Contents

Chapter 1 Introduction........................................................................................... 4

Chapter 2 Overview............................................................................................... 5

Chapter 3 MCU bootloader.................................................................................... 6

Chapter 4 blhost Utility application.........................................................................7

4.1 **blhost** Options............................................................................................................................7
  4.1.1 -? (Help menu)...............................................................................................................................7
  4.1.2 -v (Version).....................................................................................................................................7
  4.1.3 -i <name>,<address>,<speed> (Connect over I2C).................................................................. 8
  4.1.4 -p <name>,<speed> (Connect over UART)................................................................................... 8
  4.1.5 -s <name>,<speed>,<polarity>,<phase>, {lsb|msb} (Connect over SPI)......................................9
  4.1.6 -u [[{vid}],{pid}]][{path}] (Connect over USB).................................................................. 9
  4.1.7 -b (BusPal)...................................................................................................................................11
  4.1.8 -l(LPCUSBSIO).............................................................................................................................11
  4.1.9 -V (Verbose)...............................................................................................................................11
  4.1.10 -d (Debug)...............................................................................................................................12
  4.1.11 -j (JSON output type)..............................................................................................................12
  4.1.12 -n (No ping)..............................................................................................................................12
  4.1.13 -t (Timeout)..............................................................................................................................12

4.2 **blhost** Commands..........................................................................................12
  4.2.1 reset...........................................................................................................................................13
  4.2.2 get-property <tag>[<memoryID>]......................................................................................... 13
  4.2.3 set-property <tag> <value>[<memoryID>]............................................................................ 14
  4.2.4 flash-erase-region <addr> <byte_count>[memoryID].......................................................... 15
  4.2.5 flash-erase-all[memoryID]...................................................................................................... 15
  4.2.6 flash-erase-all-unsecure............................................................................................................15
  4.2.7 read-memory <addr> <byte_count> [file][memoryID]......................................................... 16
  4.2.8 write-memory <addr> [<file> | {{<hex-data>}}][memoryID].................................................. 16
  4.2.9 fill-memory <addr> <byte_count> [pattern] [word | short | byte]......................................... 16
  4.2.10 receive-sb-file <file>............................................................................................................ 16
  4.2.11 execute <address> <arg> <stackpointer>...........................................................................16
  4.2.12 call <address> <arg>.............................................................................................................17
  4.2.13 flash-security-disable <key>............................................................................................. 17
  4.2.14 flash-program-once <index> <byteCount> <data>.............................................................. 17
  4.2.15 flash-read-once <index> <byteCount>................................................................................ 17
  4.2.16 efuse-program-once <index> <data>.................................................................................. 18
  4.2.17 efuse-read-once <addr>........................................................................................................ 18
  4.2.18 flash-read-resource <addr> <byteCount> [option] [file]...................................................... 18
  4.2.19 configure-memory <memoryID> <addr>............................................................................. 18
  4.2.20 flash-image <file> [erase][memoryID]................................................................................ 18
  4.2.21 reliable-update <addr>......................................................................................................... 18
  4.2.22 generate-key-blob <dek_file> <blob_file> [keySel].............................................................. 19
  4.2.23 key-provisioning <operation> [arguments].......................................................................... 19
  4.2.24 load-image <file>.................................................................................................................. 19
  4.2.25 program-aeskey <file>...........................................................................................................20
Chapter 1
Introduction

This document describes the usage of the blhost 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 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 <key>`, 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 if that pin is enabled in the BCA. For more information on boot pins and the platforms supported for each supported mcu, see the *MCU Bootloader Demo Applications User's Guide* (document MBOOTDEMOUG).
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, Appendix C, and Appendix D.
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 and flash-based implementations. There are example applications in the package which demonstrates how to interface with the MCU bootloader.
Chapter 4
blhost Utility application

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

The blhost application supports multi-platforms, including Windows, Linux (X86-based), MACOSX, and Linux (Arm-based).

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

MCU bootloader typically supports other peripherals such as I2C, SPI, and CAN, depends on each mcu specific configuration. However, the blhost application cannot interface with the MCU bootloader over these transports without external hardware (except Arm Linux version).

See Appendix C for the description of BusPal embedded software. 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.

See Appendix D for the description of LPC USB Serial I/O(LPCUSBSIO), a firmware built in LPC Link2. The LPCUSBSIO acts as a bus translator, and establishes connection with blhost over USB-HID, and the MCU bootloader device over I2C and SPI.

The arguments for blhost consist 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

blhost utility provides an easy to use command line interface. The syntax of the usage text is:

```bash
blhost.exe [-?|--help] [-v|--version] [-p|--port <name>[,<speed>]]
    [-i|--i2c <name>[,<address>,<speed>]]
    [-s|--spi <name>[,<speed>,<polarity>,<phase>,lsb|msb]]
    [-b|--buspal [spi[,<speed>,<polarity>,<phase>,lsb|msb] | i2c[,<address>,<speed>]]
    [-l|--lpcusbsio spi[,<port>,<pin>,<speed>,<polarity>,<phase>]
    [-u|--usb [[<vid>,]<pid>] | [<path>]]] [-V|--verbose]
    [-d|--debug] [-j|--json] [-n|--noping] [-t|--timeout <ms] -- command <args...>
```

4.1.1 -? (Help menu)

The below commands can be used to get "help" guide. This guide lists all the options and commands supported by blhost utility.

blhost -?
blhost -h
blhost --help

4.1.2 -v (Version)

This below command prints the version of the blhost application.

blhost -v
4.1.3 -i <name>[,<address>,<speed>] (Connect over I2C)

The below command can be used to connect to MCU bootloader over I2C. This option is valid for Arm-Linux blhost. This option can have one or three arguments.

blhost -i <name> -- command
blhost --i2c <name> -- command
blhost -i <name>[,<address>,<speed>] -- command

name: I2C port.
address: I2C slave address (default 0x10, if not specified)
speed: I2C baud rate/frequency (default 100 kbps, if not specified)

<table>
<thead>
<tr>
<th>Obtaining the port number</th>
<th>Windows</th>
<th>Linux</th>
<th>MAC</th>
<th>Arm Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Use the command $dmesg</td>
</tr>
<tr>
<td>Example</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>$sudo ./blhost -i /dev/ i2c-1,0x10,100 -- command</td>
</tr>
<tr>
<td>Administrative rights</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Yes</td>
</tr>
<tr>
<td>required for this operation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1.4 -p <name>[,<speed>] (Connect over UART)

The below command can be used in Windows platform to connect to MCU bootloader over UART. This option has two arguments.

blhost - p <name>[,<speed>] -- command

name: serial port
speed: baud rate (default - 57600, if not specified)

blhost -p COMx -- command
blhost -- port COMx -- command

<table>
<thead>
<tr>
<th>Obtaining the port number</th>
<th>Windows</th>
<th>Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Check in the Device Manager</td>
<td>Use the command $dmesg</td>
<td>grep tty</td>
</tr>
<tr>
<td>Example</td>
<td>blhost - port COM1 --command</td>
<td>$ sudo ./blhost -p /dev/ttyACM0 -- command</td>
<td>$ sudo ./blhost -p /dev/cu.usbmodemFA121 -- command</td>
</tr>
<tr>
<td>Administrative rights</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>required for this operation.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.1.5 -s <name>[,<speed>,<polarity>,<phase>, lsb|msb] (Connect over SPI)

The below command can be used to connect to MCU bootloader over SPI. This option is valid for Arm-Linux blhost. This option can have one, two, or five arguments.

blhost -s <name> -- command
blhost --spi <name> -- command
blhost -s <name>,<speed> -- command
blhost -s <name>[,<speed>,<polarity>,<phase>, lsb|msb] -- command

name: SPI port.
speed: SPI baud rate/frequency (default 100kpbs, if not specified)
polarity: SPI polarity, 0 for active high, 1 for active low (default)
phase: SPI phase, 0 for rising edge sampling, 1 for failing edge sampling (default)
lsb|msb: SPI sequence endian, “lsb” or “msb” (default)

Table 3. Command usage in different operating systems

<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>Linux</th>
<th>MAC</th>
<th>Arm Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtaining the port number</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Use the command $dmesg</td>
</tr>
<tr>
<td>Example</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>$sudo ./blhost -s /dev/spidev0.0,100,1,1,msb --command</td>
</tr>
<tr>
<td>Administrativerights required for this operation</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4.1.6 -u [[[<vid>,]<pid>]][[<path>]] (Connect over USB)

The below command can be used to connect to MCU bootloader over USB HID. This option can have one or two arguments.

blhost -u [[[<vid>,]<pid>]] -- command
blhost --usb [[[<vid>,]<pid>]] -- command
blhost -u [[<path>]] -- command
vid: Specify USB vendor ID of device
pid: Specify USB product ID of device
path: Specify USB path ID of device

**NOTE**
- The default vid,pid is -u 0x15a2,0x0073. Hence this command can also be used as blhost -u -- command
- --u and --usb can be used interchangeably in this command
### Table 4. Command usage in different operating systems

<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtaining VID and PID</td>
<td>Check in the Device Manager</td>
<td>Use the command <code>lsusb</code></td>
<td>Click Apple logo at top left &gt; About This Mac &gt; System Report... &gt; USB</td>
</tr>
<tr>
<td>Example</td>
<td><code>blhost -u HIDVID_15A2&amp;PID_0073&amp;I4A67 53&amp;0&amp;0000</code></td>
<td><code>$ sudo ./blhost -u /dev/hidraw1 -- command</code></td>
<td><code>$ sudo ./blhost -u 0x15a2,0x0073 -- command</code></td>
</tr>
<tr>
<td>Path</td>
<td>Check in the Device Manager</td>
<td>Use the command <code>dmesg</code></td>
<td>similar to Linux</td>
</tr>
<tr>
<td>Administrative rights required for this operation</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

![Device Manager](image)

**Figure 1.** Check USB VID & PID on Windows
### 4.1.7 -b (BusPal)

See Appendix C on the usage of blhost with BusPal.

```
blhost -b
```

```
blhost --buspal
```

### 4.1.8 -l (LPCUSBSIO)

See Appendix D on the usage of blhost with LPC USB Serial I/O

```
blhost -l
```

```
blhost --lpcusbsio
```

### 4.1.9 -V (Verbose)

The below command prints detailed log information.

```
blhost -V
```

```
blhost --verbose
```
4.1.10  -d (Debug)

The below command prints every byte sent and received from the MCU bootloader. The bytes sent to the bootloader are denoted as \[xx\] and bytes received from the bootloader are denoted as \(<xx>\).

`blhost -d`
`blhost --debug`

4.1.11  -j (JSON output type)

The below command prints 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" : [ 1258423808 ],
    "status" : {
        "description" : "0 (0x0) Success."
    },
    "value" : 0
}
```

4.1.12  -n (No ping)

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 has already been established. Also, the ping can safely be suppressed when using I2C, SPI, or CAN transports. Sending a ping before every command does not cause an issue.

`blhost -n`
`blhost --noping`

4.1.13  -t (Timeout)

The below command sets serial read timeout in milliseconds. This value allows blhost to timeout when waiting on data from the MCU bootloader over a serial transport.

The default is 5000 milliseconds.

Example : `blhost -u -t 50000 -- flash-erase-all`

4.2  blhost Commands

The "help" guide of blhost lists all of the options and commands supported by the blhost utility.

All commands are not supported on all MCU bootloader platforms. If a command is not supported by the MCU bootloader, it returns `k_StatusUnknownCommand`. To know the supported command on a specific mcu, use `get-property 7`.

All the <addr> in the blhost commands is the byte address. For FTFx flash, 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 supported due to flash security settings.
4.2.1 reset

This command resets the device. A response packet is sent prior to resetting the device.
The reset command is used to reset the device.
Command: `-- reset`
Usage: `blhost -u -- reset`

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

This command is used to query the bootloader about various properties and settings. Each supported property has a unique integer tag value.
Example: `-- get-property 10`
tag:
  • 0x01 The current version of the MCU bootloader. For more information on MCU bootloader version, see section 5.1.
  • 0x02 A mask of the available peripherals. For more information on available peripherals, see section 5.2.
  • 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. For more information on available commands, see section 5.3.
  • 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. Table 5 lists the details on memoryID. For additional information on memoryID, refer to MCU Bootloader reference manual or ROM chapter in device’s reference manual
• 0x1a Reliable update status - the status of the reliable update operation.
• 0x1b Flash page size - Size of flash page, <index> is required.
• 0x1c IRQ notifier pin - The pin selected as nIRQ pin.
• 0x1d FFR key store update operation - The selected operation for FFR keystore update.
• 0x1e Byte Write Timeout(ms) - The timeout value for outgoing packet bytes.

NOTE
Not all properties are supported by all target. To check the supported properties, see the target’s user manual or the reference manual.

Table 5. Memory ID

<table>
<thead>
<tr>
<th>Internal Memory</th>
<th>Device internal memory space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Internal Memory (Default selected memory)</td>
</tr>
<tr>
<td>16 (0x10)</td>
<td>Execute-only region on internal flash (Only used for flash-erase-all)</td>
</tr>
</tbody>
</table>

Mapped External Memory

The memories that are remapped to internal space, and must be accessed by internal addresses. (IDs in this group are only used for flash-erase-all and configure-memory, and ignored by write-memory, read-memory, flash-erase-region and flash-image(use default 0))

| 1               | QuadSPI Memory                                    |
| 8               | SEMC NOR Memory                                   |
| 9               | FlexSPI NOR Memory                                |
| 10 (0xa)        | SPIFI NOR Memory                                  |

Unmapped External Memory

Memories which cannot be remapped to internal space, and only can be accessed by memories’ addresses. (Must be specified for all commands with <memoryId> argument)

| 256 (0x100)    | SEMC NAND Memory                                  |
| 257 (0x101)    | SPI NAND Memory                                   |
| 272 (0x110)    | SPI NOR/EEPROM Memory                            |
| 273 (0x111)    | I2C NOR/EEPROM Memory                            |
| 288 (0x120)    | uSDHC SD Memory                                  |
| 289 (0x121)    | uSDHC MMC Memory                                  |

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

Example: -- set-property 10 0

This command changes properties and options in the bootloader. For memoryID details, see Table 5. 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.
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.

- 0x1c IRQ notifier pin - The pin selected as nIRQ pin.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>[31]</th>
<th>[30:16]</th>
<th>[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enable nIRQ pin, 0: disable, 1: enable</td>
<td>GPIO port index</td>
<td>GPIO pin index</td>
<td></td>
</tr>
</tbody>
</table>

FFR key store update operation - The selected operation for FFR keystore update 0 for Keyprovisioning; 1 for write-memory.

- 0x1e Byte Write Timeout (ms) - The timeout value for outgoing packet bytes.

**NOTE**

Not all properties are supported by all target. To check the supported properties, see the target's user manual or the reference manual.

### 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 entire sector(s) containing the start and end address is erased. For FTFx Flash, the start address and byte count should be a multiple of the word (32-bit).

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 for FTFx flash controller. On MCU devices which do not support this command, 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 and RAM.

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 {{aabbcdddeefff001020304}}

Write memory specified 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, and RAM. 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.

For FTFx Flash, writing requires the start address to be word (32-bit) 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).

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 (32-bit), 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
For Flash-resident bootloader and flashloader in sdk package, this command 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. <arg> is the parameter passed to the called function. 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.

For some ROM bootloader, this command can be used to jump to boot image. In this case, <address> is PC of the image, which is the second word (32-bit) of the image, <arg> is the address the image reside at and <stackpointer> is the SP of the image, which should be the first word (32-bit) of the image.

The effective prototype of the called function is:

```c
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.

The effective prototype of the called function is :

```c
void function(uint32_t arg);
```

**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 FTFx 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 should be 4-byte aligned or 8-byte aligned. The write size depends on the mcu flash controller. For the alignment and write information of different devices, see the reference manual.

### 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 <index> <data>

Example: --efuse-program-once 6 0x04030201
This command writes data to a specific efuse word (32-bit). Each efuse bit can only be programmed once.

4.2.17 efuse-read-once <addr>

Example --efuse-read-once 6
Returns the contents of a specific efuse word (32-bit).

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

Example: -- flash-read-resource 0 4 1 firmwareID.txt
Reads the contents of FTFx 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 0xffe000
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.

NOTE
For more details about reliable update, refer to the MCU Bootloader Reference Manual document.
4.2.22  generate-key-blob <dek_file> <blob_file> [key_sel]

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. [key_sel] selects the blob key encryption key(BKEK) used to generate the key blob.

For devices with SNVS, the valid options of [key_sel] are:

• 0, 1 or "OTPMK": OTPMK from FUSE or OTP (default)
• 2 or "ZMK": ZMK from SNVS
• 3 or "CMK": CMK from SNVS.

For devices without SNVS, this option is ignored.

4.2.23  key-provisioning <operation> [arguments…]

The key-provisioning command is a pack of several security related commands.

• enroll
Example: -- key-provisioning enroll

Enroll key provisioning feature. No argument for this operation.

• set_user_key <type><file>[,<size>]
Example: -- key-provisioning set_user_key 0xB userKey.bin

Send the user key specified by <type> to bootloader. <file> is the binary file containing user key plain text. If <size> is not specified, the entire <file> will be sent, otherwise, blhost only sends the first <size> bytes.

• set_key <type> <size>
Example: -- key-provisioning set_key 0x1 0x100

Generate <size> bytes of the key specified by <type>.

• write_key_nonvolatile [memoryID]
Example: -- key-provisioning write_key_nonvolatile 0x110

Write the key to a nonvolatile memory.

• read_key_nonvolatile [memoryID]
Example: -- key-provisioning read_key_nonvolatile 0x110

Load the key from a nonvolatile memory to bootloader.

• write_key_store <file>[,<size>]
Send the key store to bootloader. <file> is the binary file containing key store. If <size> is not specified, the entire <file> will be sent. Otherwise, only send the first <size> bytes.

• read_key_store <file>[,<size>]
Read the key store from bootloader to host(PC). <file> is the binary file to store the key store.

<type> and corresponding <size> are target specific values, and various on different devices. For details, see the ROM chapter in the Reference Manual.

4.2.24  load-image <file>

Example: -- load-image ivt_application_0x800.bin
Send a boot image <file> to the device. Only binary file is supported. And the <file> must be a bootable image which contains the boot image header supported by MCU bootloader, for example, IVT and Boot Data.

4.2.25 program-aeskey <file>

Example: -- program-aeskey aes_key.bin

Send a raw binary, which contains an aes key, to the devices and program it to the OTP field.
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>
<thead>
<tr>
<th>Bit</th>
<th>[31:24]</th>
<th>[23:16]</th>
<th>[15:8]</th>
<th>[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Name</td>
<td>Major version</td>
<td>Minor version</td>
<td>Bugfix version</td>
</tr>
</tbody>
</table>

5.2 Available peripherals

The value of this property is a one-word bitfield that lists the peripherals supported by the bootloader.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>USB HID</td>
</tr>
<tr>
<td>3</td>
<td>CAN</td>
</tr>
<tr>
<td>2</td>
<td>SPI slave</td>
</tr>
<tr>
<td>1</td>
<td>I2C slave</td>
</tr>
<tr>
<td>0</td>
<td>UART</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Bit</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:21]</td>
<td>reserved</td>
</tr>
<tr>
<td>20</td>
<td>key-provisioning (0x15)</td>
</tr>
<tr>
<td>19</td>
<td>reserved</td>
</tr>
<tr>
<td>18</td>
<td>generate-key-blob (0x13)</td>
</tr>
<tr>
<td>17</td>
<td>reliable-update (0x12)</td>
</tr>
<tr>
<td>16</td>
<td>configure-memory (0x11)</td>
</tr>
<tr>
<td>15</td>
<td>flash-read-resource (0x10)</td>
</tr>
<tr>
<td>14</td>
<td>flash-read-once (0x0f)</td>
</tr>
<tr>
<td>13</td>
<td>flash-program-once (0x0e)</td>
</tr>
<tr>
<td>12</td>
<td>flash-erase-all-unsecure (0x0d)</td>
</tr>
</tbody>
</table>

Table continues on the next page...
<table>
<thead>
<tr>
<th>Bit</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>set-property (0x0c)</td>
</tr>
<tr>
<td>10</td>
<td>reset (0x0b)</td>
</tr>
<tr>
<td>9</td>
<td>call (0x0a)</td>
</tr>
<tr>
<td>8</td>
<td>execute (0x09)</td>
</tr>
<tr>
<td>7</td>
<td>receive-sb-file (0x08)</td>
</tr>
<tr>
<td>6</td>
<td>get-property (0x07)</td>
</tr>
<tr>
<td>5</td>
<td>flash-security-disable (0x06)</td>
</tr>
<tr>
<td>4</td>
<td>fill-memory (0x05)</td>
</tr>
<tr>
<td>3</td>
<td>write-memory (0x04)</td>
</tr>
<tr>
<td>2</td>
<td>read-memory (0x03)</td>
</tr>
<tr>
<td>1</td>
<td>flash-erase-region (0x02)</td>
</tr>
<tr>
<td>0</td>
<td>flash-erase-all (0x01)</td>
</tr>
</tbody>
</table>
Chapter 6
Revision history

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

<table>
<thead>
<tr>
<th>Revision number</th>
<th>Date</th>
<th>Substantive changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12/2014</td>
<td>Initial release</td>
</tr>
<tr>
<td>1</td>
<td>07/2015</td>
<td>Kinetis bootloader 1.2.0 updates</td>
</tr>
<tr>
<td>2</td>
<td>09/2015</td>
<td>Kinetis bootloader K80 standalone updates</td>
</tr>
<tr>
<td>3</td>
<td>04/2016</td>
<td>Kinetis Bootloader v.2.0.0 release</td>
</tr>
<tr>
<td>4</td>
<td>03/2018</td>
<td>RT1050 Flashloader 1.1 release</td>
</tr>
<tr>
<td>4.1</td>
<td>04/2018</td>
<td>MCU Bootloader v2.5.0 release</td>
</tr>
<tr>
<td>5</td>
<td>09/2018</td>
<td>MCU Bootloader v2.6.0 release</td>
</tr>
<tr>
<td>6</td>
<td>11/2018</td>
<td>MCU Bootloader v2.7.0 release</td>
</tr>
<tr>
<td>7</td>
<td>04/2020</td>
<td>Updated for blhost 2.6.2 release</td>
</tr>
<tr>
<td>8</td>
<td>11/2020</td>
<td>Updated for blhost 2.6.5 release</td>
</tr>
<tr>
<td>9</td>
<td>29 July 2021</td>
<td>Updated for blhost 2.6.7 release</td>
</tr>
</tbody>
</table>
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.

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 by one peripheral channel, for example UART, all enabled peripherals except the active/communicating peripheral are shut down.

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.
Appendix B
Updating a bootloader image

B.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.

B.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.

B.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 users 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.

B.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 host PC 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:

```plaintext
> 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 = 1258423808 (0x4b020600)
Current Version = K2.7.0

> 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 = Region0: 0x0-0x3FFF (16 KB)
Region1: 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 0x1fffe11, 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 0x1fffe11 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.

```plaintext
>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 = Region0: 0x0-0x0 (0 bytes)
Region1: 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:**

```plaintext
>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.**

```plaintext
>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 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.

```plaintext
>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 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

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 (2.7.0)

>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.
Response word 1 = 1258423808 (0x4b020600)
Current Version = K2.7.0

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.
Appendix C
BusPal: Bus-friendly blhost companion tool

C.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.

C.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.

C.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.

C.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.
C.2.3 TWR-K65F180M

The TWR-K65F180M 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.

C.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 kHz (default)
  - Polarity: 1 - ActiveLow (default), 0 - Active High
  - Phase: 1 - Falling edge (default), 0 - Rising edge
  - lsb or msb: "msb" (default), "lsb"

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

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

C.4 Pinmux and configurations

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

C.4.1 FRDM-KL25Z Freedom development platform

Table 11. UART0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART0_RX</td>
<td>PTA1</td>
<td>ALT2</td>
</tr>
<tr>
<td>UART0_TX</td>
<td>PTA2</td>
<td>ALT2</td>
</tr>
</tbody>
</table>
Table 12. SPI0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI0_PCS0</td>
<td>PTD0</td>
<td>ALT2</td>
<td>SPI_PCS0</td>
</tr>
<tr>
<td>SPI0_SCK</td>
<td>PTD1</td>
<td>ALT2</td>
<td>SPI_SCK</td>
</tr>
<tr>
<td>SPI0_MOSI</td>
<td>PTD2</td>
<td>ALT2</td>
<td>SPI_MOSI</td>
</tr>
<tr>
<td>SPI0_MISO</td>
<td>PTD3</td>
<td>ALT2</td>
<td>SPI_MISO</td>
</tr>
</tbody>
</table>

Table 13. I2C0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C0_SCL</td>
<td>PTC8</td>
<td>ALT2</td>
<td>I2C_SCL</td>
</tr>
<tr>
<td>I2C0_SDA</td>
<td>PTC9</td>
<td>ALT2</td>
<td>I2C_SDA</td>
</tr>
</tbody>
</table>

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

C.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 14. UART1

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART1_RX</td>
<td>PTE0</td>
<td>ALT3</td>
<td>TWR-SER</td>
</tr>
<tr>
<td>UART1_TX</td>
<td>PTE1</td>
<td>ALT3</td>
<td>TWR-SER</td>
</tr>
</tbody>
</table>

Table 15. DSPI0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI0_PCS0</td>
<td>PTA14</td>
<td>ALT2</td>
<td>Elev B46</td>
</tr>
<tr>
<td>SPI0_SCK</td>
<td>PTA15</td>
<td>ALT2</td>
<td>Elev B48</td>
</tr>
<tr>
<td>SPI0_MOSI</td>
<td>PTA16</td>
<td>ALT2</td>
<td>Elev B45</td>
</tr>
<tr>
<td>SPI0_MISO</td>
<td>PTA17</td>
<td>ALT2</td>
<td>Elev B44</td>
</tr>
</tbody>
</table>

Table 16. I2C0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C0_SCL</td>
<td>PTC14</td>
<td>ALT3</td>
<td>Elev A7</td>
</tr>
<tr>
<td>SPI0_SCK</td>
<td>PTC15</td>
<td>ALT3</td>
<td>Elev A8</td>
</tr>
</tbody>
</table>

Table 17. FlexCAN0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN0_TX</td>
<td>PTA12</td>
<td>ALT2</td>
<td>J13 1</td>
</tr>
<tr>
<td>CAN0_RX</td>
<td>PTA13</td>
<td>ALT2</td>
<td>J13 2</td>
</tr>
</tbody>
</table>
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)

C.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 18. UART4

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART4_RX</td>
<td>PTE25</td>
<td>ALT3</td>
<td>Elev A47</td>
</tr>
<tr>
<td>UART4_TX</td>
<td>PTE24</td>
<td>ALT3</td>
<td>Elev A48</td>
</tr>
</tbody>
</table>

UART connections:
- Elev A47 - TWR-SER J17 - 1
- Elev A48 - TWR-SER J19 - 1

Table 19. DSPI0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI0_PCS0</td>
<td>PTD11</td>
<td>ALT2</td>
<td>Elev B46</td>
</tr>
<tr>
<td>SPI0_SCK</td>
<td>PTD12</td>
<td>ALT2</td>
<td>Elev B48</td>
</tr>
<tr>
<td>SPI0_MOSI</td>
<td>PTD13</td>
<td>ALT2</td>
<td>Elev B45</td>
</tr>
<tr>
<td>SPI0_MISO</td>
<td>PTD14</td>
<td>ALT2</td>
<td>Elev B44</td>
</tr>
</tbody>
</table>

Table 20. I2C0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C0_SDA</td>
<td>PTE18</td>
<td>ALT4</td>
<td>Elev A8</td>
</tr>
<tr>
<td>I2C0_SCL</td>
<td>PTE19</td>
<td>ALT4</td>
<td>Elev A7</td>
</tr>
</tbody>
</table>

Table 21. FlexCAN0

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin name</th>
<th>ALT mode</th>
<th>Signal name on target</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN0_TX</td>
<td>PTA30</td>
<td>ALT2</td>
<td>TWR-SER J7 - 3</td>
</tr>
<tr>
<td>CAN0_RX</td>
<td>PTA31</td>
<td>ALT2</td>
<td>TWR-SER J7 - 1</td>
</tr>
</tbody>
</table>

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:
C.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.

C.6 Platform setup

As illustrated in Figure 4, "Role of BusPal in interfacing blhost and target device", the supported platform (FRDM-KL25Z/TWR-KV46F150M/TWR-K65F180M) 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.
Appendix D
LPC USB Serial I/O: A Bus bridge for LPC Link2 tool

D.1 Introduction
The LPC USB Serial I/O (LPCUSBSIO) is a firmware, which can communicate with serial I/O devices connected to LPC Link2. The LPCUSBSIO contains two parts. One, named LPC USB Serial I/O Library, is a generic API provided to PC application. This part is built in the blhost. The other one is named LPC Serial I/O port is a firmware, which receives USB messages and transfers to devices using I2C, SPI and GPIO interfaces. This part is build in the LPC Link2.

For details on LPC Link2, see LPC Link2.

For details on LPC USB Serial I/O, see LPCUSBSIO.

D.2 Usage with blhost
Currently, blhost supports I2C and SPI interfaces of LPCUSBSIO.
To use LPCUSBSIO, The -l|--lpcusbsio command-line options must be provided.
The peripheral type and parameters are required following the -l|--lpcusbsio option.
This is the syntax of the blhost command for using LPCUSBSIO.

```
blhost.exe [-u [<vid>,<pid>] | [<path>]] -l|--lpcusbsio spi,[<port>,<pin>,<speed>,<polarity>,<phase>]
| i2c,[<address>,<speed>]
```

D.3 USB parameters
[-u]: Optional. LPCUSBSIO can only use USB to communicate with the host.
[<vid>,<pid>]: Optional. The LPC Link2 uses the VID(0x1fc9) and PID(0x0090). A new NXP MCU Link uses VID(0x1fc9) and PID(0x0143). Alternatively, you can specify a custom VID and PID numbers.
[<path>]: Reserved option for LPCUSBSIO. Current LPCUSBSIO Library does not support multiple of LPC Link2 connected to the host. Reserved for extension in the future.

D.4 SPI parameters
spi: Must be specified if using SPI to communicate with MCU bootloader.

<port> and <pin>: Selects the GPIO[port][pin] on the LPC Link2 MCU to act as SPI.nCS. Default is port 0, pin 0.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is the GPIO on LPC Link2 MCU, but not the GPIO on the device running MCU bootloader.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPCUSBSIO only supports MSB for SPI data transmission.</td>
</tr>
</tbody>
</table>

<speed>: SPI baud rate/frequency(default 100 kpbs, if not specified)
<polarity>: SPI polarity, 0 for active high, 1 for active low (default)
<phase>: SPI phase, 0 for rising edge sampling, 1 for failing edge sampling (default)
D.5 I2C parameters

I2C: Must be specified if using I2C to communicate with MCU bootloader.

<address>: I2C slave address (default 0x10, if not specified).

<speed>: I2C baud rate/frequency (default 100kbps, if not specified).

D.6 How to check if LPC Link2 supports LPC USB Serial I/O

Not all LPC Link2 contains the LPC Serial I/O port. User can check using the Device Manager (Windows) or the $dmesg command (Linux, MAC).

Figure 5. Device manager

Three HID devices, one for debugger, one for CDC UART, and one for LPC Serial I/O port.
How to check if LPC Link2 supports LPC USB Serial I/O

Figure 6. Command prompt
Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltlVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetics, Layerscape, Magnivn, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Converge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.


All rights reserved.

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

Date of release: 29 July 2021
Document identifier: MCUBLHOSTUG