify that the code about to be executed has not been altered or corrupted. Attempting to execute altered or corrupted code will result in undefined or undesirable behavior. This could translate to erratic signals appearing on the I/O pins resulting in damage to hardware.

Flash Magic allows the easy generation of checksums for each Flash block that a programmed Hex file is stored in. Flash Magic will write a value to the highest location in each block to ensure that when a checksum is calculated over the whole block using a specific method, the checksum will be 55H. The checksum calculation is:

0xFF – (sum_of_all_bytes_in_block_truncated_to_8_bits) + 1

This is a method that is easy for an 8-bit microcontroller to perform.

Note that on devices such as the 89LPC9xx, which do not support the reading of Flash memory for security reasons, the checksum algorithm assumes the portions of memory not used by the hex file will be blank. Therefore if other hex files have been programmed into the device the checksum will be incorrect. As an alternative the 89LPC9xx offers a 32-bit CRC algorithm that may be read from the device then compared at run-time.

The LPC2xxx devices do not support this feature.

5.2 Checksum Demonstration Project

Included in the Flash Magic installation is a folder called ChecksumDemo that contains an example project using a pre-written checksum calculation routine that may be used in your own applications. The 8051 folder contains examples for the Keil and Raisonance 8051 C Compilers. The XA folder contains examples for the Tasking and Raisonance XA C Compilers.

The files in the 8051 projects are:

File	Description
Main.c	Application source file containing the main() function
Checksum.c	Source file containing the checksum calculation function
Checksum.h	Header file for the checksum source file
Kchecksumdemo.uv2	Keil uVision example project file
Rchecksumdemo.prj	Raisonance RIDE example project file
Kchecksumdemo.hex	Keil generated Hex File
Rchecksumdemo.hex	Raisonance generated Hex File

The files in the XA projects are:

File	Description
Main.c	Application source file containing the main() function
Checksum.c	Source file containing the checksum calculation function
Checksum.h	Header file for the checksum source file
Rchecksumdemo.prj	Raisonance RIDE example project file
Tchecksumdemo.pjt	Tasking example project file
Rchecksumdemo.hex	Raisonance generated Hex File
Tchecksumdemo.hex	Tasking generated Hex File

The example is written for the Phytec phyCORE development boards and turns on both LEDs D1 and D2 if Flash block 0 has the correct checksum.

Checksum.c and checksum.h are not specific to any particular hardware and may be used in your own applications.

5.3 Using the Example Checksum Code

The function calc_checksum has the following prototype:

```
unsigned char calc_checksum(unsigned char block);
```

Passed: 0 to check Flash block 0 (0000H – 1FFFH)
1 to check Flash block 1 (2000H – 3FFFH)
2 to check Flash block 2 (4000H – 7FFFH)
3 to check Flash block 3 (8000H – BFFFH)
4 to check Flash block 4 (C000H – FFFFH)

Returns: 1 if the block is valid (has the correct checksum)
0 if the block is invalid (it has been altered or corrupted)

It is essential that the highest location in each block is reserved for the “checksum adjuster value” and is not used by the application. It is easy to reserve individual locations in both the Keil, Raisonance and Tasking Compilers, however for convenience the following macros have been defined. Simply use one of the macros somewhere in the project for each of the Flash blocks used by the application. It will be replaced with the necessary line of code to reserve that location:

```
RESERVE_BLOCK0_CHECKSUM
RESERVE_BLOCK1_CHECKSUM
RESERVE_BLOCK2_CHECKSUM
RESERVE_BLOCK3_CHECKSUM
RESERVE_BLOCK4_CHECKSUM
```

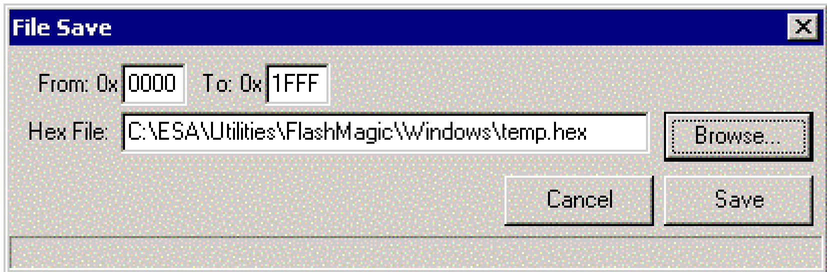
The macros are defined in checksum.h.

If using In-Application Programming be sure not to modify any location in a Flash block where a checksum is being used, otherwise calc_checksum will return indicating the block is invalid.

Chapter 6 - Additional ISP Features

6.1 Saving a Hex File

Once connected choosing Save As… from the File menu opens the Hex File save window.



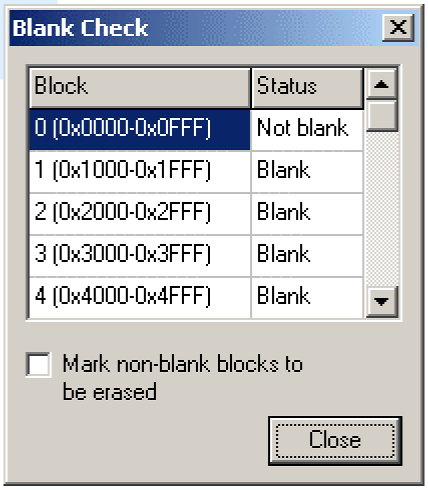
It is possible to save a section of Flash memory to a Hex File.
Enter the start address and the end address (inclusive) that you wish to save. Note that the start and end addresses do not have to correspond to a Flash block.
If desired you can save a single memory location by entering it as both the start and end address.

Next select the location and name of the Hex file to save by either entering a path in the text box or clicking on the Browse button and browsing to a folder.

Finally, click on the Save button to begin the save. Progress information on the save will be shown at the bottom of the window.

6.2 Blank Check

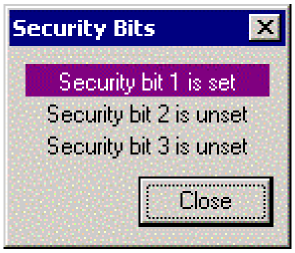
Choose Bank Check… from the ISP menu to perform a blank check on all the Flash blocks present on the device. Once complete the Blank Check window will look something like the following:



The status column indicates if a block is blank or not.
Checking the option to Mark non-blank blocks to be erased will have the effect of checking the relevant items in the Erase section of the main window. For example if Block 0 is not blank, then checking the Mark option will result in Block 0 in the Erase section of the main window being checked.

6.3 Reading the Security Settings

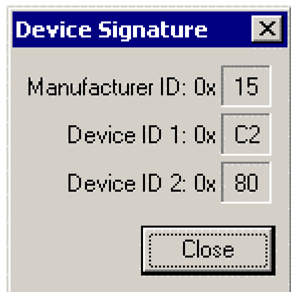
When Flash Magic is first started it will attempt to read the security bits of any device that is connected to the specified COM Port. If no device is connected then you may connect a device and choose Read Security from the ISP menu. The security settings will be read and the following window opened:



If a security bit is set then it will be highlighted.

6.4 Reading the Device Signature

The device signature is comprised of two or three bytes that identify the device or a single 32-bit value. To read the device signature choose Read Device Signature from the ISP menu. The signature will be read and the following window will open showing the three bytes:



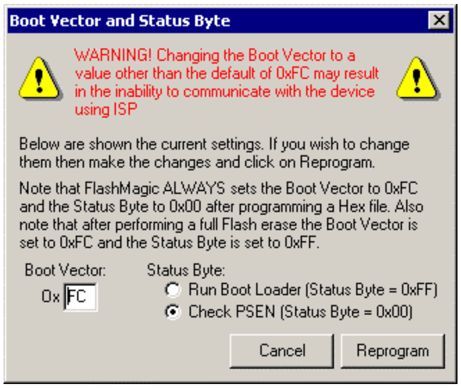
A manufacturer ID of 15H corresponds to NXP Semiconductors.

Some devices will also display the bootloader version in this window.

6.5 Modifying the Boot Vector and Status Byte

The Status Byte indicates how the device will operate after a reset. A value of 00H will result in the device checking the PSEN pin to determine whether it should run the Bootloader or the user application. Any other value will result in the device running the Bootloader.
The Boot Vector contains the page that the Bootloader (or user bootloader) starts at. The default varies depending on the device but as an example FCH corresponds to the address FC00H for 8051 non-LPC devices, and F8H, corresponds to the address F800H for the XA devices.

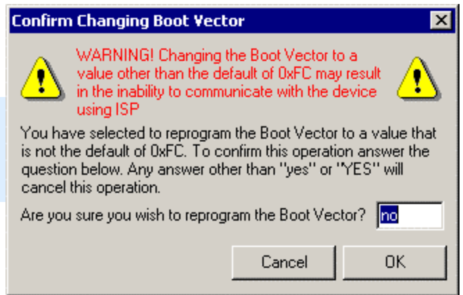
Once programming is completed it is possible to change both the Boot Vector and Status Byte by selecting Boot Vector and Status Byte from the ISP menu. You will be presented with the following window:



The current Boot Vector and Status Byte settings will be shown. Modify the settings as desired and click on the Reprogram button.

Setting the Boot Vector to a value other than the default may result in the inability to run the Bootloader on the device. This will mean that the ISP features of the device cannot be accessed until the device has been erased in a Parallel Programmer.

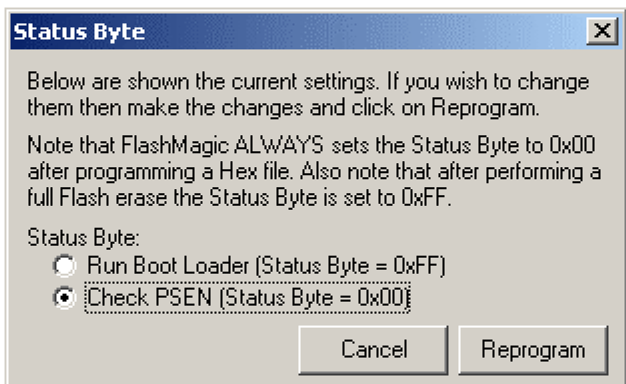
Because of this danger, you will be presented with a confirmation window if you try to reprogram the Boot Vector to a non-default value:



Following the instructions in the window carefully to reprogram the Boot Vector.

Once the Boot Vector and Status Byte have been reprogrammed a confirmation message will be displayed.

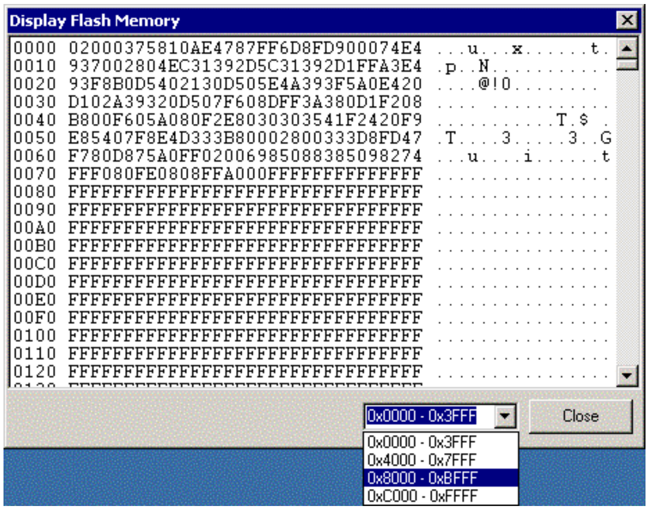
Some devices do not support a Boot Vector. However, the Status Byte may still be changed by selecting Status Byte from the ISP menu. The following dialog window will be displayed:



Modify the settings as desired and click on the Reprogram button.

6.6 Displaying Memory

A useful feature is the ability to view the contents of memory. Choosing Display Memory from the ISP menu accesses this feature.
Memory is shown one block at a time. The block being displayed may be selected from the drop-down list at the bottom of the Display Memory window:

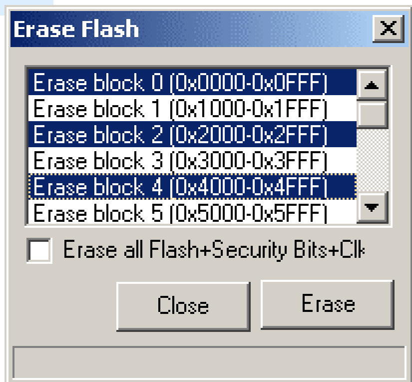


Each line shows 16 bytes of data starting at the address given at the start of the line. If a line contains question marks then data has not yet been read in for those locations. The memory is read in in the background allowing viewing of the memory that has been loaded in without having to wait for all of it to be read.
At any time a new range may be selected or the window closed.

Each line shows the 16 bytes of data first in hexadecimal format then in ASCII format. For characters that are not printable a period is displayed instead.

6.7 Erasing Flash

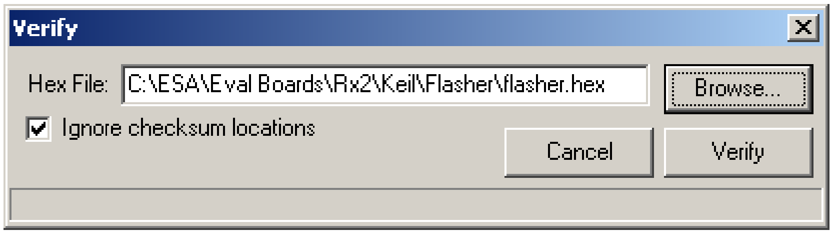
There are two ways Flash can be erased. Either as part of the five step process described earlier in this manual, or without performing any other ISP operations by choosing Erase Flash from the ISP menu.
The Erase Flash window will be opened. Its operation is identical to the erase section in the main window. Once settings have been chosen click on the Erase button to erase.



Progress information is displayed at the bottom of the window.

6.8 Verifying a Hex File

A Hex File can be verified without programming it into the device first. To access the verify feature choose Verify… from the ISP menu. You will be presented with the following dialog window:



Select a Hex File to verify either by entering the path to the file of clicking on the Browse button and choosing it.
If checksums were used when the device was programmed then check the option to ignore checksum locations, as they will be different in memory to any reserved checksum locations stored in the Hex file.
Click on the Verify button to start verification. Progress information is shown at the bottom of the window.

6.9 Reset and Execute

Selecting the Reset item on the ISP menu will cause a reset command to be sent to the device. Depending on the hardware and the status byte, the device with either reset and execute code or reset to the Bootloader.
If the Reset command is sent after successfully programming the device then the reset will execute the downloaded code. If the Reset command is sent after erasing the device then the device will reset to the Bootloader.

Selecting the Execute item on the ISP menu will cause Flash Magic to program the Boot Vector to the default and the Status Byte to zero (for devices that have this feature), followed by sending a reset command to the device. This will force any downloaded code to be executed.

6.10 Start Bootloader

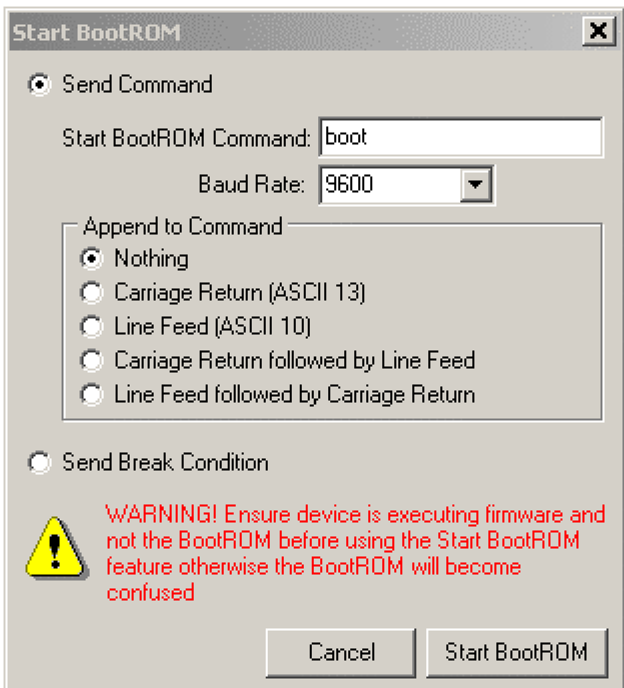
Start Bootloader Description

The Start Bootloader feature allows a textual command or break condition to be sent to the device to place it into Bootloader mode. In order for this to work however it must be supported by the user's application or the device.

For textual commands the user's application must watch the UART for the command to be received. Once received the application must echo back the command and a single full stop, then use In-Application Programming to program the Boot Vector to the default and the Status Bit/Byte to non-zero, and then reset the device. Once the device completes the reset the Bootloader will be executed, allowing the device to be reprogrammed.

If a break condition is sent, then Flash Magic does not expect the break condition to be echoed back and assumes the device has been placed in Bootloader mode.

Choosing "Start Bootloader…" from the ISP menu accesses the Start Bootloader feature. The following dialog window will be displayed.



Select to either send a textual command or a break condition.

The baud rate used is completely independent from the baud rate in the main window. This allows the user's application to use the UART at say 9600 baud, but allow Flash Magic to use say 19200 baud either with or without high-speed communications turned on. However the COM Port used is the same as the one selected in the main window.

Select the baud rate, enter the textual command (which may be anything you desire), select the append option you want and click on Start Bootloader to send. If successful the ISP features of Flash Magic can now be used.
The append options allow a carriage return and line feed to be added to the command. This is useful if your application implements a command line via the serial port and expects all commands to end in these control characters. Select the setting that matches your implementation.

Combined with the automatic Reset after programming option, devices may be programmed and tested repeatedly without ever having to touch the hardware.

An example project that accepts the command “boot” at 9600 baud is supplied with Flash Magic.

For command line access to this feature see the STARTBOOTLOADER directive.

Start Bootloader Demonstration Project

Included in the Flash Magic installation is a folder called StartBootROMDemo that contains an example project that accepts the command "boot" at 9600 baud. The 8051 folder contains examples for the Keil and Raisonance 8051 C Compilers. Currently there is no XA example, but this one can be adapted. For an LPC2xxx example, see NXP Application Note AN10356 – "Entering ISP mode from user code".

The files for the 8051 projects are:

File	Description
Main.c	Application source file containing the main() function
Rx2iaplib.a51	In Application Programming library from www.esacademy.com
Rx2iaplib.h	Header file for In Application Programming library
Reg51rx2.h	Header file for the Rx2 device
Kstartbootrom.uv2	Keil uVision example project file
Rstartbootrom.prj	Raisonance RIDE example project file
Kstartbootrom.hex	Keil generated Hex File
Rstartbootrom.hex	Raisonance generated Hex File

Advanced Features

Special characters may be inserted into the command to provide additional functionality. They consist of a special character followed by two hexadecimal characters and have the following meanings:

%HH - transmit the character HH, where HH is the hexadecimal value of the character
$HH - delay for HH x 100ms, where HH is a hexadecimal value
&AA - command. The function of the command depends upon the hexadecimal value AA and currently can be one of the following:

00 - flush RX buffer
01 - echo on
02 - echo off
03 – send break condition
04 – don't wait for a response

By default at the start of a command the echo is on.
Regardless of whether the echo is on or off, when the end of a command is reached the device must return a '.' to indicate the command was successfully received. If &04 is included anywhere in the command then the device does not need to return a response.

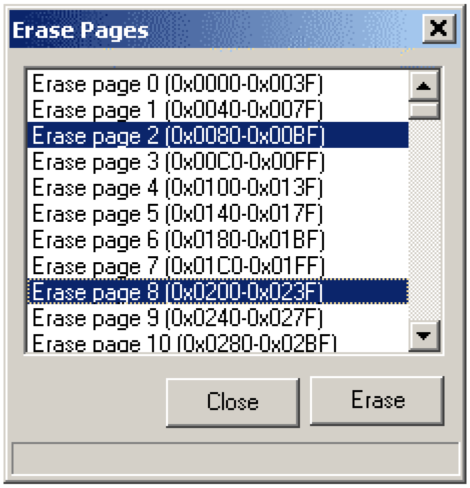
Examples:

%0D%02 Transmits a carriage return followed by an STX
%80 Transmits 80H
$14 Delays for 20 x 100ms = 2 seconds

$0A	Delays for 10 x 100ms = 1 second
&00	Flushes the RX buffer
A&02B&01C	Transmits 'A', waits for 'A' to be echoed, transmits 'B' then transmits 'C' and waits for 'C' to be echoed.
%0D$14&00A	Transmits a carriage return, waits for 2 seconds, flushes the RX buffer and transmits 'A'.

6.11 Erase Pages

Some devices allow the flash to be erased in pages. If this feature is supported by the device then the Erase Flash Pages… item on the ISP menu will be enabled. The dialog window is similar to the Erase dialog window:

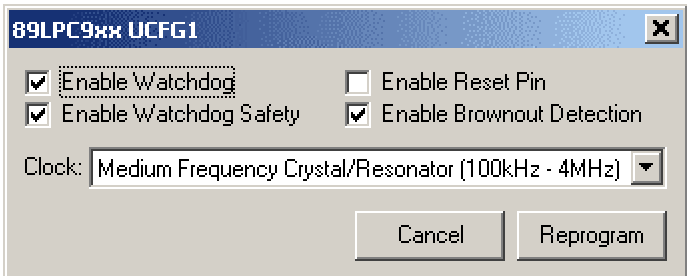


Select the pages to erase and click on the Erase button to erase them.

6.12 Device Configuration

Some devices allow configuration via ISP. If the selected device supports this feature then the Device Configuration… item on the ISP menu will be enabled. When selected the current configuration will be read from the device and displayed in the configuration dialog window. The configuration may then be changed and by clicking on Reprogram the new configuration will be programmed into the device.

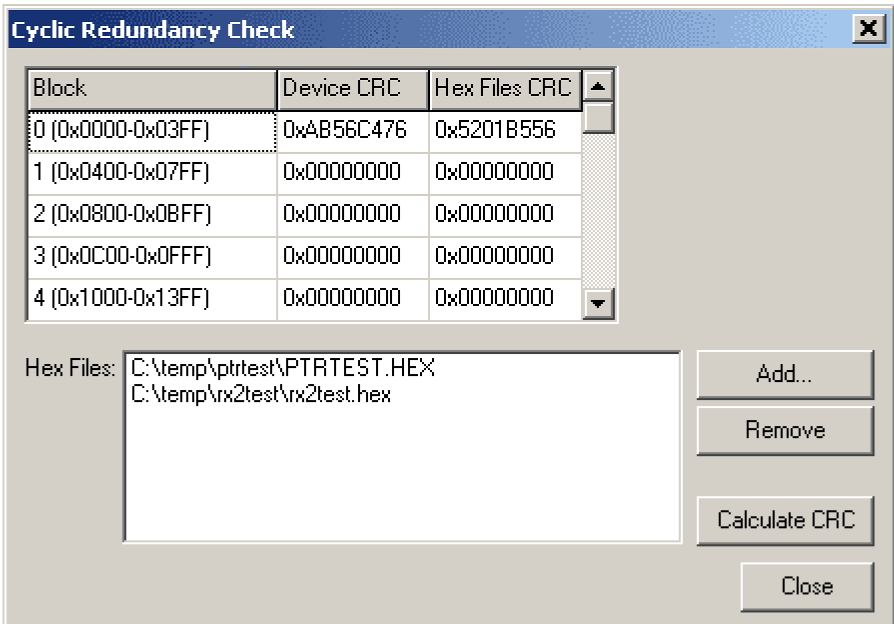
The contents of the Configuration window will vary depending on the device selected. The following image is the appearance of the 89LPC932 configuration window:



Some devices support protecting the device configuration, along with the boot vector and status bit/byte. On these devices a Clear Config Protect button will be shown, which when clicked will send the command to clear the protection for the configuration of the device.

6.13 Cyclic Redundancy Check

Some devices instead of a verify option provide a Cyclic Redundancy Check (CRC) option, where the device can calculate a 32-bit CRC value for a single Flash Block or the entire device. CRCs are a form of checksum, therefore if the contents of the memory change, there is a very high probability the CRC will also change.
If the selected device supports CRCs then the Cyclic Redundancy Check… item on the ISP menu will be enabled. Selecting it displays the CRC window.



When the window is first opened the CRC value is read from the device for each Flash Block and the entire device. These values are shown in the “Device CRC” column.
Click on the Add button and select each of the Hex files programmed into the device. Each Hex file will appear in the list at the bottom of the window.
To remove a Hex file from the list select it in the list and click on the Remove button.
Click on the Calculate CRC button to calculate the CRCs for each block and the whole device, as if the chosen Hex files were programmed into the device. These values will appear in the “Hex Files CRC” column. If the contents of the device exactly match the contents of the chosen Hex files then the CRCs on each row of the table will match.

6.14 MISR

Some devices instead of a verify option provide a MISR option, where the device can calculate a 128-bit MISR value for a single Flash Block or the entire device. MISR values are a form of checksum, therefore if the contents of the memory change, there is a very high probability the MISR value will also change.
If the selected device supports MISR values then the Read MISR… item on the ISP menu will be enabled. Selecting it displays the MISR window, which is functionally identical to the CRC window described in the previous section.

When the window is first opened the MISR value is read from the device for each Flash Block and the entire device. These values are shown in the “Device MISR” column.
Click on the Add button and select each of the Hex files programmed into the device. Each Hex file will appear in the list at the bottom of the window.
To remove a Hex file from the list select it in the list and click on the Remove button.
Click on the Calculate MISR button to calculate the MISR values for each block and the whole device, as if the chosen Hex files were programmed into the device. These values will appear in the “Hex Files MISR” column. If the contents of the device exactly match the contents of the chosen Hex files then the MISR values on each row of the table will match.

6.15 Go

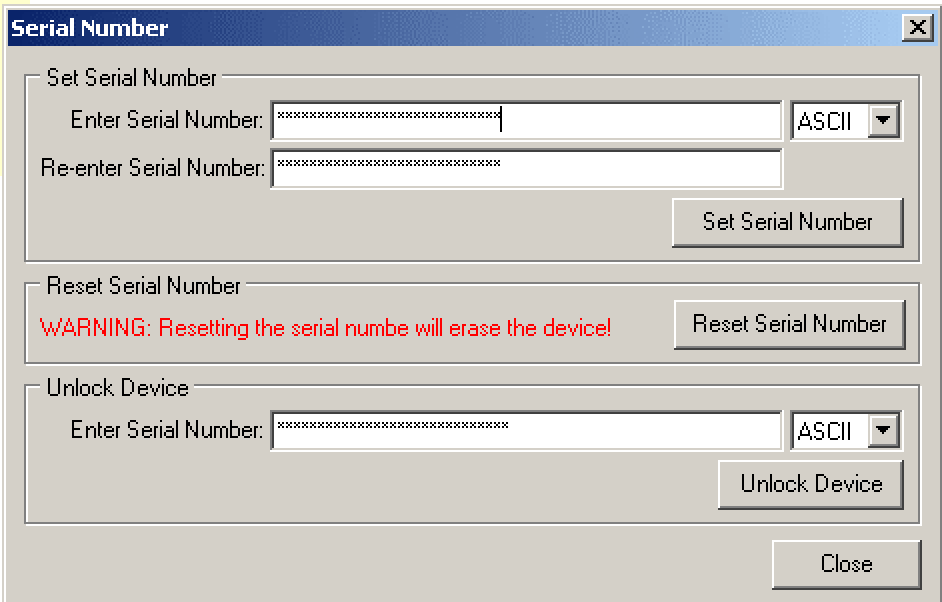
Some devices support executing from specific addresses in flash or RAM and using specific options or modes, for example the ARM devices.
To perform a Go operation and start execution choose Go… from the ISP menu.

Enter the address to start execution from and choose the mode to execute in. Click on Go to start execution.

Optionally enter a delay in milliseconds to cause Flash Magic to pause between sending the Go command and closing the COM port. This is useful when executing from RAM with an LPC2xxx device and closing the COM port may reset the device, due to DTR and RTS being used.

6.16 Serial Number

Some devices provide a serial number feature, which protects access to the device via ISP. If a serial number is supplied to the device then on the next reset the device will block most ISP operations until it is unlocked by supplying the correct serial number. The serial number may be reset however this will usually result in the device being erased stopping unauthorized access to the code.
Check the device datasheet for the exact details of the implementation.
If the selected device supports serial numbers then the Serial Number… item on the ISP menu will be enabled. Selecting it displays the serial number window.



To set a serial number, enter it into box boxes indicated. The contents of the boxes must match. Then click on Set Serial Number.

To reset a serial number click on the Reset Serial Number button.

To unlock a device using a serial number, enter the correct serial number into the box and click on Unlock Device.

Serial numbers may be entered in ASCII or Hexadecimal, by selecting from the drop-down list. Examples of ASCII serial numbers:

2A45bc2
foobar
MyOldMan

Examples of Hexadecimal serial numbers:

23BD4C2101ED3451

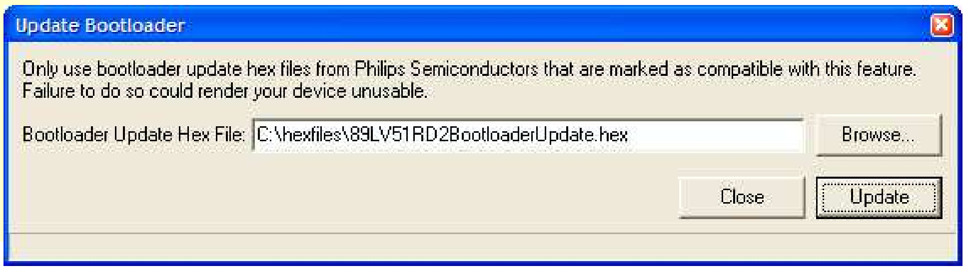
112233AABBCC

Serial numbers may be any length up to the maximum allowed. Hexadecimal serial numbers must not include the 0x prefix or the H/h suffix.

6.17 Update Bootloader

Some devices allow the bootloader to be updated via ISP. This can only be performed using a special bootloader update hex file supplied by NXP Semiconductors. Flash Magic supports these hex files.

If the currently selected device supports this feature, then "Update Bootloader…" on the ISP menu will be enabled. Choosing the menu item displays the dialog window.



Simply select the bootloader update hex file by entering the path to the file or clicking on the Browse… button to select it. Click on Update to start the update process. Status information will be shown at the bottom of the dialog window.

Once complete Flash Magic will indicate that the bootloader was updated or the reason why the update failed.

Only use special bootloader update hex files supplied by NXP Semiconductors that are created for this feature. Do not interrupt the power supply or reset the device during the update. Failure to follow these warnings may result in a device that no longer has a functional bootloader.

6.18 Additional Security Bits

Some devices support additional security bits that are not related to specific section of Flash memory. If the selected device has this feature then the Additional Security Bits… option on the ISP menu will be available. Choosing the option will display a dialog window showing the additional security bits for that device.

When the window is opened the current settings for the security bits will be read from the device and displayed in the window. Checking a security bit will set that bit. Unchecking a security bit will attempt to unset it.

Note that setting these security bits may have significant repercussions on the functionality of the device. It is strongly recommended to consult the data sheet or user manual and understand the functionality of these bits before setting them.

Chapter 7 - Advanced Options

Advanced Options are accessed by selecting Advanced Options from the Options menu.

7.1 High Speed Communications

Some devices feature the ability to switch from the initial baudrate to a high speed communications mode, allowing speeds higher than the autobaud method in the Bootloader would be able to accurately measure.

To enable the High Speed communications mode select the option in the Advanced Options window and select whether the device is operating in 6-clock mode or 12-clock mode (if applicable).

Flash Magic will calculate the highest possible baud rate that may be used by both the device and the PC COM Port and automatically switch to it after connecting at the initial baud rate specified in the main window.

If you experience problems with this feature, then try limiting how fast the high speed communications mode can go. Select the maximum speed from the drop-down list. If in doubt, select 9600 and start increasing until the problems appear.

7.2 Half-duplex Communications

When communicating with the device Flash Magic can send and receive data at the same time to achieve the fastest data rate. This type of transmission is called full-duplex. Turning the half-duplex option on will cause Flash Magic to only transmit one byte at a time, waiting for the byte to be echoed from the Bootloader before transmitting the next byte. While this will slow the data rate down it allows ISP to be performed via half-duplex serial buses, such as RS-485 and RS-485 derivatives such as J1708.

Note however that you must design your hardware such that the PC and the Bootloader do not receive the bytes they transmit otherwise each will be confused.

7.3 Hardware Configuration

For some devices, Flash Magic can control DTR and RTS to enter ISP mode or execute newly downloaded code. To implement this requires a hardware design that supports controlling of the device using DTR and RTS.

The options for controlling this feature are accessed by clicking on the Hardware Config tab. The options are different depending on the device selected in the main window. The following subsections describe the options for different devices.

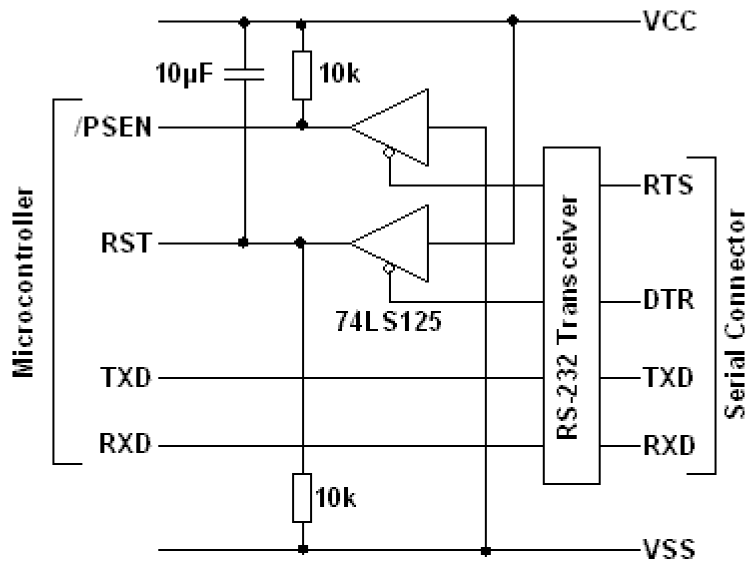
The advantages of this feature include:

- Faster development
- ISP is now possible on hardware that is hard to reach or enclosed in a box
- Removes the need for switches or jumpers on the hardware

89C51Rx2xx, 89C6xX2, 89C51Rx2Hxx, 89C66x, 89C51Rx+, XA-Gx9

DTR and RTS need to be connected to RST and /PSEN to allow Flash Magic to control the reset and ISP entry of the device.

The following simplified circuit diagram for an 89C51RD2 is one possible way of connecting the DTR and RTS signals to RST and /PSEN of the device. Note that minor changes would have to be made to use this circuit with the XA devices.



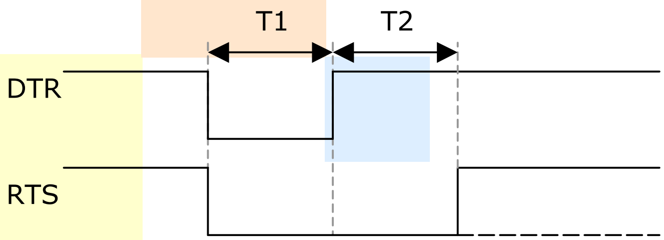
Notes:
When the COM Port is not in use or the serial cable is not connected, the RS232 signals are pulled low by the transceiver. This results in the TTL signals being high. Therefore when the TTL DTR and RTS signals are high, /PSEN must be weakly pulled high, and RST must be left to float so it can be asserted by the device and/or reset circuit.
/PSEN must be weakly pulled high to ensure the device can assert it.
The 10uF capacitor and 10k resistor form the reset circuit in this example circuit.

By checking the option to assert RTS while the COM Port is open, the RTS signal will remain asserted while the ISP operation is performed. This allows hardware to be designed that can reset or reconfigure hardware for an ISP operation.

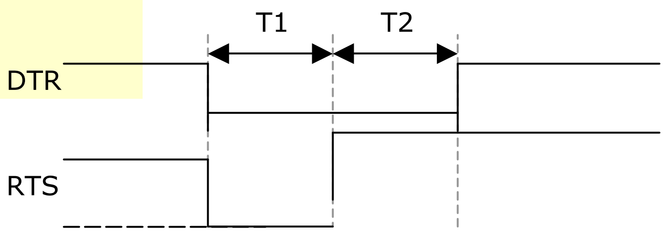
T1 and T2 are timing values for the waveforms Flash Magic generates with the DTR and RTS signals. Entering values in milliseconds into the boxes may configure these timings. Note however that the timings are approximate as they depend on what other applications are running in Windows and how fast the PC is.

The following timing diagrams show how T1 and T2 are used. Note that the signal levels are TTL (it is assumed DTR and RTS have already been passed through an RS232 transceiver).

Start of ISP operation (starting the Bootloader):



End of ISP operation (executing firmware):



The final option when checked instructs Flash Magic to assert DTR and RTS whenever Flash Magic has the COM Port open. This allows the possibility of the target hardware stealing power from the handshaking lines during ISP operations. Note however that once Flash Magic closes the COM Port at the end of an ISP operation Windows deasserts the DTR and RTS signals.

Note that when selecting to keep RTS asserted while the COM Port is open, or selecting to assert DTR and RTS while the COM Port is open, and high speed communications is selected, there will be pulses on the RTS (and DTR) signals just before the ISP operation. This is because the high speed communications feature needs to reconfigure the COM port several times before the ISP operation is performed. For each reconfigure Windows deasserts the DTR and RTS signals. Flash Magic then immediately reasserts the DTR and RTS signals resulting in a pulse.

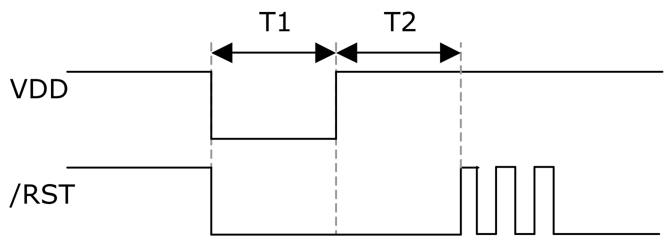
89LPC9xx

DTR and RTS need to be connected to VDD and /RST of the device. Currently Flash Magic only supports this feature with the Keil MCB 900 board, however custom hardware will be supported at a later date.

To use check the option to “Use DTR and RTS to enter ISP mode” and select the correct hardware from the drop down list.

T1 and T2 are timing values for the waveforms Flash Magic generates with the DTR and RTS signals. Entering values in milliseconds into the boxes may configure these timings. Note however that the timings are approximate as they depend on what other applications are running in Windows and how fast the PC is.

The following timing diagram shows how T1 and T2 are used. Note that the signal levels are TTL (it is assumed DTR and RTS have already been passed through an RS232 transceiver).

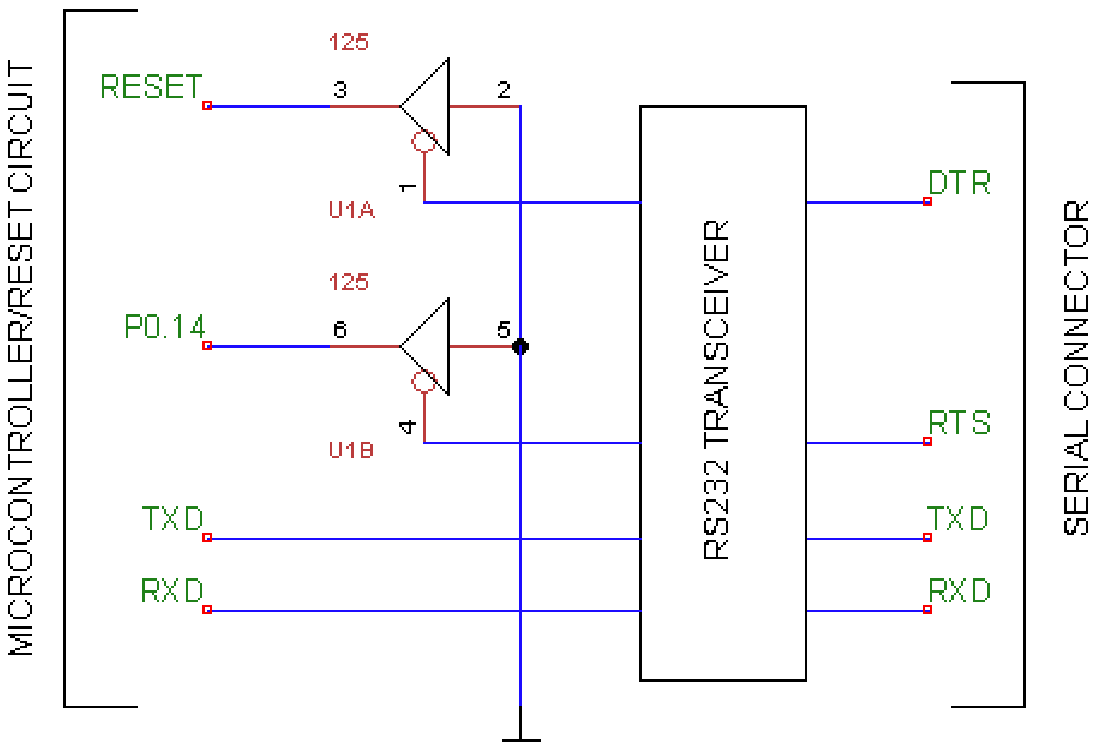


Adjusting T1 and T2 may be necessary if there is increased capacitance on VDD, causing the device to take longer to power down and back up again.

LPC2xxx and LPC1xxx

DTR and RTS need to be connected to RST and the ISP entry pin (for example P0.14)to allow Flash Magic to control the reset and ISP entry of the device.

The following simplified circuit diagram for an LPC2xxx is one possible way of connecting the DTR and RTS signals to RST and P0.14 of the device.



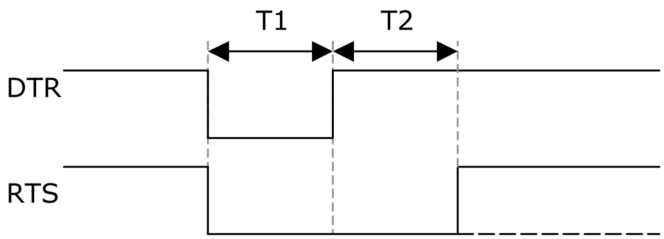
Notes:

When the COM Port is not in use or the serial cable is not connected, the RS232 signals are pulled low by the transceiver. This results in the TTL signals being high. Therefore when the TTL DTR and RTS signals are high, P0.14 and RST must be in a state that allows the device to reset normally and execute code.

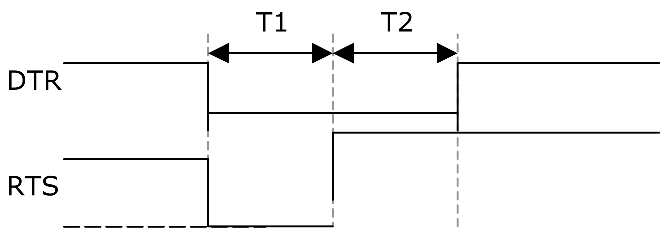
By checking the option to assert RTS while the COM Port is open, the RTS signal will remain asserted while the ISP operation is performed. This allows hardware to be designed that can reset or reconfigure hardware for an ISP operation.

T1 and T2 are timing values for the waveforms Flash Magic generates with the DTR and RTS signals. Entering values in milliseconds into the boxes may configure these timings. Note however that the timings are approximate as they depend on what other applications are running in Windows and how fast the PC is.
The following timing diagrams show how T1 and T2 are used. Note that the signal levels are TTL (it is assumed DTR and RTS have already been passed through an RS232 transceiver).

Start of ISP operation (starting the Bootloader):



End of ISP operation (executing firmware):



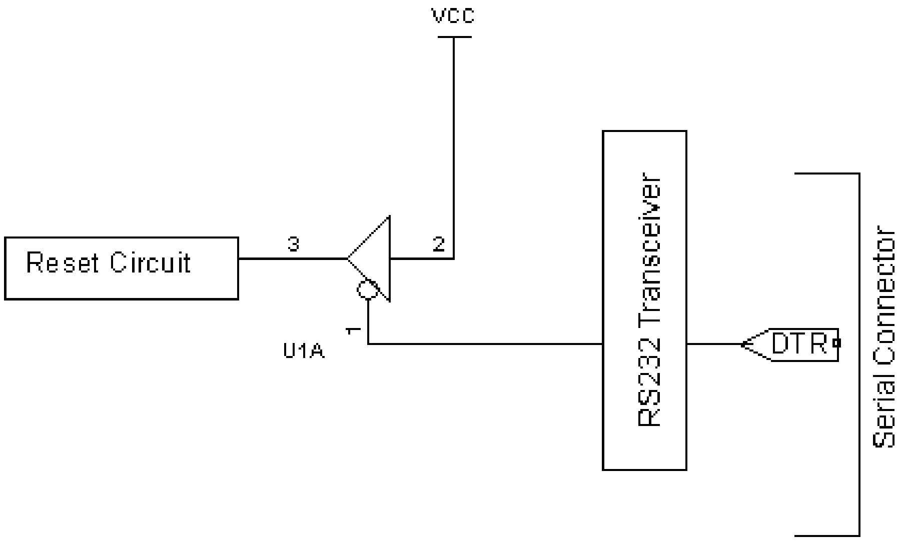
The final option when checked instructs Flash Magic to assert DTR and RTS whenever Flash Magic has the COM Port open. This allows the possibility of the target hardware stealing power from the handshaking lines during ISP operations. Note however that once Flash Magic closes the COM Port at the end of an ISP operation Windows deasserts the DTR and RTS signals.

Note that when selecting to keep RTS asserted while the COM Port is open, or selecting to assert DTR and RTS while the COM Port is open, and high speed communications is selected, there will be pulses on the RTS (and DTR) signals just before the ISP operation. This is because the high speed communications feature needs to reconfigure the COM port several times before the ISP operation is performed. For each reconfigure Windows deasserts the DTR and RTS signals. Flash Magic then immediately reasserts the DTR and RTS signals resulting in a pulse.

89V51Rx2, 89LV51Rx2

DTR needs to be connected to RST to allow Flash Magic to control the reset of the device.

The following simplified circuit diagram for an 89V51RD2 is one possible way of connecting the DTR signal to RST of the device.

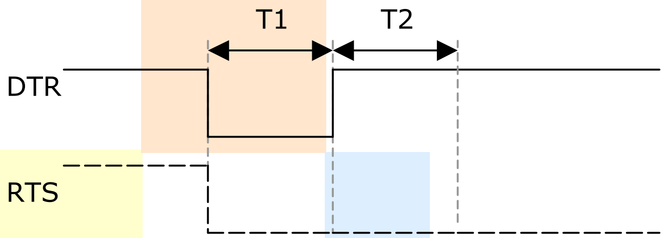


Notes:
When the COM Port is not in use or the serial cable is not connected, the RS232 signals are pulled low by the transceiver. This results in the TTL signals being high. Therefore when the TTL DTR signal is high RST must be left to float so it can be asserted by the device and/or reset circuit.

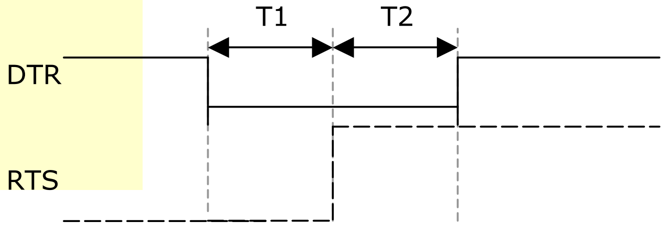
By checking the option to assert RTS while the COM Port is open, the RTS signal will remain asserted while the ISP operation is performed. This allows hardware to be designed that can reset or reconfigure hardware for an ISP operation.

T1 and T2 are timing values for the waveforms Flash Magic generates with the DTR and RTS signals. Entering values in milliseconds into the boxes may configure these timings. Note however that the timings are approximate as they depend on what other applications are running in Windows and how fast the PC is.
The following timing diagrams show how T1 and T2 are used. Note that the signal levels are TTL (it is assumed DTR and RTS have already been passed through an RS232 transceiver).

Start of ISP operation (starting the Bootloader):



End of ISP operation (executing firmware):



The final option when checked instructs Flash Magic to assert DTR and RTS whenever Flash Magic has the COM Port open. This allows the possibility of the target hardware stealing power from the handshaking lines during ISP operations. Note however that once Flash Magic closes the COM Port at the end of an ISP operation Windows deasserts the DTR and RTS signals.

Note that when selecting to keep RTS asserted while the COM Port is open, or selecting to assert DTR and RTS while the COM Port is open, and high speed communications is selected, there will be pulses on the RTS (and DTR) signals just before the ISP operation. This is because the high speed communications feature needs to reconfigure the COM port several times before the ISP operation is performed. For each reconfigure Windows deasserts the DTR and RTS signals. Flash Magic then immediately reasserts the DTR and RTS signals resulting in a pulse.

7.4 Protect ISP

Some devices contain the ISP code – which allows Flash Magic to communicate with the device – in the main Flash memory. This means that it is possible to erase or corrupt the ISP code. Once erased or corrupted ISP operations can no longer be performed and the device would have to be physically removed from the hardware and placed in a parallel programmer to be reprogrammed.
In order to stop the ISP code from being accidentally erased the Protect ISP option is provided. When checked, Flash Magic will modify erase and programming operations such that the ISP code is not erased or corrupted.
For example:

- A full device erase will be achieved by erasing Flash blocks and pages, to erase the Flash without erasing the section of Flash containing the ISP code. A side effect is

that any security bits that can only be erased with a full device erase will not be erased. **This must be considered before setting those security bits.**
- A block erase for a block that contains the ISP code will be achieved by erasing pages in the block – if they exist on the device – to avoid erasing the ISP code.
- The programming of any Hex file containing data that would corrupt the ISP code will be aborted.
- The section of Flash containing the ISP code will not be filled.
- Any checksums that are in the same locations as ISP code will not be programmed.

If the Protect ISP option is unchecked then erasing or programming a device (include fills and checksum generation) has the potential to erase or corrupt the ISP code. Any attempt to perform one of these operations will result in Flash Magic asking for confirmation first. Erasing or corrupting the ISP code will immediately render the ISP functionality of the device non-functional.

IT IS STRONGLY RECOMMENDED TO LEAVE THE PROTECT ISP OPTION TURNED ON

7.5 Just In Time Code

Just In Time Code is a powerful feature, which allows Flash Magic to call a custom 3rd party program (called a JIT Module) that generates last minute code or constant data to be programmed into the device.

Uses for this system include:

- Serial Number generation
- Copy Protection (generate authorization codes via the Internet)
- Programmer information
- Date and Time
- Lookup Table generation
- Language tables

The JIT Module can be written using any language or development tools available for PCs and may access files on the local machine or Internet to generate the data.

The JIT Module must run from the Command Prompt or DOS Command Line. The command line syntax is as follows:

executablename commandfilepath

where:

executablename	The path and name of the JIT Module
commandfilepath	The path to a command file that describes where to place the data, where to place output for the user to see and any optional parameters supplied by the user.

The command file is an ASCII file with the following format (each line terminated with a linefeed character):

datafilepath
userfilepath
[*option*]

datafilepath	The path to the data file that the JIT Module should generate containing the data to program into the device.
userfilepath	The path to the user file where an ASCII string that will be displayed to the user can be stored. Leave empty or don't create if no message is required. The message is shown after the JIT Module has finished executing.
option	One or more optional parameters entered by the user that may be used by the program.

The format of the data file is the same as an Intel Hex File. Only record types 00H and 01H are allowed however. The terminating record is optional.

The JIT Module must return a zero for failure or a one for success. In C this is achieved by returning the value from the main() function.

Free PC Compilers are available from http://www.idiom.com/free-compilers, and http://www.borland.com.

To configure the Just In Time feature click on the Just In Time Code tab in the advanced options.

Enter the path and filename of the JIT Module into the box or click on the Browse… button to browse to the file.
Enter any options desired separated by spaces. If no options are required then leave the box empty.
Enter the maximum run time of the JIT Module in seconds into the final box. If the JIT Module does not finish executing within this time then Flash Magic will give up waiting and return an error.

7.6 Timeouts

In the Timeouts section the timeouts Flash Magic should use when performing ISP operations are specified. Normally default settings are used, however if you wish to change the timeouts then check the option to use my timeouts and fill in the values in the boxes.

Flash Magic uses two timeouts, regular and long. Each timeout is specified in seconds. The regular timeout is used for most ISP operations. The long timeout is used for erasing and performing blank checks.

The default settings are four seconds for the regular timeout and 60 seconds for the long timeout. **It is strongly recommended to use the default settings.**

If you are using a USB to COM port converter then you may find that increasing the timeouts will resolve communication problems that are sometimes present with those converters.

7.7 Misc

In the Misc section miscellaneous settings relating to Flash Magic can be found.

Flash Magic includes the ability to play a Wave file when programming is completed. To do this check the option to play a Wave file, then either enter the path to the Wave file into the box or click on the Browse button and select the Wave file. To hear the Wave file click on the purple arrow.

If a device is encountered with an incorrect set of signature bytes, it is possible to turn of the signature checking by checking the Disable device signature checking option. Please report any incorrect signature bytes to support@esacademy.com. Please include the signature bytes read from the device along with all markings on the top of the device itself.

Chapter 8 - Command Line Interface

8.1 Introduction

The Command Line interface to Flash Magic allows the features of Flash Magic to be accessed from the command line/DOS. This provides a powerful and flexible way of integrating Flash Magic into:

- your project's build process
- the Integrated Development Environment you use for developing applications.

If you use a Batch file to build your project then a call to Flash Magic can be added onto the end of the batch file allowing one-step build and program of your application.

Most modern IDEs allow users to add custom menu entries that run any command you desire. Therefore it is simple to add menu entries to run Flash Magic and program the latest Hex file.

It is possible to configure Flash Magic to output the result of all operations to an ASCII text file. The output is described in this manual enabling programs to be written to parse the text file and provide automated testing of devices or gang-programming of devices.

8.2 Running Flash Magic on the Command Line

The Command Line version of Flash Magic is called:

FM.EXE

Before you can run FM.EXE on the command line you must set up your PATH environment variable to point to the folder FM.EXE is stored in. This is done automatically during installation, however you must restart your machine before the change to the PATH variable is recognized when using Windows 95/98/ME. You must restart your machine or log out then back in again before the change to the PATH variable is recognized when using Windows NT/2000/XP.

Commands are passed to FM.EXE in the form or either directives or a Command File containing directives.

The command line will have the following syntax:

```
FM [directives]
```

Where:

directives	space separated list of directives or @*commandfile*
commandfile	An ASCII file containing a space separated or newline separated list of directives

There are two types of directives:

Configuration	configure how the device is accessed and how Flash Magic operates
Operation	an operation to be performed on the device

Directives may appear in any order in the list, however the operation directives are processed in the order listed. For example the following command line:

```
FM ERASE(3, PROTECTISP) HEXFILE(TEST.HEX, CHECKSUMS, FILL)
```

Performs an erase before programming the hex file. However the following command line:

```
FM HEXFILE(TEST.HEX, CHECKSUMS, FILL) ERASE(3, PROTECTISP)
```

Programs the hex file then performs the erase.

Operation directives may appear more than once on a command line, allowing complex operations to be performed. For example:

```
FM ERASE(0, PROTECTISP) HEXFILE(TEST1.HEX, CHECKSUMS, NOFILL) ERASE(1)
HEXFILE(TEST2.HEX, CHECKSUMS, NOFILL)
```

Erases block 0 then programs a hex file, then erases block 1 and programs the second hex file. Checksums are generated for both Hex files.

Configuration directives may appear anywhere on the command line, however all configuration directives are processed before any operation directives are processed.

Directive names are case insensitive.
Whitespace is ignored except when it is between directives, where it is required.
All directives are optional. When a directive is omitted the default setting for that directive is used.

All numeric values passed to directives may be passed in decimal or hexadecimal unless otherwise specified in the directive description. Hexadecimal values are either preceded by “0x” or have the suffix “H” or “h”.

Paths may contain spaces. Paths may also contain newlines, carriage returns and tabs, however these characters are converted to spaces before the path is used, allowing word wrapping at the spaces in paths.

8.3 BLANKCHECK

Description:	Performs a blank check. For the 89LPCxx2 the blank check is performed on the entire block containing the start address; the end address is ignored. For ARM devices the start and end addresses must match with the start and end addresses of a flash block.
More than one allowed:	Yes
Type:	Operation
Syntax:	BLANKCHECK(*start*, *end*) Where: *start* The address to perform the blank check from inclusive *end* The address to stop the blank check at inclusive
Output:	Memory blank (*start*, *end*) Or: Memory not blank at *address* (*start*, *end*) Or: Memory not blank (*start*, *end*) Or: Blank check failed: *reason* (*start*, *end*) Where: *start* The start address passed to the BLANKCHECK directive *end* The end address passed to the BLANKCHECK directive *address* The first non-blank address *reason* The reason for the failure
Default:	No blank check is performed.
Example:	BLANKCHECK

8.4 BOOTVECTOR

Description:	Programs the Boot Vector with a new value.	
More than one allowed:	Yes	
Type:	Operation	
Syntax:	BOOTVECTOR(*value*)	
	Where:	
	value	new value for the Boot Vector.
Output:	Boot Vector programmed to *value* Or: Boot Vector program failed: *reason* (*value*)	
	Where:	
	value	The value the Boot Vector was programmed to in hexadecimal, prefixed with “0x”.
	reason	The reason for the failure
Default:	The Boot Vector is not programmed with a new value.	
Examples:	BOOTVECTOR(128) BOOTVECTOR(0x80) BOOTVECTOR(80H)	

8.5 COM

Description: Specifies the PC Serial (COM) port and baud rate to use for communicating with the device. If High Speed communications are being used then this directive specifies the initial baud rate.

More than one allowed: No

Type: Configuration

Syntax: COM(*port*, *baudrate*)

Where:

port	The COM Port to use
baudrate	The baud rate to use. One of: 2400 4800 9600 19200 38400 57600

Output: Connected
Or: Connection failed: *reason*

Where:

reason	The reason the connection failed

Default: COM 1, baudrate of 19200 (COM(1, 19200))

Examples:

```
COM(1H, 9600)
COM(1, 0x2580)
COM(0x01, 2580H)
```

8.6 DEVICE

Description:	Selects the device being used and the oscillator frequency being used. If the device does not require an oscillator frequency then specify zero for the frequency.
More than one allowed:	No
Type:	Configuration
Syntax:	DEVICE(*device*, *frequency*)

Where:

device — Device name in capitals. Some examples:

89V664
XA-G39
89C51RD2xx
89LPC938
LPC2129
LPC1768

To use an Ethernet bootloader append "ETH" to the device name. For example LPC2388ETH.

frequency — oscillator frequency in MHz in decimal only

Output:

Device selected
Or: Device selection failed: *reason*

Where:

reason — The reason device selection failed

Default:

89C51RD2Hxx device with 0MHz oscillator frequency (DEVICE(89C51RD2Hxx,0))

Notes:

There is some flexibility in the device names that are recognized. Device names may optionally be prefixed with a “P” for NXP, and “8x” may appear in place of “89”. See the examples.

Examples:

DEVICE(89C51RC+, 10.000)
DEVICE(89C664, 16.124)
DEVICE(XA-G49, 12.662)
DEVICE(89C668, 20.118)

DEVICE(P8xC51RC+, 10.000)
DEVICE(8xC664, 16.124)
DEVICE(P89C668, 20.118)

8.7 ERASE

Description:	Erases individual flash blocks or the whole device
More than one allowed:	Yes
Type:	Operation
Syntax:	ERASE(*type, protectisp*)

Where:

type	The type of erase, either a Flash block or the whole device. If type is a value in the range 0 to 15 then that number Flash block will be erased. If type is DEVICE then the whole device will be erased including security bits, with the boot vector and status bytes set to default values.
protectisp	Set to PROTECTISP to stop the ISP code from being erased in devices that contain the ISP code in the main Flash memory. Set to NOPROTECTISP to allow the ISP code to be erased. Ignored for other devices. It is strongly recommended to set this option to PROTECTISP.

Output:

Erase complete (*type*)
Or: Erase failed: *reason* (*type*)

Where:

type	The type of erase. Either a number in the range 0 to 15 or a hexadecimal number in the range 0x00 to 0x0F prefixed with "0x" or DEVICE.
reason	The reason erase failed

Default:

No flash blocks are erased. No security bits are erased. No changes are made to the Boot Vector or Status Byte.

Examples:

ERASE(1, PROTECTISP)
ERASE(0x02, PROTECTISP)
ERASE(02H, PROTECTISP)
ERASE(DEVICE, PROTECTISP)

8.8 HEXFILE

Description: Programs an Intel Hex file into the Flash memory.
Note that more than one instance of this directive is allowed on the command line, however if a HEXFILE directive fills unused Flash then any subsequent HEXFILE directives will fail as it will not be possible to program the Flash and any previously programmed Hex File will be corrupted.

More than one allowed: Yes

Type: Operation

Syntax: HEXFILE(*path*, *checksums*, *fill, protectisp*[, CODEREADPROTECTION])

Where:

path	The path to the Intel Hex File
checksums	Set to CHECKSUMS to generate checksums in the Flash blocks used by the Hex File. Set to NOCHECKSUMS for no checksums.
fill	Set to FILL to fill the unused Flash with the value 00H. Set to NOFILL for no filling of the Flash.
protectisp	Set to PROTECTISP to stop the ISP code from being corrupted in devices that contain the ISP code in the main Flash memory. Set to NOPROTECTISP to allow the ISP code to be corrupted. Ignored for other devices. It is strongly recommended to set this option to PROTECTISP.
CODEREADPROTECTION	Sets the code read protection on devices that support it (LPC2xxx only)

Output: Hex File programming complete (*path*)
Or: Hex File programming failed: *reason* (*path*)

Where:

path	The path of the Intel Hex file as passed to the HEXFILE directive.
reason	The reason Hex File programming failed

Default: No Hex file is programmed. Checksums are not generated. Unused Flash memory is not filled.

Examples:

```
HEXFILE(test.hex, CHECKSUMS, NOFILL, PROTECTISP)
HEXFILE(..\test.hex, NOCHECKSUMS, FILL, PROTECTISP)
```

HEXFILE(C:\work\test.hex, CHECKSUMS, FILL, PROTECTISP)
HEXFILE(foo.hex, NOCHECKSUMS, FILL, PROTECTISP, CODEREADPROTECTION);

8.9 HIGHSPEED

Description:	Uses high speed communications if the device supports it.
More than one allowed:	No
Type:	Configuration
Syntax:	HIGHSPEED(*clocks*, *highspeedmax*)

Where:

clocks	The number of clocks per cycle the device is configured to use. May be 6 or 12. The LPC9xx devices that support this command should have zero passed for the number of clocks per cycle.
highspeedmax	The fastest baudrate that can be used in high speed mode in bps. Must be a standard baud rate such as 115200, 57600, 38400, etc.

Output:

High Speed mode selected
Or: High Speed mode selection failed: *reason*

Where:

reason	The reason high speed mode selection failed

Default: High speed communications are not used.

Examples:

HIGHSPEED(6, 115200)
HIGHSPEED(12, 57600)
HIGHSPEED(0x0C, 38400)
HIGHSPEED(0CH, 19200)
HIGHSPEED(0, 57600)

8.10 READ

Description:	Reads the range of Flash memory specified and saves the contents in an Intel Hex File
More than one allowed:	Yes
Type:	Operation
Syntax:	READ(*start*, *end*, *path*)

Where:

start	The start address inclusive.
end	The end address inclusive.
path	Path to the Intel Hex file to create.

Output:

Flash read complete (*path*)
Or: Flash read failed: *reason* (*path*)

Where:

path	Path of the Intel Hex File to create as passed to the READ directive.
reason	The reason Flash read failed

Default: Flash is not read.

Examples:

READ(0x2000, 0x343C, test.hex)
READ(2000H, 0x343C, C:\work\test.hex)
READ(2000H, 13372, ..\test.hex)

8.11 READSECURITY

Description:	Reads the security bits.
More than one allowed:	Yes
Type:	Operation
Syntax:	READSECURITY
Output:	The results are listed with each security bit on a separate line, with the format: Bit *n*: *result* Or: Block *b* bit *n*: *result* Or: Code Read Protection: *result* Or: Security bit read failed: *reason* Where: *n* 1 – 3 *b* 0 - 7 *result* set or not set or enabled or disabled *reason* The reason security bit read failed
Default:	The security bits are not read.
Example:	READSECURITY

8.12 READSIGNATURE

Description:	Reads the signature bytes.
More than one allowed:	Yes
Type:	Operation
Syntax:	READSIGNATURE
Output:	The results are listed with each signature byte on a separate line, with the format: *byte*: *value* Or: Signature read failed: *reason* Where:

byte	Manufacturer ID or Device ID 1 or Device ID 2 or Device ID
value	The value of the signature byte in hexadecimal prefixed with “0x”.
reason	The reason signature read failed

Default:	The signature bytes are not read.
Example:	READSIGNATURE

8.13 SECURITY

Description:	Sets the security bits. Note you cannot unset the security bits with this directive. Instead the ERASE directive must be used, performing a full device erase, or block erase – depending on the device.
More than one allowed:	Yes
Type:	Operation
Syntax:	For devices with one, two or three security bits:

SECURITY(*bit1*, *bit2*, *bit3*)

Where:

bit1	Set to 1 to set security bit 1, set to 0 to keep it unset.
bit2	Set to 1 to set security bit 2, set to 0 to keep it unset.
bit3	Set to 1 to set security bit 3, set to 0 to keep it unset.

For devices with three security bits per block:

SECURITY(*block0bit1*, *block0bit2*, *block0bit3*, *block1bit0*, *block1bit1*, …)

Where:

block0bit0	Set to 1 to set security bit 0 of block 0, set to 0 to keep it unset
block0bit1	Set to 1 to set security bit 1 of block 0, set to 0 to keep it unset
…	
blocknbit2	Set to 1 to set security bit 2 of block n, set to 0 to keep it unset

For the ARM devices, bit1 = code read protection.

Output:

Security bit programming complete
Or: Security bit programming failed: *reason*

Where:

reason	The reason security bit programming failed

Default: No security bits are set.

Examples:	SECURITY(1, 0x00, 0x01) SECURITY(0H, 1, 0x01) SECURITY(1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,0)

8.14 STATUSBYTE

Description:	Programs the Status Byte with a new value.
More than one allowed:	Yes
Type:	Operation
Syntax:	STATUSBYTE(*value*) Where: *value* new value for the Status Byte.
Output:	Status Byte programmed to *value* Or: Status Byte programming failed: *reason* (*value*) Where: *value* The value the Status Byte was programmed to in hexadecimal, prefixed with “0x”. *reason* The reason Status Byte programming failed
Default:	The Status Byte is not programmed with a new value.
Examples:	STATUSBYTE(128) STATUSBYTE (0x80) STATUSBYTE (80H)

8.15 VERIFY

Description:	Verifies that an Intel Hex file has been programmed correctly into Flash.
More than one allowed:	Yes
Type:	Operation
Syntax:	VERIFY(*path*, *checksums*) Where: *path* — Path of the Intel Hex File to verify *checksums* — Set to CHECKSUMS if checksums were generated for the Hex File. Set to NOCHECKSUMS if no checksums were generated for the Hex File.
Output:	The result is displayed on the standard output with a line of the format: Verify *result* (*path*) Or: Verify failed: *reason* (*path*) Where *result* — passed or failed *path* — The path to the Intel hex file as passed to the VERIFY directive *reason* — The reason verify failed
Default:	No verification is performed.
Examples:	VERIFY(test.hex) VERIFY(..\test.hex) VERIFY(C:\work\test.hex)

8.16 QUIET

Description:	Outputs information about the processing of each directive to an ASCII output file rather than the standard output, where a program can process it.
More than one allowed:	No
Type:	Configuration
Syntax:	QUIET(*path*) Where: *path* — Path of the output file to generate.
Output:	None. Or: Output file creation failed (*path*) Where: *path* — The path of the output file to generate as passed to the QUIET directive.
Default:	Output is sent to the standard output.
Examples:	QUIET(test.txt) QUIET(..\test.txt) QUIET(C:\work\test.txt)

8.17 READBOOTVECTOR

Description:	Reads the value of the Boot Vector
More than one allowed:	Yes
Type:	Operation
Syntax:	READBOOTVECTOR
Output:	Boot Vector: *value*
	Or: Boot Vector read failed: *reason*
	Where:
	value — The value of the Boot Vector in hexadecimal prefixed with “0x”.
	reason — The reason the read failed
Default:	The Boot Vector is not read.
Example:	READBOOTVECTOR

8.18 READSTATUSBYTE

Description:	Reads the value of the Status Byte
More than one allowed:	Yes
Type:	Operation
Syntax:	READSTATUSBYTE
Output:	Status Byte: *value*
	Or: Status Byte read failed: *reason*
	Where:
	value The value of the Status Byte in hexadecimal prefixed with "0x"
	reason The reason the read failed
Default:	The Status Byte is not read.
Example:	READSTATUSBYTE

8.19 HALFDUPLEX

Description:	When used Flash Magic waits for each byte to be echoed back before sending the next byte, allowing half-duplex buses such as J1708 to be used with ISP.
More than one allowed:	No
Type:	Configuration
Syntax:	HALFDUPLEX
Output:	None
Default:	Full-duplex communications are used.
Example:	HALFDUPLEX

8.20 RESET

Description:	Resets the device. If the device has been successfully programmed then the new code will be executed. Only supported on Rx2 and 66x families revision G or higher.
More than one allowed:	Yes
Type:	Operation
Syntax:	RESET
Output:	Reset complete Or: Reset failed: *reason* Where: *reason* The reason the reset failed.
Default:	The device is not reset.
Example:	RESET

8.21 STARTBOOTLOADER

Description:

Specifies that a command or break condition should be sent to the device to start the Bootloader. This requires support in the currently executing application. For commands the baud rate to use for communicating with the device (note that a baud rate different to the one specified with the COM directive may be used) must be specified. The COM port used is specified with the COM directive. Whether to use half-duplex communications or not is specified with the HALFDUPLEX directive. To send a break condition, pass the word "BREAK" to the directive.
Special characters may be inserted into the command to provide additional functionality. They consist of a special character followed by two hexadecimal characters and have the following meanings:

%HH	- transmit the character HH, where HH is the hexadecimal value of the character
$HH	- delay for HH x 100ms, where HH is a hexadecimal value
&AA	- command. The function of the command depends upon the hexadecimal value AA and currently can be one of the following:

00	- flush RX buffer
01	- echo on
02	- echo off
03	- send break condition

By default at the start of a command the echo is on.
Regardless of whether the echo is on or off, when the end of a command is reached the device must return a '.' to indicate the command was successfully received.

More than one allowed: No

Type: Configuration

Syntax:

STARTBOOTLOADER(*baudrate*, *command, control*)
STARTBOOTLOADER(BREAK)

Where:

baudrate The baud rate to use. One of:
2400
4800
9600
19200

	38400 57600
command	The command to send to the device
control	Control characters to append to the command or NOCONTROL if no control characters are required. One of: CR LF CRLF LFCR NOCONTROL

Output: Bootloader started
Or: Start Bootloader failed: *reason*

Where:

reason The reason the command failed

Default: No command is sent to the device

Examples:

STARTBOOTLOADER(9600, boot, NOCONTROL)
STARTBOOTLOADER(0x2580, go to bootrom, CR)
STARTBOOTLOADER(BREAK)
STARTBOOTLOADER(9600, %0D%02, NOCONTROL)
    Transmits a carriage return followed by an STX
STARTBOOTLOADER(9600, %80, NOCONTROL)
    Transmits 80H
STARTBOOTLOADER(9600, $14, NOCONTROL)
    Delays for 20 x 100ms = 2 seconds
STARTBOOTLOADER(9600, $0A, NOCONTROL)
    Delays for 10 x 100ms = 1 second
STARTBOOTLOADER(9600, &00, NOCONTROL)
    Flushes the RX buffer
STARTBOOTLOADER(9600, A&02B&01C, NOCONTROL)
    Transmits 'A', waits for 'A' to be echoed, transmits 'B' then transmits 'C' and waits for 'C' to be echoed.
STARTBOOTLOADER(9600, %0D$14&00A, NOCONTROL)
    Transmits a carriage return, waits for 2 seconds, flushes the RX buffer and transmits 'A'.

8.22 READCLOCKS

Description:	Reads the clocks bit.
More than one allowed:	Yes
Type:	Operation
Syntax:	READCLOCKS
Output:	Device is using 6 clocks/cycle Or: Device is using SFR to select clocks (default 12 clocks/cycle) Or: Clocks bit read failed: *reason* Where: *reason* The reason security bit read failed
Default:	The clocks bit is not read.
Example:	READCLOCKS

8.23 CLOCKS

Description:	Sets the clocks bit which will result in the device using 6 clocks/cycle. Note you cannot unset the clocks bit with this directive. Instead the ERASE directive must be used, performing a full device erase.
More than one allowed:	Yes
Type:	Operation
Syntax:	CLOCKS
Output:	Clocks bit programming complete Or: Clocks bit programming failed: *reason* Where: *reason* The reason clocks bit programming failed
Default:	The clocks bit is not set
Examples:	CLOCKS

8.24 HARDWARE

Description:	Configures how Flash Magic uses the handshaking signals of the PC's COM Port. The handshaking signals DTR and RTS may be used to control the /PSEN and RST pins on the microcontroller to place the device in Bootloader mode or execute firmware automatically before and after ISP operations. Alternatively the signals may be asserted allowing power to be drawn from the COM Port. To obtain the equivalent functionality of the old HARDWARERESET directive use: HARDWARE(BOOTEXEC, 100, 100).
More than one allowed:	No
Type:	Configuration
Syntax:	HARDWARE(*config*[, *t1*, *t2*])

where:

config The configuration of the signals. One of:

BOOTEXEC	- DTR and RTS are used to control /PSEN and RST or P0.14 and RST. t1 and t2 are required.
BOOTEXECRTS	- DTR and RTS are used to control /PSEN and RST or P0.14 and RST. RTS remains asserted while the COM Port is in use by Flash Magic. t1 and t2 are required
ASSERT	- DTR and RTS are asserted while the COM Port is in use by Flash Magic. t1 and t2 are not required.
KEILMCB900	- DTR and RTS are used to generate the necessary signals to place a device on the Keil MCB 900 board into ISP mode.

t1 Period for time t1 in milliseconds
t2 Period for time t2 in milliseconds

Output:	None
Default:	DTR and RTS are not used.
Example:	HARDWARE(BOOTEXEC, 50, 100) HARDWARE(BOOTEXECRTS, 200, 250) HARDWARE(ASSERT)

HARDWARE(KEILMCB900, 250, 120)

8.25 READCRC

Description:	Reads the CRC of a block or the entire device.
More than one allowed:	Yes
Type:	Operation
Syntax:	READCRC(*type*)

Where:

type The Flash to read the CRC of, either a Flash block or the whole device. If type is a value in the range 0 to 15 then the CRC of that number Flash block will be read. If type is DEVICE then the CRC of the whole device will be read.

Output:

CRC *result* (*type*)
Or: CRC read failed: *reason* (*type*)

Where:

result	The CRC value in hexadecimal prefixed with “0x”.
type	The Flash to read the CRC of. Either a decimal number in the range 0 to 15 or a hexadecimal number in the range 0x0 to 0x0F prefixed with “0x” or DEVICE.
reason	The reason the read failed

Default: The CRC is not read.

Examples:

READCRC(0x4)
READCRC(4)
READCRC(DEVICE)

8.26 ERASEPAGE

Description:	Erases a single page in the device
More than one allowed:	Yes
Type:	Operation
Syntax:	ERASEPAGE(*type, protectisp*)

Where:

type	The number of the page to erase.
protectisp	Set to PROTECTISP to stop the ISP code from being erased in devices that contain the ISP code in the main Flash memory. Set to NOPROTECTISP to allow the ISP code to be erased. Ignored for other devices. It is strongly recommended to set this option to PROTECTISP.

Output:

Page erase complete (*type*)
Or: Page erase failed: *reason* (*type*)

Where:

type	The number of the page to erase, as passed to the directive.
reason	The reason the erase failed

Default: The page is not erased

Examples:

ERASEPAGE(0, PROTECTISP)
ERASEPAGE(0x5, PROTECTISP)

8.27 READCONFIG

Description:	Reads the configuration of the device if supported by the device. For the 89LPC9xx reads the UCFG1 byte. For devices that support UCFG1 and UCFG2, both bytes are read.
More than one allowed:	Yes
Type:	Operation
Syntax:	READCONFIG
Output:	Configuration: *value* Or: Configuration: UCFG1=*value* UCFG2=*value* Or: Configuration read failed: *reason* Where: *value* The configuration in hexadecimal prefixed with “0x”. *reason* The reason the erase failed
Default:	The configuration is not read
Example:	READCONFIG

8.28 CONFIG

Description:	Configures a device – if supported by the device. For the 89LPC9xx bits 0 – 7 configure UCFG1. For devices that support UCFG1 and UCFG2, bits 0 – 7 configure UCFG1, bits 8 – 15 configure UCFG2.
More than one allowed:	Yes
Type:	Operation
Syntax:	CONFIG(*value*) Where: *value* The value for the configuration.
Output:	Device configured (*value*) Or: Device configuration failed: *reason* (*value*) Where: *value* The configuration value, as passed to the directive. *reason* The reason the erase failed
Default:	The device is not configured
Examples:	CONFIG(0x11223344) CONFIG(5)

8.29 STATUSBIT

Description:	Programs the Status Bit with a new value. Although an 8-bit value can be passed only bit 0 is used. Use for those devices that implement a Status Bit as opposed to a Status Byte, such as the 89LPC932.
More than one allowed:	Yes
Type:	Operation
Syntax:	STATUSBIT(*value*) Where: *value* new value for the Status Byte.
Output:	Status Bit programmed to *value* Or: Status Bit programming failed: *reason* (*value*) Where: *value* The value the Status Bit was programmed to (0 or 1) *reason* The reason Status Bit programming failed
Default:	The Status Bit is not programmed with a new value.
Examples:	STATUSBIT(1) STATUSBIT(0) STATUSBIT(0x01)

8.30 READSTATUSBIT

Description:	Reads the value of the Status Bit. Use for those devices that implement a Status Bit as opposed to a Status Byte, such as the 89LPC932.
More than one allowed:	Yes
Type:	Operation
Syntax:	READSTATUSBIT
Output:	Status Bit: *value* Or: Status Bit read failed: *reason* Where: *value* The value of the Status Bit (0 or 1) *reason* The reason the read failed
Default:	The Status Bit is not read.
Example:	READSTATUSBIT

8.31 EXECUTE

Description:	Causes the firmware to be executed.
More than one allowed:	Yes
Type:	Operation
Syntax:	EXECUTE(*address*, *options*) Where: *address* = the address to execute from (ARM devices only) *options* = the execution options (ARM devices only). Can be one of: ARM THUMB For non-ARM devices, set address and options to zero.
Output:	Execute complete Or: Execute failed: *reason* Where: *reason* The reason the execute failed.
Default:	The device is not reset.
Example:	EXECUTE

8.32 TIMEOUTS

Description:	Sets the timeouts used for ISP operations. Two timeouts are used: regular and long. The regular timeout is used for most operations. The long timeout is used for erasing and blank check operations. If the values used are less than the minimum allowed then the minimums are used, which are 4 seconds for the regular and 15 seconds for the long.
More than one allowed:	No
Type:	Configuration
Syntax:	TIMEOUTS(*regular*, *long*) where: *regular* — The regular timeout in seconds *long* — The long timeout in seconds
Output:	None
Default:	The default timeouts of 4 seconds for the regular and 60 seconds for the long are used.
Example:	TIMEOUTS(5, 35)

8.33 ERASEUSED

Description:	Erases the blocks used by a specific hex file
More than one allowed:	Yes
Type:	Operation
Syntax:	ERASEUSED(*path, protectisp*)

Where:

type	Path of the hex file to analyze for which blocks to erase.
protectisp	Set to PROTECTISP to stop the ISP code from being erased in devices that contain the ISP code in the main Flash memory. Set to NOPROTECTISP to allows ISP code to be erased. Ignored for other devices. It is strongly recommended to set this option to PROTECTISP.

Output: Erase complete (*type*)
Or: Erase failed: *reason* (*type*)

Where:

type	The type of erase – "USED"
reason	The reason erase failed

Default: No flash blocks are erased. No security bits are erased. No changes are made to the Boot Vector or Status Byte.

Examples: ERASEUSED(C:\Development\test.hex, NOPROTECTISP)

8.34 READADDLSECURITY

Description:	Reads the additional security bits.
More than one allowed:	Yes
Type:	Operation
Syntax:	READADDLSECURITY
Output:	The results are listed with each security bit on a separate line, with the format: *name*: *result* Or: Additional security bit read failed: *reason* Where: *name* The name of the bit, e.g. AWP *result* set or not set *reason* The reason security bit read failed
Default:	The additional security bits are not read.
Example:	READADDLSECURITY

8.35 ADDLSECURITY

Description:	Sets or unsets the additional security bits. Some devices support additional security bits that are not related to specific sections of flash memory. This directive allows those bits to be set or unset.
More than one allowed:	Yes
Type:	Operation
Syntax:	ADDLSECURITY(*awp*, *cwp*, *dccp*) Where: *awp* Set to 1 to set AWP, set to 0 to unset it. *cwp* Set to 1 to set CWP, set to 0 to keep it unchanged. *dccp* Set to 1 to set DCCP, set to 0 to unset it.
Output:	Additional security bit programming complete Or: Additional security bit programming failed: *reason* Where: *reason* The reason security bit programming failed
Default:	No additional security bits are set.
Examples:	ADDLSECURITY(1, 0x00, 0x01) SECURITY(0H, 1, 0x01)

8.36 READMISR

Description:	Reads the MISR of a block or the entire device.
More than one allowed:	Yes
Type:	Operation
Syntax:	READMISR(*type*)

Where:

type The Flash to read the MISR of, either a Flash block or the whole device. If type is a value in the range 0 to 15 then the MISR of that number Flash block will be read. If type is DEVICE then the MISR of the whole device will be read.

Output: MISR *result* (*type*)
Or: MISR read failed: *reason* (*type*)

Where:

result	The MISR value in hexadecimal prefixed with “0x”.
type	The Flash to read the MISR of. Either a decimal number in the range 0 to 15 or a hexadecimal number in the range 0x0 to 0x0F prefixed with “0x” or DEVICE.
reason	The reason the read failed

Default: The MISR is not read.

Examples:

READMISR(0x4)
READMISR(4)
READMISR(DEVICE)

8.37 READEEPROMSECURITY

Description:	Reads the EEPROM security bits.
More than one allowed:	Yes
Type:	Operation
Syntax:	READEEPROMSECURITY
Output:	The results are listed with each security bit on a separate line, with the format:

EEPROM page *b* bit *n*: *result*

Or: EEPROM security bit read failed: *reason*

Where:

n	1 – 3
b	0 - 7
result	set or not set or enabled or disabled
reason	The reason security bit read failed

Default:	The security bits are not read.
Example:	READEEPROMSECURITY

8.38 EEPROMSECURITY

Description:	Sets the EEPROM security bits. Note you cannot unset the security bits with this directive. Instead the ERASEEEPROMPAGE directive must be used.
More than one allowed:	Yes
Type:	Operation
Syntax:	EEPROMSECURITY(*page0bit1*, *page0bit2*, *page0bit3*, *page1bit0*, *page1bit1*, …)

Where:

page0bit0	Set to 1 to set security bit 0 of page 0, set to 0 to keep it unset
page0bit1	Set to 1 to set security bit 1 of page 0, set to 0 to keep it unset
…	
pagenbit2	Set to 1 to set security bit 2 of page n, set to 0 to keep it unset

Output:

EEPROM security bit programming complete
Or: EEPROM security bit programming failed: *reason*

Where:

reason	The reason security bit programming failed

Default: No security bits are set.

Examples: EEPROMSECURITY(1,0,1,1,0,1,0,1,0)

8.39 ERASEEEPROMPAGE

Description:	Erases a single EEPROM page in the device
More than one allowed:	Yes
Type:	Operation
Syntax:	ERASEEEPROMPAGE(*type*) Where: *type* The number of the page to erase.
Output:	EEPROM page erase complete (*type*) Or: EEPROM page erase failed: *reason* (*type*) Where: *type* The number of the page to erase, as passed to the directive. *reason* The reason the erase failed
Default:	The page is not erased
Examples:	ERASEEEPROMPAGE(0) ERASEEEPROMPAGE(0x5)

8.40 READEEPROMCRC

Description:	Reads the CRC of the EEPROM.
More than one allowed:	Yes
Type:	Operation
Syntax:	READEEPROMCRC(*page*) Where: *page* The number of the page to erase
Output:	EEPROM CRC *result* Or: EEPROM CRC read failed: *reason* Where: *result* The CRC value in hexadecimal prefixed with "0x". *reason* The reason the read failed
Default:	The CRC is not read.
Examples:	READEEPROMCRC

8.41 EEPROMHEXFILE

Description:	Programs an Intel Hex file into the EEPROM. Note that more than one instance of this directive is allowed on the command line.
More than one allowed:	Yes
Type:	Operation
Syntax:	EEPROMHEXFILE(*path*) Where: *path* The path to the Intel Hex File
Output:	EEPROM hex File programming complete (*path*) Or: EEPROM hex File programming failed: *reason* (*path*) Where: *path* The path of the Intel Hex file as passed to the EEPROMHEXFILE directive. *reason* The reason Hex File programming failed
Default:	No Hex file is programmed.
Examples:	EEPROMHEXFILE(test.hex) EEPROMHEXFILE(..\test.hex) EEPROMHEXFILE(C:\work\test.hex)

8.42 INTERFACE

Description:	Specifies the interface to use, if needed.
More than one allowed:	No
Type:	Configuration
Syntax:	INTERFACE(*interfacetype*) Where: *interfacetype* The interface to use. One of: NXPICPBRIDGE FDIUSBICP80C51ISP FDIUSBICPLPC9XX FDIUSBICPLPC2K FDIUSBDONGLE
Output:	None
Default:	No interface is used
Examples:	INTERFACE(NXPICPBRIDGE)

8.43 ETHERNET

Description:	Specifies the Ethernet parameters to use for communicating with the device.
More than one allowed:	No
Type:	Configuration
Syntax:	ETHERNET(*ipaddress*, *macaddress*) Where: *ipaddress* — The IP address to assign to the bootloader. Must have the format a.b.c.d where a, b, c and d are decimal values 0 to 255. *macaddress* — The MAC address of the bootloader. Must have the format aa-bb-cc-dd-ee-ff where aa, bb, cc, dd, ee and ff are two character hexadecimal values.
Output:	Connected Or: Connection failed: *reason* Where: *reason* — The reason the connection failed
Default:	192.168.0.20, 0c-1d-12-e0-1f-10
Examples:	ETHERNET(10.90.31.20, 0c-1d-12-e0-1f-10)

8.44 SET89V51SERIAL

Description:	Sets the 89(L)V51Rx2 serial number, used for security.
More than one allowed:	Yes
Type:	Operation
Syntax:	SET89V51SERIAL(*string*) SET89V51SERIAL(value) Where: *string* — An ASCII string to use as the serial number. Maximum of 31 characters. value — A hexadecimal value to used as the serial number. Must be prefixed with "0x". Maximum of 31 bytes.
Output:	SET89V51SERIAL programming completed Or: SET89V51SERIAL programming failed: *reason* Where: *reason* — The reason programming failed
Default:	No serial number is programmed.
Examples:	SET89V51SERIAL(myserialstring) SET89V51SERIAL(0x11223344556677889900)

Chapter 9 - 89LPC9xx Recommended Settings

9.1 Baud Rate

When using the internal RC oscillator, it has been found that 9600 baud works reliably most of the time. Some devices in the 89LPC9xx family which feed the watchdog during ISP operation work reliably at 7200 baud.

Recommendation: Try 9600 baud first. If it does not work or does not work reliably then try 7200 baud.

9.2 ISP Entry

By default the pulse entry method is turned on, as it is the most commonly used entry. If your circuit does not use DTR or RTS then you can turn this option off or ignore it. The option is located in the Advanced Options under the Hardware Config tab. If you are using an 89LPC932 revision C then you need to turn this option off.

Recommendation: Leave pulse entry turned on unless you are entering ISP mode using break detect or a custom method.

9.3 Oscillator Frequency

The oscillator frequency is entered into the box in section 1 of the main window. It is used for the high speed communications feature, present in some members of the 89LPC9xx family. Enter the oscillator frequency as accurately as possible. If you are using the internal RC oscillator, enter "7.3728" into the box. When using the watchdog timer, enter "0.4" into the box.

Recommendation: When using the internal RC oscillator enter "7.3728" into the box.

Chapter 10 - FlashMagic and IDEs

10.1 Introduction

Flash Magic may be integrated into development tool IDEs by using the command line functionality of Flash Magic.
This section gives examples on how various IDEs may be used with Flash Magic. Please let us know if you have integrated Flash Magic with IDEs not listed here.

10.2 Keil uVision

Flash Magic may be added to the Tools menu by completing the following steps:

1. Select Customize Tools Menu from the Tools Menu
2. Click on the New (Insert) button. An empty title box will be created.
3. Enter into the title box:
   ```
   Program device with %H
   ```
4. Select the title.
5. Click on the "…" button next to the Command box and browse to and select FM.EXE.
6. In the arguments box enter all the directives you wish to use. For the HEXFILE directive use "#H" in place of the Hex File path. For example you may enter:

   ```
   DEVICE(89C51RD2, 20.000) HEXFILE(#H, NOCHECKSUMS, NOFILL) COM(1, 19200)
   ```

7. Check the Run Minimized option.
8. Click on OK

You will now have an entry on the Tools menu that when selected will program the Hex file for the current project.

Chapter 11 - Settings Files

Although the settings made in Flash Magic are automatically saved when Flash Magic is closed, it is possible to save the settings in separate Flash Magic Settings files (extension .fms), allowing easy use of Flash Magic when using many different targets at the same time.

To save the current settings in a file choose Save Settings… from the File menu.

To open a settings file either choose Open Settings… from the File menu or double-click/open the settings file.

Chapter 12 - Miscellaneous Features and Options

12.1 Enabling and Disabling Embedded Hints Update

The Embedded Hints are the constantly changing internet links displayed at the bottom of the Flash Magic main window.

The Embedded Hints update feature allows Flash Magic to periodically check for and download new hints that contain useful information such as:

- Alerts telling you a new version of Flash Magic is available
- Upcoming events where you can meet NXP and Embedded Systems Academy
- Alerts telling you about new Flash programming related items available on the ESA web site

The update feature does **NOT** transmit any personally identifiable information.

The Embedded Hints update feature can be turned off by choosing Disable Hints Update… from the Options menu. It can be re-enabled at any time by choosing Enable Hints Update from the Options menu.

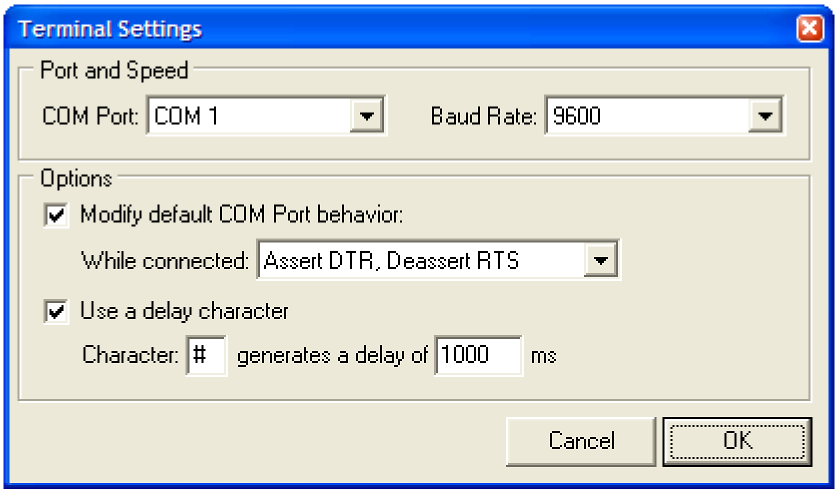
Note that with the update feature turned off, the hints you currently have will still be displayed.

Chapter 13 – Terminal Interface

Flash Magic features a simple terminal interface, designed to communicate with a microcontroller during the testing of firmware. Note that it is not a comprehensive implementation, and if VT100, character swapping, colors and other features are required, we recommend using TeraTerm.

To start the terminal interface choose Terminal… from the Tools menu.

Initially the configuration window will be displayed.



The default setting when the COM port is opened for the terminal interface, is that DTR and RTS are both asserted. This is standard COM port behavior. Because sometimes target hardware requires certain states for DTR and RTS it is possible to configure the state they will be in while the COM port is opened.

To do this check "Modify default COM Port behavior" and choose the appropriate option from the drop down list.

The terminal interface allows cut and pasting of blocks of text into it. In order to allow commands to be pasted a delay character is supported. When the option is enabled and the delay character is entered into the terminal window, Flash Magic will pause sending the rest of the data for the specified time period.

For example, suppose that the delay character was enabled and set to '#' with a delay of 1000ms. Pasting the following into the terminal window:

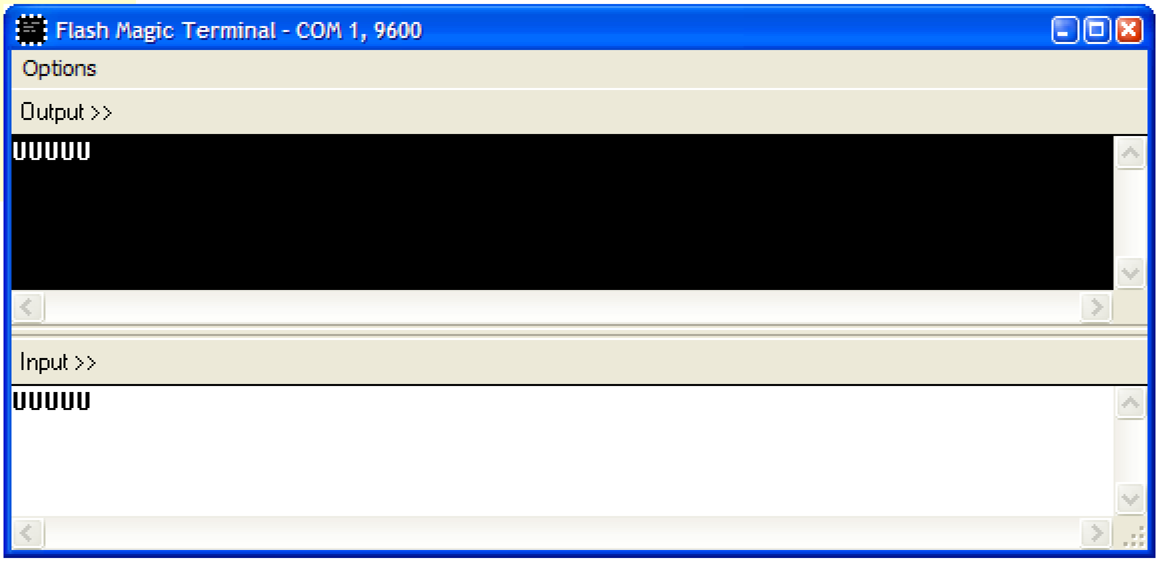
```
1#S400##3
```

would result in '1' being sent out of the COM Port followed by a delay of 1000ms, followed by 'S400' being transmitted, followed by a delay of 2000ms. followed by '3 being transmitted.

Note that the delay character itself is not transmitted, so any delay character selected must be a character that is not used in the serial command set implemented in the firmware.

Connections are always made with eight bits, no parity, one stop bit and no flow control.

Once the settings have been accepted the terminal window will open.



The top part of the window shows the output from the device. The bottom part is where characters are entered for transmission. The sizes of the areas may be adjusted by resizing the window and dragging the splitter located between the output and input areas.

To transmit data, simply type or cut and paste into the input box. Output will appear in the output window. By keeping the input and output separate, it is possible to copy the input and output into documentation such as test reports. It also allows for the replaying of inputs simply by copying it, saving it in a file for pasting later and perhaps adding delay characters if needed.

To change the settings choose Settings… from the Options menu. To one or both of the input and output areas, choose the relevent entry from the Options menu.

The COM port is open whenever the terminal window is open, so there is no need to click on connect or disconnect buttons. Window content and settings are retained when the window is closed, allowing quick and easy switching between the terminal window and the ISP operations of Flash Magic.

Chapter 14 – Scripts

The Flash Magic Production System contains support for scripts written in the Python programming language. The scripts can be executed by Flash Magic and have full access to the low level API, allowing any ISP operation to be performed. This allows automated testing and programming to be performed, for example on a production line.

Simple scripts may be combined into more complex test scripts, culminating in complete test suites.

Scripts are simple ASCII files that can be easily shared between users.

14.1 Environment

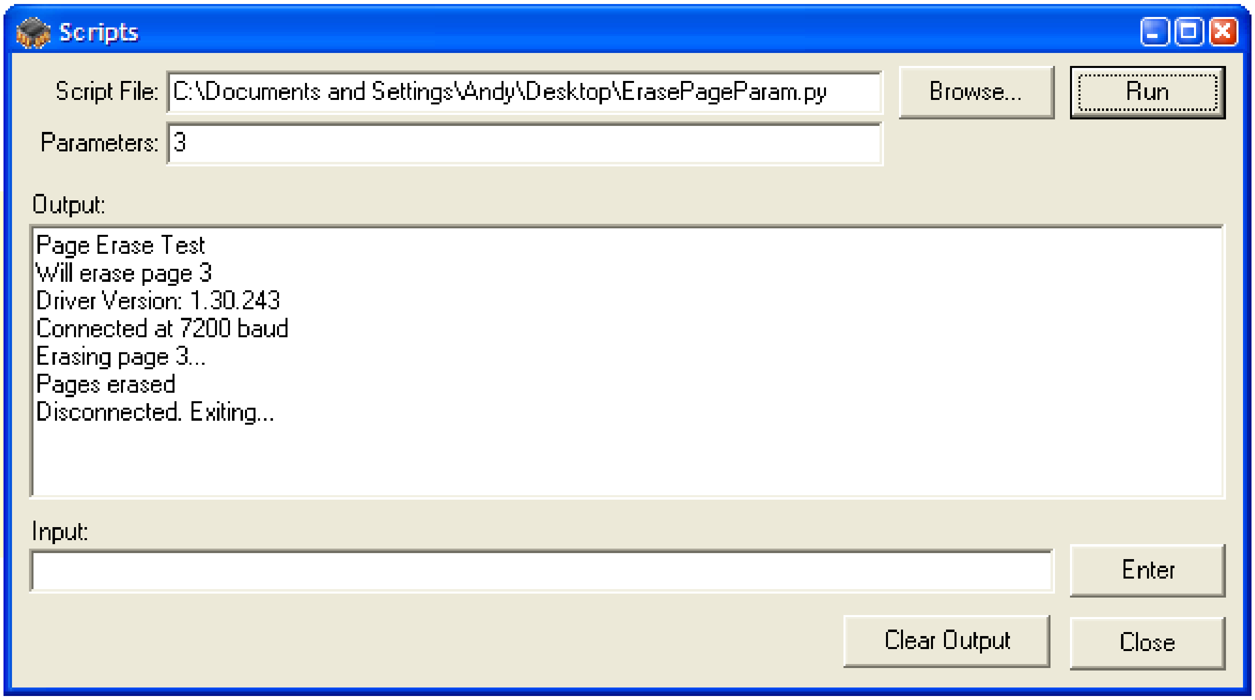
The scripts environment uses Python 2.4 with the complete standard libraries available. This allows scripts to not only perform ISP operations, but generate test report files, construct custom hex files, access the internet for test data, commands or download hex files, etc.

All output from the script is displayed in a window and the user can be prompted for any input needed, for example entering test parameters or choosing from a menu of options.

Scripts can also have user defined parameters passed to them, allowing a script to be fully automated but with customizable options.

14.2 Getting Started

The scripts interface is available by choosing Scripts… from the Tools menu.



Choose the script file to run by clicking on the Browse… button. Enter any necessary parameters into the Parameters box and click on Run. The script can be stopped at any time by clicking on the Stop button.

The output from the script appears in the Output box and user input goes into the Input box. To send input to the script either press Enter/Return or click on the Enter button.

14.3 Script Execution

A minimal script defines two functions, Cancel and Main. The Main function is executed when the user clicks on the Run button. The Cancel function is executed when the user clicks on the Stop button.

Here is a minimal script that prints "Hello World":

```
import FlashMagic

# specify driver to use
FlashMagic.Driver = "flashmagic.dll"

# global cancel flag
cancel = 0

def Cancel():
  global cancel
  cancel = 1
```

```
def Main():
  global cancel
  print "Hello World"
```

Indentation in Python is important and takes the place of curly braces in C.

The import FlashMagic line tells Python that we want to use the Flash Magic module. This is a custom module that provides Python with access to all the ISP functionality available in Flash Magic, known as the Application Programmer Interface (API).

The driver line tells the Flash Magic module which DLL should be used. **This must be set one time and before any functions are called in the Flash Magic module.** It is not currently possible to change the driver during the execution of a script.

A global cancel flag is defined and set to zero.

The Cancel function simply sets the cancel flag when called. The Main function prints the hello world message and exits.

In a real script the Main function should perodically check the cancel flag to see if it is set. If it is, then the function should exit. This is important in order to allow the user to stop a running script. The more frequently the cancel flag is checked the more responsive the script will be. An example check would be:

```
if cancel == 1:
  return
```

14.4 Flash Magic API

The FlashMagic module implements the low level Flash Magic API. This is a collection of constants and functions which allow ISP to be used. The Python object is a simple wrapper around the C functions.

For details of the API please refer to the DLL Manual, which is available on the Start menu for users of the Flash Magic Production System.

14.5 Windows API

The scripts environment contains support for a module called Windows. Currently this is a very simple module that allows a script to prompt the user for a file to open or for a location to save a file to.

GetOpenFileName

Example to prompt the user for a file to open:

```
import Windows

filename = Windows.GetOpenFileName()
if len(filename) == 0:
  print "You must select a file to open"
  return

print "File name:", filename
```

GetSaveFileName

Example to prompt the user for a location to save to:

```
import Windows

filename = Windows.GetSaveFileName()
if len(filename) == 0:
  print "You must select a file to save to"
  return

print "File name:", filename
```

Note that these functions do not actually open or save files.

GetAdapterIndex

Gets the index of an Ethernet adapter. Used for calls to fm_connect when using an Ethernet bootloader. Example:

```
import Windows

index = Windows.GetAdapterIndex()
```

14.6 HexFile API

The scripts enviroment contains support for a module called HexFile. This module contains functions for manipulating Intel hex records.

DataToHexRecord

This function converts record type, address and a list of data into a hex record. The checksum is automatically calculated. Example:

```
import HexFile

type = 0x01
address = 0x1000
data = [0x11, 0x22, 0x33]
```

```
record = HexFile.DataToHexRecord(type, address, data)
```

HexRecordToData

This function converta a hex record in the form of a string into record type, address and a list of data. The checksum in the record is checked to make sure it is correct. Example:

```
import HexFile

recordparts = HexFile.HexRecordToData(record)
type = recordparts[0]
address = recordparts[1]
data = recordparts[2]
```

14.7 Examples

Example scripts are provided on the Start menu for users of the Flash Magic Production System.

Chapter 15 – Ethernet Bootloader

This chapter provides an overview of the Ethernet bootloader for LPC17xx/LPC2xxx devices, including a quick start guide.

The Ethernet bootloader has been developed by NXP and is bundled with Flash Magic. To access the files go to Start Menu -> All Programs -> Flash Magic -> Ethernet Bootloader. The bootloader will run on any LPC17xx/LPC2xxx device with an on-chip Ethernet controller.

The bootloader is provided in the form of a pre-built hex file and Keil Realview project. Also included is a sample blinky application and an application note from NXP. For full details of the bootloader please refer to the application note.

Note that Vista/7 users will need to run Flash Magic as an administrator in order to use the Ethernet bootloader. This is achieved by right-clicking on the application and choosing "Run As Administrator" from the menu. Administrator rights are required for the IP address assignment.

15.1 Quick Start For Keil MCB1700/MCB2300

The following are steps to quickly get started using the bootloader with the Keil MCB1700 and MCB2300 boards. For other boards please see the later sections of this chapter on changes that might be needed.

Initially the MCB1700/MCB2300 board should be set up so that the device can be programmed using the UART (COM0 connector). This is required in order to program the Ethernet bootloader into the device.

Select to erase all flash and program the pre-built ethernet_bootloader.hex. This file can be found in the Bootloader/Obj folder inside the Ethernet Bootloader folder, accessible from the Start Menu. The bootloader occupies the first two flash blocks of the device.

Once programmed disconnect the serial cable and connect an Ethernet cable.

If you are using a LPC175x then set jumper E/U to connect positions 2 and 3 and set jumper E/C to connect positions 2 and 3. This connects pin P2.8 to MDC and pin P2.9 to MDIO for the software-based MIIM layer. If you wish to use a different evaluation board with different GPIO pins for MDC and MDIO then you will need to edit the init_emac function and mdio.c. Note that this is not required for the LPC176x devices as they use a hardware-based MIIM layer.

Reset the board to execute the bootloader. LED P2.6 (MCB1700) or P2.7 (MCB2300) should start flashing to indicate the bootloader is executing.

In Flash Magic select the same device but with the "Ethernet" suffix, for example "LPC2388 Ethernet". This will display the IP and MAC address options in Step 1 of the main window.

The MAC address must match the value hard-coded into the bootloader. Set the MAC address to the default value of "0c-1d-12-e0-1f-10". Note that MAC addresses are a set of six two-digit hexadecimal values separated by hyphens.

Initially the bootloader does not have an IP address, so it must be assigned one. This is achieved by entering the desired IP address into Flash Magic. IP addresses are four decimal values separated by periods. You must pick an address that is on the same subnet as your PC and is not currently used by any other device on the network. For example if your PC has the IP address "192.168.0.10" then you might enter "192.168.0.11". Note that only the last value in the address is changed. This indicates the address is on the same subnet.

Select "Erase Blocks Used By Hex File", and select the Blinky.hex example application hex file. The hex file can be found in the User Code Sample Blinky\Flash folder inside the Ethernet Bootloader folder, accessible from the Start Menu. Program the hex file into the device.

Reset the device to execute the Blinky example application.

To get back to the bootloader use the UART bootloader to erase block 2 then reset the device. This method is used only for illustrative purposes for this quick start guide. See the NXP application note for details of other options, such as using CRP and a dedicated bootloader entry pin.

15.2 IP Address

The IP address is only assigned to the bootloader while Flash Magic is performing programming operations. Once programming operations have finished the IP address is no longer assigned and a different one may be chosen.

To find the current IP address of your PC go to Start Menu -> Run and enter "cmd" into the box. Click on "OK". At the command prompt in the window that opens enter "ipconfig" and press Enter. The IP address is shown on the line starting with "IP Address".

15.3 Bootloader Configuration

The default settings for the bootloader will likely need to be changed. The settings are contained in sbl_config.h, and Keil's uVision provides an easy to use wizard for changing the settings. Open the project in uVision, open the sbl_config.h file and click on the "Configuration Wizard" tab at the bottom of the editor window.

The method of activating the Ethernet bootloader can be controlled using the Code Read Protection setting and/or the Update Entry Pin. The Update Entry Pin must be different to the one used for the UART bootloader. Additionally the Ethernet bootloader always executes if no user code is found in block 2 of the device.

See the NXP application note for full details of the bootloader configuration.

15.4 Protocol Timeout

Embedded Systems Academy has added an additional configuration option that is not described in the application note. The option is found in sbl_config.h and it is called "Timeout", under the "Protocol Options" heading.

When the bootloader first executes it waits for handshaking commands. Once the handshaking has completed the bootloader will only respond to packets from the same PC that performed the handshaking.

If no packets are received for the length of the timeout period then the bootloader will go back to waiting for handshaking commands. This ensures that if communications are disrupted or become confused for some reason the bootloader will automatically return to a known state.

The value is specified in milliseconds and has a maximum value of 1000000ms (16.67 minutes).