NXP Semiconductors
User's Guide

Document Number: MCUBLHOSTUG
Rev 4, 05/2018

blhost User's Guide

Contents

Chapter 1
Introduction

This document describes the usage of the blhost PC application. The blhost application is used on a host computer to issue commands to an NXP platform running an implementation of the MCU bootloader. The blhost application, in conjuction with the MCU bootloader, allows a user to program a firmware application onto the MCU device without a programming tool.

NOTE

1. When Flash security is enabled, blhost cannot read, write, or erase the flash. See Section 4.2.13, "flash-security-disable", for how to disable the flash security.
2. blhost cannot connect to the ROM or flash-resident bootloader if the application is running on the device. It is possible to get back into the bootloader by pressing the boot pin. However, the boot pin must be enabled in the BCA. See the *MCU Bootloader Demo Applications User's Guide* (document MBOOTDEMOUG) for more information on boot pins and the platforms supported.

Chapter 2
Overview

This user's guide describes how to interface with the MCU bootloader using blhost application. There is a brief introduction of the MCU bootloader followed by detailed descriptions of blhost options and commands. A description of the MCU bootloader operation, as it relates to blhost application, is provided in Appendix A, Appendix B, and Appendix C.

Chapter 3
MCU bootloader

The MCU bootloader is intended to be the standard bootloader for all MCU devices. It provides a standard interface to the device using any of the peripherals supported by the bootloader on a given NXP MCU device.

MCU bootloader implementations include:

- ROM-based bootloader
- one-shot flash memory programming aid (referred to as a flashloader)
- flash-resident bootloader

The MCU bootloader is available as source code for customer, flash-based implementations. Example applications are provided to demonstrate how to interface with the MCU bootloader.

Chapter 4
blhost Utility application

The blhost application is a command-line utility used by the host computer to initiate communication and inject commands to the MCU bootloader. The application only sends one command per invocation.

The blhost application can communicate directly with the MCU bootloader over the host computer's UART (Serial Port) or USB connections.

Although the MCU bootloader typically supports other peripherals such as I2C, SPI, and CAN. the blhost application cannot interface with the MCU bootloader over these transports without external hardware. See Appendix C for the description of BusPal embedded software included with the MCU bootloader release. The BusPal acts as a bus translator running on selected platforms. BusPal assists blhost in carrying out commands and responses from the target device through an established connection with blhost over UART, and the target device over I2C, SPI, or CAN.

The arguments for blhost consists of a set of options followed a command description. The options and the command are separated with a '--'.

Ex. `blhost [options] -- [command]`

The following sections describe all of blhost options and commands.

4.1 *blhost* Options

The usage screen of blhost lists all of the options and commands supported by the blhost utility.

4.1.1 -?, -h/--help

Prints the usage screen.

4.1.2 -v/--version

Prints the version of the blhost application.

4.1.3 -p/--port<name>[<speed>]

Connect to MCU bootloader over UART

name: Specify COM port.

speed: Baud rate (optional).

The default is *-p COM1,57600*

Do not use the *-u* option with the *-p* option.

4.1.4 -u/--usb [[<vid>,]<pid>]|[<path>]

Connect to MCU bootloader over USB HID.

vid: Specify the USB vendor ID of the MCU bootloader.

pid: Specify the USB vendor ID of the MCU bootloader.

The default is *-u 0x15a2,0x0073*

path: Specify the USB device instance path. For windows system, the path can used from the Device Manager as shown in the figure below. For Linux system, the path parameter is the absolute path of the USB node in the /dev

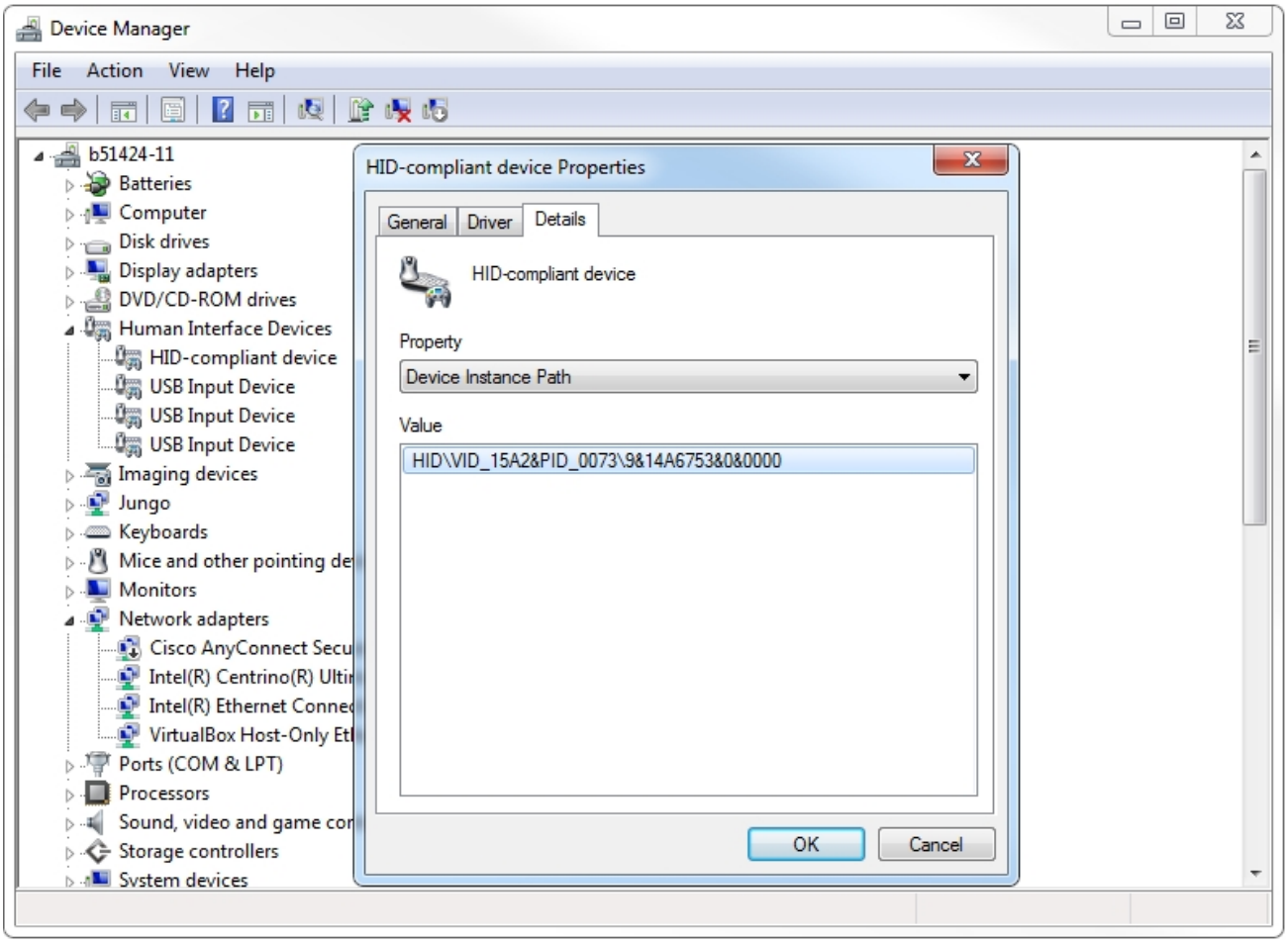

Figure 1. Device Manage

4.1.5 -b/--buspal

See Appendix C on the usage of blhost with BusPal.

4.1.6 -V/--verbose

Print extra detailed log information.

4.1.7 -d/--debug

Prints every byte sent to and received from the MCU bootloader. Bytes sent to the bootloader are denoted with [xx] and bytes received from the bootloader are denoted by <xx>.

4.1.8 -j/--json

Print output in JSON format to aid automation.

All output other than the JSON formatted command response is suppressed. For example:

```
>blhost -p COM23 -j -- get-property 1
{
```

```
    "command" : "get-property",
    "response" : [ 1258357504 ],
    "status" : {
       "description" : "0 (0x0) Success.",
       "value" : 0
    }
}
```

4.1.9 -n/--noping

Skip the initial ping of a serial target. By default, blhost sends an initial ping before every command. The initial ping is used by the UART peripheral to automatically establish the baud rate of the UART. If sending successive commands to a UART, subsequent commands can use the –n option to skip the ping because the baud rate was already established. Also, the ping can safely be suppressed when using I2C, SPI, or CAN transports. Sending a ping before every command has no side effects.

4.1.10 -t/--timeout<ms>

Set serial read timeout in milliseconds. This value allows blhost to time out when waiting on data from the MCU bootloader over a serial transport.

The default is 5000 milliseconds.

4.2 blhost Commands

The usage screen of blhost lists all of the options and commands supported by the blhost utility.

Not all commands are supported on all MCU bootloader platforms. If a command is not supported by the MCU bootloader, it returns *k_StatusUnkownCommand*.

When flash security is enabled, only the get-property, set property, reset, flash-security-disable, and flash-erase-all-unsecure commands are supported. The MCU bootloader returns *kStatus_SecurityViolation* if a command is received that is not allowed because of the flash security settings.

See the *MCU Bootloader v2.5.0 Reference Manual* (document MCUBOOTRM) for the specific MCU platform for a list of supported commands and properties.

4.2.1 reset

Example: -- `reset`

Resets the device. The response packet is sent prior to resetting the device.

The reset command is used to switch boot from flash after successful flash image provisioning via ROM bootloader. After issuing the reset command, allow 5 seconds for the user application to start running from flash.

4.2.2 get-property <tag>[<memoryID>]

Example: `-- get-property 10`

This command is used to query the bootloader about various properties and settings. Each supported property has a unique integer *tag* value.

tag:

- 0x01 The Current Version of the MCU bootloader
- 0x02 A mask of the Available Peripherals

- 0x03 The starting address of the on-chip flash
- 0x04 The size of the on-chip flash
- 0x05 The size in bytes of one sector of program flash. This is the minimum erase size
- 0x06 The number of blocks in the on-chip flash
- 0x07 A mask of the Available Commands
- 0x08 Status code from the last CRC check operation. Available only if the CRC check feature is supported
- 0x09 Reserved for future use
- 0x0a Verify Writes Flag - Boolean controlling whether the bootloader verifies writes to flash. A value of 0 means no verification is done, a value of 1 enables verification. This feature is enabled by default
- 0x0b The maximum supported packet size for the currently active peripheral interface
- 0x0c Reserved Regions - List of memory regions reserved by the bootloader. Returned as value pairs (<start-address-of-region>,<end-address-of-region>). If HasDataPhase flag is not set, response packet parameter count indicates number of pairs. If HasDataPhase flag is set, second parameter is the number of bytes in the data phase
- 0x0e The starting address of the on-chip RAM
- 0x0f The size in bytes of the on-chip RAM
- 0x10 The value of the Kinetis System Device Identification register
- 0x11 Flash Security Enabled Flag - Boolean indicating whether flash security is enabled. 0 means disabled, 1 means enabled
- 0x12 The values of the Device Unique ID. The number of ID words is indicated by the parameter count in the response packet
- 0x13 FAC Supported Flag - Boolean indicating whether Flash Access Control (FAC) module is supported. 0 means not supported, 1 means supported
- 0x14 The number of bytes per FAC segment
- 0x15 The number of segments available in the FAC
- 0x16 The flash read margin setting. A value of 0 indicates Normal read margin, a value of 1 indicates User read margin, and a value of 2 indicates Factory read margin. The default is User
- 0x17 Reserved for future use
- 0x18 Target Version - the target build version number
- 0x19 External Memory Attributes - memoryId is required. For additional information on memoryID, refer to MCU Bootloader Reference Manual

4.2.3 set-property <tag> <value>[<memoryID>]

Example: `-- set-property 10 0`

This command changes properties and options in the bootloader. The command accepts the same property tags used with the get-property command; however, only some properties are writable. If an attempt to write a read-only property is made, an error is returned indicating the property is read-only and cannot be changed.

Properties that can be changed all have 32-bit values.

tag:

- 0x0a Verify Writes Flag - Boolean controlling whether the bootloader verifies writes to flash. A value of 0 means no verification is done, a value of 1 enables verification. This feature is enabled by default.
- 0x16 The flash read margin setting. A value of 0 indicates Normal read margin, a value of 1 indicates User read margin, and a value of 2 indicates Factory read margin. The default is User.

4.2.4 flash-erase-region <addr> <byte_count>[memoryID]

Example: `-- flash-erase-region 0xa000 1024`

Erases one or more sectors of flash memory specified by memoryID. The default memoryID is 0 if the memory parameter is not provided.

The start address and count must be a multiple of the word size. The entire sector(s) containing the start and end address is erased.

If the VerifyWrites property is enabled, the command performs a flash verify erase operation.

4.2.5 flash-erase-all[memoryID]

Example: `-- flash-erase-all`

Performs an erase of the entire flash memory specified by memoryID. The default memoryID is 0 if the memoryID parameter is not provided.

If any flash regions are protected, the command fails with an error. If any flash regions are reserved by the bootloader, they are ignored (not erased).

If the VerifyWrites property is enabled, the flash-erase-all command performs a flash verify erase all operation, or multiple flash verify erase options if decomposed due to reserved regions.

4.2.6 flash-erase-all-unsecure

Example: `-- flash-erase-all-unsecure`

This command is only supported in new versions of the flash controller. Most MCU devices do not support this command, and the bootloader sends a *kStatus_UnknownCommand* error in response.

Performs a mass erase of the flash memory, including protected sectors and any reserved regions in flash. Flash security is immediately disabled if it was enabled and the FSEC byte in the Flash Configuration Field at address 0x40C is programmed to 0xFE.

The Mass Erase Enable option in the FSEC field is honored by this command. If mass erase is disabled, then this command fails.

This command is only useful and only present in ROM configurations of the bootloader because it erases reserved regions in flash.

4.2.7 read-memory <addr> <byte_count> [<file>][memoryID]

Example: `-- read-memory 0x3c0 32 myConfigData.dat`

Read memory specified by MemoryID and write to file or stdout if no file specified. The default MemoryID is 0 if the memoryID parameter is not provided.

Returns the contents of memory at the given address, for a specified number of bytes. This command can read any region of memory accessible by the CPU and not protected by security. This includes flash, RAM, and peripheral registers. Note that the minimum profile does not support reading the peripheral register space.

4.2.8 write-memory <addr> [<file> | {{<hex-data>}}][memoryID]

Example: `-- write-memory 0x3c0 myConfigData.dat`

Example: `-- write-memory 0xa000 "{{aa bb cc dd ee ff 00 01 02 03 04}}" (spaces require quotes)`

Example: `-- write-memory 0xa000 {{aabbccddeeff0001020304}}`

Write memory specifiled by memoryID at *addr* from *file* or string of hex values. The default memoryID is 0 if the memoryID parameter is not provided.

Writes a provided buffer to a specified range of bytes in memory. Can write to all accessible memory, including flash, RAM, and peripheral registers. However, if flash protection is enabled, writes to protected sectors fails. Data specified by *file* is treated as binary data.

Any flash sector written to must be previously erased with either a *flash-erase-all, flash-erase-region*, or *flash-erase-all-unsecure* command.

Writing to flash requires the start address to be word aligned. The byte count is rounded up to a multiple of the word size, and trailing bytes are filled with the flash erase pattern (0xff).

Word and halfword-aligned and sized writes to RAM and peripheral registers use appropriately sized writes. This enables writing to registers larger than a byte in a single bus transaction. Note that the minimum profile does not support writing to the peripheral register space.

If the VerifyWrites property is enabled, writes to flash performs a flash verify program operation.

4.2.9 fill-memory <addr> <byte_count> <pattern> [word | short | byte]

Example: `-- fill-memory 0x3c0 32 0xff byte`

Fill memory with pattern; size is word (default), short or byte. To fill 32-bit memory words with a repeating byte pattern, specify a byte-sized pattern argument followed by the 'byte' keyword. To fill memory words with a repeating 16-bit pattern, specify a short-sized pattern followed by the 'short' keyword.

Follows the same rules as the *write-memory* command.

4.2.10 receive-sb-file <file>

Example: -- receive-sb-file mySecureImage.sb

Receive a file in Secure Binary (SB) format. An SB file is an encapsulated, binary stream of bootloader commands that can be optionally encrypted. The SB file format is described in the document *elftosb User's Guide* (document MCUELFTOSBUG) and can be created using the elftosb tool.

Note that if the SB file contains a JUMP command, the receive-sb-file command is aborted at the point of the jump, and a status of *kStatus_AbortDataPhase* is returned.

4.2.11 execute <address> <arg> <stackpointer>

Example: `-- execute 0x6000 0x21 0x1fff8400`

Jumps to code at the provided *address* and does not return to the bootloader. The system is returned to reset state prior to the jump. The function *arg* parameter is passed in R0 to the called code. The main stack pointer and process stack pointer registers are set to the *stackpointer* parameter, which can be zero. If set to zero, the code being called should set the stack pointer before using the stack.

The effective prototype of the called function is:

void function(uint32_t arg);

4.2.12 call <address> <arg>

Example: `-- call 0x6000 0x21`

This invokes code at the provided *address* with the expectation that control returns to the bootloader.

The function that is called has the same prototype as for the one called by the execute command.

NOTE

Because the intention is to return to the bootloader after the function executes, the function must not perform any action that would interfere with the bootloader operation. In particular, the following restrictions apply:

- Do not use interrupts because the interrupt vectors are still owned by the bootloader.
- Do not modify any memory locations used by the bootloader (use "get-property 12" to determine reserved regions).
- Do not modify any pin mux or clock settings used by bootloader peripherals.

4.2.13 flash-security-disable <key>

Example: `-- flash-security-disable 0102030405060708`

Performs the flash security disable operation by comparing the provided 8-byte backdoor key against the backdoor key stored in the Flash Configuration Field at address 0x400 in flash.

If the backdoor key comparison fails, further attempts to disable security with this command fails until the system is reset.

Backdoor key access must be enabled by setting the KEYEN bitfield of the FSEC byte in the Flash Configuration Field to 0b10. It is disabled by default. The backdoor key in the Flash Configuration Field must also be set to a value other than all zeros or all ones.

4.2.14 flash-program-once <index> <byteCount> <data>

Example: `-- flash-program-once 0 4 0x01020304`

This writes provided data to a specific program once field.

Special care must be taken when writing to program once field. The program once field only supports programming once.

Any attempts to reprogram a program once field gets an error response. The number of bytes to be written must be 4-byte aligned for non-FAC fields, and be 8-byte aligned for FAC fields.

4.2.15 flash-read-once <index> <byteCount>

Example: `-- flash-read-once 0 4`

Returns the contents of a specific program once field.

4.2.16 efuse-program-once <addr> <data>

Example: `--efuse-program-once 6 0x04030201`

This write provided data to a specific efuse word. Each efuse bit can only be programed once.

4.2.17 efuse-read-once <addr>

Example `--efuse-read-once 6`

Returns the contents of a specific efuse word.

4.2.18 flash-read-resource <addr> <byteCount> <option> [<file>]

Example: `-- flash-read-resource 0 4 1 firmwareID.txt`

Reads the contents of Flash IFR or Flash Firmware ID as specified by *option* and writes result to *file* or stdout if *file* is not specified.

byteCount: The number of bytes to read and returns to the caller. Must be 4-byte aligned.

option:Indicates which area to be read. 0 means Flash IFR, 1 means Flash Firmware ID.

4.2.19 configure-memory <memoryID> <addr>

Example: `-- configure-memory 1 0x20001000`

Apply the configuration block at <addr> to external memory with ID <memoryID>. The specified configuration block must have been previously written to memory using the write-memory command. The format of the configuration block is described in the *MCU Bootloader v2.5.0 Reference Manual* (document MCUBOOTRM).

4.2.20 flash-image <file> [erase][memoryID]

Example: `-- flash-image myImage.s19 erase`

Write the formatted image in <file> to the memory specified by memoryID. Supported file types are S-Record (.srec and .s19), and Hex (.hex). Flash is erased before writing if [erase] is ‘erase’ or 1. This blhost command does not directly correspond to a bootloader command, but may send multiple bootloader commands to perform the operation.

4.2.21 reliable-update <addr>

For software implementation:

Example: `-- reliable-update 0x105000`

Checks the validity of backup application at <addr>, then copies the contents of backup application from <addr> to main application region.

For hardware implementation:

Example: `-- reliable-update 0xfe000`

Verifies if the provided <addr> is a valid swap indicator address for flash swap system, then checks the validity of backup application resided in upper flash block. After that, it swaps the flash system.

4.2.22 generate-key-blob

Example: `-- generate-key-blob dek.bin blob.bin`

Generate the blob for the given dek key -- dek.bin, and write the blob to the file -- blob.bin. DEK key file is generated by CST tool.

Chapter 5
MCU bootloader properties

5.1 Current version

The value of this property is a 4-byte structure containing the current version of the bootloader. This property is encoded in a one-word value.

Table 1. Bit ranges for version components

Bit	[31:24]	[23:16]	[15:8]	[7:0]
Field	Name	Major version	Minor version	Bugfix version

5.2 Available peripherals

The value of this property is a one-word bitfield that lists the peripherals supported by the bootloader and the hardware on which it is running.

Table 2. Peripheral bits

Bit	5	4	3	2	1	0
Peripheral	Reserved	USB HID	CAN	SPI slave	I2C slave	UART

5.3 Available commands

This property value is a bitfield with bits set corresponding to commands enabled in the bootloader.

The bit number that identifies whether a command is present is the command's tag value minus 1. To get the bit mask for a given command, use this expression.

mask = 1 << (tag - 1)

Table 3. Available commands

Bit	**Command**
[31:19]	reserved
18	generate-key-blob (0x13)
17	reliable-update (0x12)
16	configure-memory (0x11)
15	flash-read-resource (0x10)

Table continues on the next page...

Table 3. Available commands (continued)

Bit	Command
14	flash-read-once (0x0f)
13	flash-program-once (0x0e)
12	flash-erase-all-unsecure (0x0d)
11	set-property (0x0c)
10	reset (0x0b)
9	call (0x0a)
8	execute (0x09)
7	receive-sb-file (0x08)
6	get-property (0x07)
5	flash-security-disable (0x06)
4	fill-memory (0x05)
3	write-memory (0x04)
2	read-memory (0x03)
1	flash-erase-region (0x02)
0	flash-erase-all (0x01)

Chapter 6
Revision history

The following table contains a history of changes made to this user's guide.

Table 4. Revision history

Revision number	Date	Substantive changes
0	12/2014	Initial release
1	07/2015	Kinetis bootloader 1.2.0 updates
2	09/2015	Kinetis bootloader K80 standalone updates
3	04/2016	Kinetis Bootloader v.2.0.0 release
4	03/2018	RT1050 Flashloader 1.1 release
4.1	04/2018	MCU Bootloader v2.5.0 release

Chapter 7
Appendix A - Bootloader operation

The correct use of blhost program requires a connection to a MCU device running the MCU bootloader command interface. The diagram shows a simplified view of the MCU bootloader state machine that shows the states relevant to blhost application.

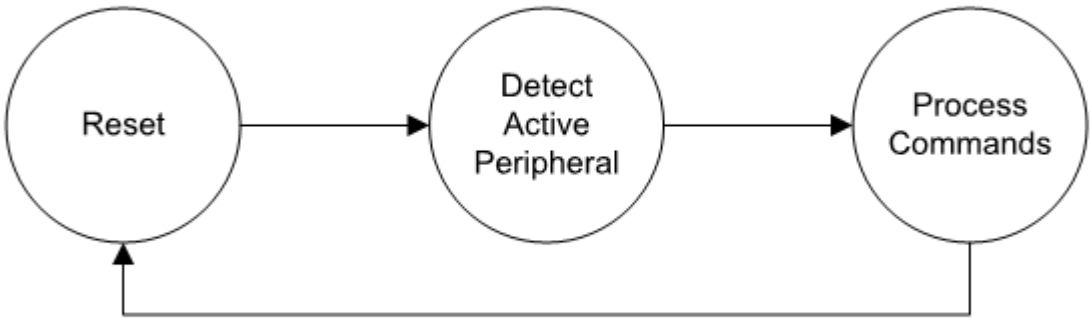

Figure 2. Simplified MCU bootloader state diagram

After reset, the bootloader monitors all enabled peripheral devices, UART, I2C, SPI, CAN, USB-HID, and USB-MSC for active communication. After communication is established, all peripherals except the active peripheral are shut down, and the bootloader enters the command processing state.

If the bootloader is in the “Detect Active Peripheral” state, the first use of blhost application establishes the active peripheral and the bootloader transitions to the “Process Commands” state. The active peripheral is established according to the transport used by the initial blhost invocation.

For example, if the -u option was successfully used to send the initial command, the USB-HID is the active peripheral and all future commands must use the *-u* option unless the bootloader is reset.

If the *-p COMx* option is used, the UART is the active peripheral and all future commands must use the *–p COMx* option unless the bootloader is reset. For the UART peripheral, the baud rate is automatically determined by the bootloader when processing the initial ping. Therefore, subsequent blhost invocations must specify the same baud rate as was used for the initial invocation unless the bootloader is reset. If the baud rate is not specified using the *-p COMx,<baudrate>* option, the UART baud rate is set to 57600.

NOTE

After the MCU bootloader is in the “Process Commands” state, the device has to be reset to communicate over a different peripheral or at a different baud rate over the same UART peripheral.

Chapter 8
Appendix B - Updating a bootloader image

8.1 Introduction

The flash-resident bootloader's main objective upon invocation is to provide a means for the host to update the application image residing on the flash along with the bootloader image.

If the flash-resident bootloader itself requires an upgrade, the MCU bootloader release package contains a solution. The RAM-based flashloader project is available in the release package for supported platforms, and can be used in the manner described below to upgrade the flash-resident bootloader.

8.2 Checklist

The setup and software required for upgrading the bootloader image:

1. Pre-built image for the RAM-based flashloader.
2. New flash-resident bootloader binary. In this example, the bootloader version number in src/bootloader/bl_version.h is changed to distinguish the new bootloader image from the current image.
3. Host PC for running blhost software and interfacing over UART/USB to the target device.
4. Target device connected to the host PC with either UART/USB.

8.3 Procedure

These steps explain how to upgrade the bootloader image on the flash:

1. Invoke flash-resident bootloader on the target device.
2. Establish a connection between the bootloader and the host PC over UART or USB.
3. Use the host-side command line tool (blhost) available with the release package on the host PC to start communicating with the bootloader on the target device. See additional information in this user's guide for using the blhost command line tool.
4. Download the pre-built image for the RAM-based flashloader into the internal RAM of the target device using the blhost write-memory command.
5. Invoke the blhost execute command to run the RAM-based flashloader on the target device.
6. Use blhost commands to communicate with the RAM-based flashloader, and replace the flash-resident bootloader with the new binary on the flash.
7. Reset the device to boot with the new flash-resident bootloader binary.

8.4 Example

Here is an example of how to update a bootloader for the KV46 32K RAM hardware and software environment:

Kinetis Tower System:

- Tower Serial port module.
- KV46 Rev A Tower System module with the current version of the bootloader and the LED demo application flashed on the device.
- The demo start address is 0xA000.

Detailed procedures:

- Power on the KV46 Tower System module. The LED demo should be running.
- Connect the KV46 Tower System module to Windows® OS 7 through UART port (e.g., COM1).
- Power cycle the KV46 Tower System module and send the following command within 5 seconds, which should enter bootloader mode.

This is an example command sequence for updating a bootloader image:

```
> blhost.exe -p COM1 -- get-property 1

Ping responded in 1 attempt(s)

Inject command 'get-property'

Response status = 0 (0x0) Success.

Response word 1 = 1258358016 (0x4b010500)

Current Version = K1.4.0

>blhost.exe -p COM1 -- get-property 12

Ping responded in 1 attempt(s)

Inject command 'get-property'

Response status = 0 (0x0) Success.

Response word 1 = 0 (0x0)

Response word 2 = 16383 (0x3fff)

Response word 3 = 536862720 (0x1fffe000)

Response word 4 = 536863999 (0x1fffe4ff)

Reserved Regions = Flash: 0x0-0x3FFF (16 KB), RAM: 0x1FFFE000-0x1FFFE4FF (1.250 KB)
```

Load the prebuilt RAM-based flashloader into RAM and execute from the flashloader. The example assumes the RAM start address is 0x1fffe600, entry address is 0x1fffea11, and stack pointer is 0x20001718. Confirm this by reading the first 8 bytes in bin file.

```
> blhost.exe -p COM1 -- write-memory 0x1fffe600 <path>flashloader.bin

Ping responded in 1 attempt(s)

Inject command 'write-memory'

Preparing to send 11360 (0x2c60) bytes to the target.

Successful generic response to command 'write-memory'
```

```
Successful generic response to command 'write-memory'

Response status = 0 (0x0) Success.

Wrote 11360 of 11360 bytes.

>blhost.exe -p COM1 -- execute 0x1fffea11 0 0x20001718

Ping responded in 1 attempt(s)

Inject command 'execute'

Successful generic response to command 'execute'

Response status = 0 (0x0) Success.
```

The flashloader should now be running. By checking the reserved area <get-property 12> you should see the RAM reserved area is different (much bigger) compared with the bootloader.

```
>blhost.exe -p COM1 -- get-property 12

Ping responded in 1 attempt(s)

Inject command 'get-property'

Response status = 0 (0x0) Success.

Response word 1 = 0 (0x0)

Response word 2 = 0 (0x0)

Response word 3 = 536864256 (0x1fffe600)

Response word 4 = 536876823 (0x20001717)

Reserved Regions = Flash: 0x0-0x0 (0 bytes), RAM: 0x1FFFE600-0x20001717 (12.273 KB)
```

Erase the flash memory by using the flash-erase-all or flash-erase-region commands. If you want to update the bootloader and erase the application at the same time, use the flash-erase-all command. Otherwise, use the flash-erase-region command in order to protect your application from being erased.

Erase entire flash memory command:

```
>blhost.exe -p COM1 -- flash-erase-all

Ping responded in 1 attempt(s)

Inject command 'flash-erase-all'

Successful generic response to command 'flash-erase-all'

Response status = 0 (0x0) Success.
```

Confirm if the flash memory has been erased by using the read-memory command.

```
>blhost.exe -p COM1 -- read-memory 0 32

Ping responded in 1 attempt(s)

Inject command 'read-memory'

Successful response to command 'read-memory'

ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

Successful generic response to command 'read-memory'

Response status = 0 (0x0) Success.
```

```
Response word 1 = 32 (0x20)

Read 32 of 32 bytes.
```

Write the new bootloader into the flash area and confirm if the writing is properly done by reading memory.

```
>blhost.exe -p COM1 -- write-memory 0 <path>new_tower_bootloader.bin

Ping responded in 1 attempt(s)

Inject command 'write-memory'

Preparing to send 12924 (0x327c) bytes to the target.

Successful generic response to command 'write-memory'

Successful generic response to command 'write-memory'

Response status = 0 (0x0) Success.

Wrote 12924 of 12924 bytes.

>blhost.exe -p COM1 -- read-memory 0 32

Ping responded in 1 attempt(s)

Inject command 'read-memory'

Successful response to command 'read-memory'

00 e5 ff 1f 01 08 00 00 37 0f 00 00 11 14 00 00

13 14 00 00 15 14 00 00 17 14 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 37 0f 00 00

37 0f 00 00 00 00 00 00 37 0f 00 00 a7 0b 00 00

67 13 00 00 67 13 00 00 67 13 00 00 67 13 00 00

67 13 00 00 67 13 00 00 67 13 00 00 67 13 00 00

67 13 00 00

Successful generic response to command 'read-memory'

Response status = 0 (0x0) Success.

Response word 1 = 100 (0x64)

Read 100 of 100 bytes.
```

Reset. Try to get the version and you should be able to run under a new version of the bootloader by checking the revision number (1.5.0)

```
>blhost.exe -p COM1 -- reset

Ping responded in 1 attempt(s)

Inject command 'reset'

Successful generic response to command 'reset'

Response status = 0 (0x0) Success.

>blhost.exe -p COM1 -- get-property 1

Ping responded in 1 attempt(s)

Inject command 'get-property'

Response status = 0 (0x0) Success.
```

Example

```
Response word 1 = 1258358016 (0x4b010500)

Current Version = K1.5.0
```

Now that the bootloader update is complete, you can use it to flash new or original applications into it.

```
>blhost.exe -p COM1 -- write-memory 0xa000 <path>led_demo.bin

Ping responded in 1 attempt(s)

Inject command 'write-memory'

Preparing to send 2004 (0x7d4) bytes to the target.

Successful generic response to command 'write-memory'

Successful generic response to command 'write-memory'

Response status = 0 (0x0) Success.

Wrote 2004 of 2004 bytes.
```

After reset and a short wait, you should see that the LED demo running.

```
>blhost.exe -p COM1 -- reset

Ping responded in 1 attempt(s)

Inject command 'reset'

Successful generic response to command 'reset'

Response status = 0 (0x0) Success.
```

Warnings and exceptions:

- If you have only 16K RAM on the chip, you need to ensure the flashloader is able to fit into the RAM range supported by the original bootloader.
- If you only want to update the bootloader but preserve the existing application on the flash, you should not use the '-- flash-erase-all command'. Use the '-- flash-erase-region' command to only erase the original bootloader flash area.

Chapter 9
Appendix C - BusPal: Bus-friendly blhost companion tool

9.1 Introduction

BusPal is an embedded software tool that is available as a companion to blhost. The tool acts as a bus translator with an established connection with blhost over UART and with the target device over I2C, SPI, or CAN, and assists blhost in carrying out commands and responses from the USB target device. The BusPal is available for selected platforms. The source code for BusPal is provided with the MCU bootloader release, and can be customized to run on other platforms.

The diagram below illustrates the role BusPal plays in blhost communication with the target device.

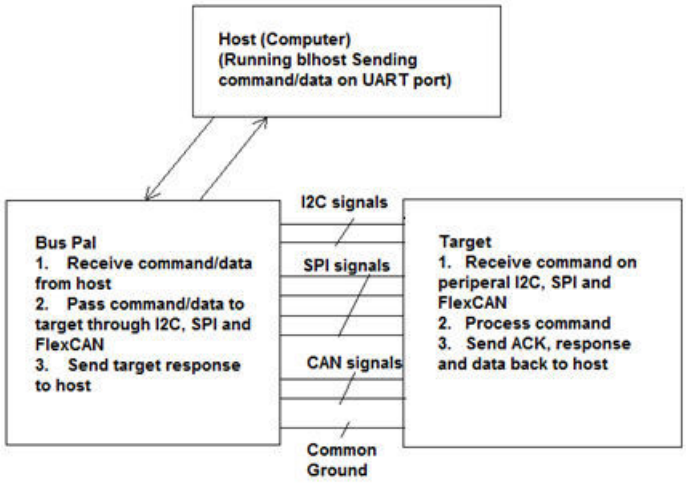

Figure 3. Role of BusPal in interfacing blhost and target device

9.2 Supported platforms and features

BusPal software is currently supported for selected platforms. The entire source code is available with the MCU bootloader release, and can be easily customized for any other platform of the customer's choice.

9.2.1 FRDM-KL25Z

The FRDM-KL25Z NXP Freedom development platform is available for order on the NXP website. The BusPal supports low speed I2C and SPI peripherals to interface with the target device, and the UART to communicate with blhost running on host PC.

9.2.2 TWR-KV46F150M

The TWR-KV46F150M NXP Tower System module is available for order on the NXP website. The BusPal supports high-speed I2C, SPI, and CAN peripherals to interface with the target device. The TWR-SER Tower System module is required for UART connection to the blhost running on a host PC.

9.2.3 TWR-KV65F180M

The TWR-KV65F180M Tower System module is available for order at www.nxp.com. The BusPal supports high-speed I2C, SPI, and CAN peripherals to interface with the target device. The TWR-SER Tower System module can be used for UART connection to the blhost running on a host PC. The K65 version of BusPal also supports communication with the host PC over USB-HID.

9.3 Usage with blhost

The BusPal assists blhost in sending a command to the target device over I2C, SPI, or CAN interface, and collects responses from the peripheral and provides feedback to the blhost.

The blhost command line provides a way to make blhost aware of the BusPal. The -b or --buspal command line options of blhost are used for this purpose. The peripheral and its connection parameters are required following the -b or --buspal option. This is the syntax of the blhost command for using BusPal:

blhost.exe -p COMi [-b|--buspal spi[,<speed>,<polarity>,<phase>,lsb|msb] | i2c,[<address>,<speed>] | can,<speed>,<txid>,<rxid>]

- SPI parameters:

- Speed: 100 K (default)
- Polarity: ActiveLow (default)
- Phase: SecondEdge (default)
- Lsb or msb: Msb first (default)

- I2C parameters:

- Address: Slave I2C node (7 bit address -- default is 0x10)
- Spead: 100 K (default)

- CAN parameters:

- Speed: 0, 1, 2, 4 (125 K, 250 K, 500 K, 1 M. default is 1 M)
- TxId: Standard Format 11 bits only ID (default is 0x321)
- RxId: Standard Format 11 bits only ID (default is 0x123)

9.4 Pinmux and configurations

The following sections describe the Pinmux configuration used in BusPal software for each peripheral of the supported platforms.

9.4.1 FRDM-KL25Z Freedom development platform

Table 5. UART0

Signal name	Pin name	ALT mode
UART0_RX	PTA1	ALT2
UART0_TX	PTA2	ALT2

Table 6. SPI0

Signal name	Pin name	ALT mode	Signal name on target
SPI0_PCS0	PTD0	ALT2	SPI_PCS0
SPI0_SCK	PTD1	ALT2	SPI_SCK
SPI0_MOSI	PTD2	ALT2	SPI_MOSI
SPI0_MISO	PTD3	ALT2	SPI_MISO

Table 7. I2C0

Signal name	Pin name	ALT mode	Signal name on target
I2C0_SCL	PTC8	ALT2	I2C_SCL
I2C0_SDA	PTC9	ALT2	I2C_SDA

Note that there are no pull-up resistors on PTC8 and PTC9 pins.

9.4.2 TWR-KV46F150M Tower System module

Because the TWR-KV46F150M Tower System module only has one OpenSDA USB port, use the COM1 port on the TWR-SER Tower System module to communicate with the TWR-KV46F150M BusPal.

Table 8. UART1

Signal name	Pin name	ALT mode	Signal name on target
UART1_RX	PTE0	ALT3	TWR-SER
UART1_TX	PTE1	ALT3	TWR-SER

Table 9. DSPI0

Signal name	Pin name	ALT mode	Signal name on target
SPI0_PCS0	PTA14	ALT2	Elev B46
SPI0_SCK	PTA15	ALT2	Elev B48
SPI0_MOSI	PTA16	ALT2	Elev B45
SPI0_MISO	PTA17	ALT2	Elev B44

Table 10. I2C0

Signal name	Pin name	ALT mode	Signal name on target
I2C0_SCL	PTC14	ALT3	Elev A7
SPI0_SCK	PTC15	ALT3	Elev A8

Table 11. FlexCAN0

Signal name	Pin name	ALT mode	Signal name on target
CAN0_TX	PTA12	ALT2	J13 1
CAN0_RX	PTA13	ALT2	J13 2

FlexCAN support speed:

The maximum speed supported on the current FlexCAN IP is 1 MHz.

The speedIndex value is 0, 1, 2, and 4, and represents the real speed as follows:

- 0 - 125 KHz
- 1 - 250 KHz
- 2 - 500 KHz
- 4 - 1 MHz (default)

9.4.3 TWR-K65F180M Tower System module

The COM1 port on the TWR-SER board is used to communicate with the K65 BusPal. The USB MINIAB port on the TWR-SER board is used for BusPal over USB.

Table 12. UART1

Signal name	Pin name	ALT mode	Signal name on target
UART4_RX	PTE25	ALT3	Elev A47
UART4_TX	PTE24	ALT3	Elev A48

UART connections:

- Elev A47 - TWR-SER J17 - 1
- Elev A48 - TWR-SER J19 - 1

Table 13. DSPI0

Signal name	Pin name	ALT mode	Signal name on target
SPI0_PCS0	PTA11	ALT2	Elev B46
SPI0_SCK	PTA12	ALT2	Elev B48
SPI0_MOSI	PTA13	ALT2	Elev B45
SPI0_MISO	PTA14	ALT2	Elev B44

Table 14. I2C0

Signal name	Pin name	ALT mode	Signal name on target
I2C0_SDA	PTE18	ALT4	Elev A8
SPI0_SCL	PTE19	ALT4	Elev A7

Table 15. FlexCAN0

Signal name	Pin name	ALT mode	Signal name on target
CAN0_TX	PTA30	ALT2	TWR-SER J7 - 3
CAN0_RX	PTA31	ALT2	TWR-SER J7 - 1

FlexCAN support speed:

The maximum speed supported on the current FlexCAN IP is 1 MHz.

The speedIndex value is 0, 1, 2, and 4, and represents the real speed as follows:

- 0 - 125 KHz
- 1 - 250 KHz
- 2 - 500 KHz
- 4 - 1 MHz (default)

FlexCAN connections:

- Elev A47 - TWR=SER J17 - 1
- Elev A28 - TWR-SER 19 - 1

9.5 Build the BusPal project

The BusPal source code is available in the `apps\bus_pal` directory in NXP_Kinetis_Bootloader_2.0.0 package on www.nxp.com/MCUBOOT.

A separate subproject is available to build the binary for each supported platform. The KV46 and K64 platforms support IAR Embedded Workbench for ARM, KDS, and KEIL tool chains. The K65 platform supports the IAR tool chain only.

9.6 Platform setup

As illustrated in Figure 3, "Role of BusPal in interfacing blhost and target device", the supported platform (FRDM-KL25Z/TWR-KV46F150M/TWR-K65F150M) should be connected to host PC via UART (USB-HID is also supported on K65). The BusPal image should be running on the platform, and blhost should be running on the host PC. The target platform runs the bootloader image.

The connection between the target platform and BusPal platform hardware depends on what peripheral is being used to interface with the bootloader. For example, if the I2C peripheral is used for interfacing with the bootloader, the I2C probe pins for SCL and SDA available on the boards for BusPal and target platforms should be physically connected using fly-wires. The connection should be well secured for reliable data transfers.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support