MCUX Flashloader Reference Manual

Rev. 2, 01/2018

Contents

Section number **Title** **Page**

Chapter 1
Introduction

Chapter 2
Functional description

Chapter 3
Kinetis bootloader protocol

Chapter 4
Bootloader packet types

Chapter 5

Kinetis bootloader command API

Chapter 6
Supported peripherals

Chapter 7
Peripheral interfaces

Chapter 8
External Memory Support

Chapter 9
Security Utilities

Chapter 10
Appendix A: status and error codes

Chapter 11
Appendix B: GetProperty and SetProperty commands

Chapter 12
Revision history

Chapter 1
Introduction

1.1 Introduction

The Kinetis bootloader is a configurable flash programming utility that operates over a serial connection on Kinetis MCUs. It enables quick and easy programming of Kinetis MCUs through the entire product life cycle, including application development, final product manufacturing, and beyond. The bootloader is delivered in two ways. The Kinetis bootloader is provided as full source code that is highly configurable. The bootloader is also preprogrammed by Freescale into ROM or flash on select Kinetis devices. Host-side command line and GUI tools are available to communicate with the bootloader. Users can utilize host tools to upload/download application code via the bootloader.

1.2 Terminology

target

The device running the bootloader firmware (aka the ROM).

host

The device sending commands to the target for execution.

source

The initiator of a communications sequence. For example, the sender of a command or data packet.

destination

Receiver of a command or data packet.

incoming

From host to target.

outgoing

From target to host.

1.3 Block diagram

This block diagram describes the overall structure of the Kinetis bootloader.

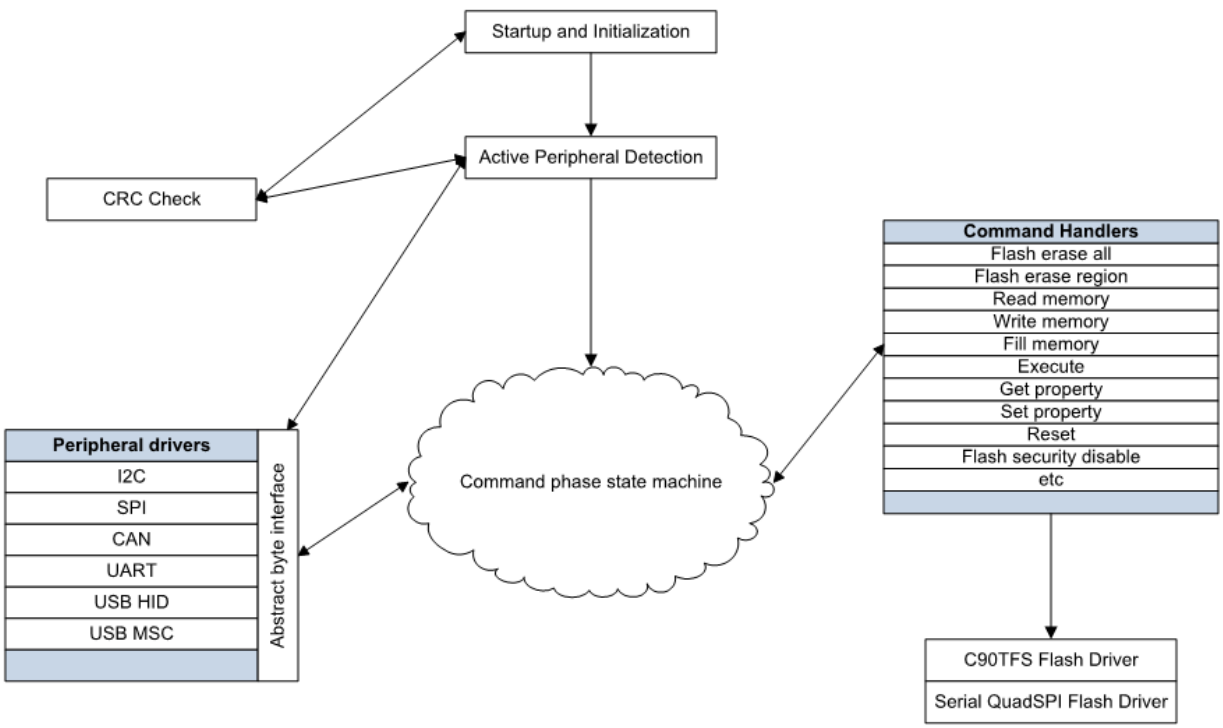

Figure 1-1. Block diagram

1.4 Features supported

Here are some of the features supported by the Kinetis bootloader:

- Supports UART, I2C, SPI, CAN, and USB peripheral interfaces.
- Automatic detection of the active peripheral.

- Ability to disable any peripheral.
- UART peripheral implements autobaud.
- Common packet-based protocol for all peripherals.
- Packet error detection and retransmit.
- Flash-resident configuration options.
- Fully supports flash security, including ability to mass erase or unlock security via the backdoor key.
- Protection of RAM used by the bootloader while it is running.
- Provides command to read properties of the device, such as Flash and RAM size.
- Multiple options for executing the bootloader either at system start-up or under application control at runtime.
- Support for internal flash, serial QuadSPI and other external memories.
- Support for encrypted image download.

1.5 Components supported

Components for the bootloader firmware:

- Startup code (clocking, pinmux, etc.)
- Command phase state machine
- Command handlers
    - GenericResponse
    - FlashEraseAll
    - FlashEraseRegion
    - ReadMemory
    - ReadMemoryResponse
    - WriteMemory
    - FillMemory
    - GetProperty
    - GetPropertyResponse
    - Execute
    - Call
    - Reset
    - SetProperty
    - FlashProgramOnce/EfuseProgramOnce
    - FlashReadOnce/EfuseReadOnce
    - FlashReadOnceResponse
    - ConfigureQuadSPI
    - ConfigureMemory
    - ReliableUpdate

- SB file state machine
  - Encrypted image support (AES-128)
- Packet interface
  - Framing packetizer
  - Command/data packet processor
- Memory interface
  - Abstract interface
  - FlexSPI NOR Memory Interface
  - FlexSPI NAND Memory Interface
  - SEMC NOR Memory Interface
  - SEMC NAND Memory Interface
  - SD Card Memory Interface
  - eMMC Memory Interface
- Peripheral drivers
  - UART
    - Auto-baud detector
  - USB device
    - USB controller driver
    - USB framework
    - USB HID class

Chapter 2
Functional description

2.1 Introduction

The following subsections describe the Kinetis bootloader functionality.

2.2 Memory map

See the Kinetis bootloader chapter of the reference manual of the particular SoC for the ROM and RAM memory map used by the bootloader.

2.3 Start-up process

It is important to note that the startup process for bootloader in ROM, RAM (flashloader), and flash (flash-resident) are slightly different. See the chip-specific reference manual for understanding the startup process for the ROM bootloader and flashloader.

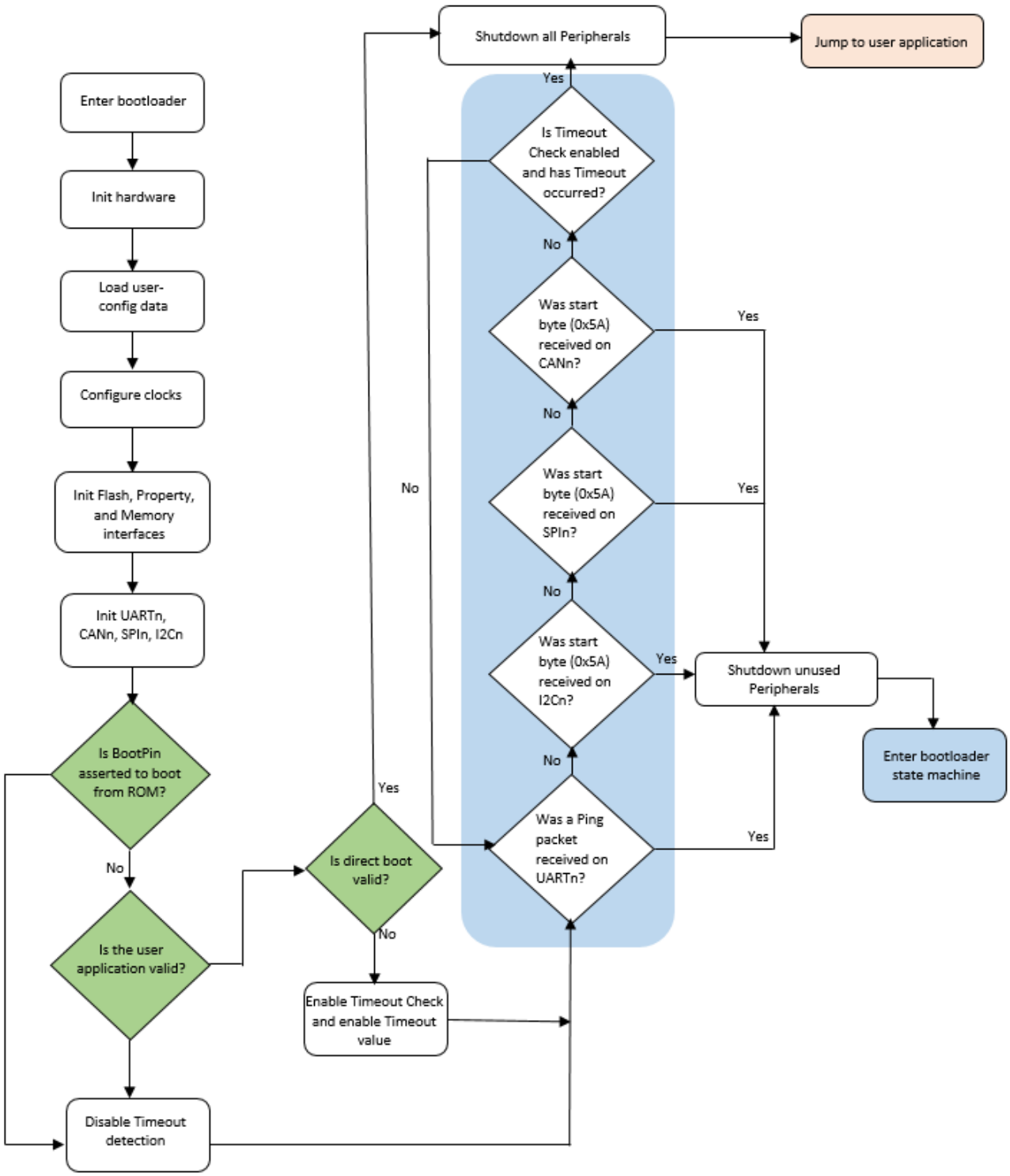

Figure 2-1. Kinetis bootloader start-up flowchart

Chapter 3
Kinetis bootloader protocol

3.1 Introduction

This section explains the general protocol for the packet transfers between the host and the Kinetis bootloader. The description includes the transfer of packets for different transactions, such as commands with no data phase and commands with incoming or outgoing data phase. The next section describes various packet types used in a transaction.

Each command sent from the host is replied to with a response command.

Commands may include an optional data phase.

- If the data phase is incoming (from the host to Kinetis bootloader), it is part of the original command.
- If the data phase is outgoing (from Kinetis bootloader to host), it is part of the response command.

3.2 Command with no data phase

NOTE

In these diagrams, the Ack sent in response to a Command or Data packet can arrive at any time before, during, or after the Command/Data packet has processed.

Command with no data phase

The protocol for a command with no data phase contains:

- Command packet (from host)
- Generic response command packet (to host)

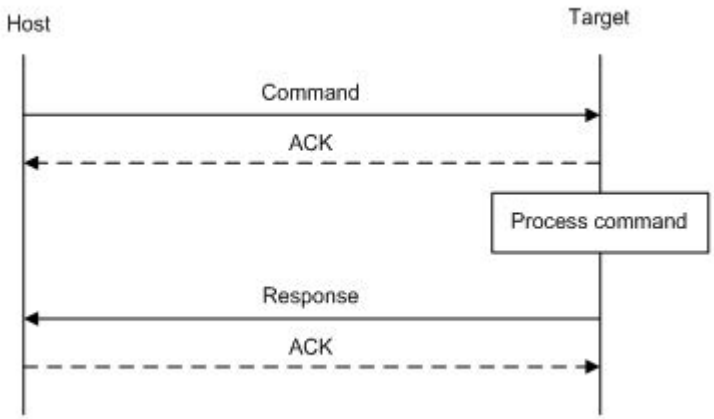

Figure 3-1. Command with no data phase

3.3 Command with incoming data phase

The protocol for a command with incoming data phase contains:

- Command packet (from host)(kCommandFlag_HasDataPhase set)
- Generic response command packet (to host)
- Incoming data packets (from host)
- Generic response command packet (to host)

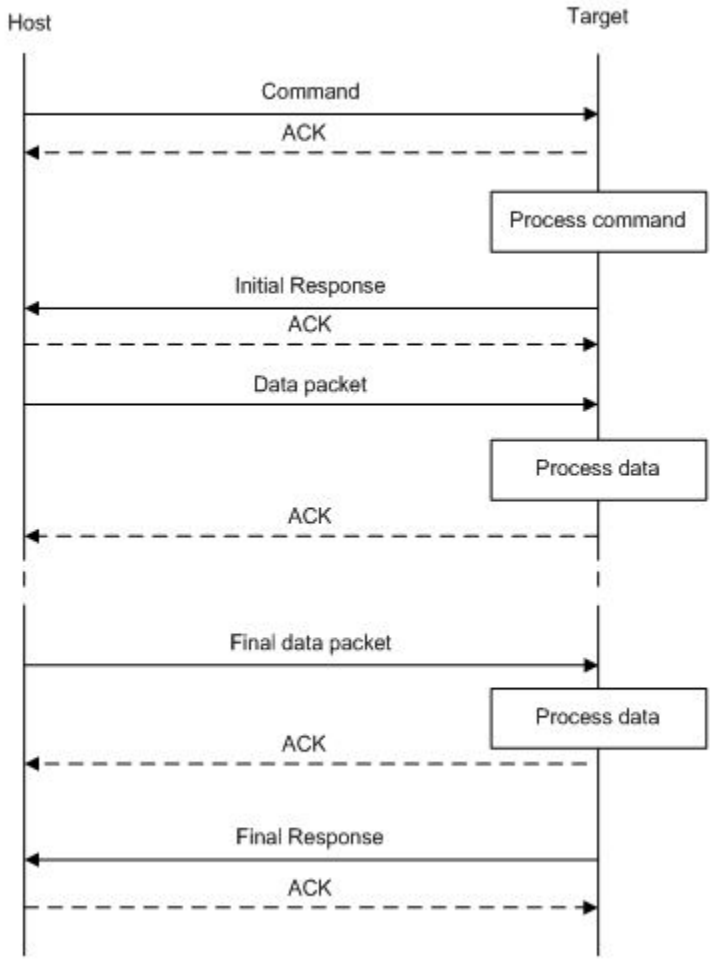

Figure 3-2. Command with incoming data phase

Notes

- The host may not send any further packets while it is waiting for the response to a command.
- The data phase is aborted if the Generic Response packet prior to the start of the data phase does not have a status of kStatus_Success.
- Data phases may be aborted by the receiving side by sending the final Generic Response early with a status of kStatus_AbortDataPhase. The host may abort the data phase early by sending a zero-length data packet.
- The final Generic Response packet sent after the data phase includes the status for the entire operation.

3.4 Command with outgoing data phase

The protocol for a command with an outgoing data phase contains:

- Command packet (from host)
- ReadMemory Response command packet (to host)(kCommandFlag_HasDataPhase set)
- Outgoing data packets (to host)
- Generic response command packet (to host)

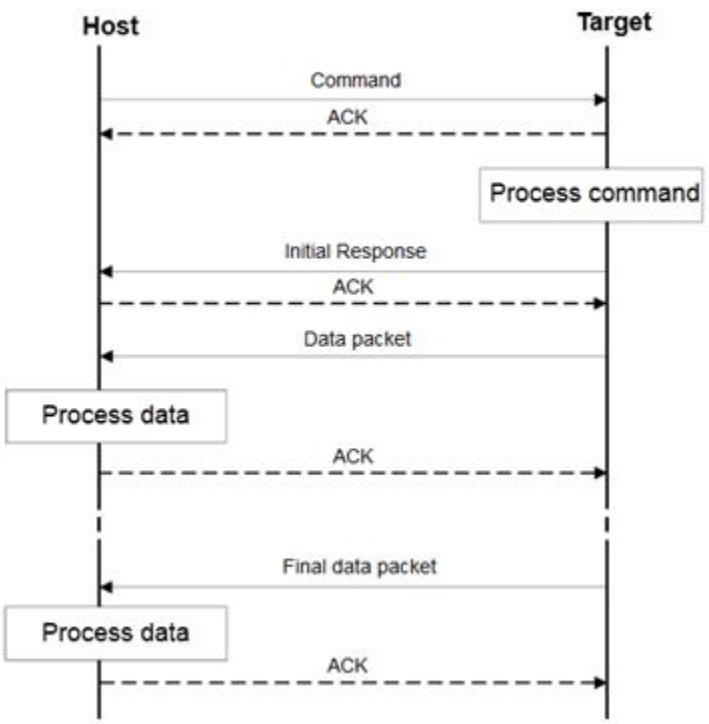

Figure 3-3. Command with outgoing data phase

Note

- The data phase is considered part of the response command for the outgoing data phase sequence.
- The host may not send any further packets while the host is waiting for the response to a command.
- The data phase is aborted if the ReadMemory Response command packet, prior to the start of the data phase, does not contain the kCommandFlag_HasDataPhase flag.

- Data phases may be aborted by the host sending the final Generic Response early with a status of kStatus_AbortDataPhase. The sending side may abort the data phase early by sending a zero-length data packet.
- The final Generic Response packet sent after the data phase includes the status for the entire operation.

Chapter 4
Bootloader packet types

4.1 Introduction

The Kinetis bootloader device works in slave mode. All data communication is initiated by a host, which is either a PC or an embedded host. The Kinetis bootloader device is the target, which receives a command or data packet. All data communication between host and target is packetized.

NOTE

The term "target" refers to the "Kinetis bootloader device".

There are 6 types of packets used:

- Ping packet
- Ping Response packet
- Framing packet
- Command packet
- Data packet
- Response packet

All fields in the packets are in little-endian byte order.

4.2 Ping packet

The Ping packet is the first packet sent from a host to the target to establish a connection on selected peripheral in order to run autobaud. The Ping packet can be sent from host to target at any time that the target is expecting a command packet. If the selected peripheral is UART, a ping packet must be sent before any other communications. For other serial peripherals it is optional, but is recommended in order to determine the serial protocol version.

In response to a Ping packet, the target sends a Ping Response packet, discussed in later sections.

Table 4-1. Ping Packet Format

Byte #	Value	Name
0	0x5A	start byte
1	0xA6	ping

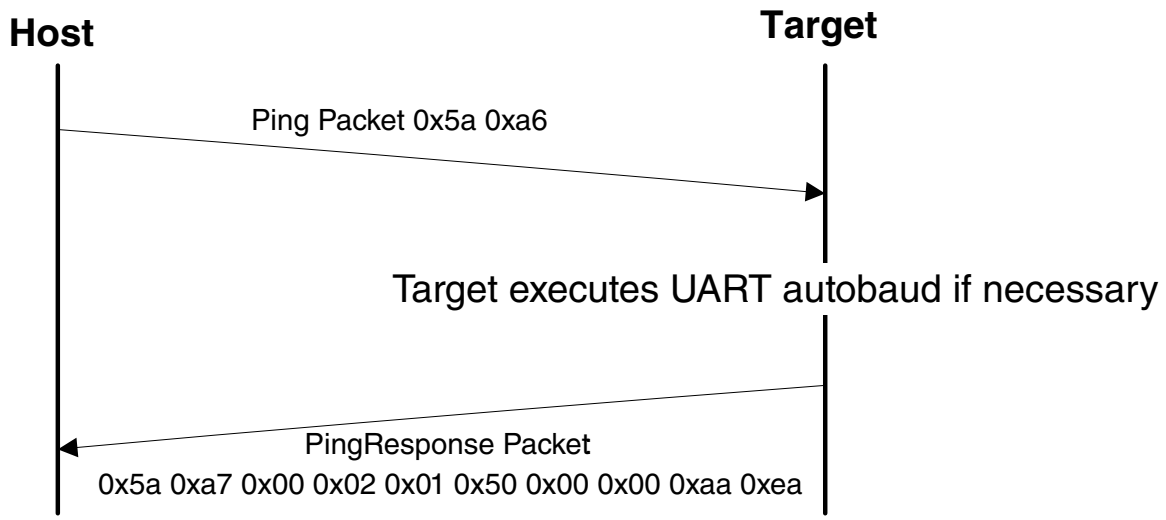

Figure 4-1. Ping Packet Protocol Sequence

4.3 Ping response packet

The target sends a Ping Response packet back to the host after receiving a Ping packet. If communication is over a UART peripheral, the target uses the incoming Ping packet to determine the baud rate before replying with the Ping Response packet. Once the Ping Response packet is received by the host, the connection is established, and the host starts sending commands to the target.

Table 4-2. Ping Response packet format

Byte #	Value	Parameter
0	0x5A	start byte
1	0xA7	Ping response code
2		Protocol bugfix
3		Protocol minor
4		Protocol major
5		Protocol name = 'P' (0x50)
6		Options low
7		Options high

Table continues on the next page...

Table 4-2. Ping Response packet format (continued)

Byte #	Value	Parameter
8		CRC16 low
9		CRC16 high

The Ping Response packet can be sent from host to target any time the target expects a command packet. For the UART peripheral, it must be sent by host when a connection is first established, in order to run autobaud. For other serial peripherals it is optional, but recommended to determine the serial protocol version. The version number is in the same format at the bootloader version number returned by the GetProperty command.

4.4 Framing packet

The framing packet is used for flow control and error detection for the communications links that do not have such features built-in. The framing packet structure sits between the link layer and command layer. It wraps command and data packets as well.

Every framing packet containing data sent in one direction results in a synchronizing response framing packet in the opposite direction.

The framing packet described in this section is used for serial peripherals including the UART, I2C, and SPI. The USB HID peripheral does not use framing packets. Instead, the packetization inherent in the USB protocol itself is used.

Table 4-3. Framing Packet Format

Byte #	Value	Parameter	
0	0x5A	start byte	
1		packetType	
2		length_low	Length is a 16-bit field that specifies the entire command or data packet size in bytes.
3		length_high	
4		crc16_low	This is a 16-bit field. The CRC16 value covers entire framing packet, including the start byte and command or data packets, but does not include the CRC bytes. See the CRC16 algorithm after this table.
5		crc16_high	
6 . . .n		Command or Data packet payload	

A special framing packet that contains only a start byte and a packet type is used for synchronization between the host and target.

Table 4-4. Special Framing Packet Format

Byte #	Value	Parameter
0	0x5A	start byte
1	0xA*n*	packetType

The Packet Type field specifies the type of the packet from one of the defined types (below):

Table 4-5. packetType Field

packetType	Name	Description
0xA1	kFramingPacketType_Ack	The previous packet was received successfully; the sending of more packets is allowed.
0xA2	kFramingPacketType_Nak	The previous packet was corrupted and must be re-sent.
0xA3	kFramingPacketType_AckAbort	Data phase is being aborted.
0xA4	kFramingPacketType_Command	The framing packet contains a command packet payload.
0xA5	kFramingPacketType_Data	The framing packet contains a data packet payload.
0xA6	kFramingPacketType_Ping	Sent to verify the other side is alive. Also used for UART autobaud.
0xA7	kFramingPacketType_PingResponse	A response to Ping; contains the framing protocol version number and options.

4.5 CRC16 algorithm

This section provides the CRC16 algorithm.

The CRC is computed over each byte in the framing packet header, excluding the crc16 field itself, plus all of the payload bytes. The CRC algorithm is the XMODEM variant of CRC-16.

The characteristics of the XMODEM variant are:

width	16
polynomial	0x1021
init value	0x0000
reflect in	false
reflect out	false
xor out	0x0000
check result	0x31c3

The check result is computed by running the ASCII character sequence "123456789" through the algorithm.

```
uint16_t crc16_update(const uint8_t * src, uint32_t lengthInBytes
      {
          uint32_t crc = 0;
          uint32_t j;
          for (j=0; j < lengthInBytes; ++j)
         {
              uint32_t i;
              uint32_t byte = src[j];
             crc ^= byte << 8;
              for (i = 0; i < 8; ++i)
              {
                 uint32_t temp = crc << 1;
                  if (crc & 0x8000)
                  {
                      temp ^= 0x1021;
                }
                  crc = temp;
              }
          }
          return crc;
      }
```

4.6 Command packet

The command packet carries a 32-bit command header and a list of 32-bit parameters.

Table 4-6. Command Packet Format

Command Packet Format (32 bytes)										
Command Header (4 bytes)				28 bytes for Parameters (Max 7 parameters)						
Tag	Flags	Rsvd	Param Count	Param1 (32-bit)	Param2 (32-bit)	Param3 (32-bit)	Param4 (32-bit)	Param5 (32-bit)	Param6 (32-bit)	Param7 (32-bit)
byte 0	byte 1	byte 2	byte 3	-	-	-	-	-	-	-

Table 4-7. Command Header Format

Byte #	Command Header Field	
0	Command or Response tag	The command header is 4 bytes long, with these fields.
1	Flags	
2	Reserved. Should be 0x00.	
3	ParameterCount	

The header is followed by 32-bit parameters up to the value of the ParameterCount field specified in the header. Because a command packet is 32 bytes long, only 7 parameters can fit into the command packet.

Command packets are also used by the target to send responses back to the host. As mentioned earlier, command packets and data packets are embedded into framing packets for all of the transfers.

Table 4-8. Command Tags

Command Tag	Name	
0x01	FlashEraseAll	The command tag specifies one of the commands supported by the Kinetis bootloader. The valid command tags for the Kinetis bootloader are listed here.
0x02	FlashEraseRegion	
0x03	ReadMemory	
0x04	WriteMemory	
0x05	FillMemory	
0x06	FlashSecurityDisable	
0x07	GetProperty	
0x08	Reserved	
0x09	Execute	
0x10	FlashReadResource	
0x11	Reserved	
0x0A	Call	
0x0B	Reset	
0x0C	SetProperty	
0x0D	FlashEraseAllUnsecure	
0x0E	eFuseProgram	
0x0F	eFuseRead	
0x10	FlashReadResource	
0x11	ConfigureMemory	
0x12	ReliableUpdate	

Table 4-9. Response Tags

Response Tag	Name	
0xA0	GenericResponse	The response tag specifies one of the responses the Kinetis bootloader (target) returns to the host. The valid response tags are listed here.
0xA7	GetPropertyResponse (used for sending responses to GetProperty command only)	
0xA3	ReadMemoryResponse (used for sending responses to ReadMemory command only)	
0xAF	FlashReadOnceResponse (used for sending responses to FlashReadOnce command only)	
0xB0	FlashReadResourceResponse (used for sending responses to FlashReadResource command only)	

Flags: Each command packet contains a Flag byte. Only bit 0 of the flag byte is used. If bit 0 of the flag byte is set to 1, then data packets follow in the command sequence. The number of bytes that are transferred in the data phase is determined by a command-specific parameter in the parameters array.

ParameterCount: The number of parameters included in the command packet.

Parameters: The parameters are word-length (32 bits). With the default maximum packet size of 32 bytes, a command packet can contain up to 7 parameters.

4.7 Response packet

The responses are carried using the same command packet format wrapped with framing packet data. Types of responses include:

- GenericResponse
- GetPropertyResponse
- ReadMemoryResponse
- FlashReadOnceResponse
- FlashReadResourceResponse

GenericResponse: After the Kinetis bootloader has processed a command, the bootloader sends a generic response with status and command tag information to the host. The generic response is the last packet in the command protocol sequence. The generic response packet contains the framing packet data and the command packet data (with generic response tag = 0xA0) and a list of parameters (defined in the next section). The parameter count field in the header is always set to 2, for status code and command tag parameters.

Table 4-10. GenericResponse Parameters

Byte #	Parameter	Descripton
0 - 3	Status code	The Status codes are errors encountered during the execution of a command by the target. If a command succeeds, then a kStatus_Success code is returned.
4 - 7	Command tag	The Command tag parameter identifies the response to the command sent by the host.

GetPropertyResponse: The GetPropertyResponse packet is sent by the target in response to the host query that uses the GetProperty command. The GetPropertyResponse packet contains the framing packet data and the command packet data, with the command/response tag set to a GetPropertyResponse tag value (0xA7).

The parameter count field in the header is set to greater than 1, to always include the status code and one or many property values.

Table 4-11. GetPropertyResponse Parameters

Byte #	Value	Parameter
0 - 3		Status code
4 - 7		Property value
. . .		. . .
		Can be up to maximum 6 property values, limited to the size of the 32-bit command packet and property type.

ReadMemoryResponse: The ReadMemoryResponse packet is sent by the target in response to the host sending a ReadMemory command. The ReadMemoryResponse packet contains the framing packet data and the command packet data, with the command/response tag set to a ReadMemoryResponse tag value (0xA3), the flags field set to kCommandFlag_HasDataPhase (1).

The parameter count set to 2 for the status code and the data byte count parameters shown below.

Table 4-12. ReadMemoryResponse Parameters

Byte #	Parameter	Descripton
0 - 3	Status code	The status of the associated Read Memory command.
4 - 7	Data byte count	The number of bytes sent in the data phase.

FlashReadOnceResponse: The FlashReadOnceResponse packet is sent by the target in response to the host sending a FlashReadOnce command. The FlashReadOnceResponse packet contains the framing packet data and the command packet data, with the command/response tag set to a FlashReadOnceResponse tag value (0xAF), and the flags field set to 0. The parameter count is set to 2 plus *the number of words* requested to be read in the FlashReadOnceCommand.

Table 4-13. FlashReadOnceResponse Parameters

Byte #	Value	Parameter
0 – 3		Status Code
4 – 7		Byte count to read
…		…
		Can be up to 20 bytes of requested read data.

The FlashReadResourceResponse packet is sent by the target in response to the host sending a FlashReadResource command. The FlashReadResourceResponse packet contains the framing packet data and command packet data, with the command/response tag set to a FlashReadResourceResponse tag value (0xB0), and the flags field set to kCommandFlag_HasDataPhase (1).

Table 4-14. FlashReadResourceResponse Parameters

Byte #	Value	Parameter
0 – 3		Status Code
4 – 7		Data byte count

Chapter 5
Kinetis bootloader command API

5.1 Introduction

All Kinetis bootloader command APIs follows the command packet format wrapped by the framing packet as explained in previous sections.

See Table 4-8 for a list of commands supported by Kinetis bootloader.

For a list of status codes returned by Kinetis bootloader see Appendix A.

5.2 GetProperty command

The GetProperty command is used to query the bootloader about various properties and settings. Each supported property has a unique 32-bit tag associated with it. The tag occupies the first parameter of the command packet. The target returns a GetPropertyResponse packet with the property values for the property identified with the tag in the GetProperty command.

Properties are the defined units of data that can be accessed with the GetProperty or SetProperty commands. Properties may be read-only or read-write. All read-write properties are 32-bit integers, so they can easily be carried in a command parameter.

For a list of properties and their associated 32-bit property tags supported by Kinetis bootloader, see Appendix B.

The 32-bit property tag is the only parameter required for GetProperty command.

Table 5-1. Parameters for GetProperty Command

Byte #	Command
0 - 3	Property tag
4 - 7	External Memory Identifier (only applies to get property for external memory)

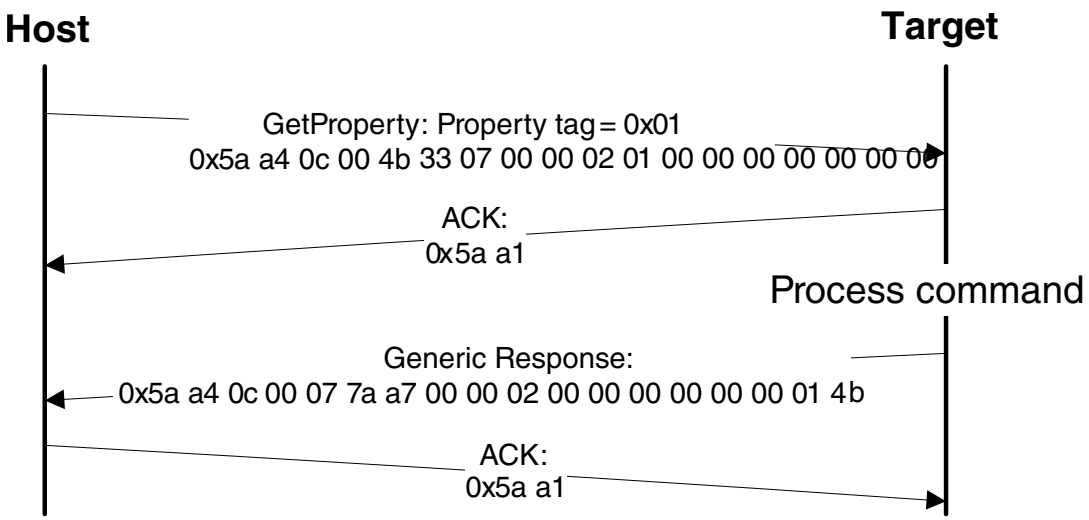

Figure 5-1. Protocol Sequence for GetProperty Command

Table 5-2. GetProperty Command Packet Format (Example)

GetProperty	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x0C 0x00
	crc16	0x4B 0x33
Command packet	commandTag	0x07 – GetProperty
	flags	0x00
	reserved	0x00
	parameterCount	0x02
	propertyTag	0x00000001 - CurrentVersion
	Memory ID	0x00000000 - Internal Flash (0x00000001 - QSPI0 Memory)

The GetProperty command has no data phase.

Response: In response to a GetProperty command, the target sends a GetPropertyResponse packet with the response tag set to 0xA7. The parameter count indicates the number of parameters sent for the property values, with the first parameter showing status code 0, followed by the property value(s). The next table shows an example of a GetPropertyResponse packet.

Table 5-3. GetProperty Response Packet Format (Example)

GetPropertyResponse	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command

Table continues on the next page...

Table 5-3. GetProperty Response Packet Format (Example) (continued)

GetPropertyResponse	Parameter	Value
	length	0x0c 0x00 (12 bytes)
	crc16	0x07 0x7a
Command packet	responseTag	0xA7
	flags	0x00
	reserved	0x00
	parameterCount	0x02
	status	0x00000000
	propertyValue	0x0000014b - CurrentVersion

5.3 SetProperty command

The SetProperty command is used to change or alter the values of the properties or options of the bootloader. The command accepts the same property tags used with the GetProperty command. However, only some properties are writable--see Appendix B. 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.

The property tag and the new value to set are the two parameters required for the SetProperty command.

Table 5-4. Parameters for SetProperty Command

Byte #	Command
0 - 3	Property tag
4 - 7	Property value

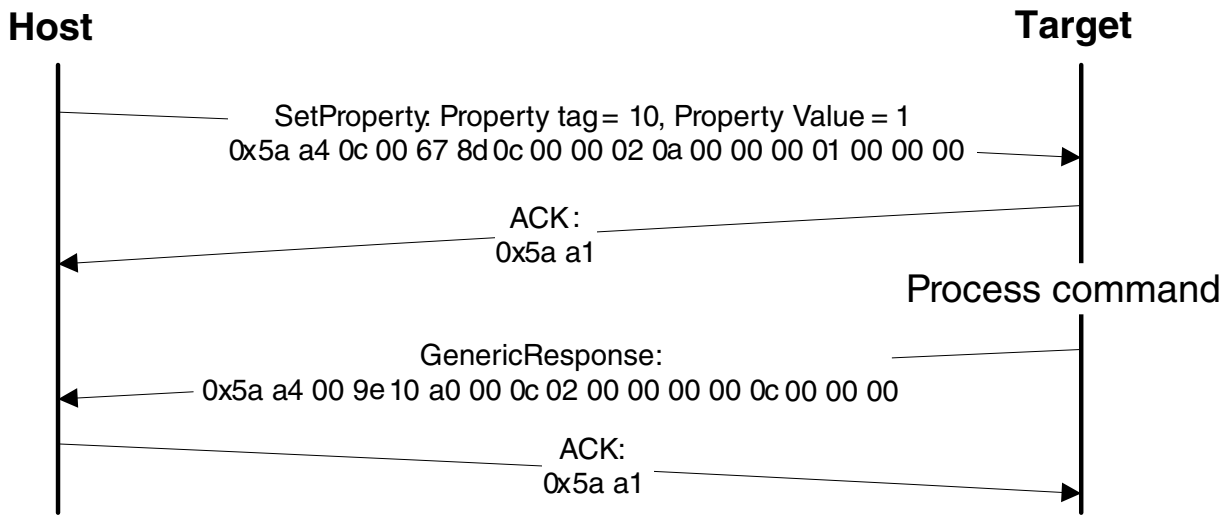

Figure 5-2. Protocol Sequence for SetProperty Command

Table 5-5. SetProperty Command Packet Format (Example)

SetProperty	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x0C 0x00
	crc16	0x67 0x8D
Command packet	commandTag	0x0C – SetProperty with property tag 10
	flags	0x00
	reserved	0x00
	parameterCount	0x02
	propertyTag	0x0000000A - VerifyWrites
	propertyValue	0x00000001

The SetProperty command has no data phase.

Response: The target returns a GenericResponse packet with one of following status codes:

Table 5-6. SetProperty Response Status Codes

Status Code
kStatus_Success
kStatus_ReadOnly
kStatus_UnknownProperty
kStatus_InvalidArgument

5.4 FlashEraseAll command

The FlashEraseAll command performs an erase of the entire flash memory. If any flash regions are protected, then the FlashEraseAll command fails and returns an error status code. Executing the FlashEraseAll command releases flash security if it (flash security) was enabled, by setting the FTFA_FSEC register. However, the FSEC field of the flash configuration field is erased, so unless it is reprogrammed, the flash security is re-enabled after the next system reset. The Command tag for FlashEraseAll command is 0x01 set in the commandTag field of the command packet.

The FlashEraseAll command requires memory ID. If memory ID is not specified, the internal flash (memory ID =0) will be selected as default.

Table 5-7. Parameter for FlashEraseAll Command

Byte #	Parameter	
0-3	Memory ID	
	0x000	Internal Flash
	0x010	Execute-only region in Internal Flash
	0x001	Serial NOR through QuadSPI
	0x008	Parallel NOR through SEMC
	0x009	Serial NOR through FlexSPI
	0x100	SLC Raw NAND through SEMC
	0x101	Serial NAND through FlexSPI
	0x110	Serial NOR/EEPROM through SPI
	0x120	SD through uSDHC
	0x121	eMMC through uSDHC

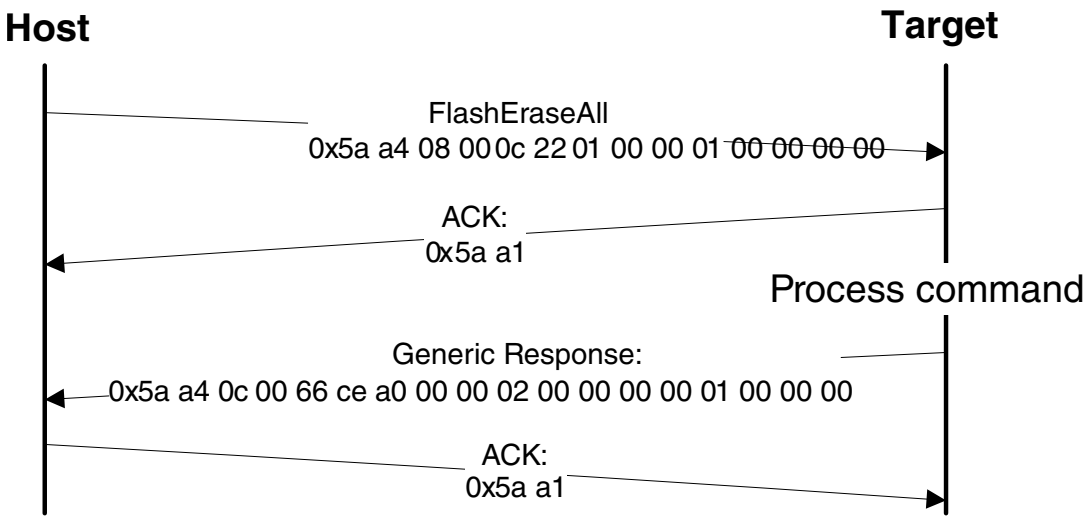

Figure 5-3. Protocol Sequence for FlashEraseAll Command

Table 5-8. FlashEraseAll Command Packet Format (Example)

FlashEraseAll	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x08 0x00
	crc16	0x0C 0x22
Command packet	commandTag	0x01 - FlashEraseAll
	flags	0x00
	reserved	0x00
	parameterCount	0x01
	Memory ID	refer to the above table

The FlashEraseAll command has no data phase.

Response: The target returns a GenericResponse packet with status code either set to kStatus_Success for successful execution of the command, or set to an appropriate error status code.

5.5 FlashEraseRegion command

The FlashEraseRegion command performs an erase of one or more sectors of the flash memory.

The start address and number of bytes are the 2 parameters required for the FlashEraseRegion command. The start and byte count parameters must be 4-byte aligned ([1:0] = 00), or the FlashEraseRegion command fails and returns kStatus_FlashAlignmentError(101). If the region specified does not fit in the flash memory space, the FlashEraseRegion command fails and returns kStatus_FlashAddressError(102). If any part of the region specified is protected, the FlashEraseRegion command fails and returns kStatus_MemoryRangeInvalid(10200).

Table 5-9. Parameters for FlashEraseRegion Command

Byte #	Parameter
0 - 3	Start address
4 - 7	Byte count
8 - 11	Memory ID

The FlashEraseRegion command has no data phase.

Response: The target returns a GenericResponse packet with one of following error status codes.

Table 5-10. FlashEraseRegion Response Status Codes

Status Code
kStatus_Success (0)
kStatus_MemoryRangeInvalid (10200)
kStatus_FlashAlignmentError (101)
kStatus_FlashAddressError (102)
kStatus_FlashAccessError (103)
kStatus_FlashProtectionViolation (104)
kStatus_FlashCommandFailure (105)

5.6 FlashEraseAllUnsecure command

The FlashEraseAllUnsecure command performs a mass erase of the flash memory, including protected sectors. Flash security is immediately disabled if it (flash security) was enabled, and the FSEC byte in the flash configuration field at address 0x40C is programmed to 0xFE. However, if the mass erase enable option in the FSEC field is disabled, then the FlashEraseAllUnsecure command fails.

The FlashEraseAllUnsecure command requires no parameters.

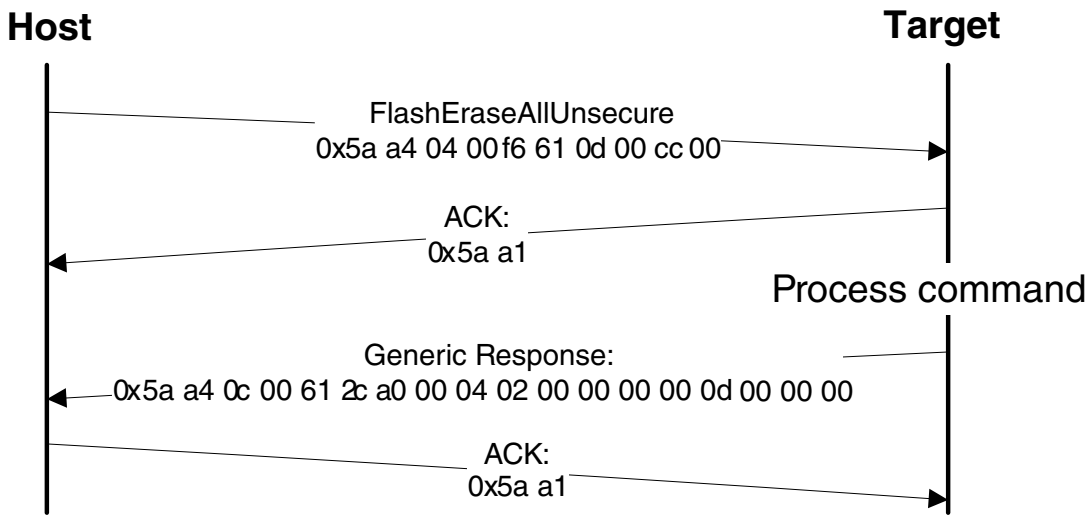

Figure 5-4. Protocol Sequence for FlashEraseAll Command

Table 5-11. FlashEraseAllUnsecure Command Packet Format (Example)

FlashEraseAllUnsecure	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x04 0x00
	crc16	0xF6 0x61
Command packet	commandTag	0x0D - FlashEraseAllUnsecure
	flags	0x00
	reserved	0x00
	parameterCount	0x00

The FlashEraseAllUnsecure command has no data phase.

Response: The target returns a GenericResponse packet with status code either set to kStatus_Success for successful execution of the command, or set to an appropriate error status code.

NOTE

When the MEEN bit in the NVM FSEC register is cleared to disable the mass erase, the FlashEraseAllUnsecure command will fail. FlashEraseRegion can be used instead skipping the protected regions.

5.7 ReadMemory command

The ReadMemory command 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.

The start address and number of bytes are the two parameters required for ReadMemory command. The memory ID is optional. Internal memory will be selected as default if memory ID is not specified.

Table 5-12. Parameters for read memory command

Byte	Parameter	Description
0-3	Start address	Start address of memory to read from
4-7	Byte count	Number of bytes to read and return to caller
8-11	Memory ID	Internal or extermal memory Identifier

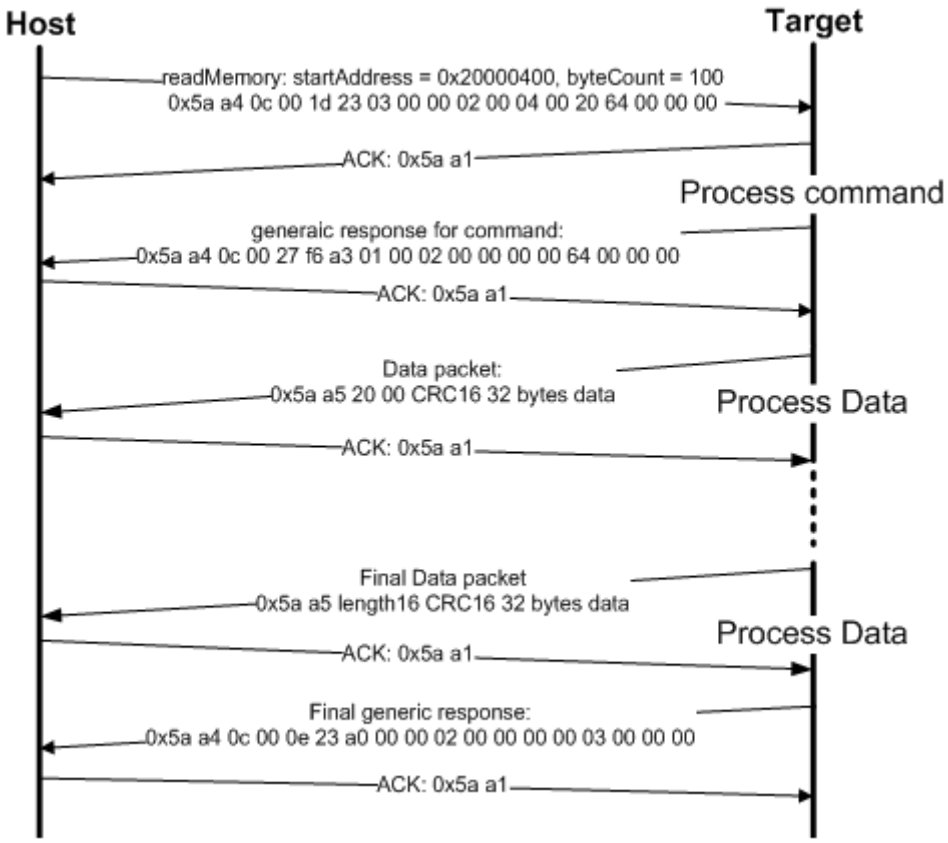

Figure 5-5. Command sequence for read memory

ReadMemory	Parameter	Value
Framing packet	Start byte	0x5A0xA4,
	packetType	kFramingPacketType_Command
	length	0x10 0x00
	crc16	0xF4 0x1B
Command packet	commandTag	0x03 - readMemory
	flags	0x00
	reserved	0x00
	parameterCount	0x03
	startAddress	0x20000400
	byteCount	0x00000064
	memoryID	0x0

Data Phase: The ReadMemory command has a data phase. Because the target works in slave mode, the host needs to pull data packets until the number of bytes of data specified in the byteCount parameter of ReadMemory command are received by host.

Response: The target returns a GenericResponse packet with a status code either set to kStatus_Success upon successful execution of the command, or set to an appropriate error status code.

5.8 WriteMemory command

The WriteMemory command writes data provided in the data phase to a specified range of bytes in memory (flash or RAM). However, if flash protection is enabled, then writes to protected sectors fail.

Special care must be taken when writing to flash.

- First, any flash sector written to must have been previously erased with a FlashEraseAll, FlashEraseRegion, or FlashEraseAllUnsecure command.
- First, any flash sector written to must have been previously erased with a FlashEraseAll or FlashEraseRegion command.
- Writing to flash requires the start address to be 4-byte aligned ([1:0] = 00).
- The byte count is rounded up to a multiple of 4, and trailing bytes are filled with the flash erase pattern (0xff).
- If the VerifyWrites property is set to true, then writes to flash also performs a flash verify program operation.

When writing to RAM, the start address does not need to be aligned, and the data is not padded.

The start address and number of bytes are the 2 parameters required for WriteMemory command. The memory ID is optional. Internal memory will be selected as default if memory ID is not specified.

Table 5-13. Parameters for WriteMemory Command

Byte #	Command
0 - 3	Start address
4 - 7	Byte count
8 - 11	Memory ID

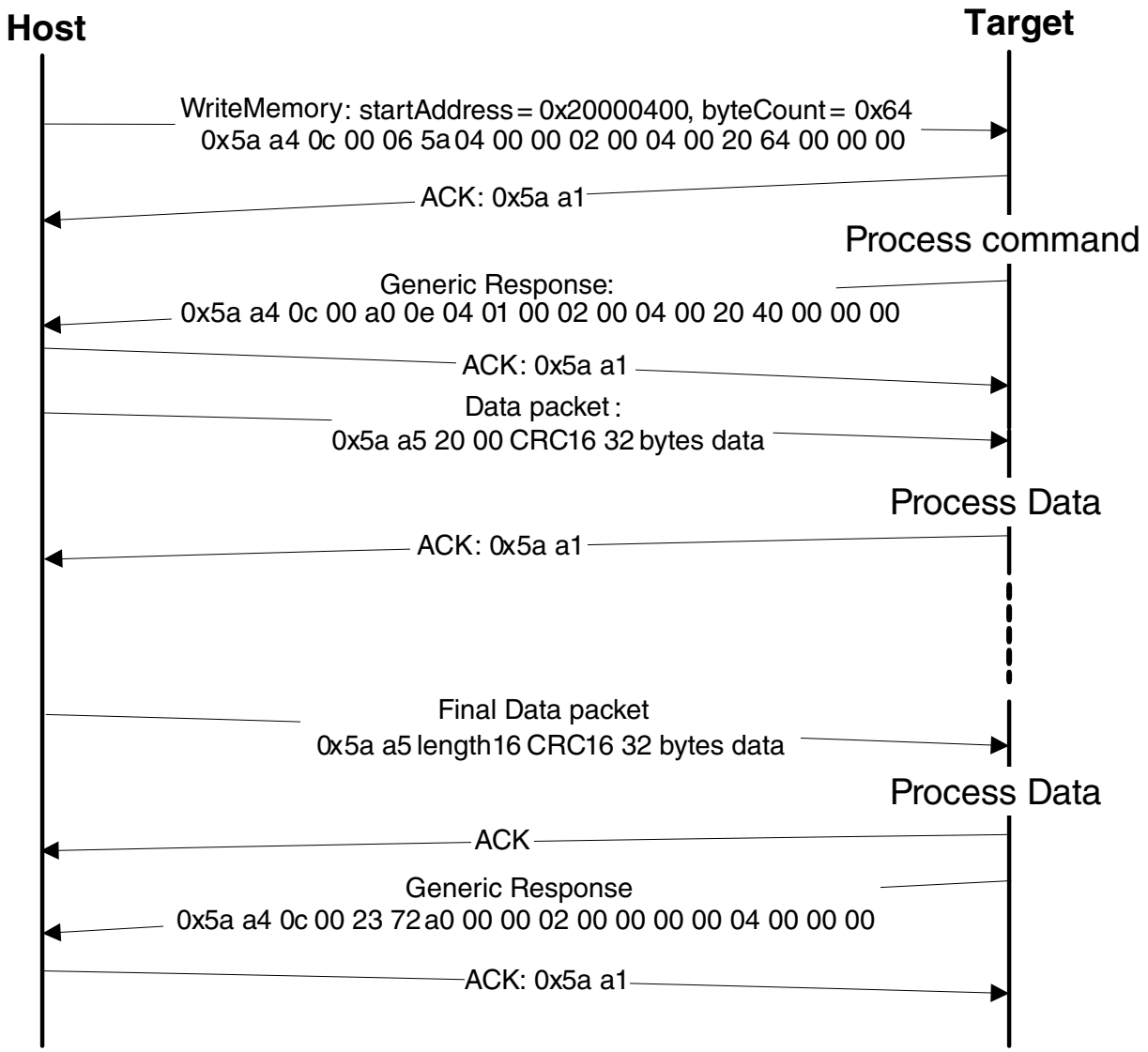

Figure 5-6. Protocol Sequence for WriteMemory Command

Table 5-14. WriteMemory Command Packet Format (Example)

WriteMemory	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x10 0x00
	crc16	0x97 0xDD
Command packet	commandTag	0x04 - writeMemory
	flags	0x01
	reserved	0x00
	parameterCount	0x03
	startAddress	0x20000400

Table continues on the next page...

Table 5-14. WriteMemory Command Packet Format (Example) (continued)

WriteMemory	Parameter	Value
	byteCount	0x00000064
	memoryID	0x0

Data Phase: The WriteMemory command has a data phase; the host sends data packets until the number of bytes of data specified in the byteCount parameter of the WriteMemory command are received by the target.

Response: The target returns a GenericResponse packet with a status code set to kStatus_Success upon successful execution of the command, or to an appropriate error status code.

5.9 FillMemory command

The FillMemory command fills a range of bytes in memory with a data pattern. It follows the same rules as the WriteMemory command. The difference between FillMemory and WriteMemory is that a data pattern is included in FillMemory command parameter, and there is no data phase for the FillMemory command, while WriteMemory does have a data phase.

Table 5-15. Parameters for FillMemory Command

Byte #	Command
0 - 3	Start address of memory to fill
4 - 7	Number of bytes to write with the pattern • The start address should be 32-bit aligned. • The number of bytes must be evenly divisible by 4. (Note: for a part that uses FTFE flash, the start address should be 64-bit aligned, and the number of bytes must be evenly divisible by 8).
8 - 11	32-bit pattern

- To fill with a byte pattern (8-bit), the byte must be replicated 4 times in the 32-bit pattern.
- To fill with a short pattern (16-bit), the short value must be replicated 2 times in the 32-bit pattern.

For example, to fill a byte value with 0xFE, the word pattern is 0xFEFEFEFE; to fill a short value 0x5AFE, the word pattern is 0x5AFE5AFE.

Special care must be taken when writing to flash.

- First, any flash sector written to must have been previously erased with a FlashEraseAll, FlashEraseRegion, or FlashEraseAllUnsecure command.
- First, any flash sector written to must have been previously erased with a FlashEraseAll or FlashEraseRegion command.
- Writing to flash requires the start address to be 4-byte aligned ([1:0] = 00).
- If the VerifyWrites property is set to true, then writes to flash also performs a flash verify program operation.

When writing to RAM, the start address does not need to be aligned, and the data is not padded.

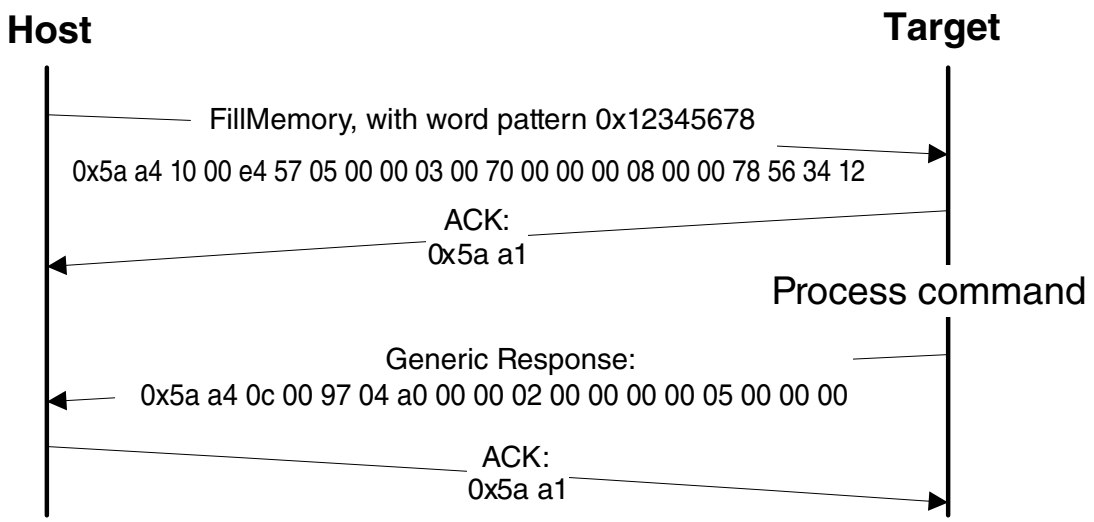

Figure 5-7. Protocol Sequence for FillMemory Command

Table 5-16. FillMemory Command Packet Format (Example)

FillMemory	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x10 0x00
	crc16	0xE4 0x57
Command packet	commandTag	0x05 – FillMemory
	flags	0x00
	Reserved	0x00
	parameterCount	0x03
	startAddress	0x00007000
	byteCount	0x00000800
	patternWord	0x12345678

The FillMemory command has no data phase.

Response: upon successful execution of the command, the target (Kinetis bootloader) returns a GenericResponse packet with a status code set to kStatus_Success, or to an appropriate error status code.

5.10 Execute command

The execute command results in the bootloader setting the program counter to the code at the provided jump address, R0 to the provided argument, and a Stack pointer to the provided stack pointer address. Prior to the jump, the system is returned to the reset state.

The Jump address, function argument pointer, and stack pointer are the parameters required for the Execute command. If the stack pointer is set to zero, the called code is responsible for setting the processor stack pointer before using the stack.

If QSPI is enabled, it is initialized before the jump. QSPI encryption (OTFAD) is also enabled if configured.

Table 5-17. Parameters for Execute Command

Byte #	Command
0 - 3	Jump address
4 - 7	Argument word
8 - 11	Stack pointer address

The Execute command has no data phase.

Response: Before executing the Execute command, the target validates the parameters and return a GenericResponse packet with a status code either set to kStatus_Success or an appropriate error status code.

5.11 Call command

The Call command executes a function that is written in memory at the address sent in the command. The address needs to be a valid memory location residing in accessible flash (internal or external) or in RAM. The command supports the passing of one 32-bit argument. Although the command supports a stack address, at this time the call still takes place using the current stack pointer. After execution of the function, a 32-bit return value is returned in the generic response message.

QSPI must be initialized prior to executing the Call command if the call address is on QSPI. The Call command does not initialize QSPI.

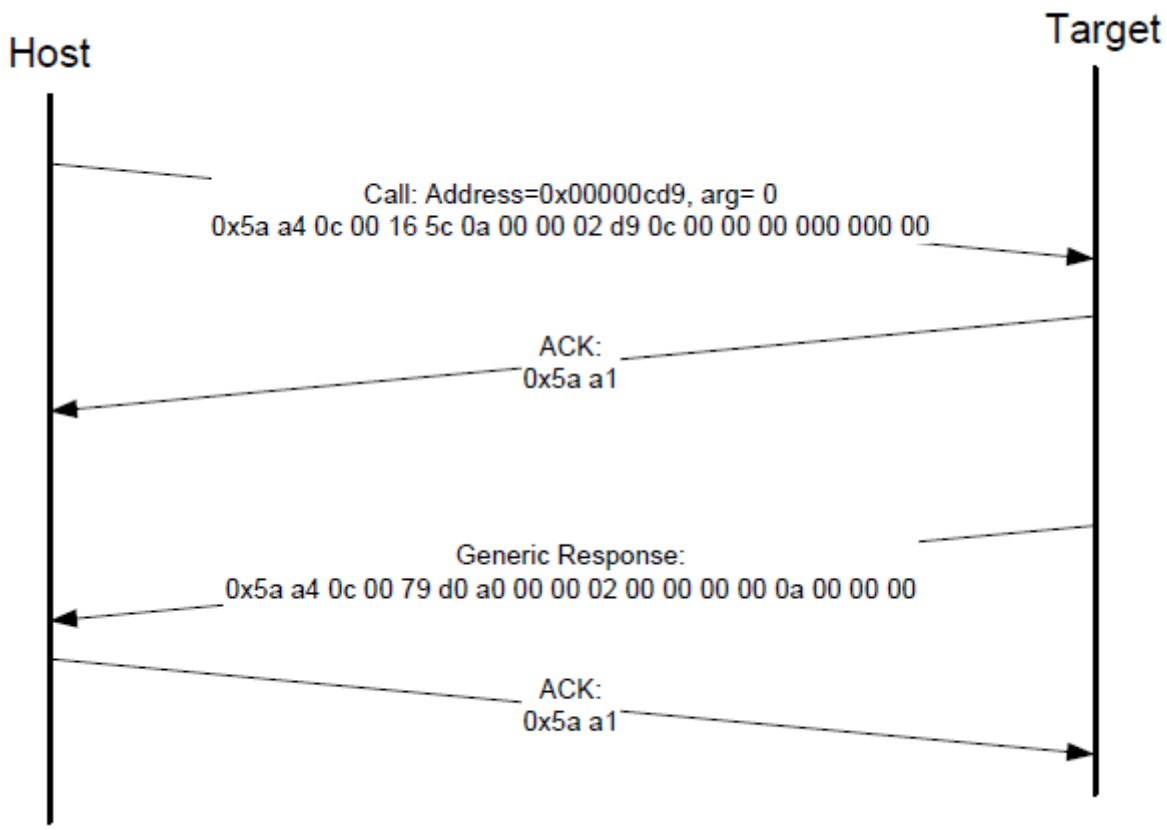

Figure 5-8. Protocol sequence for call command

Table 5-18. Parameters for Call Command

Byte #	Command
0 - 3	Call address
4 - 7	Argument word
8 - 11	Stack pointer

Response: The target returns a GenericResponse packet with a status code either set to the return value of the function called or set to kStatus_InvalidArgument (105).

5.12 Reset command

The Reset command results in the bootloader resetting the chip.

The Reset command requires no parameters.

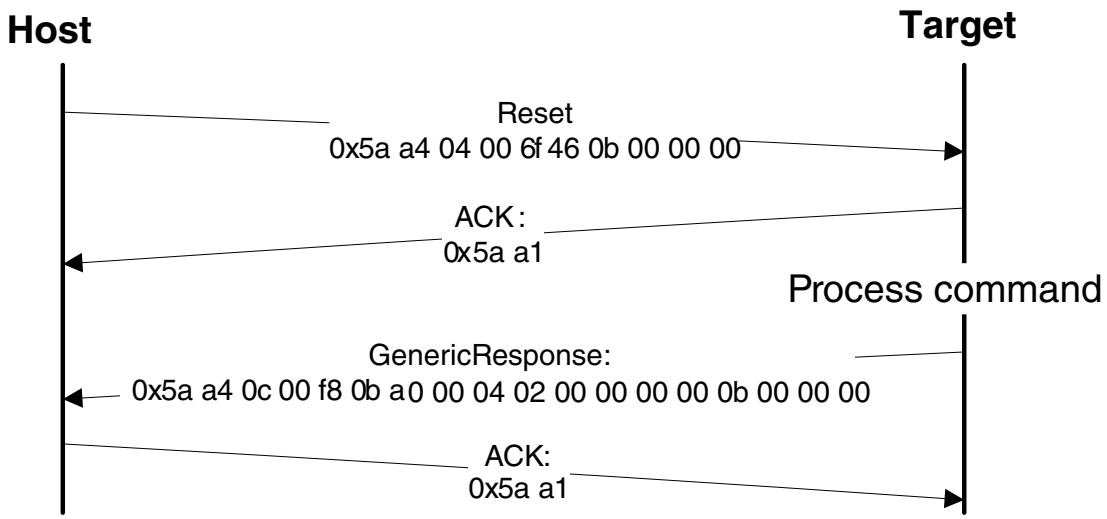

Figure 5-9. Protocol Sequence for Reset Command

Table 5-19. Reset Command Packet Format (Example)

Reset	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x04 0x00
	crc16	0x6F 0x46
Command packet	commandTag	0x0B - reset
	flags	0x00
	reserved	0x00
	parameterCount	0x00

The Reset command has no data phase.

Response: The target returns a GenericResponse packet with status code set to kStatus_Success, before resetting the chip.

The reset command can also be 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.

5.13 eFuseProgramOnce command

The FlashProgramOnce command writes data (that is provided in a command packet) to a specified range of bytes in the program once field. Special care must be taken when writing to the program once field.

- The program once field only supports programming once, so any attempted to reprogram a program once field gets an error response.
- Writing to the program once field requires the byte count to be 4-byte aligned or 8-byte aligned.

The FlashProgramOnce command uses three parameters: index 2, byteCount, data.

Table 5-20. Parameters for FlashProgramOnce Command

Byte #	Command
0 - 3	Index of program once field
4 - 7	Byte count (must be evenly divisible by 4)
8 - 11	Data
12 - 16	Data

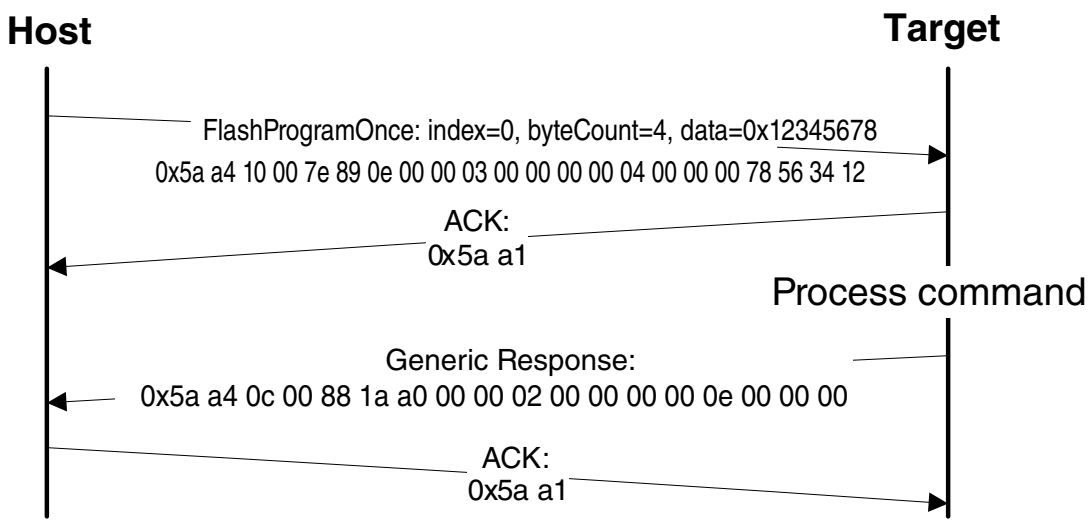

Figure 5-10. Protocol Sequence for FlashProgramOnce Command

Table 5-21. FlashProgramOnce Command Packet Format (Example)

FlashProgramOnce	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4, kFramingPacketType_Command
	length	0x10 0x00
	crc16	0x7E4 0x89
Command packet	commandTag	0x0E – FlashProgramOnce
	flags	0
	reserved	0
	parameterCount	3
	index	0x0000_0000

Table continues on the next page...

Table 5-21. FlashProgramOnce Command Packet Format (Example) (continued)

FlashProgramOnce	Parameter	Value
	byteCount	0x0000_0004
	data	0x1234_5678

Response: upon successful execution of the command, the target (Kinetis bootloader) returns a GenericResponse packet with a status code set to kStatus_Success, or to an appropriate error status code.

5.14 eFuseReadOnce command

The FlashReadOnce command returns the contents of the program once field by given index and byte count. The FlashReadOnce command uses 2 parameters: index and byteCount.

Table 5-22. Parameters for FlashReadOnce Command

Byte #	Parameter	Description
0 - 3	index	Index of the program once field (to read from)
4 - 7	byteCount	Number of bytes to read and return to the caller

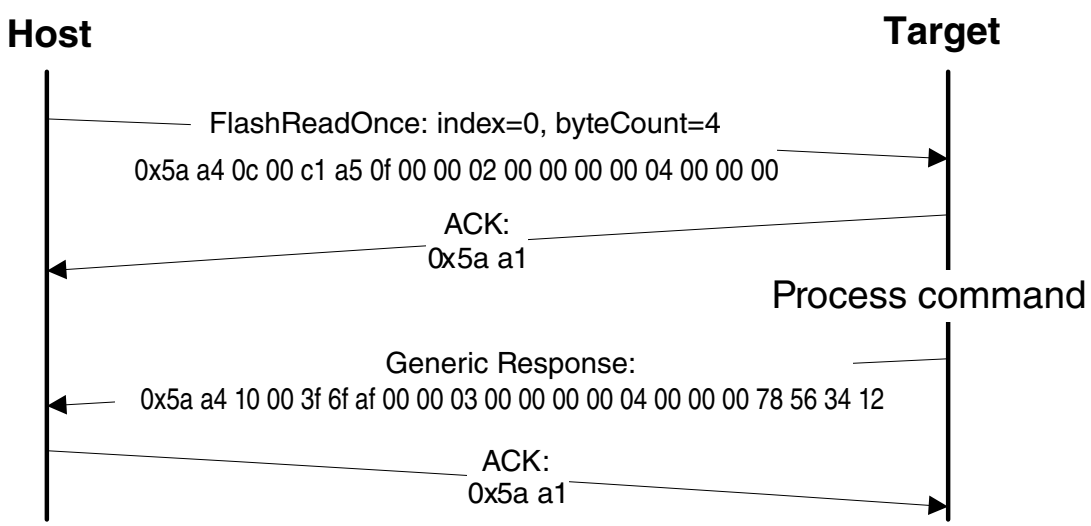

Figure 5-11. Protocol Sequence for FlashReadOnce Command

Table 5-23. FlashReadOnce Command Packet Format (Example)

FlashReadOnce	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4
	length	0x0C 0x00
	crc	0xC1 0xA5
Command packet	commandTag	0x0F – FlashReadOnce
	flags	0x00
	reserved	0x00
	parameterCount	0x02
	index	0x0000_0000
	byteCount	0x0000_0004

Table 5-24. FlashReadOnce Response Format (Example)

FlashReadOnce Response	Parameter	Value
Framing packet	start byte	0x5A
	packetType	0xA4
	length	0x10 0x00
	crc	0x3F 0x6F
Command packet	commandTag	0xAF
	flags	0x00
	reserved	0x00
	parameterCount	0x03
	status	0x0000_0000
	byteCount	0x0000_0004
	data	0x1234_5678

Response: upon successful execution of the command, the target returns a FlashReadOnceResponse packet with a status code set to kStatus_Success, a byte count and corresponding data read from Program Once Field upon successful execution of the command, or returns with a status code set to an appropriate error status code and a byte count set to 0.

5.15 Configure Memory command

The Configure Memory command configures the external memory device using a pre-programmed configuration image. The parameters passed in the command are the memory ID, and then the memory address from which the configuration data can be

loaded from. Options for loading the data can be a scenario where the configuration data is written to a RAM or flash location and then this command directs the bootloader to use the data at that location to configure the external memory devices.

Table 5-25. Parameters for Configure Memory Command

Byte #	Command
0 – 3	Memory ID
4 – 7	Configuration block address

Response: The target (Kinetis Bootloader) returns a GenericResponse packet with a status code either set to kStatus_Success upon successful execution of the command, or set to an appropriate error code.

5.16 ReceiveSBFile command

The Receive SB File command (ReceiveSbFile) starts the transfer of an SB file to the target. The command only specifies the size in bytes of the SB file that is sent in the data phase. The SB file is processed as it is received by the bootloader.

Table 5-26. Parameters for Receive SB File Command

Byte #	Command
0 - 3	Byte count

Data Phase: The Receive SB file command has a data phase; the host sends data packets until the number of bytes of data specified in the byteCount parameter of the Receive SB File command are received by the target.

Response: The target returns a GenericResponse packet with a status code set to the kStatus_Success upon successful execution of the command, or set to an appropriate error code.

Chapter 6
Supported peripherals

6.1 Introduction

This section describes the peripherals supported by the Kinetis bootloader. To use an interface for bootloader communications, the peripheral must be enabled in the BCA. If the BCA is invalid (such as all 0xFF bytes), then all peripherals are enabled by default.

6.2 UART Peripheral

The Kinetis bootloader integrates an autobaud detection algorithm for the UART peripheral, thereby providing flexible baud rate choices.

Autobaud feature: If UART*n* is used to connect to the bootloader, then the UART*n*_RX pin must be kept high and not left floating during the detection phase in order to comply with the autobaud detection algorithm. After the bootloader detects the ping packet (0x5A 0xA6) on UART*n*_RX, the bootloader firmware executes the autobaud sequence. If the baudrate is successfully detected, then the bootloader sends a ping packet response [(0x5A 0xA7), protocol version (4 bytes), protocol version options (2 bytes) and crc16 (2 bytes)] at the detected baudrate. The Kinetis bootloader then enters a loop, waiting for bootloader commands via the UART peripheral.

NOTE

The data bytes of the ping packet must be sent continuously (with no more than 80 ms between bytes) in a fixed UART transmission mode (8-bit data, no parity bit and 1 stop bit). If the bytes of the ping packet are sent one-by-one with more than 80 ms delay between them, then the autobaud detection algorithm may calculate an incorrect baud rate. In this instance, the autobaud detection state machine should be reset.

Supported baud rates: The baud rate is closely related to the MCU core and system clock frequencies. Typical baud rates supported are 9600, 19200, 38400, and 57600. Of course, to influence the performance of autobaud detection, the clock configuration in BCA can be changed.

Packet transfer: After autobaud detection succeeds, bootloader communications can take place over the UART peripheral. The following flow charts show:

- How the host detects an ACK from the target
- How the host detects a ping response from the target
- How the host detects a command response from the target

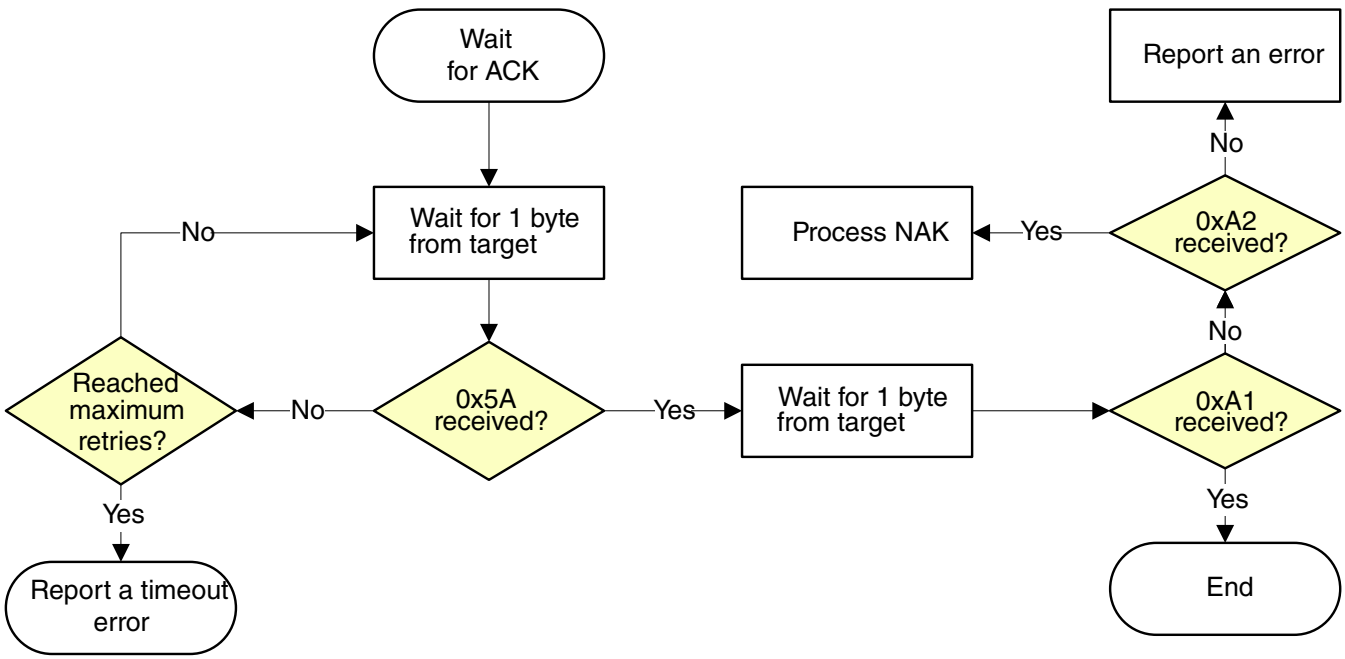

Figure 6-1. Host reads an ACK from target via UART

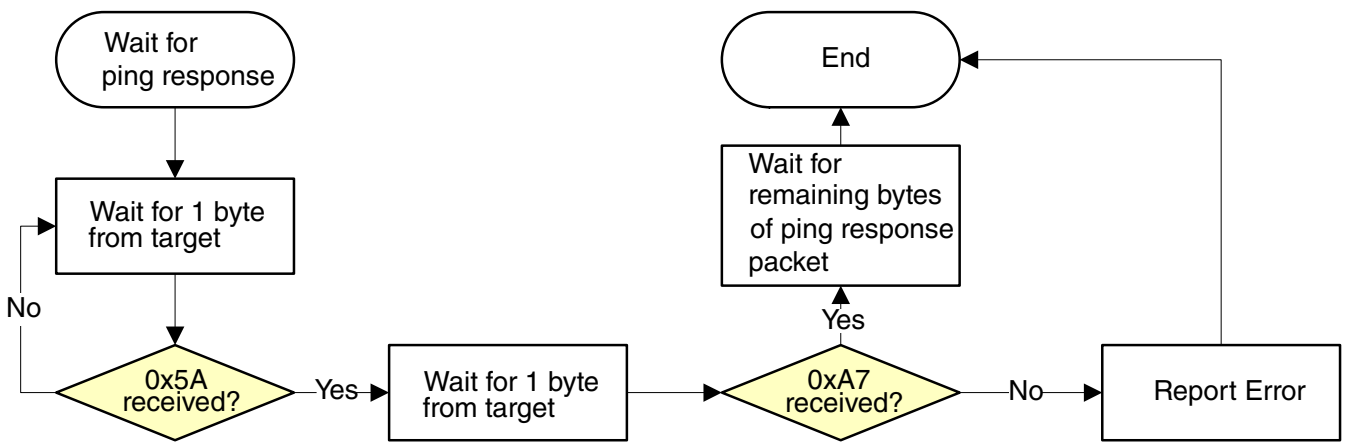

Figure 6-2. Host reads a ping response from target via UART

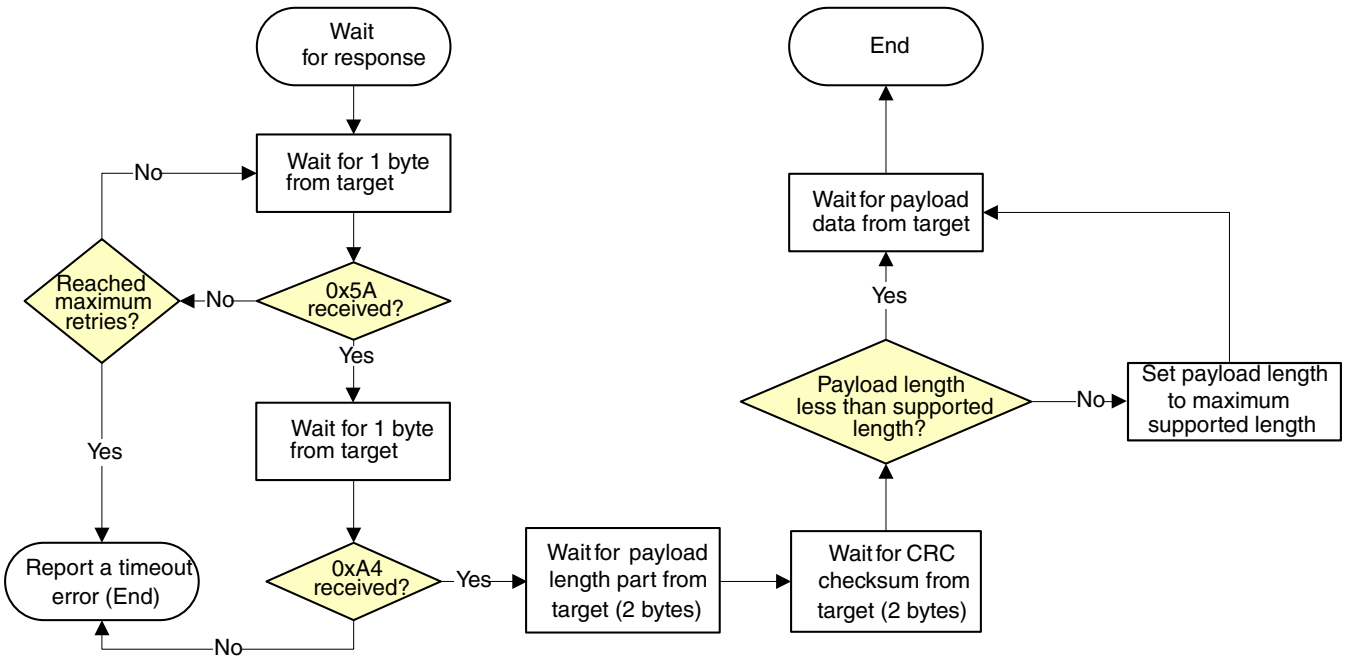

Figure 6-3. Host reads a command response from target via UART

6.2.1 Performance Numbers for UART

The table below provides reference to the expected performance of write speeds to Flash and RAM memories using Kinetis bootloader SPI interface. The numbers have been measured on a number of platforms running Kinetis bootloader either from ROM or the RAM (in case of flashloaders).

UART Baud Rate	**Flash Average Writing Speed (KB/s)**							**Ram Average Writing Speed (KB/s)**						
	KL27	KL28	KL43	KL80	K80	KL03	KS22	KL27	KL28	KL43	KL80	K80	KL03	KS22
19200	1.47	1.47	1.43	1.47	1.46	1.43	1.45	1.51	1.52	1.48	1.52	1.52	1.49	1.51
38400	2.81	2.82	2.75	2.82	2.79	2.81	2.75	2.99	3.03	2.95	3.03	3.03	2.9	3.00
57600	4.07	4.07	3.97	4.08	4.01	-	3.93	4.46	4.53	4.4	4.54	4.51	-	4.47
115200	7.3	7.31	7.12	7.35	7.1	-	6.88	8.69	8.97	8.65	8.98	8.85	-	8.73
230400	12.14	-	11.83	12.27	11.42	-	11.01	16.57	-	16.77	17.58	16.73	-	16.65

Table continues on the next page...

Default core Frequency (MHz)	48	48	48	48	48	8	48	48	48	48	48	48	8	48
Default bus Frequency (MHz)	24	24	24	24	24	4	24	24	24	24	24	24	4	24

NOTE

1. Every test covers all flash or RAM region with 0x0 - 0xf.
2. Run every test three times and calculate the average.

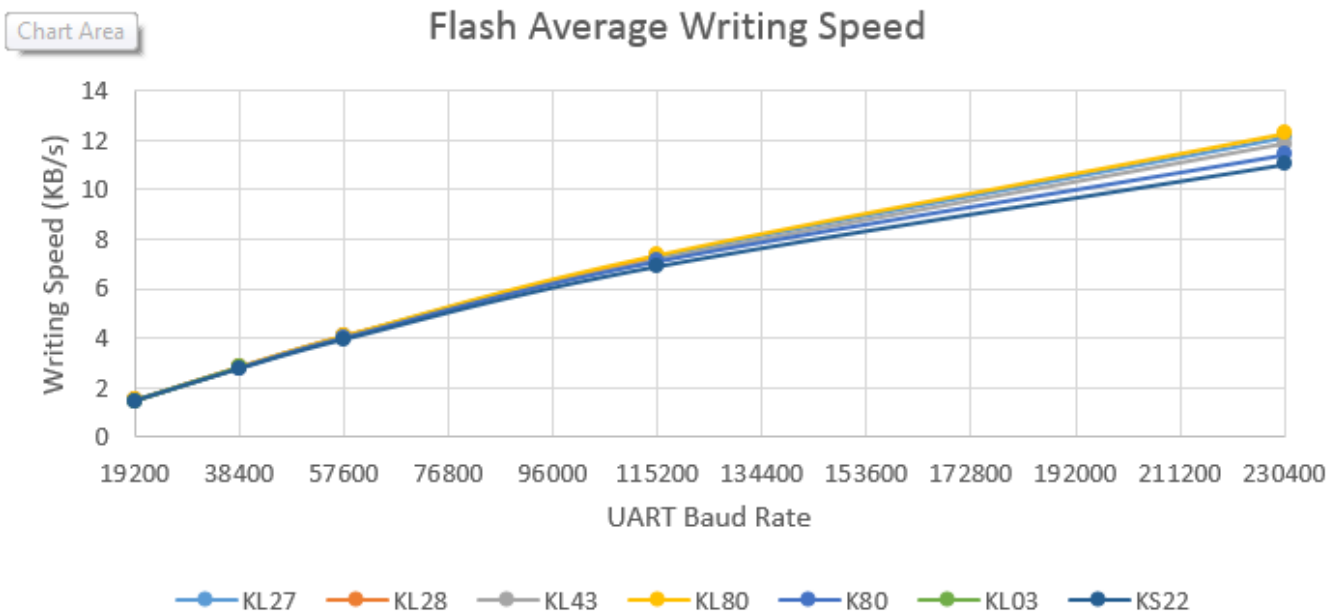

Figure 6-4. Flash Average Writing Speed

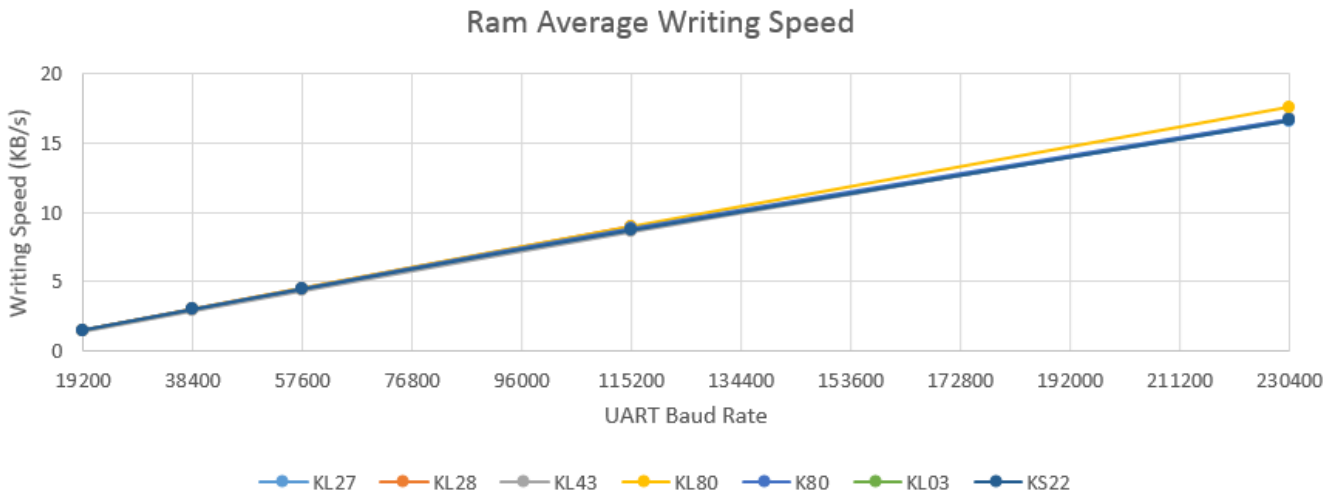

Figure 6-5. RAM Average Writing Speed

6.3 USB HID Peripheral

The Kinetis bootloader supports loading data into flash via the USB peripheral. The target is implemented as a USB HID class.

USB HID does not use framing packets; instead the packetization inherent in the USB protocol itself is used. The ability for the device to NAK Out transfers (until they can be received) provides the required flow control; the built-in CRC of each USB packet provides the required error detection.

6.3.1 Device descriptor

The Kinetis bootloader configures the default USB VID/PID/Strings as below:

Default VID/PID:

- VID = 0x15A2
- PID = 0x0073

Default Strings:

- Manufacturer [1] = "Freescale Semiconductor Inc."
- Product [2] = "Kinetis bootloader"

The USB VID, PID, and Strings can be customized using the Bootloader Configuration Area (BCA) of the flash. For example, the USB VID and PID can be customized by writing the new VID to the usbVid(BCA + 0x14) field and the new PID to the

usbPid(BCA + 0x16) field of the BCA in flash. To change the USB strings, prepare a structure (like the one shown below) in the flash, and then write the address of the structure to the usbStringsPointer(BCA + 0x18) field of the BCA.

```
g_languages = { USB_STR_0,
sizeof(USB_STR_0),
(uint_16)0x0409,
(const uint_8 **)g_string_descriptors,
g_string_desc_size};
the USB_STR_0, g_string_descriptors and g_string_desc_size are defined as below.
USB_STR_0[4] =    {0x02,
0x03,
0x09,
0x04
};
g_string_descriptors[4] =
{ USB_STR_0,
USB_STR_1,
USB_STR_2,
USB_STR_3};
g_string_desc_size[4] =
{ sizeof(USB_STR_0),
sizeof(USB_STR_1),
sizeof(USB_STR_2),
sizeof(USB_STR_3)};
```

- USB_STR_1 is used for the manufacturer string.
- USB_STR_2 is used for the product string.
- USB_STR_3 is used for the serial number string.

By default, the 3 strings are defined as below:

```
USB_STR_1[] =
{  sizeof(USB_STR_1),
USB_STRING_DESCRIPTOR,
'F',0,
'r',0,
'e',0,
'e',0,
's',0,
'c',0,
'a',0,
'l',0,
'e',0,
' ',0,
'S',0,
'e',0,
'm',0,
'i',0,
'c',0,
'o',0,
'n',0,
'd',0,
'u',0,
'c',0,
't',0,
'o',0,
'r',0,
' ',0,
'I',0,
'n',0,
'c',0,
```

```
'.',0
};

USB_STR_2[] =
{  sizeof(USB_STR_2),
USB_STRING_DESCRIPTOR,
'M',0,
'K',0,
' ',0,
'M',0,
'a',0,
's',0,
's',0,
' ',0,
'S',0,
't',0,
'o',0,
'r',0,
'a',0,
'g',0,
'e',0
};

USB_STR_3[] =
{  sizeof(USB_STR_3),
USB_STRING_DESCRIPTOR,
'0',0,
'1',0,
'2',0,
'3',0,
'4',0,
'5',0,
'6',0,
'7',0,
'8',0,
'9',0,
'A',0,
'B',0,
'C',0,
'D',0,
'E',0,
'F',0
};
```

6.3.2 Endpoints

The HID peripheral uses 3 endpoints:

- Control (0)
- Interrupt IN (1)
- Interrupt OUT (2)

The Interrupt OUT endpoint is optional for HID class devices, but the Kinetis bootloader uses it as a pipe, where the firmware can NAK send requests from the USB host.

6.3.3 HID reports

There are 4 HID reports defined and used by the bootloader USB HID peripheral. The report ID determines the direction and type of packet sent in the report; otherwise, the contents of all reports are the same.

Report ID	Packet Type	Direction
1	Command	OUT
2	Data	OUT
3	Command	IN
4	Data	IN

For all reports, these properties apply:

Usage Min	1
Usage Max	1
Logical Min	0
Logical Max	255
Report Size	8
Report Count	34

Each report has a maximum size of 34 bytes. This is derived from the minimum bootloader packet size of 32 bytes, plus a 2-byte report header that indicates the length (in bytes) of the packet sent in the report.

NOTE

In the future, the maximum report size may be increased, to support transfers of larger packets. Alternatively, additional reports may be added with larger maximum sizes.

The actual data sent in all of the reports looks like:

0	Report ID
1	Packet Length LSB
2	Packet Length MSB
3	Packet[0]
4	Packet[1]
5	Packet[2]
	...
N+3-1	Packet[N-1]

This data includes the Report ID, which is required if more than one report is defined in the HID report descriptor. The actual data sent and received has a maximum length of 35 bytes. The Packet Length header is written in little-endian format, and it is set to the size (in bytes) of the packet sent in the report. This size does not include the Report ID or the Packet Length header itself. During a data phase, a packet size of 0 indicates a data phase abort request from the receiver.

6.4 USB Peripheral

The Kinetis bootloader supports loading data into flash or RAM using the USB peripheral. The target is implemented as USB-HID and USB MSC (Mass Storage Class) composite device classes.

When transfer data through USB-HID device class, USB-HID does not use framing packets. Instead, the packetization inherent in the USB protocol itself is used. The ability for the device to NAK Out transfers (until they can be received) provides the required flow control. The built-in CRC of each USB packet provides the required error detection.

When transfer data through USB MSC device class, USB MSC does not use framing packets. Instead, the packetization inherent in the USB protocol itself is used. As with any mass storage class device, a device drive letter appears in the file manager of the operating system, and the file image can be dragged and dropped to the storage device. Right now, the USB MSC download only supports SB file drag-and-drop. Reading the SB file from the MSC device is not supported.

The USB peripheral can work as HID + MSC in Composite device mode. For HID-only mode or MSC-only mode, this is configured using macros during compile time. If configured as the HID and MSC composite device, users can either send commands to the HID interface, or drag/drop SB files to the MSC device.

6.4.1 Device descriptor

```
uint8_t *g_string_descriptors[USB_STRING_COUNT + 1] = { g_usb_str_0,

g_usb_str_1,

g_usb_str_2,

g_usb_str_3,

                                                      g_usb_str_n };
      usb_language_t g_usb_lang[USB_LANGUAGE_COUNT] = { {
```

```
        g_string_descriptors, g_string_desc_size, (uint16_t)0x0409,
      } };
      usb_language_list_t g_language_list = {
         g_usb_str_0, sizeof(g_usb_str_0), g_usb_lang, USB_LANGUAGE_COUNT,
      };
      uint8_t g_usb_str_1[USB_STRING_DESCRIPTOR_1_LENGTH +
      USB_STRING_DESCRIPTOR_HEADER_LENGTH] = {
         sizeof(g_usb_str_1),
         USB_DESCRIPTOR_TYPE_STRING,
'F',
0,
'R',
0,
'E',
0,
'E',
0,
'S',
0,
'C',
0,
'A',
0,
'L',
0,
'E',
0,
' ',
0,
'S',
0,
'E',
0,
'M',
0,
'I',
0,
'C',
0,
'O',
0,
'N',
0,
'D',
0,
'U',
0,
'C',
0,
'T',
0,
'O',
0,
'R',
0,
' ',
0,
'I',
0,
'N',
0,
'C',
0,
'.',
0
uint8_t g_usb_str_2[USB_STRING_DESCRIPTOR_2_LENGTH +
USB_STRING_DESCRIPTOR_HEADER_LENGTH] = {
       sizeof(g_usb_str_2),
       USB_DESCRIPTOR_TYPE_STRING,
      'U',
```

```
       0,
      'S',
       0,
      'B',
       0,
      ' ',
       0,
      'C',
       0,
      'O',
       0,
      'M',
       0,
      'P',
       0,
      'O',
       0,
      'S',
       0,
      'I',
       0,
      'T',
       0,
      'E',
       0,
      ' ',
       0,
      'D',
       0,
      'E',
       0,
      'V',
       0,
      'I',
       0,
      'C',
       0,
      'E',
       0
};
```

For HID and MSC composite devices.

```
uint8_t g_usb_str_3[USB_STRING_DESCRIPTOR_3_LENGTH +
USB_STRING_DESCRIPTOR_HEADER_LENGTH] = {
      sizeof(g_usb_str_3),
      USB_DESCRIPTOR_TYPE_STRING,
     'M',
      0,
     'C',
      0,
     'U',
      0,
     ' ',
      0,
     'M',
      0,
     'S',
      0,
     'C',
      0,
     ' ',
      0,
     'A',
      0,
     'N',
      0,
     'D',
      0,
```

```
      ' ',
       0,
      'H',
       0,
      'I',
       0,
      'D',
       0,
      ' ',
       0,
      'G',
       0,
      'E',
       0,
      'N',
       0,
      'E',
       0,
      'R',
       0,
      'I',
       0,
      'C',
       0,
      ' ',
       0,
      'D',
       0,
      'E',
       0,
      'V',
       0,
      'I',
       0,
      'C',
       0,
      'E',
       0};
```

For HID-only devices.

```
uint8_t g_usb_str_3[USB_STRING_DESCRIPTOR_3_LENGTH +
USB_STRING_DESCRIPTOR_HEADER_LENGTH] = {
      sizeof(g_usb_str_3),
      USB_DESCRIPTOR_TYPE_STRING,
      'M',
       0,
      'C',
       0,
      'U',
       0,
      ' ',
       0,
      'H',
       0,
      'I',
       0,
      'D',
       0,
      ' ',
       0,
      'G',
       0,
      'E',
       0,
      'N',
       0,
      'E',
       0,
```

```
     'R',
      0,
     'I',
      0,
     'C',
      0,
     ' ',
      0,
     'D',
      0,
     'E',
      0,
     'V',
      0,
     'I',
      0,
     'C',
      0,
     'E',
      0
};
```

For MSC-only devices.

```
uint8_t g_usb_str_3[USB_STRING_DESCRIPTOR_3_LENGTH +
USB_STRING_DESCRIPTOR_HEADER_LENGTH] = {
      sizeof(g_usb_str_3),
      USB_DESCRIPTOR_TYPE_STRING,
     'M',
      0,
     'C',
      0,
     'U',
      0,
     ' ',
      0,
     'M',
      0,
     'S',
      0,
     'C',
      0,
     ' ',
      0,
     'D',
      0,
     'E',
      0,
     'V',
      0,
     'I',
      0,
     'C',
      0,
     'E',
      0
};
```

6.4.2 Endpoints

```
USB MSC device uses 2 endpoints, in addition to the default pipe that is required by USB HID
device
#define USB_MSC_BULK_IN_ENDPOINT (3), which
#define USB_MSC_BULK_OUT_ENDPOINT (4)
```

Chapter 7
Peripheral interfaces

7.1 Introduction

The block diagram shows connections between components in the architecture of the peripheral interface.

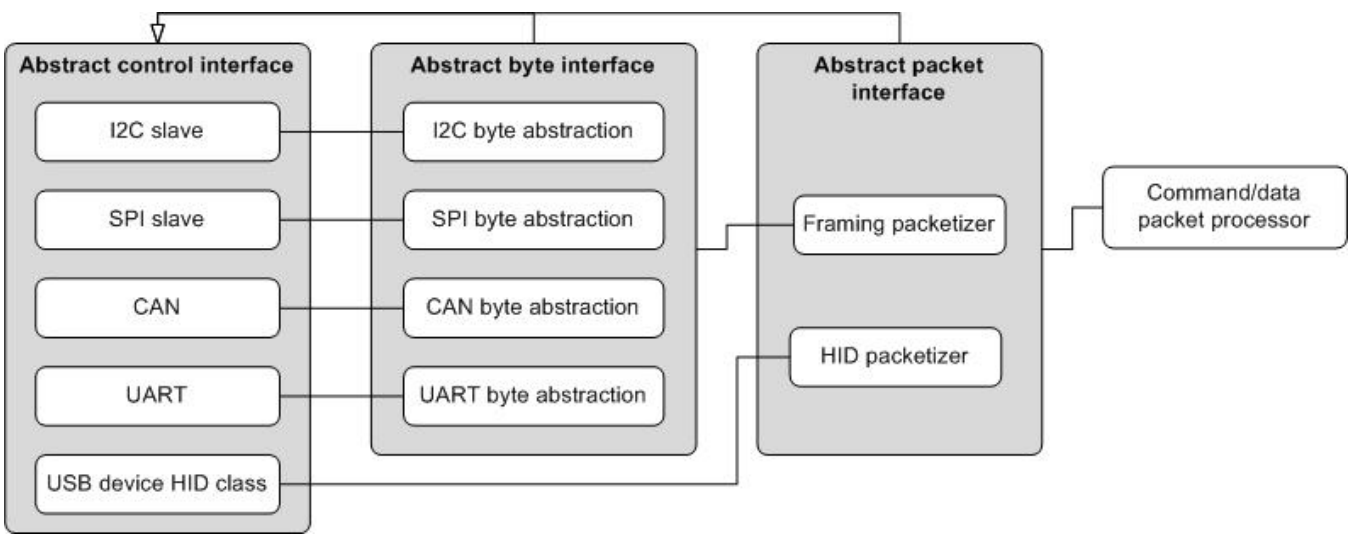

Figure 7-1. Components peripheral interface

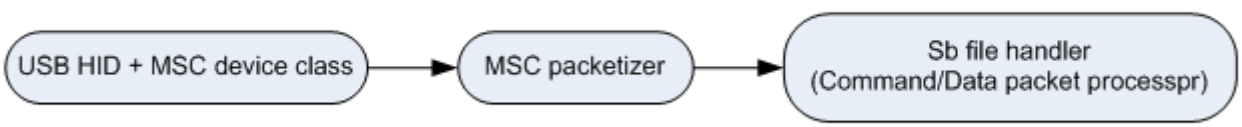

Figure 7-2. USB/MSC Peripheral interface

In this diagram, the byte and packet interfaces are shown to inherit from the control interface.

All peripheral drivers implement an abstract interface built on top of the driver's internal interface. The outermost abstract interface is a packet-level interface. It returns the payload of packets to the caller. Drivers which use framing packets have another abstract interface layer that operates at the byte level. The abstract interfaces allow the higher layers to use exactly the same code regardless which peripheral is being used.

The abstract packet interface feeds into the command and data packet processor. This component interprets the packets returned by the lower layer as command or data packets.

7.2 Abstract control interface

This control interface provides a common method to initialize and shutdown peripheral drivers. It also provides the means to perform the active peripheral detection. No data transfer functionality is provided by this interface. That is handled by the interfaces that inherit the control interface.

The main reason this interface is separated out from the byte and packet interfaces is to show the commonality between the two. It also allows the driver to provide a single control interface structure definition that can be easily shared.

```
struct PeripheralDescriptor {
    //! @brief Bit mask identifying the peripheral type.
    //!
    //! See #_peripheral_types for a list of valid bits.
    uint32_t typeMask;

    //! @brief The instance number of the peripheral.
    uint32_t instance;

    //! @brief Configure pinmux setting for the peripheral.
    void (*pinmuxConfig)(uint32_t instance, pinmux_type_t pinmux);

    //! @brief Control interface for the peripheral.
    const peripheral_control_interface_t * controlInterface;

    //! @brief Byte-level interface for the peripheral.
    //!
    //! May be NULL because not all periperhals support this interface.
    const peripheral_byte_inteface_t * byteInterface;

    //! @brief Packet level interface for the peripheral.
    const peripheral_packet_interface_t * packetInterface;
};

struct PeripheralControlInterface
{
    bool (*pollForActivity)(const PeripheralDescriptor * self);
    status_t (*init)(const PeripheralDescriptor * self, BoatloaderInitInfo * info);
    void (*shutdown)(const PeripheralDescriptor * self);
```

```
    void (*pump)(const peripheral_descriptor_t *self);
}
```

Table 7-1. Abstract control interface

Interface	Description
pollForActivity()	Check whether communications has started.
init()	Fully initialize the driver.
shutdown()	Shutdown the fully initialized driver.
pump	Provide execution time to driver.

7.3 Abstract byte interface

This interface exists to give the framing packetizer, which is explained in the later section, a common interface for the peripherals that use framing packets.

The abstract byte interface inherits the abstract control interface.

```
struct PeripheralByteInterface
{
    status_t (*init)(const peripheral_descriptor_t * self);
    status_t (*write)(const peripheral_descriptor_t * self, const uint8_t *buffer, uint32_t
byteCount);
};
```

Table 7-2. Abstract byte interface

Interface	Description
init()	Initialize the interface.
write()	Write the requested number of bytes.

NOTE

The byte interface has no read() member. Interface reads are performed in an interrupt handler at the packet level.

7.4 Abstract packet interface

The abstract packet interface inherits the abstract control interface.

```
    status_t (*init)(const peripheral_descriptor_t *self);
    status_t (*readPacket)(const peripheral_descriptor_t *self,
                           uint8_t **packet,
                           uint32_t *packetLength,
                           packet_type_t packetType);
    status_t (*writePacket)(const peripheral_descriptor_t *self,
```

```
                            const uint8_t *packet,
                            uint32_t byteCount,
                            packet_type_t packetType);
    void (*abortDataPhase)(const peripheral_descriptor_t *self);
    status_t (*finalize)(const peripheral_descriptor_t *self);
    uint32_t (*getMaxPacketSize)(const peripheral_descriptor_t *self);
    void (*byteReceivedCallback)(uint8_t byte);
```

Table 7-3. Abstract packet interface

Interface	Description
init()	Initialize the peripheral.
readPacket()	Read a full packet from the peripheral.
writePacket()	Send a complete packet out the peripheral.
abortDataPhase()	Abort receiving of data packets.
finalize()	Shut down the peripheral when done with use.
getMaxPacketSize	Returns the current maximum packet size.
byteReceivedCallback	Notification of received byte.

7.5 Framing packetizer

The framing packetizer processes framing packets received via the byte interface with which it talks. It builds and validates a framing packet as it reads bytes. And it constructs outgoing framing packets as needed to add flow control information and command or data packets. The framing packet also supports data phase abort.

7.6 USB HID packetizer

The USB HID packetizer implements the abstract packet interface for USB HID, taking advantage of the USB's inherent flow control and error detection capabilities. The USB HID packetizer provides a link layer that supports variable length packets and data phase abort.

7.7 USB HID packetizer

The USB HID packetizer implements the abstract packet interface for USB HID, taking advantage of the USB's inherent flow control and error detection capabilities.

The below image shows the USB MSC command/data/status flow chart:

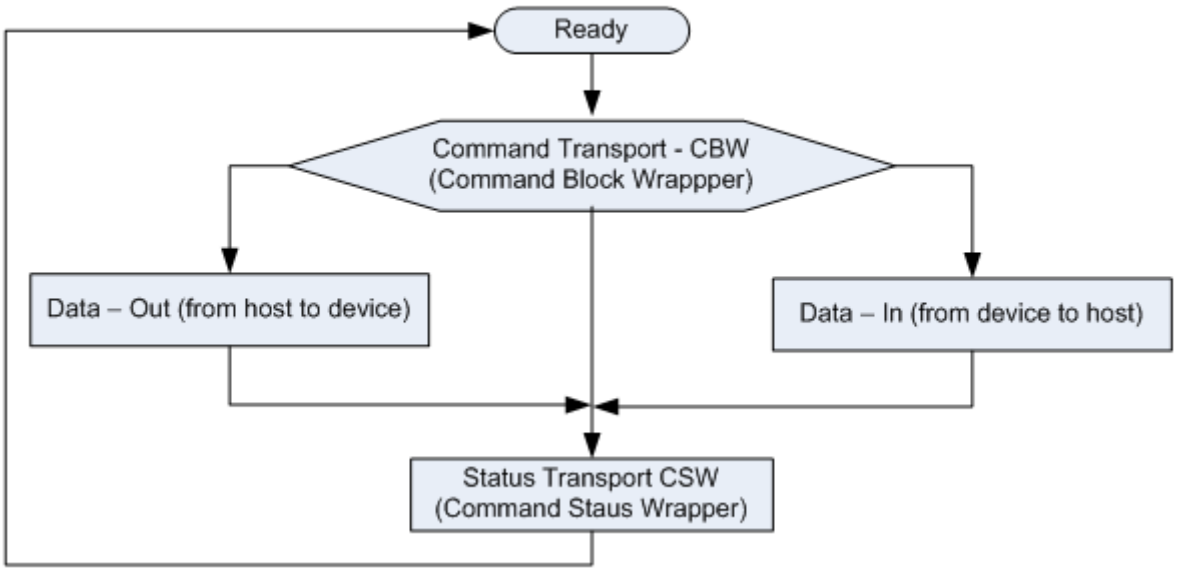

Figure 7-3. USB MSC status flow chart

- The CBW begins on a packet boundary, and ends as a short packet. Exactly 31 bytes are transferred.
- The CSW begins on a packet boundary, and ends as a short packet. Exactly 13 bytes are transferred.
- The data packet begins on a packet boundary, and ends as a short packet. Exactly 64 bytes are transferred.

7.8 Command/data processor

This component reads complete packets from the abstract packet interface, and interprets them as either command packets or data packets. The actual handling of each command is done by command handlers called by the command processor. The command handler tells the command processor whether a data phase is expected and how much data it is expected to receive.

If the command/data processor receives a unexpected command or data packet, it ignores it. In this instance, the communications link resynchronizes upon reception of the next valid command.

Chapter 8
External Memory Support

8.1 Introduction

This section describes the external memory devices supported by the MCU Flashloader. To use an external memory device correctly, the device must be enabled with corresponding configuration profile. If the external memory device is not enabled, then it cannot be accessed by Flashloader. The MCU Flashloader enables specific external memory device using pre-assigned memory identifier, supported external memory devices and memory identifiers are shown below.

Table 8-1. Memory ID for external memory devices

Memory Identifier	External Memory device
0x01	'Serial NOR over QuadSPI module'
0x08	'Parallel NOR over SEMC module'
0x09	'Serial NOR over FlexSPI module'
0x100	'SLC raw NAND over SEMC module'
0x101	'Serial NAND over FlexSPI module'
0x110	'Serial NOR/EEPROM over LPSPI module'
0x120	'SD over uSDHC'
0x121	'eMMC over uSDHC'

8.2 Serial NOR Flash through FlexSPI

The MCU Flashloader supports read, write and erase external Serial NOR Flash devices via the FlexSPI Module. Before accessing Serial NOR Flash devices, the FlexSPI module must be configured properly, using a simplified FlexSPI NOR Config option block or a complete 512-byte FlexSPI NOR Configuration Block. Flashloader can generate the 512-byte FlexSPI NOR Configuration Block based on the simplified Flash Configuration

Option Block for most Serial NOR Flash devices in the market. To protect Intellectual Property on external Serial NOR Flash, Flashloader also supports image encryption and programming using OTPMK/SNVS key if the chip supports BEE module. Refer to the Security Utility chalter for further details.

8.2.1 FlexSPI NOR Configuration Block

Table 8-2. Memory ID for external memory devices

Name	Offset	Size(Bytes)	Description
Tag	0x000	4	0x42464346, ascii:"FCFB"
Version	0x004	4	0x56010000 [07:00] bugfix [15:08] minor [23:16] major = 1 [31:24] ascii 'V'
-	0x008	4	Reserved
readSampleClkSrc	0x00c	1	0 – Internal loopback 1 – loopback from DQS pad 3 – Flash provided DQS
csHoldTime	0x00d	1	Serial Flash CS Hold Time Recommend default value is 0x03
csSetupTime	0x00e	1	Serial Flash CS Setup Time Recommend default value is 0x03
columnAdressWidth	0x00f	1	3 – For HyperFlash/ HyperRAM 12/13 – For Serial NAND, see datasheet to find correct value 0 – Other devices
deviceModeCfgEnable	0x010	1	Device Mode Configuration Enable feature 0 – Disabled 1 – Enabled
deviceModeType	0x011	1	Specify the Configuration command type 0 - Generic Command 1 - Quad Enable 2 - SPI to OPI Others - Reserved

Table continues on the next page...

Table 8-2. Memory ID for external memory devices (continued)

Name	Offset	Size(Bytes)	Description
waitTimeCfgCommands	0x012	2	Wait time for all configuration commands, unit: 100us. 0 - Use read status command to determine the busy status for configuration commands Others - Delay "waitTimeCfgCommads" * 100us for configuration commands
deviceModeSeq	0x014	4	Sequence parameter for device mode configuration [7:0] LUT sequence Id [15:8] LUT sequence number for this sequence [31:16] Reserved for future use
deviceModeArg	0x018	4	Device Mode argument, effective only when deviceModeCfgEnable = 1
configCmdEnable	0x01c	1	Config Command Enable feature 0 – Disabled 1 – Enabled
configModeType	0x01d	3	Configure mode type, the same definition as "deviceModeType"
configCmdSeqs	0x020	12	Sequences for Config Command, allow 4 separate configuration command sequences
-	0x02c	4	Reserved
cfgCmdArgs	0x030	12	Arguments for each separate configuration command sequence
-	0x03c	4	Reserved
controllerMiscOption	0x040	4	Bit0 – Enable differential clock Bit2 – Enable Parallel Mode Bit3 – Enable Word Addressable Bit4 – Enable Safe Config Freq Bit5 – Enable Pad Setting Override Bit6 – Enable DDR Mode Others - Reserved

Table continues on the next page...

Table 8-2. Memory ID for external memory devices (continued)

Name	Offset	Size(Bytes)	Description
deviceType	0x044	1	1 - Serial NOR 2 - Serial NAND
sflashPadType	0x045	1	1 – Single pad 2 – Dual pads 4 – Quad pads 8 – Octal pads Others - Invalid value
serialClkFreq	0x046	1	Device specific value, check System Boot chapter in the SoC RM for more details
lutCustomSeqEnable	0x047	1	0 - Use pre-defined LUT sequence index and number 1 - Use LUT sequence parameters provided in this block
Reserved	0x048	8	Reserved
sflashA1Size	0x050	4	For SPI NOR, need to fill with actual size For SPI NAND, need to fill with actual size * 2
sflashA2Size	0x054	4	For SPI NOR, need to fill with actual size For SPI NAND, need to fill with actual size * 2
sflashB1Size	0x058	4	For SPI NOR, need to fill with actual size For SPI NAND, need to fill with actual size * 2
sflashB2Size	0x05c	4	For SPI NOR, need to fill with actual size For SPI NAND, need to fill with actual size * 2
csPadSettingOverride	0x060	4	Set to 0 if it is not supported
sclkPadSettingOverride	0x064	4	Set to 0 if it is not supported
dataPadSettingOverride	0x068	4	Set to 0 if it is not supported
dqsPadSettingOverride	0x06c	4	Set to 0 if it is not supported
timeoutInMs	0x070	4	Maximum wait time during read/write Not used in ROM
commandInterval	0x074	4	Unit: ns Currently, it is used for SPI NAND at high working frequency

Table continues on the next page...

Table 8-2. Memory ID for external memory devices (continued)

Name	Offset	Size(Bytes)	Description
dataValidTime	0x078	4	Time from clock edge to data valid edge, unit ns This field is used when the FlexSPI Root clock is less than 100MHz and the read sample clock source is device provided DQS signal without CK2 support [31:16] data valid time for DLLB in terms of 0.1ns [15:0] data valid time for DLLA in terms of 0.1ns
busyOffset	0x07c	2	busy bit offset, valid range : 0-31
busyBitPolarity	0x07e	2	0 – busy bit is 1 if device is busy 1 – busy bit is 0 if device is busy
lookupTable	0x080	256	Lookup table
lutCustomSeq	0x180	48	Customized LUT sequence, see below table for details
-	0x1b0	16	Reserved
pageSize	0x1c0	4	Flash Page size
sectorSize	0x1c4	4	Flash Sector Size
ipCmdSerialClkFreq	0x1c8	4	IP Command Clock Frequency, the same definition as "serialClkFreq"
isUniformBlockSize	0x1c9	4	Sector / Block size is identical or not
-	0x1ca	2	-
serialNorType	0x1cc	1	Serial NOR Flash Type: 0 - Extended SPI 1 - HyperBus 2 - Octal DDR
needExitNoCmdMode	0x1cd	4	Reserved, set to 0
halfClkForNonReadCmd	0x1ce	1	Divide the clock for SDR command by 2 Need to set for the device that only support DDR read and other commands are SDR commands
needrestorNoCmdMode	0x1cf	1	Reserved, set 0
blockSize	0x1d0	4	Flash Block size
-	0x1d4	44	Reserved

NOTE

To customize the LUT sequence for some specific device, users need to enable "**lutCustomSeqEnable**" and fill in corresponding "**lutCustomSeq**" field specified by command index below.

For Serial (SPI) NOR, the pre-defined LUT index is as following:

Table 8-3. Lookup Table index pre-assignment for FlexSPI NOR

Name	Index in lookup table	Description
Read	0	Read command Sequence
ReadStatus	1	Read Status command
ReadStatusXpi	2	Read Status command under OPI mode
WriteEnable	3	Write Enable command sequence
WriteEnableXpi	4	Write Enable command under OPI mode
EraseSector	5	Erase Sector Command
EraseBlock	8	Erase Block Command
PageProgram	9	Page Program Command
ChipErase	11	Full Chip Erase
ExitNoCmd	15	Exit No Command Mode as needed
Reserved	6,7,10,12,13,14	All reserved indexes can be freely used for other purpose

8.2.2 FlexSPI NOR Configuration Option Block

The FlexSPI NOR Configuration Option Block is organized by 4-bit unit, and it is expandable, current definition of the block is as shown in below table.

The Flashloader detects FNORCB using read SFDP command which is supported by most flash devices those are JESD216(A/B) compliant. However, JESD216A/B only defines the dummy cycles for Quad SDR read. In order to get the dummy cycles for DDR/DTR read mode, flashloader supports auto probing by writing test patterns to offset 0x200 on the external memory devices. To get optimal timing, the readSampleClkSrc is

set to 1 in Flashloader for Flash devices that do not support external provided DQS pad input. It is set to 3 in Flashloader for flash devices that support external provided DQS pad input such as HyperFLASH. FlexSPI_DQS pad is not used for other purpose.

Table 8-4. FlexSPI NOR Configuration Option Block

<table>
<tr><th>Offset</th><th>Field</th><th>Description</th></tr>
<tr><td>0</td><td>Option0</td><td>
<table>
<tr><th>TAG [31:28]</th><th>Option size [27:24]</th><th>Device Detection Type [23:20]</th><th>Query CMD Pad(s) [19:16]</th><th>CMD Pad(s) [15:12]</th><th>Quad Enable Type[11:8]</th><th>Misc[7:4]</th><th>Max Freq [3:0]</th></tr>
<tr><td>0x0c</td><td>Size in bytes = (Option Size + 1) * 4</td><td>0 - QuadSPI SDR
1 - QuadSPI DDR
2 - HyperFLASH 1V8
3 - HyperFLASH 3V
4 - MXIC OPI DDR
6 - Micron OPI DDR
8 - Adesto OPI DDR</td><td>0 - 1
2 - 4
3 - 8</td><td>0 - 1
2 - 4
3 - 8</td><td>1 - QE bit is bit 6 in StatusReg1
2 - QE bit is bit 1 in StatusReg2
3 - QE bit is in bit7 in StatusReg2
4 - QE bit is bit 1 in StatusReg2, enable command is 0x31</td><td>3 - Byte order is swapped under OPI DDR mode</td><td>Device specific, see System Boot chapter in SoC RM for more details</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>Option1 Optional</td><td>
<table>
<tr><th>Reserved [31:8]</th><th>Dummy Cycle [7:0]</th></tr>
<tr><td>Reserved for future use</td><td>0 - Use auto-probing dummy cycle
Others - dummy cycles provided in data sheet</td></tr>
</table>
</td></tr>
</table>

- Tag - Fixed as 0x0C
- Option Size - Provide scalability for future use, the option block size equals to (Option size + 1) * 4 bytes
- Device Detection type - SW defined device types used for config block auto detection
- Query Command Pad(s) - Command pads (1/4/8) for the SFDP command

- CMD pad(s) - Commands pads for the Flash device (1/4/8), for device that works under 1-1-4,1-4-4,1-1-8 or 1-8-8 mode, CMD pad(s) value is always 0x0, for devices that only support 4-4-4 mode for high performance, CMD pads value is 2, for devices that only support 8-8-8 mode for high performance, CMD pads value is 3
- Quad Enable Type - Specify the Quad Enable sequence, only applicable for device that only JESD216 compliant, this field is ignored if device support JESD216A or later version
- Misc - Specify miscellaneous mode for selected flash type
- Max Frequency - the maximum work frequency for specified Flash device
- Dummy Cycle - user provided dummy cycles for SDR/DDR read command

8.2.2.1 Typical use cases for FlexSPI NOR Configuration Block

- QuadSPI NOR - Quad SDR Read: option0 = 0xc0000006 (100MHz)
- QuadSPI NOR - Quad DDR Read: option0 = 0xc0100003 (60MHz)
- HyperFLASH 1V8: option0 = 0xc0233007 (133MHz)
- HyperFLASH 3V0: option0 = 0xc0333006 (100MHz)
- MXIC OPI DDR (OPI DDR enabled by default): option=0xc0433006(100MHz)
- Micron Octal DDR: option0=0xc0600006 (100MHz)
- Micron OPI DDR: option0=0xc0603006 (100MHz), SPI->OPI DDR
- Micron OPI DDR (DDR read enabled by default): option0=0xc0633006(100MHz)
- Adesto OPI DDR: option0=0xc0803007(133MHz)

8.2.2.2 Program Serial NOR Flash device using FlexSPI NOR Configuration Option

The MCU Flashloader supports generating complete FNORCB using configure-memory command. It also supports programming the generated FNORCB to the start of the flash memory using a specific option "0xF000000F". Here is the example for configuring and accessing HyperFLASH (Assuming it is a blank HyperFLASH device).

```
blhost -u -- fill-memory 0x2000 0x04 0xc0233007 (write option block to SRAM address 0x2000)
blhost -u -- configure-memory 0x09 0x2000 (configure HyperFLASH using option block)
blhost -u -- fill-memory 0x3000 0x04 0xf000000f (write specific option to SRAM address
0x3000)
blhost -u -- configure-memory 0x09 0x3000 (program FNORCB to the start of HyperFLASH)
blhost -u -- write-memory <addr> image.bin
```

8.3 Serial NAND Flash through FlexSPI

Some MCU support booting from Serial NAND Flash devices via BootROM, the MCU Flashloader works as a companion to program the boot image into the Serial NAND, the Flashloader supports generating corresponding boot data structure like FlexSPI NAND Firmware Configuration Block (FCB) and Discovered Bad Block Table (DBBT) required by the BootROM. Please refer to System Boot Chapter in device reference manual for details regarding FlexSPI NAND boot flow. This chapter only focuses on generating FCB, DBBT and programming FCB, DBBT and boot images using Flashloader. Flashloader can configure Serial NAND devices using FCB, or a simplified FCB option block, Flashloader can generate a complete FCB based on the simplified FCB option block.

8.3.1 FlexSPI NAND Firmware Configuration Block(FCB)

FCB is a 1024-byte data structure that contains the optimum NAND timings, page address of Discovered Bad Block Table (DBBT) Search Area and firmware info (including start page address and page count), etc.

Table 8-5. FlexSPI NAND Firmware Configuration Block Definition

Name	Offset	Size(Bytes)	Description
crcChecksum	0x000	4	Checksum
fingerprint	0x004	4	0x4E46_4342 ASCII: “NFCB”
version	0x008	4	0x0000_0001
DBBTSearchStartPage	0x00c	4	Start Page address for bad block table search area
searchStride	0x010	2	Search stride for DBBT and FCB search Not used by ROM, max value is defined in Fusemap. See the Fusemap in SoC RM for more details.
searchCount	0x012	2	Copies on DBBT and FCB Not used by ROM, max value is defined in Fusemap. See the Fusemap in SoC RM for more details.
firmwareCopies	0x014	4	Firmware copies Valid range 1-8

Table continues on the next page...

Table 8-5. FlexSPI NAND Firmware Configuration Block Definition (continued)

Name	Offset	Size(Bytes)	Description
Reserved	0x018	40	Reserved for future use Must be set to 0
firmwareInfoTable	0x40	64	This table consists of (up to 8 entries): Field / Size(Bytes) / Description StartPage / 4 / Start page of this firmware pageCount / 4 / Pages in this firmware
Reserved	0x080	128	Reserved Must be set to 0
spiNandConfigBlock	0x100	512	Serial NAND configuration block over FlexSPI
Reserved	0x300	256	Reserved Must be set to 0

8.3.2 FlexSPI NAND Configuration Block

The optimum Serial NAND parameters are defined in FlexSPI NAND Configuration Block (FNANDCB), FNANDCB is a 512-byte data structure as shown in below table.

Table 8-6. FlexSPI NAND Configuration Block Definition

Name	Offset	Size (Bytes)	Description
memCfg	0x00	480	The same definition as the first 480 bytes in FlexSPI NOR Configuration Block
pageDataSize	0x1c0	480	Page size in bytes, in general, it is 2048 or 4096
pageTotalSize	0x1c4	4	It equals to 2 ^ width of column address
pagesPerBlock	0x1c8	4	Pages per Block
bypassReadStatus	0x1cc	1	0 - Perform Read Status 1 - Bypass Read Status

Table continues on the next page...

Table 8-6. FlexSPI NAND Configuration Block Definition (continued)

Name	Offset	Size (Bytes)	Description
bypassEccRead	0x1cd	1	0 - Perform ECC Read 1 - Bypass ECC Read
hasMultiPlanes	0x1ce	1	0 - Device has only 1 plane 1 - Device has 2 planes
-	0x1cf	1	Reserved
eccCheckCustomEnable	0x1d0	1	0 - Use the commonly used ECC check command and masks 1 - Use ECC check related masks provide in this configuration block
ipCmdSerialClkFreq	0x1d1	1	Chip specific value, set to 0
readPageTimeUs	0x1d2	2	Wait time during page read, only effective if "bypassReadStatus" is set to 1
eccStatusMask	0x1d4	4	ECC status mask, only effective if "eccCheckCustomEnable" is set to 1
eccFailureMask	0x1d8	4	ECC Check Failure mask, only effective if "eccCheckCustomEnable" is set to 1
blocksPerDevice	0x1dc	4	Blocks in a Serial NAND device
-	0x1e0	32	Reserved

NOTE

For Serial (SPI) NAND, the pre-defined LUT index is as following:

Table 8-7. Lookup Table index pre-assignment for FlexSPI

Command Index	Name	Index in lookup table	Description
0	ReadFromCache	0	Read From cache
1	ReadStatus	1	Read Status
2	WriteEnable	3	Write Enable
3	BlockErase	5	Erase block
4	ProgramLoad	9	Program Load
5	ReadPage	11	Read page to cache
6	ReadEccStatus	13	Read ECC Status
7	ProgramExecute	14	Program Execute

Table continues on the next page...

Table 8-7. Lookup Table index pre-assignment for FlexSPI (continued)

Command Index	Name	Index in lookup table	Description
8	ReadFromCacheOdd	4	Read from Cache while page in odd plane
9	ProgramLoadOdd	10	-
-	Reserved	2,6,7,8,12,15	All reserved indexes can be freely used for other purposes

8.3.3 FlexSPI NAND FCB option block

FlexSPI NAND FCB option block defines the major parameters required by FCB, such as image info. The detailed configuration block definition is shown below.

Table 8-8. FlexSPI NAND FCB option block

<table>
<tr><th>Offset</th><th>Field</th><th>Size</th><th>Description</th></tr>
<tr><td>0</td><td>option0</td><td>4</td><td>
<table>
<tr><th>Offset</th><th>Field</th><th>Description</th></tr>
<tr><td>31:28</td><td>tag</td><td>Fixed to 0x0E</td></tr>
<tr><td>27:24</td><td>searchCount</td><td>Valid value:
1-4</td></tr>
<tr><td>23:20</td><td>searchStride</td><td>0 - 64 pages
1 - 128 pages
2 - 256 pages
3 - 32 pages
NOTE: This field is aligned with Fuse definition</td></tr>
<tr><td>19:12</td><td>Reserved</td><td>-</td></tr>
<tr><td>11:8</td><td>Reserved</td><td>0 - byte address
1 - block address</td></tr>
<tr><td>7:4</td><td>Reserved</td><td>-</td></tr>
<tr><td>3:0</td><td>Reserved</td><td>Option size in longword, Min size is 3, Max size is 10</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>nandOptionAddr</td><td>4</td><td>Address of NAND option defined above</td></tr>
</table>

Table continues on the next page...

Table 8-8. FlexSPI NAND FCB option block (continued)

8	imageInfo	4-32	Image info is a map of below info, maximum entry size is 8 **Field** / **Size** / **Description** blockCount / 2 / Maximum allowed blocks for this image blockId / 2 / Start block index for this image

NOTE

- "searchCount" should match the one provisioned in eFUSE
- "searchStride" should match the one provisioned in eFUSE
- "addressType" specifies the address type for the start address of erase, write and read operation in Flashloader
- "Option size" specifies the total size of the option block size in longwords
- "nandOptionAddr" specifies the address that stores FlexSPI NAND Configuration Option
- "imageInfo" is an array that holds each image info used during boot. For example, 0x00040002 means the block Id is 4, maximum allowed block count is 2

8.3.4 FlexSPI NAND Configuration Option Block

Currently, all the Serial NAND devices in the market support the same commands, differences are the NAND size, page size, etc. This option block will focus on these differences, the detailed block definitions are shown below

Table 8-9. FlexSPI NAND Configuration Option Block

Offset	Field	Description							
0	option 0	**TAG [31:28]**	**Option size [27:24]**	**Reserved [23:20]**	**Flash size [19:16]**	**Has multiplanes [15:12]**	**Pages Per Block [11:8]**	**Page Size (Kbytes) [7:4]**	**Max Freq [3:0]**

Table continues on the next page...

Table 8-9. FlexSPI NAND Configuration Option Block (continued)

<table>
<tr><th>Offset</th><th>Field</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>0x0c</td><td>Size in bytes = (Option Size + 1) * 4</td><td>0</td><td>0 - 512Mb
1 - 1Gb
2 - 2Gb
4 - 4Gb</td><td>0 - 1 plane
1 - 2 planes</td><td>0 - 64
1 - 128
2 - 256
3 - 32</td><td>2 - 2KB
4 - 4KB</td><td>NAND Freq: Device specific</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>option 1</td><td>This field is optional, it is effective if option size in option0 is greater than 0
<table>
<tr><th>Reserved [31:8]</th><th>Manufacturer ID [7:0]</th></tr>
<tr><td>Reserved for future use</td><td>Actual Manufacturer ID provided in Serial NAND device datasheet
For example, 0x2C is the manufacture ID assigned to Micron</td></tr>
</table>
</td></tr>
</table>

8.3.5 Example usage with Flashloader

Flashloader can generate FCB and DBBT based on specified FlexSPI NAND FCB option block.

Assuming FCB parameters are:

- FCB and DBBT copies are 2
- Firmware copies are 2
- Firmware 0 starts at block 4, maximum block count is 2
- Firmware 1 starts at block 8, maximum block count is 2

Assuming Serial NAND parameters are:

- Flash size: 1Gbit
- Plane number:1
- Pages Per Block:64
- Page Size: 2KB
- Maximum Frequency: 80MHz

Based on above info, here is an example steps for generating FlexSPI NAND Configuration Option block. Write FlexSPI NAND Configuration Option Block to SRAM

```
blhost -u -- fill-memory 0x2030 0x4 0xc0010025
```

Write FlexSPI NAND FCB Option Block to SRAM

```
blhost - u -- fill-memory 0x2000 0x4 0xc2000104
blhost -u -- fill-memory 0x2004 0x4 0x2030 // nandOptionAddr = 0x2030
blhost -u -- fill-memory 0x2008 0x4 0x00040002 // blockId = 4, blockCount = 2
blhost -u -- fill-memory 0x200c 0x4 0x00080002 // blockId = 8, blockCount = 2
```

Configure Serial NAND using FCB option and NAND option

```
blhost -u -- configure-memory 0x101 0x2000
```

Erase and Program image

```
blhost -u -- flash-erase-region 0x4 0x2 0x101              // Erase 2 blocks starting from
block 4
blhost -u -- write-memory 0x4 image.bin 0x101          // Program image.bin to block 4
blhost -u -- flash-erase-region 0x8 0x2 0x101              // Erase 2 blocks starting from
block 8
blhost -u -- write-memory 0x8 image.bin 0x101          // Program image.bin to block 8
```

8.4 SD/eMMC through uSDHC

Some MCU BootROM supports booting from SD/eMMC devices, then the MCU Flashloader supports to flash the boot image into the SD/eMMC devices. This Charpter explains the usage of SD/eMMC via Flashloader.

8.4.1 SD Configuration Block

SD Card must be initialized before Flashloader access SD memory. The SD configuration block is a combination of several necessary SD configurations used by Flashloader to initialize the card.

Table 8.3.1 Lists the detailed description of each bits in the SD configuration block.

Table 8-10. SD Configuration Block Definition

Word Index	Bit Field	Name	Description
Word0	[31:28]	TAG	SD configuration block tag used to mark if the block is valid or not. 0xD: Valid block

Table continues on the next page...

Table 8-10. SD Configuration Block Definition (continued)

			Others: Invalid
	[27:26]	RSV	0x0
	[25:24]	PWR_DOWN_TIME	SD power down delay time before power up the SD card. Only valid when PWR_CYCLE_ENABLE is enable. 0: 20ms 1: 10ms 2: 5ms 3: 2.5ms
	23	PWR_POLARITY	SD power control polarity. Only valid when PWR_CYCLE_ENABLE is enable. 0: Power down when uSDHC.RST set low. 1: Power down when uSDHC.RST set high.
	[22:21]	RSV	0x0
	20	PWR_UP_TIME	SD power up delay time to wait voltage regulator output stable. Only valid when PWR_CYCLE_ENABLE is enable. 0: 5ms 1: 2.5ms
	19	PWR_CYCLE_ENABLE	Execute a power cycle before start the initialization progress.[1] 0: disable for non-UHSI timing,[2] enable for UHSI timing 1: enable
	[18:15]	RSV	0x0
	[14:12]	TIMING_INTERFACE	SD speed timing selection. 0: Normal/SDR12 1: High/SDR25 2: SDR50 3: SDR104 4: DDR50 (Not support yet) 5-7: Reserved

Table continues on the next page...

Table 8-10. SD Configuration Block Definition (continued)

	[11:9]	RSV	0x0
	8	BUS_WIDTH	SD bus width selection. 0: 1 bit, 4bit for UHSI timing 1: 4 bit
	[7:0]	RSV	0x0
Word1	[31:0]	RSV	0x0

NOTE

Flashloader toggles the uSDHC.RST pin to execute the power cycle progress. This needs board-level hardware support. If the hardware doesn't support controlling SD power, the power cycle progress cannot really reset the SD card.

NOTE

UHSI timing includes SDR50, SDR104 and DDR50.

8.4.2 Example usage with Flashloader

Here uses the SDR25 timing and 4bit bus width as an example. To make sure the SD card is reset before the initialization progress, it is suggested to enable the power cycle. Here choose the default settings of power cycle.

So, the hex of the SD configuration block is 0xD0082100.

- Write the configuration block to MCU internal RAM. `blhost -u – fill-memory 0x20000000 0x4 0xC0082100` RAM address 0x20000000 is selected as an example. User can select any RAM position which are available to use. User also can select an address locates at an XIP external memory, such as Flex SPI NOR Flash.
- Execute the initialization progress using configure-memory command. `blhost -u – configure-memory 0x120 0x20000000` 0x120 is the memory ID of eMMC card device. If the eMMC card is initialized successfully, then a "Success" will be received and SD memory is available to be accessed by Flashloader. If an error occurred, please refer the *Charpter 10 Appendix A: status and error codes* for debugging.
- After SD is initialized, user can use get property 25 command to check the SD card capacity. `blhost -u – get-property 25 0x120`

- To program the boot image, user needs to erase the SD card memory firstly and then program the image.

```
      blhost -u – flash-erase-region 0x0 0x1000 0x120
blhost -u – write-memory 0x400 C:\Image\bootImage.bin 0x120
```

  0x0 at the flash-erase-region command line and 0x400 at the write-memory command line is the byte offset of the SD memory, not the sector offset. That means 4K bytes starting from the start address of SD memory will be erased, then the boot image *C:\Image\bootImage.bin* will be written to the space starting from SD second Block.

- To check if the boot image is programmed successfully, user can read the data out.

```
blhost –u – read-memory 0x400 0x1000 0x120
```

  In most cases, user won't need to read the data out to verify if the boot image is written successfully or not, Flashloader will guarantee it.

8.4.3 eMMC Configuration Block

Similar to SD Card, eMMC also must be initialized before accessing it. The eMMC configuration block is used to tell Flashloader how to initialize the eMMC device. To use the fast boot feature offered by BootROM, eMMC also must be pre-configured. The fast boot configuration is also included in the eMMC configuration block.

Table 8.3.3 Lists the detailed description of each bits in the eMMC configuration block.

Table 8-11. eMMC Configuration Block Definition

Word Index	Bit Field	Name	Description
Word0	[31:28]	TAG	eMMC configuration block tag used to mark if the block is valid or not. 0xC: Valid block Others: Invalid
	27	RSV	0x0
	[26:24]	PARTITION_ACCESS	Select eMMC partition which the Flashloader write the image or data to 0: User data area 1: Boot partition 1 2: Boot partition 2 3: RPMB

Table continues on the next page...

Table 8-11. eMMC Configuration Block Definition (continued)

			4: General Purpose parition 1 5: General Purpose parition 2 6: General Purpose parition 3 7: General Purpose parition 4
	23	RSV	0x0
	[22:20]	BOOT_PARTITION_ENABLE	Select the boot partition used for fast boot. Only valid when BOOT_CONFIG_ENABLE is set 0: Not enabled 1: Boot partition 1 2: Boot partition 2 3-6: Reserved 7: User data area
	[19:18]	RSV	0x0
	[17:16]	BOOT_BUS_WIDTH	Select the bus width used for fast boot. 0: x1(SDR), x4(DDR) 1: x4(SDR,DDR) 2: x8(SDR,DDR) 3: Reserved
	[15:12]	TIMING_INTERFACE	Select the bus timing when Flashloader accesses eMMC memory. 0: Normal 1: HS 2: HS200(Not support yet) 3: HS400(Not support yet) 4-15: Reserved
	[11:8]	BUS_WIDTH	Select the bus width when Flashloader accesses eMMC memory. 0: x1 SDR 1: x4 SDR 2: x8 SDR 3-4: Reserved 5: x4 DDR 6: x8 DDR 7-15: Reserved

Table continues on the next page...

Table 8-11. eMMC Configuration Block Definition (continued)

	[7:6]	RSV	0x0
	[5:4]	BOOT_MODE	0: Normal 1: HS 2: DDR 3: Reserved
	3	RESET_BOOT_BUS_CONDITIONS	Configure eMMC behavior after exiting fast boot 0: Reset to x1,SDR,Normal 1: Retain boot config
	2	BOOT_ACK	Configure eMMC ACK behavior at fast boot. 0: NO ACK 1: ACK
	1	RSV	0x0
	0	BOOT_CONFIG_ENABLE	Determine if write fast boot configurations into eMMC or not.[2] 0: Boot configuration will be ignored. 1: Boot configuration will be written into device
Word1	[31:26]	RSV	0x0
	[25:24]	PWR_DOWN_TIME	eMMC power down delay time before power up the eMMC card. Only valid when PWR_CYCLE_ENABLE is enable 0: 20ms 1: 10ms 2: 5ms 3: 2.5ms
	23	PWR_POLARITY	eMMC power control polarity. Only valid when PWR_CYCLE_ENABLE is enable 0: Power down when uSDHC.RST set low. 1: Power down when uSDHC.RST set high.
	[22:21]	RSV	0x0
	20	PWR_UP_TIME	eMMC power up delay time to wait voltage regulator output stable.

Table continues on the next page...

Table 8-11. eMMC Configuration Block Definition (continued)

			Only valid when PWR_CYCLE_ENABLE is enable 0: 5ms 1: 2.5ms
	19	PWR_CYCLE_ENABLE	Execute a power cycle before start the SD initialization progress. 0: disable 1: enable
	18	1V8_ENABLE	Select if set uSDHC.VSELECT pin. 0: not set vselect pin 1: set vselect pin high
	[17:0]	RSV	0x0

NOTE

Fast boot configuration includes BOOT_PARTITION_ENABLE, BOOT_BUS_WIDTH, BOOT_MODE, RESET_BOOT_BUS_CONDITIONS and BOOT_ACK.

8.4.4 Example usage with Flashloader

Here uses the 8bit DDR mode as an example, and boot image is written to user data area. After writing the boot image, user want bootROM to boot the image via fast boot to decrease the boot time. Fast boot also uses the same mode – 8it DDR mode. ACK is enabled for fast boot.

So the hex of the eMMC configuration block is 0xC0721625, 0x00000000

- Write the configuration block to MCU internal RAM.

```
blhost -u – fill-memory 0x20000000 0x4 0xC0721625
        blhost -u – fill-memory 0x20000004 0x4 0x00000000
```

RAM address 0x20000000 is selected as an example. User can select any RAM position which are available to use. User also can select an address locates at an XIP external memory, such as Flex SPI NOR Flash.

- Execute the initialization progress using configure-memory command. `blhost -u – configure-memory 0x121 0x20000000` 0x121 is the memory ID of eMMC card device. If the eMMC card is initialized successfully, then a “Success” will be recieved. If an error occurred, please refer the *Charpter 10 Appendix A: status and error codes* for debugging.
- After step 2 , eMMC is available to access. User can use get property 25 command to check the eMMC card capacity. `blhost -u – get-property 25 0x121`
- To program the boot image, user shall erase the eMMC card memory before program the image.

```
blhost -u – flash-erase-region 0x0 0x2000 0x121
blhost -u – write-memory 0x400 C:\Image\bootImage.bin 0x121
```

  the address of eMMC memory in the command line is byte address, not sector address. That means 8K bytes starting from the start address of eMMC memory will be erased, then the boot image *C:\Image\bootImage.bin* will be written to eMMC 1st Block.
- To check if the boot image is programmed successfully, user can read the data out.

```
blhost –u – read-memory 0x200 0x2000 0x121
```

  In most cases, user won’t need to read the data out to verify if the boot image is written successfully or not, Flashloader will guarantee it when user gets a “Success” status for write-memory command.

  If user want to swith to other partitions of the eMMC device, user has to re-configure the eMMC devices two times.
- Select the Boot partition 1, bus width and speed timing are kept unchanged. Fast boot configuration is not necessary if user doesn’t want to update it.

```
blhost -u – fill-memory 0x20000000 0x4 0xC1001600
blhost -u – configure-memory 0x121 0x20000000
blhost -u – flash-erase-region 0x0 0x1000 0x121
blhost -u – write-memory 0x400 C:\Image\bootPartitionOneImage.bin 0x121
```

Chapter 9
Security Utilities

9.1 Introduction

The MCU Flashloader supports some security utilities that can generate some security related blocks easily, to enable these security utilities, be aware that the Flashloader itself must be signed first to enable the security utilities correctly.

9.2 Image Encryption and Programming

For device with BEE module, it supports two encrypted regions using two unique crypto keys. Each encrypted region can support up to 3 sub-divided FAC regions, totally, 3 FAC regions are supported by two encrypted regions. See the details of the image decryption and data structure required for image decryption in System Boot Chapter in SoC's RM, in the section, it focuses on encrypted image generation and programming using Flashloader. Be aware that Flashloader only supports image encryption and programming for the first encrypted region using OTPMK/SNVS key. Flashloader generates encrypted image based on a simplified PRDB option block, which is defined as below.

Table 9-1. PRDB option block

Offset	Field	Size(Bytes)	Description							
			Tag [31:28]	Key source [27:24]	Mode [23:20]	FAC Region Count [19:16]	Region1 Protection Mode [15:12]	Region2 Protection Mode [11:8]	Region3 Protection Mode [7:4]	LockOption [3:0]
0	Option	4	0xE	0 - OTPMK / SNVS [255:128]	1-AES CTR	1/2/3	0/1/	0/1	0/1	0 - No Lock

Table continues on the next page...

Table 9-1. PRDB option block (continued)

<table>
<tr><th>Offset</th><th>Field</th><th>Size(Bytes)</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Tag [31:28]</th><th>Key source [27:24]</th><th>Mode [23:20]</th><th>FAC Region Count [19:16]</th><th>Region1 Protection Mode [15:12]</th><th>Region2 Protection Mode [11:8]</th><th>Region3 Protection Mode [7:4]</th><th>LockOption [3:0]</th></tr>
<tr><td></td><td>1 - OTPMK / SNVS [127:0]</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>4</td><td>Fac Region info</td><td>8-24</td><td>
<table>
<tr><th>Offset</th><th>Field</th><th>Description</th></tr>
<tr><td>0</td><td>Start</td><td>Fac Region Start</td></tr>
<tr><td>4</td><td>Size</td><td>Fac Region Size</td></tr>
</table>
</td></tr>
</table>

NOTE

- Tag is fixed as 0x0E
- Key Source can be OTPMK/SNVS [255:128] or OTPMK/SNVS [127:0]
- Mode: it is recommended to use AES-CTR mode
- FAC Region Count: Maximum allowed FAC region number is 3 (shared by encrypted region 0 and encrypted region 1)
- Region n Protection mode: 0 – No protection, 1 – Debug disabled
- Lock Option: must be 0

9.2.1 Example to generate encrypted image and program to Flash

Take HyperFLASH as an example, assuming the encrypted info is:

- Key source: OTPMK/SNVS [255:128]
- FAC region Count:2
- Region Protection mode: 1

Here are the steps to create PRDB option block

Configure HyperFLASH using FlexSPI NOR Configuration Option Block

```
blhost -u -- fill-memory 0x2000 0x04 0xc0233007 (133MHz)
blhost -u -- configure-memory 0x9 0x2000
blhost -u -- fill-memory 0x3000 0x04 0xf000000f
blhost -u -- configure-memory 0x09 0x3000
```

Prepare PRDB0 info using PRDB option block

```
blhost -u -- fill-memory 0x4000 0x04 0xe0121100
blhost -u -- configure-memory 0x09 0x4000
Program HyperFLASH
blhost -u -- write-memory <addr> image.bin
```

9.3 KeyBlob Generation and Programming

9.3.1 KeyBlob

KeyBlob is a data structure that wraps the DEK for image decryption using AES-CCM algorithm. The whole KeyBlob data structure is shown below.

Table 9-2. KeyBlob Data structure

<table>
<tr><th>Field</th><th>Size(Bytes)</th><th>Description</th></tr>
<tr><td>Header</td><td>4</td><td>
<table>
<tr><th>Offset</th><th>Field</th><th>Description</th></tr>
<tr><td>0</td><td>tag</td><td>Fixed value: 0x81</td></tr>
<tr><td>1-2</td><td>len</td><td>Length of KeyBlob block, 16-bit big-endian order</td></tr>
<tr><td>3</td><td>par</td><td>KeyBlob Version, set to 0x42 or 0x43</td></tr>
</table>
</td></tr>
<tr><td>AEAD</td><td>4</td><td>
<table>
<tr><th>Offset</th><th>Field</th><th>Description</th></tr>
<tr><td>0</td><td>mode</td><td>Fixed to 0x66, CCM mode</td></tr>
<tr><td>1</td><td>alg</td><td>Fixed to 0x55, Crypto Algorithm: AES</td></tr>
<tr><td>2</td><td>mac_bytes</td><td>Fixed to 16</td></tr>
<tr><td>3</td><td>aad_bytes</td><td>Fixed to 0</td></tr>
</table>
</td></tr>
<tr><td>EBK</td><td>16 / 32</td><td>Blob key is used for DEK encryption, it is a random number generated by TRNG engine</td></tr>
</table>

Table continues on the next page...

Table 9-2. KeyBlob Data structure (continued)

Field	Size(Bytes)	Description
		Blob key is encrypted to EBK by a key derived from Security Engine such as SNVS or CAAM
EDEK	16 / 24 / 32	DEK is used for boot image encryption, it is encrypted to EDEK by the BK with AES algorithm using AES-CCM mode
MAC	16	MAC is generated during DEK encryption

9.3.2 KeyBlob Option Block

The MCU Flashloader supports KeyBlob generation and programming using a simplified option block called KeyBlob Option Block.

Table 9-3. KeyBlob Data structure

Offset	Field	Size	Description
0	option	4	<table><tr><th>Offset</th><th>Field</th><th>Description</th></tr><tr><td>31:28</td><td>Tag</td><td>Fixed to 0x0B</td></tr><tr><td>27:24</td><td>type</td><td>0 - Update, used to update the keyblob context 1 - Program - used to notify memory driver to program Keyblob to destination</td></tr><tr><td>23:20</td><td>size</td><td>keyblob_info size must equal to 3 if type = 0, ignored if type = 1</td></tr><tr><td>19:8</td><td>Reserved</td><td>-</td></tr><tr><td>7:4</td><td>dek_size</td><td>DEK size 0 - 128 bits 1 - 192 bits 2 - 256 bits Effective if type = 0, ignored if type = 1</td></tr><tr><td>3:0</td><td>image_index</td><td>Boot image index</td></tr></table>

Table continues on the next page...

Table 9-3. KeyBlob Data structure (continued)

Offset	Field	Size	Description
			<table><tr><th>Offset</th><th>Field</th><th>Description</th></tr><tr><td></td><td></td><td>For example, index for firmwareTable effective if type = 1, ignored if type = 0</td></tr></table>
1	dek_addr	4	Start address for the memory holds DEK
2	keyblob_offset	4	The relative Keyblob offset in the selected image For example, a signed image that contains IVT, encrypted application, CSF, Key blob IVT is at offset 0x400 Encrypted image is at offset 0x2000 CSF is at offset 0xA000 KeyBlob is at offset 0xB000 **NOTE:** For NAND device, keyblob_offset must be page aligned

9.3.3 Example to generate and program KeyBlob

Generate KeyBlob

// Write DEK to RAM

```
blhost -u -- write-memory 0x2100 dek.bin
```

// Construct KeyBlob option

```
blhost -u -- fill-memory 0x2080 4 0xb0300000 // tag = 0x0b, type = 0, size = 3, dek_size = 0
(128bits)
blhost -u -- fill-memory 0x2084 4 0x2100           // dek_addr = 0x2100
blhost -u -- fill-memory 02088 4 0x80000          // keybob_offset = 0x80000, keybob is
located at offset 0x80000 in application image
```

// Update KeyBlob Info

```
blhost -u -- configure-memory 0x101 0x2080 // Update KeyBlob Info (memory id: 0x101 -
FlexSPI NAND)
```

// Program KeyBlob

```
blhost -u -- fill-memory 0x2088 0xb1000000   // tag = 0x0b, type = 1, image_index = 0
blhost -u -- configure-memory 0x101 0x2080 // Generate KeyBlob and program it into offset
<keyblob_offset> in the selected Image <image_idex> memory region
```

Chapter 10
Appendix A: status and error codes

Status and error codes are grouped by component. Each component that defines errors has a group number. This expression is used to construct a status code value.

status_code = (group * 100) + code

Component group numbers are listed in this table.

Table 10-1. Component group numbers

Group	Component
0	Generic errors
1	Flash driver
4	QuadSPI driver
5	OTFAD driver
100	Bootloader
101	SB loader
102	Memory interface
103	Property store
104	CRC checker
105	Packetizer
106	Reliable update

The following table lists all of the error and status codes.

Table 10-2. Error and status codes

Name	Value	Description
kStatus_Success	0	Operation succeeded without error.
kStatus_Fail	1	Operation failed with a generic error.
kStatus_ReadOnly	2	Property cannot be changed because it is read-only.
kStatus_OutOfRange	3	Requested value is out of range.
kStatus_InvalidArgument	4	The requested command's argument is undefined.
kStatus_Timeout	5	A timeout occurred.

Table continues on the next page...

Table 10-2. Error and status codes (continued)

Name	Value	Description
kStatus_NoTransferInProgress	6	The current transfer status is idle.
kStatus_FlashSizeError	100	Not used.
kStatus_FlashAlignmentError	101	Address or length does not meet required alignment.
kStatus_FlashAddressError	102	Address or length is outside addressable memory.
kStatus_FlashAccessError	103	The FTFA_FSTAT[ACCERR] bit is set.
kStatus_FlashProtectionViolation	104	The FTFA_FSTAT[FPVIOL] bit is set.
kStatus_FlashCommandFailure	105	The FTFA_FSTAT[MGSTAT0] bit is set.
kStatus_FlashUnknownProperty	106	Unknown Flash property.
kStatus_FlashEraseKeyError	107	Error in erasing the key.
kStatus_FlashRegionOnExecuteOnly	108	The region is execute only region.
kStatus_FlashAPINotSupported	115	Unsupported Flash API is called.
kStatus_QspiFlashSizeError	400	Error in QuadSPI flash size.
kStatus_QspiFlashAlignmentError	401	Error in QuadSPI flash alignment.
kStatus_QspiFlashAddressError	402	Error in QuadSPI flash address.
kStatus_QspiFlashCommandFailure	403	QuadSPI flash command failure.
kStatus_QspiFlashUnknownProperty	404	Unknown QuadSPI flash property.
kStatus_QspiNotConfigured	405	QuadSPI not configured.
kStatus_QspiCommandNotSupported	406	QuadSPI command not supported.
kStatus_QspiCommandTimeout	407	QuadSPI command timed out.
kStatus_QspiWriteFailure	408	QuadSPI write failure.
kStatus_QspiModuleBusy	409	QuadSPI module is busy.
kStatus_OtfadSecurityViolation	500	Security violation in OTFAD module.
kStatus_OtfadLogicallyDisabled	501	OTFAD module is logically disabled.
kStatus_OtfadInvalidKey	502	The key is invalid.
kStatus_OtfadInvalidKeyBlob	503	The Key blob is invalid.
kStatus_SDMMC_NotSupportYet	1800	Not supported this feature.
kStatus_SDMMC_TransferFailed	1801	Failed to communicate with the device.
kStatus_SDMMC_SetCardBlockSizeFailed	1802	Failed to set the block size.
kStatus_SDMMC_HostNotSupport	1803	Host doesn't support this feature.
kStatus_SDMMC_CardNotSupport	1804	The card does not support this feature.
kStatus_SDMMC_AllSendCidFailed	1805	Failed to send CID.
kStatus_SDMMC_SendRelativeAddressFailed	1806	Failed to send relative address.
kStatus_SDMMC_SendCsdFailed	1807	Failed to send CSD.
kStatus_SDMMC_SelectCardFailed	1808	Failed to select card.
kStatus_SDMMC_SendScrFailed	1809	Failed to send SCR.
kStatus_SDMMC_SetDataBusWidthFailed	1810	Failed to set bus width.
kStatus_SDMMC_GoIdleFailed	1811	Go idle failed.

Table continues on the next page...

Table 10-2. Error and status codes (continued)

Name	Value	Description
kStatus_SDMMC_HandShakeOperationConditionFailed	1812	Failed to send operation condition.
kStatus_SDMMC_SendApplicationCommandFailed	1813	Failed to send application command.
kStatus_SDMMC_SwitchFailed	1814	Switch command failed.
kStatus_SDMMC_StopTransmissionFailed	1815	Stop transmission failed.
kStatus_SDMMC_WaitWriteCompleteFailed	1816	Failed to wait write complete.
kStatus_SDMMC_SetBlockCountFailed	1817	Failed to set block count.
kStatus_SDMMC_SetRelativeAddressFailed	1818	Failed to set relative address.
kStatus_SDMMC_SwitchBusTimingFailed	1819	Failed to switch high speed.
kStatus_SDMMC_SendExtendedCsdFailed	1820	Failed to send EXT_CSD.
kStatus_SDMMC_ConfigureBootFailed	1821	Failed to configure boot.
kStatus_SDMMC_ConfigureExtendedCsdFailed	1822	Failed to configure EXT_CSD.
kStatus_SDMMC_EnableHighCapacityEraseFailed	1823	Failed to enable high capacity erase.
kStatus_SDMMC_SendTestPatternFailed	1824	Failed to send test pattern.
kStatus_SDMMC_ReceiveTestPatternFailed	1825	Failed to receive test pattern.
kStatus_SDMMC_InvalidVoltage	1829	Invalid voltage.
kStatus_SDMMC_TuningFail	1833	Tuning failed.
kStatus_SDMMC_SwitchVoltageFail	1834	Failed to switch voltage.
kStatus_SDMMC_SetPowerClassFail	1837	Set power class fail.
kStatus_UnknownCommand	10000	The requested command value is undefined.
kStatus_SecurityViolation	10001	Command is disallowed because flash security is enabled.
kStatus_AbortDataPhase	10002	Abort the data phase early.
kStatus_Ping	10003	Internal: Received ping during command phase.
kStatus_NoResponse	10004	There is no response for the command.
kStatus_NoResponseExpected	10005	There is no response expected for the command.
kStatusRomLdrSectionOverrun	10100	ROM SB loader section overrun.
kStatusRomLdrSignature	10101	ROM SB loader incorrect signature.
kStatusRomLdrSectionLength	10102	ROM SB loader incorrect section length.
kStatusRomLdrUnencryptedOnly	10103	ROM SB loader does not support plain text image.
kStatusRomLdrEOFReached	10104	ROM SB loader EOF reached
kStatusRomLdrChecksum	10105	ROM SB loader checksum error.
kStatusRomLdrCrc32Error	10106	ROM SB loader CRC32 error.

Table continues on the next page...

Table 10-2. Error and status codes (continued)

Name	Value	Description
kStatusRomLdrUnknownCommand	10107	ROM SB loader unknown command.
kStatusRomLdrIdNotFound	10108	ROM SB loader ID not found.
kStatusRomLdrDataUnderrun	10109	ROM SB loader data underrun.
kStatusRomLdrJumpReturned	10110	ROM SB loader return from jump command occurred.
kStatusRomLdrCallFailed	10111	ROM SB loader call command failed.
kStatusRomLdrKeyNotFound	10112	ROM SB loader key not found.
kStatusRomLdrSecureOnly	10113	ROM SB loader security state is secured only.
kStatusRomLdrResetReturned	10114	ROM SB loader return from reset occurred.
kStatusMemoryRangeInvalid	10200	Memory range conflicts with a protected region.
kStatusMemoryReadFailed	10201	Failed to read from memory range.
kStatusMemoryWriteFailed	10202	Failed to write to memory range.
StatusMemoryCumulativeWrite	10203	Failed to write to unerased memory range.
kStatusMemoryAppOverlapWithExecuteOnlyRegion	10204	Memory range contains a protected executed only region.
kStatusMemoryNotConfigured	10205	Failed to access to un-configured external memory.
kStatusMemoryAlignmentError	10206	Address alignment Error.
kStatusMemoryVerifyFailed	10207	Failed to verify the write operation.
kStatusMemoryWriteProtected	10208	Memory range contains protected memory region.
kStatus_UnknownProperty	10300	The requested property value is undefined.
kStatus_ReadOnlyProperty	10301	The requested property value cannot be written.
kStatus_InvalidPropertyValue	10302	The specified property value is invalid.
kStatus_AppCrcCheckPassed	10400	CRC check passed.
kStatus_AppCrcCheckFailed	10401	CRC check failed.
kStatus_AppCrcCheckInactive	10402	CRC checker is not enabled.
kStatus_AppCrcCheckInvalid	10403	Invalid CRC checker due to blank part of BCA not present.
kStatus_AppCrcCheckOutOfRange	10404	CRC check is valid but addresses are out of range.
kStatus_NoPingResponse	10500	Packetizer did not receive any response for the ping packet.
kStatus_InvalidPacketType	10501	Packet type is invalid.
kStatus_InvalidCRC	10502	Invalid CRC in the packet.
kStatus_NoCommandResponse	10503	No response received for the command.
kStatus_ReliableUpdateSuccess	10600	Reliable update process completed successfully.
kStatus_ReliableUpdateFail	10601	Reliable update process failed.
kStatus_ReliableUpdateInacive	10602	Reliable update feature is inactive.
kStatus_ReliableUpdateBackupApplicationInvalid	10603	Backup application image is invalid.
kStatus_ReliableUpdateStillInMainApplication	10604	Next boot will still be with Main Application image.
kStatus_ReliableUpdateSwapSystemNotReady	10605	Cannot swap flash by default because swap system is not ready.

Table continues on the next page...

Table 10-2. Error and status codes (continued)

Name	Value	Description
kStatus_ReliableUpdateBackupBootloaderNotReady	10606	Cannot swap flash because there is no valid backup bootloader image.
kStatus_ReliableUpdateSwapIndicatorAddressInvalid	10607	Cannot swap flash because provided swap indicator is invalid.

Chapter 11
Appendix B: GetProperty and SetProperty commands

Properties are the defined units of data that can be accessed with the GetProperty or SetProperty commands. Properties may be read-only or read-write. All read-write properties are 32-bit integers, so they can easily be carried in a command parameter. Not all properties are available on all platforms. If a property is not available, GetProperty and SetProperty return kStatus_UnknownProperty.

The tag values shown in the table below are used with the GetProperty and SetProperty commands to query information about the bootloader.

Table 11-1. Tag values GetProperty and SetProperty

Name	Writable	Tag value	Size	Description
CurrentVersion	no	0x01	4	The current bootloader version.
AvailablePeripherals	no	0x02	4	The set of peripherals supported on this chip.
FlashStartAddress	no	0x03	4	Start address of program flash.
FlashSizeInBytes	no	0x04	4	Size in bytes of program flash.
FlashSectorSize	no	0x05	4	The size in bytes of one sector of program flash. This is the minimum erase size.
FlashBlockCount	no	0x06	4	Number of blocks in the flash array.
AvailableCommands	no	0x07	4	The set of commands supported by the bootloader.
CRCCheckStatus	no	0x08	4	The status of the application CRC check.
Reserved	n/a	0x09	n/a	

Table continues on the next page...

Table 11-1. Tag values GetProperty and SetProperty (continued)

Name	Writable	Tag value	Size	Description
VerifyWrites	yes	0x0a	4	Controls whether the bootloader verifies writes to flash. The VerifyWrites feature is enabled by default. 0 - No verification is done 1 - Enable verification
MaxPacketSize	no	0x0b	4	Maximum supported packet size for the currently active peripheral interface.
ReservedRegions	no	0x0c	n	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, then the Response packet parameter count indicates number of pairs. • If HasDataPhase flag is set, then the second parameter is the number of bytes in the data phase.
RAMStartAddress	no	0x0e	4	Start address of RAM.
RAMSizeInBytes	no	0x0f	4	Size in bytes of RAM.
SystemDeviceId	no	0x10	4	Value of the Kinetis System Device Identification register.
FlashSecurityState	no	0x11	4	Indicates whether Flash security is enabled. 0 - Flash security is disabled 1 - Flash security is enabled
UniqueDeviceId	no	0x12	n	Unique device identification, value of Kinetis Unique Identification registers

Table continues on the next page...

Table 11-1. Tag values GetProperty and SetProperty (continued)

Name	Writable	Tag value	Size	Description
				(16 for K series devices, 12 for KL series devices)
FlashFacSupport	no	0x13	4	FAC (Flash Access Control) support flag 0 - FAC not supported 1 - FAC supported
FlashAccessSegmentSize	no	0x14	4	The size in bytes of 1 segment of flash.
FlashAccessSegmentCount	no	0x15	4	FAC segment count (The count of flash access segments within the flash model.)
FlashReadMargin	yes	0x16	4	The margin level setting for flash erase and program verify commands. 0=Normal 1=User 2=Factory
QspiInitStatus	no	0x17	4	The result of the QSPI or OTFAD initialization process. 405 - QSPI is not initialized 0 - QSPI is initialized
TargetVersion	no	0x18	4	Target build version number.
ExternalMemoryAttributes	no	0x19	24	List of attributes supported by the specified memory Id (0=Internal Flash, 1=QuadSpi0). See description for the return value in the section ExternalMemoryAttributes Property.
ReliableUpdateStatus	-	0x1a	4	Result of last Reliable Update operation. See Table 12-2.

Chapter 12
Revision history

12.1 Revision History

This table shows the revision history of the document.

Table 12-1. Revision history

Revision number	Date	Substantive changes
0	04/2016	Kinetis Bootloader v2.0.0 release
1	10/2017	Update for Flashloader application for i.MX RT Series of devices
2	01/2018	Update for Flashloader application for QuadSPI NOR Flash device that is only JESD216 compliant

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number function_description
Revision 2, 01/2018