AN11229

UUencoding for UART ISP

Rev. 1 — 22 June 2012 Application note

Document information

Info	Content
Keywords	LPC2XXX, LPC11AXX, LPC11XX, LPC11UXX, LPC11EXX, LPC12XX, LPC12DXX, LPC13XX, LPC17XX, LPC18XX, LPC43XX
Abstract	This application note introduces the UUencoding scheme used in the UART ISP for NXP's LPC microcontroller family.

Revision history

Rev	Date	Description
1	20120622	Initial release

Contact information

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

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

1. Introduction

The UART ISP routines used in the UART ISP aware NXP controllers require data to be encoded in the UUencode format. This application note explains how UUencode works and some basic UART ISP calls.

2. UUencode scheme

UUencode is a form of binary-to-ASCII encoding originating from the UNIX environment. UUencode takes an 8-bit value and converts it into an ASCII equivalent value. UUencode works on groups of three 8-bit data bytes. If the data bytes are not in groups of three, padded bytes must be added. The maximum number of data bytes that can be encoded in a data line is 45 data bytes. Each data line cannot exceed 61 characters.

A data line of UUencode data uses the format:

<character length><formatted characters><newline>

<character length> is one character indicating the number of data bytes encoded in the line. The character length is calculated by adding 32 to the number of data bytes being transferred before encoding has occurred.

<formatted characters> is the encoded data bytes.

<newline> indicates the end of the data bytes. The new line is indicated with a <CR><LF>, carriage return and line feed, respectively.

3. UUencode conversion

The flow of the UUencode conversion is as follows:

1. The data is subdivided into 3-byte groups forming a 24-bit stream
2. The 24-bit stream is then subdivided into 6-bit groups
3. A value of 0x20 is added to the 6-bit group
4. If a 6-bit group has a value of 0x00, a value of 0x60 is added to it
5. The number of data bytes is calculated and converted into its ASCII equivalent

If the number of bytes is not a multiple of three, padded bytes are added to create a multiple of three. The padded bytes can be of any value since the decoding process discards the padded bytes. The value of 0x00 is recommended. For instance, for a payload consisting of 4 bytes, two padded bytes are added to create a 6 byte payload. Each Uuencode line cannot exceed 61 characters/45 data bytes.

3.1 UUencode example – Three byte data

This example describes how to convert three data bytes consisting of 0x14, 0x0F, and 0xA8 into a UUencode stream.

The first step is to convert the data bytes into a 24-bit stream.

Data Byte	0x14								0x0F								0xA8							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0

The 24-bit stream is subdivided into 6-bit groups.

Data Byte	0x14								0x0F								0xA8							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0
6-Bit Group	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0
6-Bit Value	0x05								0x00				0x3E				0x28							

A value of 0x20 is added to the 6-bit value. The result is a character in the ASCII table. For the 6-bit data whose value is 0x00, a value of 0x60 is added to it as opposed to 0x20.

Data Byte	0x14								0x0F								0xA8							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0
6-Bit Group	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0
6-Bit Value	0x05								0x00				0x3E				0x28							
Encoding	0x25								0x60				0x5E				0x48							

Use the ASCII table to determine the ASCII character associated with the encoded value.

Data Byte	0x14								0x0F								0xA8							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0
6-Bit Group	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0
6-Bit Value	0x05								0x00				0x3E				0x28							
Encoding	0x25								0x60				0x5E				0x48							
ASCII	%								`				^				H							

The number of bytes to be transferred is calculated and then converted into its ASCII equivalent. The data bytes consist of 3 bytes: 0x14, 0x0F, and 0xA8. The character length is determined by adding 32 to the number of data bytes. For this example, the character length is “#”.

3 + 32 = 35 = 0x23 = “#”

The line sent to the NXP controller is:

#%`^H<CR><LF>

3.2 UUencode example – 4 byte data

This example describes how to convert four bytes consisting of 0x14, 0x0F, 0xA8, and 0x17 into a UUencode stream.

UUencode expects the data to be in multiples of 3. The 4 bytes of data are padded with 2 data bytes of 0x00 so it is a multiple of 3.

The first step is to convert the data bytes into a 24-bit stream.

Data Byte	0x14								0x0F								0xA8								0x17								0x00								0x00							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The 24-bit stream is subdivided into 6-bit groups.

Data Byte	0x14								0x0F								0xA8								0x17								0x00								0x00							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-Bit Group	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-Bit Value	0x05								0x00								0x3E								0x28								0x05								0x30								0x00								0x00							

A value of 0x20 is added to the 6-bit value. The result is a character in the ASCII table. For the 6-bit data whose value is 0x00, a value of 0x60 is added to it as opposed to 0x20.

Data Byte	0x14								0x0F								0xA8								0x17								0x00								0x00							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-Bit Group	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-Bit Value	0x05								0x00								0x3E								0x28								0x05								0x30								0x00								0x00							
Encoding	0x25								0x60								0x5E								0x48								0x25								0x50								0x60								0x60							

Use the ASCII table to determine the ASCII character associated with the encoded value.

Data Byte	0x14								0x0F								0xA8								0x17								0x00								0x00							
Bit Stream	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-Bit Group	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-Bit Value	0x05								0x00								0x3E								0x28								0x05								0x30								0x00								0x00							
Encoding	0x25								0x60								0x5E								0x48								0x25								0x50								0x60								0x60							
ASCII	%								`								^								H								%								P								`								`							

The number of bytes to be transferred is calculated then converted into its ASCII equivalent. The data bytes consist of four bytes, 0x14, 0x0F, 0xA8, and 0x17. The character length is determined by adding 32 to the number of data bytes. For this example, the character length is “$”.

4 + 32 = 36 = 0x24 = “$”

The line sent to the NXP controller is:

$%`^H%P``<CR><LF>

4. Calculating checksum

The checksum is the sum of transferred bytes. The checksum is needed when read/write commands are issued. The checksum to be sent to the controller is in decimal form.

As an example, if the data bytes consisting of 0x14, 0x0F, and 0xA8 are to be sent to the NXP controller, the checksum would be:

0x14 + 0x0F + 0xA8 = 0xCB

The decimal equivalent of 0xCB is 203. The number 203 is sent to the NXP controller as the checksum.

The line sent to the NXP controller is:

203<CR><LF>

5. UART ISP example

For the following ISP examples, the host used is a Windows 7 system running TeraTerm. The UART port is set to 9600, 8, N, 1, XON/XOFF.

The test board used is the LPCXpresso base board with a LCP1114/302.

All UART ISP commands should be sent as single ASCII strings. Strings need to be terminated with Carriage Return (CR) and Line Feed (LF) control characters. Extra <CR> and <LF> characters are ignored. All ISP responses are sent as <CR><LF> terminated ASCII strings. Data is sent and received in UUencoded format. All other commands and responses are in ASCII format.

5.1 ISP initialization

The controller must first be put into ISP mode. For the LCP1114, this is accomplished by grounding the PIO0_1 pin during reset. Once ISP mode is initialized, the host prepares the controller for ISP control. The steps are as follows:

1. The host sends an ASCII character of “?”
2.
3. The controller responds with “Synchronized”
4. The host acknowledges this with “Synchronized”
5. The controller responds with “OK”
6. The host now sends the frequency of the crystal in kHz. For example “12000” is sent for a 12 MHz crystal
7. The controller responds with “OK”
8. The host can now set a new baud rate if desired

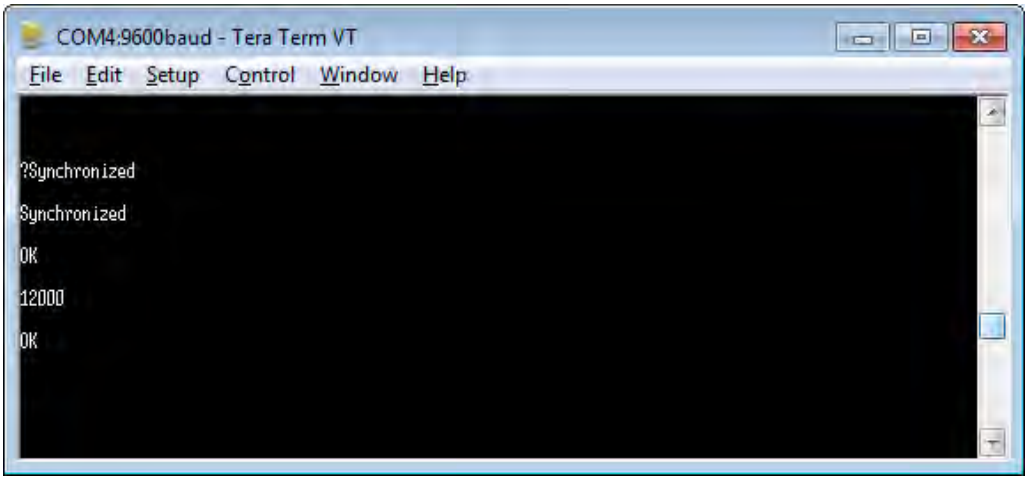

Fig 1. UART ISP Initialization

5.2 Reading memory

The command format to read from the RAM/Flash memory is as follows:

R <address> <number of bytes>

<address> is the desired address in decimal. The address must be a word boundary.

<number of bytes> is the desired bytes. The number of bytes must be in multiples of 4.

When a read is issued, the controller responds with the requested data, encoded in the UUencode format, and the checksum of the requested data.

The checksum is sent after the request amount of data is transmitted or 20 UUencoded lines, whichever comes first. The checksum is generated by adding the raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. When the data fits in less than 20 UU-encoded lines then the checksum is of actual number of bytes sent.

As an example, to read 4 bytes of data from the address 0x10000000, the following sequence occurs:

1. The host sends the command “R 268435456 4”
2.
3. The controller responds with a Return Code, the data, and the checksum
4. The host sends an “OK” if the checksum is correct. If the checksum is incorrect, a “RESEND” command is issued so the controller can resend the data

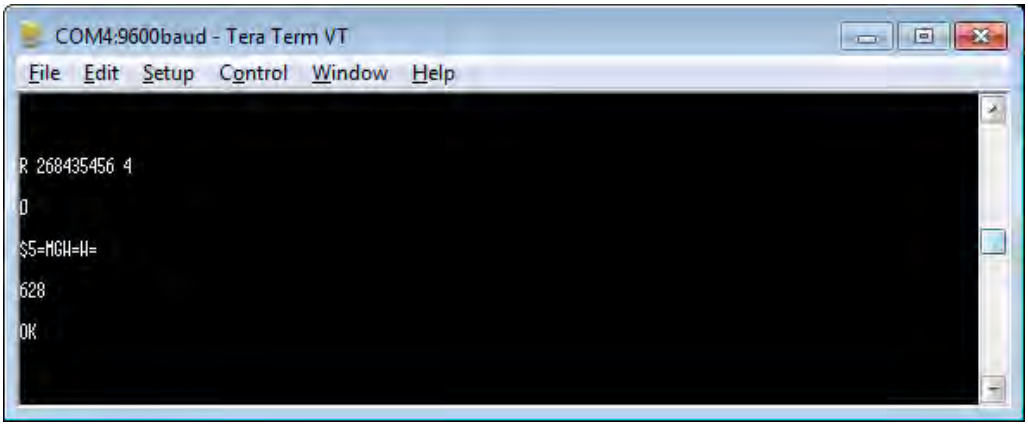

Fig 2. UART ISP reading from memory

5.3 Writing to RAM

The command format to write to the RAM is as follows:

W <start address> <number of bytes>

<address> is the desired address in decimal. The address must be a word boundary.

<number of bytes> is the desired bytes. The number of bytes must be in multiples of 4.

As an example, to write the value of 0x14, 0x0F, 0xA8, and 0x17 to the RAM address of 0x10000000, the following sequence occurs:

1. The host sends the command “W 268435456 4”
2. The controller responds with a Return Code
3. The host sends the data in UUencode format, “$%`^H%P``”
4. The host sends the checksum, “226”
5. The controller responds with an “OK” if the checksum matches the data. If the checksum does not match, then a “RESEND” is sent back to the host

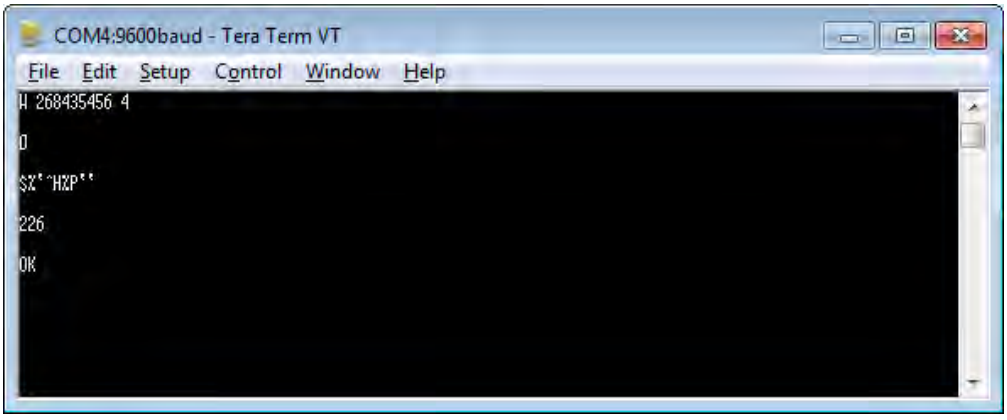

Fig 3. UART ISP writing to RAM

5.4 Copying RAM to flash

The command format to copy data from the RAM to flash is as follows:

C <Flash address> <RAM address> <number of bytes>

<Flash address> is the destination address in decimal. The destination address should be a 256 byte boundary.

<RAM address> is the source address in decimal.

<number of bytes> is the desired bytes. Valid values are 256, 512, 1024, and 4096.

When writing to the flash, the following limitations apply:

1. The smallest amount of data that can be written to flash by the copy RAM to flash command is 256 bytes (equal to one page).
2.

3. One page consists of 16 flash words (lines), and the smallest amount that can be modified per flash write is one flash word (one line). This limitation follows from the application of ECC to the flash write operation.
4. To avoid write disturbance (a mechanism intrinsic to flash memories), an erase should be performed after following 16 consecutive writes inside the same page. Note that the erase operation erases the entire sector.

Remark: Once a page has been written to 16 times, it is still possible to write to other pages within the same sector without performing a sector erase (assuming that those pages have been erased previously).

As an example, to copy 256 bytes from the RAM at 0x10000000 to flash at 0x00, the following sequence occurs:

1. The host sends the unlock command, “U 23130”
2. The controller responds with a Return Code
3. The host sends the Prepare Sector for write command, “P 0 0”
4. The controller responds with a Return code
5. The host sends the copy command, “C 0 268435456 256”
6. The controller responds with a Return code

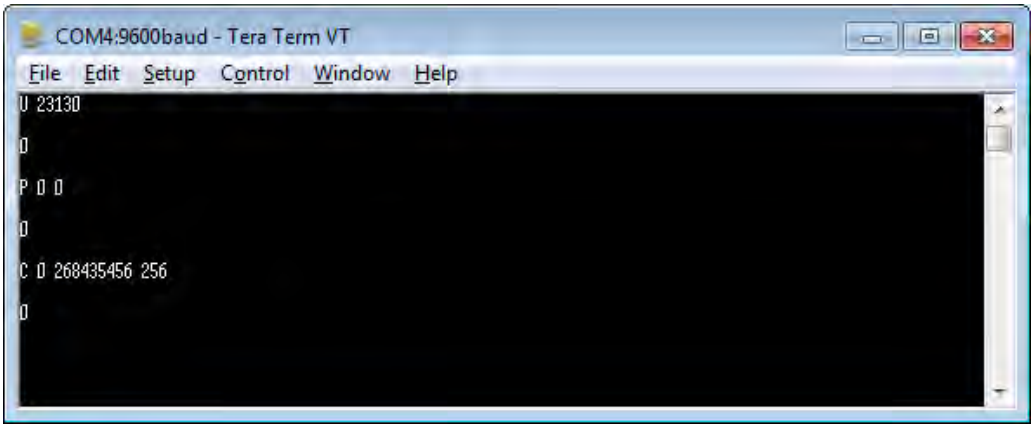

Fig 4. UART ISP Copying RAM to Flash

6. Tips and hints

The UART ISP command controller by default will echo back received data. To maximize the speed of the UART transmission, the echo functionality can be turned off. This is accomplished by sending a command of **A 0** to the controller.

For a read operation, a checksum will be sent after every 20 UUencode lines. The host ISP code should account for this. For a write operation, a checksum must be sent after every 20 UUencode lines.

RAM is used by the ISP controller in the ISP mode. Avoid using this section of RAM in ISP if possible. Refer to the controller’s user manual to determine the block of RAM being used in ISP.

In ISP mode, flash programming commands use the top 32 bytes of the RAM. The maximum stack usage is 256 bytes and it grows downward.

The Unlock command must be issued before any flash operation.

The Prepare Sector command must be executed before Copy RAM to Flash and Erase Sector commands.

7. Additional resources

[1] UART ISP implementation on the LPC1768 using the mbed platform http://mbed.org/cookbook/lpc-bootloader

[2] UART ISP for the LPC1100/LPC1300/LPC1700/ LPC2000 http://sourceforge.net/projects/lpc21isp/

[3] UART ISP for the Linux platform http://code.google.com/p/lpcflash/

[4] UART ISP written in Python http://sourceforge.net/projects/nxpprog/

[5] UART ISP written in Python for the LPC2000 series http://sourceforge.net/projects/pylpctools/

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. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

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 and its suppliers accept 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 competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.3 Trademarks

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

9. List of figures

10. Contents

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

Date of release: 22 June 2012
Document identifier: AN11229