AN10986

USB In-System Programming with the LPC1300

Rev. 1 — 24 September 2010 **Application note**

Document information

Info	Content
Keywords	LPC1300, USB, In-System Programming (ISP), Mass Storage Class (MSC), Cortex-M3, Windows, Apple OS-X, Linux
Abstract	This application note explains how to use the on-chip USB In-System Programming feature of the LPC1300 Cortex-M3 based microcontroller to update firmware in on-chip flash memory on Windows, Apple OS-X, and Linux operating systems.

Revision history

Rev	Date	Description
1	20100924	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The LPC1300 microcontroller family is based on the ARM Cortex-M3 CPU architecture for embedded applications featuring a high level of support block integration and low power consumption. The peripheral complement of the LPC1300 series includes up to 32 kB of flash memory, up to 8 kB of data memory, USB Device interface, 1 UART, 1 SSP controller, SPI interface, I2C interface, 8 channel 10-bit ADC, 4 general purpose timer/PWMs, and up to 40 general purpose I/O pins.

Also present is an on-chip ROM containing In-System Programming capability (a bootloader) supporting UART and USB flash programming, as well as APIs for user code. The flash API implements a simple interface to the on-board flash programming functionality and allows entry to ISP mode at any time. The USB API supports development of Human Interface Devices (HID) and Mass Storage Class (MSC) devices without requiring driver code to be written by the customer or stored in Flash.

The various topics covered in this application note are as follows:

1. USB In-System Programming Overview
2. USB ISP Details
3. Automating USB ISP
4. Automating entry of USB ISP
5. Sample Software
6. Conclusion

2. USB In-System Programming (ISP) Overview

The LPC1300’s on-chip USB ISP firmware enables programming and updating of firmware in the field by end users using standard personal computer operating systems. This document will reference the LPC1343 in particular, but the procedures should also apply to other LPC1300 family products with on-chip USB.

Holding PIO0_1 low during power-up will trigger the on-chip ISP firmware to enter ISP mode (unless it is disabled by the NO_ISP code read protection [CRP] mode). Once ISP mode has been entered, the USB VBUS line PIO0_3 is checked. If high, then USB ISP will be entered. If low, UART ISP will be entered instead. The diagram in the User’s Manual titled “Boot Process Flowchart” explains this process in greater detail.

Upon entry to USB ISP mode, the LPC1300 part will enable the on-chip USB full-speed interface as a mass storage class device. This disk device will contain a FAT12 filesystem which will appear as a standard disk device in most operating systems. The label of the disk will indicate the CRP status and the disk will contain a single file, firmware.bin. Deleting and rewriting this file will write to the flash memory if allowed by the code protect settings. Reading the contents of flash memory is as simple as copying the `firmware.bin` file.

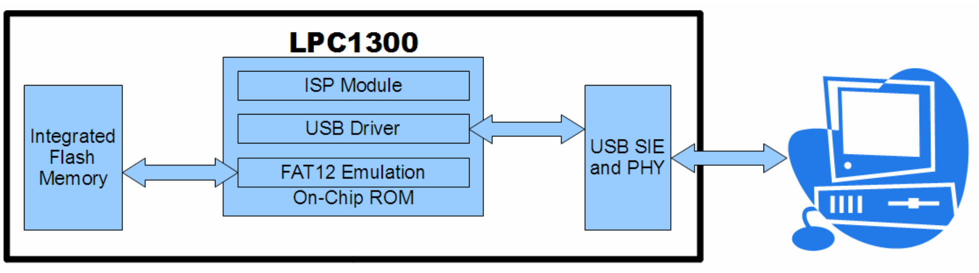
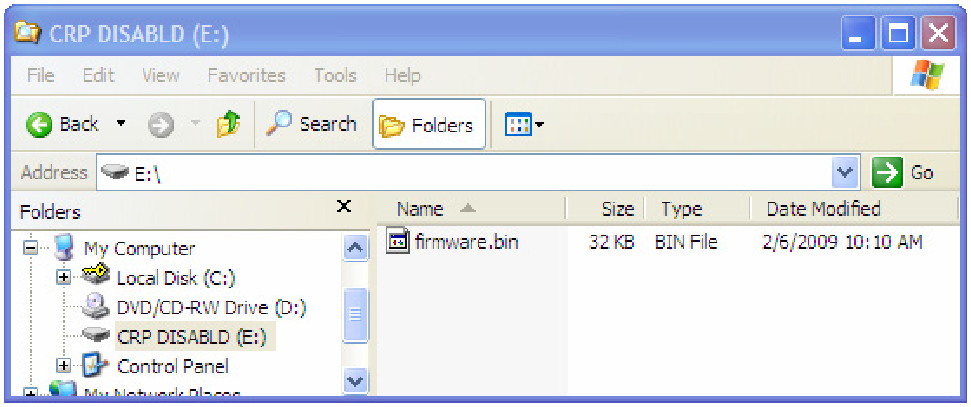


Fig 1. LPC1300 USB ISP system



Name	Size	Type	Date Modified
firmware.bin	32 KB	BIN File	2/6/2009 10:10 AM

Fig 2. USB ISP- firmware.bin file as seen in Windows XP

3. USB ISP details

The LPC1300’s on-chip USB ISP firmware emulates a FAT filesystem to facilitate firmware reading and writing by PC software. We will describe details of the filesystem emulation and techniques to update the firmware using several standard host operating systems.

When LPC1300 is connected to a USB host and USB ISP mode is initiated, it enumerates as USB Vendor ID `0x04CC` and Device ID `0x0003`. This information is usually hidden from the end user, but it can be used by firmware updating software to find the LPC1300 among the USB devices connected to a PC. Once the device has been found, a Mass Storage Class Inquiry command can be sent. This will return a string describing the LPC microcontroller product attached. The Inquiry string for the LPC134x products reads “`NXP LPC134X IFLASH 1.0`” The device can also be recognized by its device vendor which is “`NXP`” and device model which is “`LPC134X IFLASH`.” Finally, there is a model ID which is “`NXP_LPC13XX_IFLASH`”

The disk emulated by the LPC1300 has different volume labels depending on the Code Read Protect (CRP) settings and slightly different behavior during reprogramming. A brief summary of this behavior is printed below, but the complete documentation is available in the LPC1300 User's Manual UM10375.

If CRP1 or CRP2 is enabled, the user flash is erased when the file is deleted and reprogrammed when the new file is copied.

If CRP1 is enabled or no CRP is selected, the user flash is erased and reprogrammed when the new file is copied. However, only the area occupied by the new file is erased and reprogrammed. Because of this, ideally, the new programming file would contain the full flash contents (32KB for the LPC1343) so that all of the flash would be in a known state. Using a padded (32KB or flash size) programming file also provides a means for the programming tool to error-check that the correct LPC part is connected (by comparing file sizes of the new firmware and the firmware.bin on the device) without additional configuration information.

Remark: The only Windows commands supported for the LPC1300 flash image folder are copy and delete. Overwrite operations using the Windows Explorer will not be successful as there is insufficient room on the emulated disk to store the temporary file created by Windows during the overwrite process.

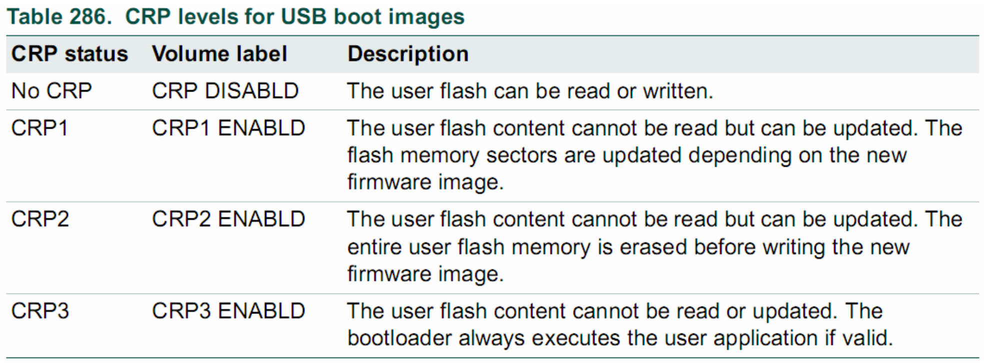


Table 286. CRP levels for USB boot images

CRP status	Volume label	Description
No CRP	CRP DISABLD	The user flash can be read or written.
CRP1	CRP1 ENABLD	The user flash content cannot be read but can be updated. The flash memory sectors are updated depending on the new firmware image.
CRP2	CRP2 ENABLD	The user flash content cannot be read but can be updated. The entire user flash memory is erased before writing the new firmware image.
CRP3	CRP3 ENABLD	The user flash content cannot be read or updated. The bootloader always executes the user application if valid.

Fig 3. LPC1300 User's Manual- CRP disk volume labels

The FAT filesystem emulated by the boot ROM consist of a single file called `firmware.bin` which contains the entire flash contents of the part. Normally, a disk must be slightly larger than the required storage due to the directory and allocation table overhead of the filesystem. In the case of the LPC1300's emulated FAT filesystem, four extra blocks are needed for the boot block, root directory, and file allocation table. Because it is required to be able to program all of the flash memory in the MCU using ISP, these extra blocks are emulated using data from RAM and ROM rather than being mapped into the flash used for code storage. Because of this, no filesystem metadata is saved when power is lost- only data programmed into flash via file writes is saved.

The data written to the filesystem is organized in flash by disk block order, with the beginning of flash starting at block 4. If firmware.bin is deleted, PCs running windows will allocate any new file starting at block 4 and using increasing block numbers as more data is written. This means that in Windows, any standard program or tool can be used to write new firmware to the LPC1300. In a Windows Explorer window, a user can delete `firmware.bin` and drag over a new file to program the flash. Unfortunately, FAT filesystems on Mac and Linux machines tend to allocate blocks to files in a different order which results in data being written onto the ISP disk, and consequently firmware being written to flash, being reordered. This will cause the firmware update to be unsuccessful. There are two workarounds for this. The most general workaround is to overwrite the `firmware.bin` file in place. A more “brute-force” option that requires administrative privileges to do direct disk device writes to `/dev`.

4. Automating USB ISP

Sometimes a system requirement is for firmware updates to be performed without user intervention. Asking a user to determine which disk drive links to a USB device, or asking them to manually delete and rewrite a file, is often too complex. Firmware often needs to be updated automatically under control of a PC program. This section will describe how this can be accomplished. To fully automate this process on the PC side, a program should find the correct USB device to be updated, convert the USB device “handle” into a file path, check the CRP mode, write to the file to program flash, and finally unmount the disk device to ensure that the flash contents are written. These steps will be discussed separately. It can also be useful to validate the firmware file to be programmed. This can be verified using the “Criterion for Valid User Code” in the LPC134x User’s Manual, which is implemented in the Windows ISP tool. It is also possible to validate that the file is the same length as the flash space on the device to be programmed. This is implemented on the Mac and Linux ISP examples. Note: Some deviation from these exact steps is okay to simplify implementation on various operating systems.

4.1 Finding the correct USB device

The technique used to enumerate USB devices varies depending on the operating system. On Linux, most distributions provide a program called `lsusb` which can search for a USB device with a specific Vendor ID and Product ID (`04CC:0003`). Afterward, `udevadm` (present on systems using the `udev` device filesystem) can be used to list the USB Model ID and system device path. The Model ID should be “`NXP_LPC13XX_IFLASH`” for the LPC1300 family. Once the Vendor ID, Product ID, and Model ID are confirmed, you are assured to have found an NXP LPC1300 microcontroller that is in ISP mode. Under Windows and Mac OS-X, this step is a little bit different. Since the USB device model information is available from the disk volume database on the Mac, no attempt is made to find the actual USB device. Instead, the disk devices are checked until one with the correct Vendor and Device Model is found. A similar process is used under Windows, with the help of the `iTuner UsbManager` class to retrieve disk information under Windows.

4.2 Converting a USB device handle into a file path

In Linux, a USB device handle can be converted into a file path by iterating through all of the disk drives in the `udev` filesystem and querying `udevadm` for their system device paths until a match is found with the ISP USB device.

1. `lsusb` is called with the Vendor ID and Product ID of the NXP ISP device (`04CC:0003`). `lsusb` outputs the bus and device ID to standard out (the console). This output is captured and used to create a `udev` path for the USB device. For example, bus 2 device 3 would be `/dev/bus/usb/002/003`.
2. A disk device name is pulled from `/dev`. In our sample script we use a wildcard match to `/dev/sd[a-z]` and loop through all devices matching this pattern- `/dev/sda`, `/dev/sdb`, `/dev/sdc`, etc. This will find all disk drives on the system including hard disks as well as USB mass storage devices.
3. The complete low-level disk device file path is looked up with `udevadm info -q path -n /dev/sd`*`X`*. The file path returned should begin with the low-level usb device file path if that disk device is associated with the USB device we found.
4. Finally the output of the `mount` command is used to determine where the disk device name tested in step 2 is mounted in the linux filesystem. Now we have a filesystem path for our USB ISP device and can access `firmware.bin`.

Under Mac and Windows, we start out with a disk handle (instead of a USB device handle) and query the operating system for the disk letter or filesystem path.

4.3 Checking for Code Read Protection (CRP)

The LPC1300 USB ISP feature sets the label of the USB disk to indicate CRP mode. A robust design for a USB ISP flash programmer would check the disk label to make sure that CRP is not enabled. In Linux, FAT disk filesystem labels can be read with the `mtools` package. Unfortunately, this is not recommended because it requires administrative access. Another way to read filesystem labels in Linux is to use the `mount -l` command if it is supported by your system’s version of `mount`. This will list all of the mounted filesystems and their labels. On the Mac and on Windows the disk label information shows up in the `DiskDescription` dictionary or in the `Volume` object retrieved from `WMI` respectively.

If CRP 1 or CRP 2 is enabled, the `firmware.bin` file on the USB ISP disk must be deleted to disable code protection before new firmware can be written. After deleting `firmware.bin`, the device needs to be powered down and reconnected in order for the change to CRP settings to take effect so that the firmware can be updated. If CRP 3 is enabled, the device cannot be erased and the firmware cannot be updated.

4.4 Writing the file into program flash

In Windows, almost any standard filesystem write sequence will work properly as the Windows variant of the FAT filesystem allocates blocks sequentially beginning at the first free block on the ISP disk. In Linux, the device itself could be written directly (in `/dev`) using `dd` to ensure block ordering. This technique is not recommended because it requires that the user have administrative permissions. A second option is to open the existing `firmware.bin` file on the USB ISP disk and overwrite the contents. If the file is opened without truncation, then the new data will be written with the same order as the existing file that is set up by the ISP firmware. (remember the “file metadata” is created by the rom code and not stored in flash) Overwriting can be accomplished using `dd` to write to `firmware.bin` with the `conv=nocreat,notrunc`.option. If coding in the C language, it is possible to use `open(path, O_RDWR)` or `fopen(path, “r+”)`. Again in Windows there is no requirement to overwrite the existing file, so any file write scheme can be used as long as it either overwrites or deletes the current file so that free space is available to write the new data. In our Windows sample program we use the C# function `System.IO.File.WriteAllBytes(filePath, firmwareData)` which does a truncate operation before writing.

4.5 Unmounting the disk device

Most operating systems will eventually complete writes to disk after a delay, but forcing an unmount is a helpful final step to ensure that the operating system has finished writing data to disk so the user can be informed that it is okay to remove the device. For the unmount operation to succeed, all open files including the `firmware.bin` file must be closed first. In Linux, the posix standard `umount` command can be used. Usually no administrator access is required to unmount automatically mounted USB mass-storage devices. On the Mac, since mounting is handled automatically by the Disk Arbiter, the best route is to use the `DADiskUnmount` system call. On Windows, most users are used to unmounting disks themselves or the `CM_Request_Device_Eject` function in the `SetupAPI` can be used to do it for them. Our sample Windows ISP updater does not unmount the device when it is finished, instead it displays a dialog reminding the user to do it.

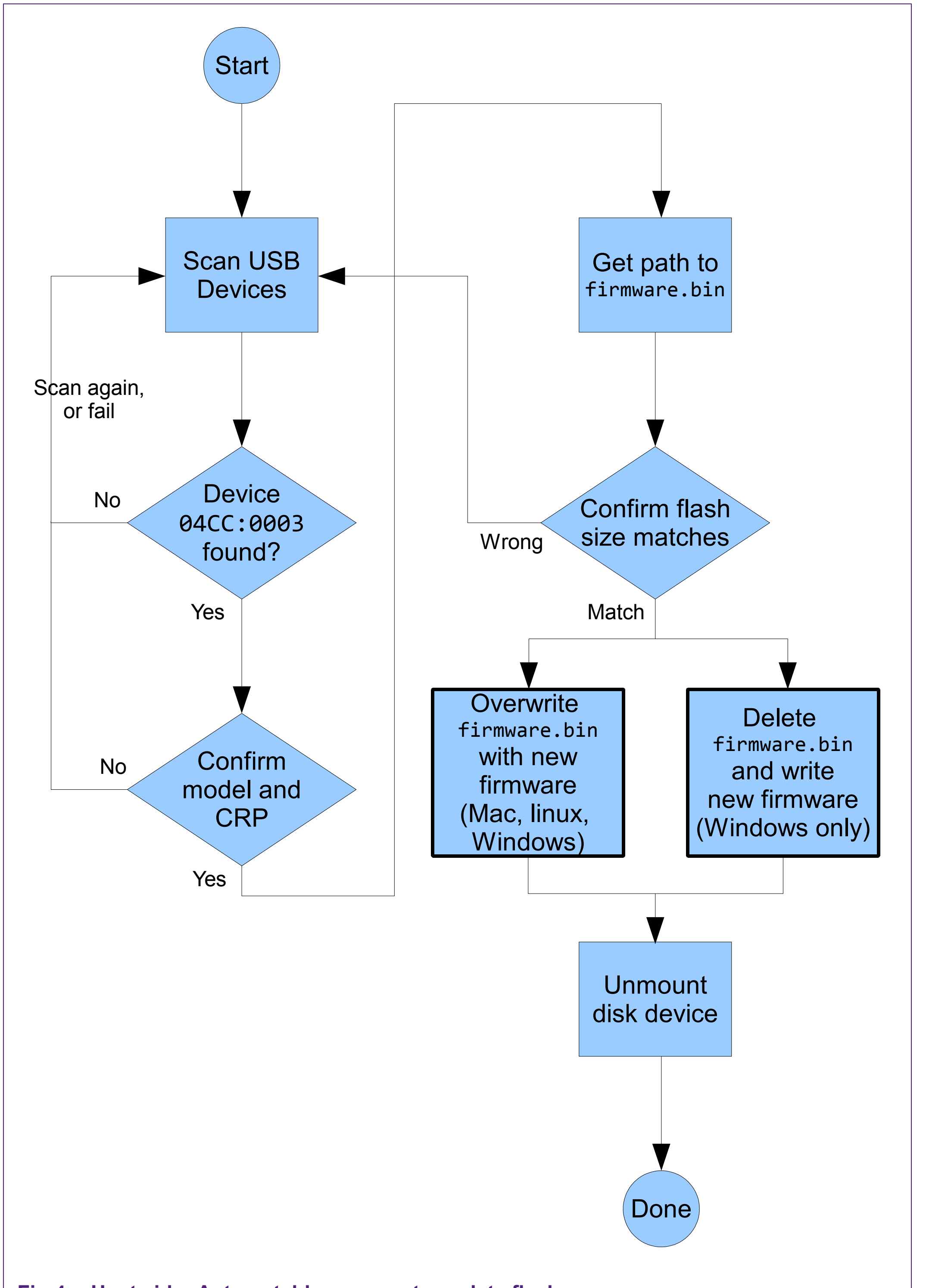


Fig 4. Host side- Automatable process to update flash

5. Automating entry of USB ISP

In the previous section we explained the PC process to update the firmware of a device once it has already been placed in USB ISP mode. NXP LPC1300 microcontrollers enter ISP mode by default when no firmware is programmed, or they can be placed in ISP mode by pulling a pin low (PIO0.1 on the LPC134X family) and resetting or power-cycling the device. Sometimes it is not desired to require a user to press a button while connecting the device to invoke ISP mode for firmware updates. In this case, the firmware can be designed to allow programmatic entry into ISP mode.

Programmatic entry into ISP mode is accomplished with a call to the In-Application Programming API in the LPC1300 on-chip ROM. After the firmware is updated, the user will need to power-cycle the device in order to start the new firmware. Alternately the firmware can pre-configure the watchdog timer to reset the LPC1300 after the new firmware has been downloaded. A flow chart is printed below showing the device-side automated ISP process. A software example is provided called “autoisp” to demonstrate this technique.

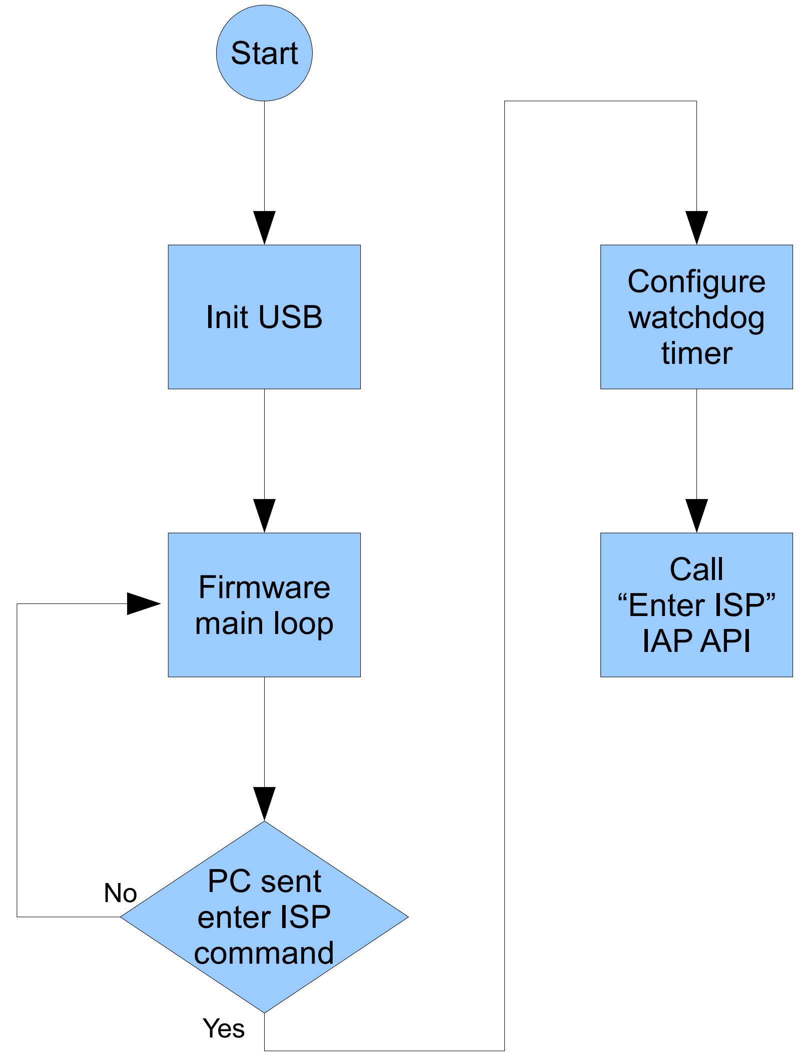


Fig 5. Target side- Automatable process to update flash

6. Sample software

To accelerate the process of building a customer-friendly firmware updating tool, NXP has released three implementations of an ISP download program and provided source code. A Windows Forms-based C# application running in Windows XP, Windows Vista, and Windows 7 is provided. The Windows application was developed using Microsoft Visual Studio C# Express 2010. A command-line tool written in C is provided for the Mac. It was developed in Apple Xcode and runs on OS-X 10.5 or higher. Finally, a bash shell script is provided for Ubuntu Linux. It has been tested in Ubuntu 10.04 and may work in other variants of Linux that have the same underlying `udev` filesystem and tools installed.

All of these tools have open unrestricted licenses for reuse except the Windows application, which depends on a USB disk device class library from the iTuner (http://ituner.codeplex.com/) project. It is covered under an open-source license that requires disclosure of the source code of derivative works.

Also included with this application note is a blinky program implementing code read protection. This program, developed under the LPCXpresso IDE, will flash the LED on the LPCXpresso board and count numbers on the 7-segment LED display on an Embedded Artists base board. There are nine compiled versions of the program included so that a test device can be repeatedly reprogrammed with various CRP modes and LED blinking/counting rates for testing.

A program called “autoisp” is included. This sample demonstrates how to enter ISP mode under program control. It flashes an LED for ten seconds and then ISP mode is entered. If the device is connected to a PC, it should enumerate and the firmware should be flashed. During this time, while the LPC1343 is in ISP mode, the watchdog timer continues to operate. After it times out, the LPC1343 will reset and run the newly flashed firmware.

Finally, a small command-line program called “padto” is included. This program is used for taking the binary output from development tools, and padding them with 0xFF bytes until they match the size of the flash memory in the LPC microcontroller being used. This size matching is useful to help the ISP download program on the PC host ensure that the firmware is designed for the particular LPC134x part connected to USB.

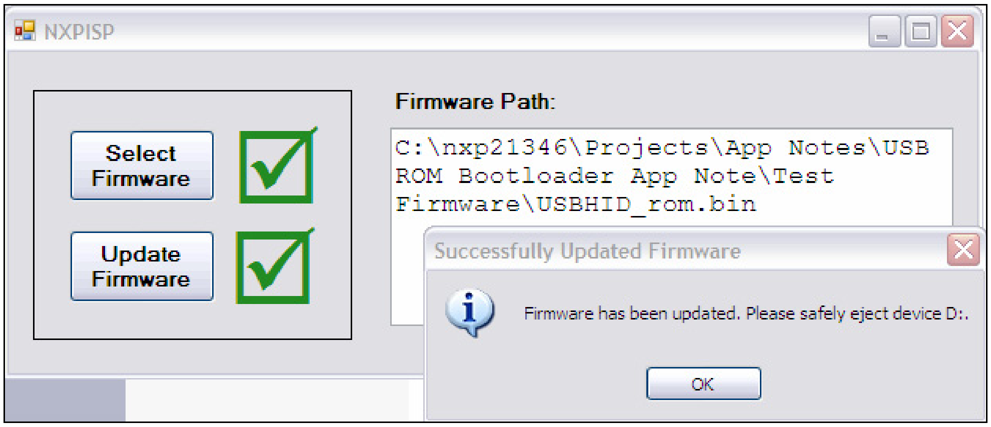
Below are screen shots and descriptions of each of the three firmware downloading tools.

6.1 Windows NXPISP utility

This program is based on Microsoft .NET, and when the setup.exe program is run to start it, the bundled Visual Studio Installer will check the PC and download the latest version of .NET if it is not already installed. **For this reason, make sure you are connected to the Internet through a broadband connection the first time you run this setup.exe program.** After the Windows .NET subsystem is updated, the NXPISP program will be installed and automatically started. Once NXPISP is running, click “Select Firmware.” A file chooser dialog will pop up and allow you to select a .bin file. This .bin file will be checked against the “Criterion for Valid User Code” which is simply that the first 8 32-bit words of the vector table sum to zero. If the .bin file looks good, a green checkmark will be displayed to the right of the “Select Firmware” button. Otherwise, a red X will be displayed.

To actually program a device, click “Update Firmware.” The “Update Firmware” button works like a toggle and can stay depressed when clicked or release when it is clicked again. When the button is pressed, the PC will be scanned for NXP ISP devices. If one is found, its firmware will be updated and a green checkmark will be displayed to the right of the “Update Firmware” button, and the button will release. If no devices are found, the “Update Firmware” button will stay depressed and the tool will wait for a device to be connected. After a device is found and updated, a green checkmark will be displayed and the “Update Firmware” button will pop out. If the utility is awaiting a device connection (“Update Firmware” button is still depressed) it can be canceled by clicking on the “Update Firmware” button again and causing it to pop out.

Once the update has completed, it is important to safely eject the USB ISP device to ensure that the new firmware has been written to it since the Windows NXPISP tool does not unmount the device. Safe ejection can be performed with Window’s eject hardware icon or the eject option in the Windows Explorer. Failure to complete this step may result in corrupt firmware.



Firmware Path:

C:\nxp21346\Projects\App Notes\USB ROM Bootloader App Note\Test Firmware\USBHID_rom.bin

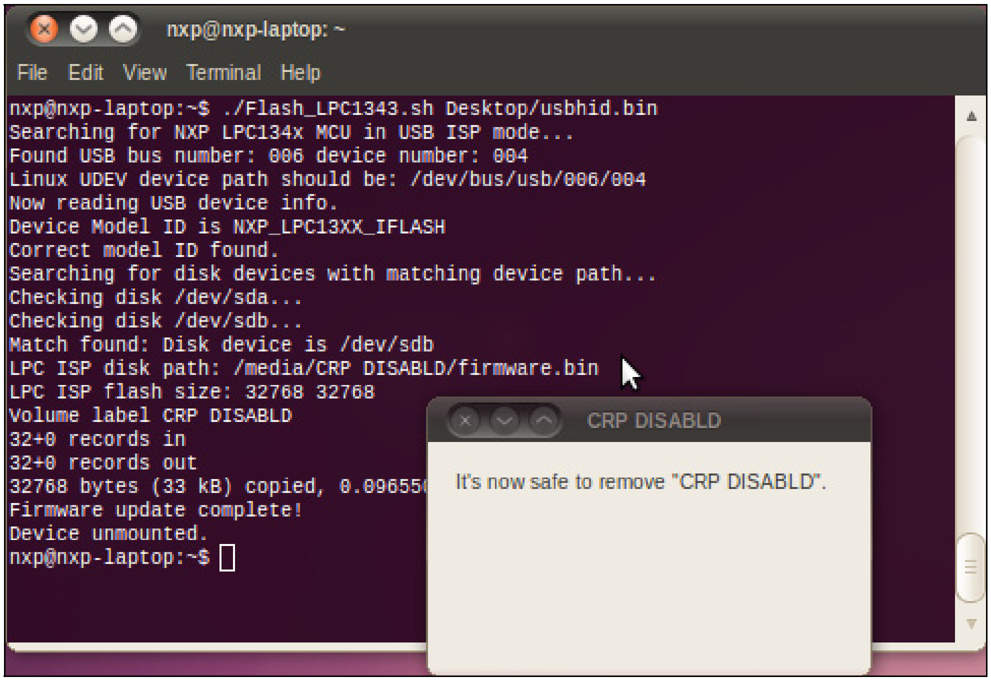
Successfully Updated Firmware

Firmware has been updated. Please safely eject device D:.

Fig 6. Windows NXPISP Utility (C# language, Microsoft Visual Studio 2010)

6.2 Linux LinuxNXPISP.sh ISP Utility

The `LinuxNXPISP.sh` ISP utility is a bash shell script that will run in Ubuntu 10.04. To run it, open a Terminal window, which can be found in the Accessories submenu of the Applications menu on the Ubuntu desktop. In the terminal window, navigate to where you have placed the script, using the `cd` command. If the script has been extracted from a .zip file, it may not be marked with executable permissions. Use `chmod +x filename` to remedy this. Finally, ensure that the USB ISP device to be programmed is already connected to the PC, and run the script with a single argument, the path to the .bin file you wish to program.



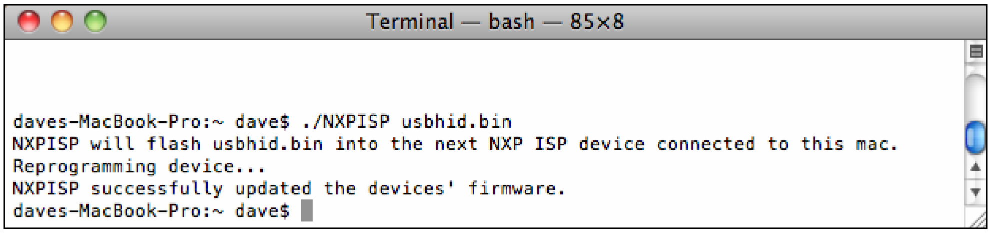
```
nxp@nxp-laptop:~$ ./Flash_LPC1343.sh Desktop/usbhid.bin
Searching for NXP LPC134x MCU in USB ISP mode...
Found USB bus number: 006 device number: 004
Linux UDEV device path should be: /dev/bus/usb/006/004
Now reading USB device info.
Device Model ID is NXP_LPC13XX_IFLASH
Correct model ID found.
Searching for disk devices with matching device path...
Checking disk /dev/sda...
Checking disk /dev/sdb...
Match found: Disk device is /dev/sdb
LPC ISP disk path: /media/CRP DISABLD/firmware.bin
LPC ISP flash size: 32768 32768
Volume label CRP DISABLD
32+0 records in
32+0 records out
32768 bytes (33 kB) copied, 0.09655
Firmware update complete!
Device unmounted.
nxp@nxp-laptop:~$
```

It's now safe to remove "CRP DISABLD".

Fig 7. Ubuntu NXPISP Utility (bash script)
chmod +x LinuxNXPISP.sh
then run from command prompt with ./LinuxNXPISP.sh firmware.bin

6.3 Mac NXPISP ISP utility

The Mac NXPISP ISP utility is an executable that should run on any Intel Mac running OS 10.5 or newer. To run it, open a Terminal window, which can be found using a Spotlight search. In the terminal window, navigate to where you have placed the program, using the `cd` command. If the executable has been extracted from a .zip file, it may not be marked with execute permissions. Use `chmod +x filename` to remedy this. Finally, run the program with a single argument, the path to the .bin file you wish to program. If the ISP device is already connected to the Mac, it will be updated immediately and the tool will exit. If no device is connected, the tool will wait until a USB ISP device is connected to the Mac, then update it and quit.



```
daves-MacBook-Pro:~ dave$ ./NXPISP usbhid.bin
NXPISP will flash usbhid.bin into the next NXP ISP device connected to this mac.
Reprogramming device...
NXPISP successfully updated the devices' firmware.
daves-MacBook-Pro:~ dave$
```

Fig 8. Mac OS-X NXPISP Utility (C language- Xcode)

7. Conclusion

In conclusion, the on-chip USB mass-storage in-circuit programming feature in the LPC1300 family of microcontrollers can simplify manual firmware downloads during the development process as well as support automated updates in the field.

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an please send an email to:
salesaddresses@nxp.com

Date of release: 24 September 2010

Document identifier: AN10986