PIC32CM MC00 Curiosity Nano MDFU Client Solution API Documentation

Table of Contents

1. Specific MISRA C:2012 Deviations

Global FRAME_CHECK_SIZE

Advisory: misra-c2012- 2.5

Justification: This is a false positive.

2. Module Documentation

2.1. 32-Bit MDFU Client Library

Core firmware update APIs for supporting device firmware updates using an MDFU ecosystem and MDFU Protocol.

2.1.1. Module description

Core firmware update APIs for supporting device firmware updates using an MDFU ecosystem and MDFU Protocol.

2.1.1.1. Data structures

- struct bl_unlock_boot_metadata_t
  Structure containing metadata required to unlock the bootloader.
- struct bl_command_header_t
  Operational data orientation for each operation.
- struct bl_block_header_t
  Header data orientation for each block.

2.1.1.2. Definitions

- #define BL_IMAGE_FORMAT_MAJOR_VERSION (0x1)
  Represents the major version of the image format that is understood by the bootloader core.
  .
- #define BL_IMAGE_FORMAT_MINOR_VERSION (0x0)
  Represents the minor version of the image format that is understood by the bootloader core.
  .
- #define BL_IMAGE_FORMAT_PATCH_VERSION (0x0)
  Represents the patch version of the image format that is understood by the bootloader core.
  .
- #define BL_VECTORED_INTERRUPTS_ENABLED (0)
  Indicates that the bootloader supports vectored interrupts in the application.
- #define BL_APPLICATION_START_ADDRESS (0x1000U)
  Start of the application memory space.
- #define BL_DEVICE_ID_START_ADDRESS_U (0x41002018U)
  Device ID address.
- #define BL_APPLICATION_END_ADDRESS (0x1FFFF)
  End of the application memory space.
- #define BL_IMAGE_PARTITION_SIZE (0x1F000)
  Defined size of the application memory space.
- #define BL_STAGING_IMAGE_START (BL_APPLICATION_START_ADDRESS)
  Start of the application download space.
- #define BL_STAGING_IMAGE_END (BL_APPLICATION_END_ADDRESS)
  End of the application download space.
- #define BL_STAGING_IMAGE_ID (0U)
  Image area ID that identifies the download location of the transferred data.

- #define BL_APPLICATION_IMAGE_COUNT (1U)
  Number to represent how many image spaces are configured by the bootloader.
- #define BL_SOFTWARE_ENTRY_PATTERN_START (0x20000000)
  Start address of the software entry pattern array.
- #define BL_SOFTWARE_ENTRY_PATTERN (0x5048434DU)
  32-bit pattern used to indicate that a software entry has been requested.
- #define ASM_VECTOR asm("bx %0"::"r" (reset_vector))
  Macro defined to jump the program to the application reset location.
- #define BL_COMMAND_HEADER_SIZE (4U)
- #define BL_BLOCK_HEADER_SIZE (3U)
  Total size of the basic block header part.
- #define BL_WRITE_BYTE_LENGTH (NVMCTRL_FLASH_PAGESIZE)
  Maximum number of bytes that the bootloader can hold inside of its process buffer.
- #define BL_MAX_BUFFER_SIZE (BL_BLOCK_HEADER_SIZE + BL_COMMAND_HEADER_SIZE + BL_WRITE_BYTE_LENGTH)
  Maximum length of data in bytes that the bootloader can receive from the host in each operational block.

2.1.1.3. Enumerations

- enum bl_block_type_t { UNLOCK_BOOTLOADER = 0x01U, WRITE_FLASH = 0x02U }
  Contains codes corresponding to the various types of data blocks that the bootloader supports.
- enum bl_result_t { BL_PASS = 0x81U, BL_BUSY = 0x3CU, BL_FAIL = 0xC3U, BL_ERROR_COMMUNICATION_FAIL = 0x18U, BL_ERROR_FRAME_VALIDATION_FAIL = 0xFFU, BL_ERROR_BUFFER_OVERLOAD = 0xBDU, BL_ERROR_INVALID_ARGUMENTS = 0xE7U, BL_ERROR_UNKNOWN_COMMAND = 0x42U, BL_ERROR_ADDRESS_OUT_OF_RANGE = 0x24U, BL_ERROR_COMMAND_PROCESSING = 0x7EU, BL_ERROR_VERIFICATION_FAIL = 0xDBU, BL_ERROR_BUFFER_UNDERLOAD = 0xAAU, BL_ERROR_ROLLBACK_FAILURE = 0xF0 }
  Enumeration of bootloader API return codes.

2.1.1.4. Functions

- static void CRC32_Calculate (uint32_t startAddress, uint32_t length, uint32_t *crc)
  Calculates the CRC32 checksum for a specified memory region.
- static bl_result_t CRC32_Validate (uint32_t startAddress, uint32_t length, uint32_t crcAddress)
  Validates the CRC32 checksum for a specified memory region.
- bl_result_t BL_ImageVerify (void)
  Performs a verification sequence on the staging area image memory space.
- static bl_result_t BootloaderProcessorUnlock (uint8_t *bufferPtr)
  Unlocks the bootloader processor using the provided buffer.
- static void DownloadAreaErase (uint32_t startAddress)
  Erases the entire area used to download the image data.
- bl_result_t BL_Initialize (void)
  Performs the initialization steps required to configure the bootloader peripherals.
- bl_result_t BL_BootCommandProcess (uint8_t *commandBuffer, uint16_t commandLength)
  Executes the required action based on the block type received in the bootloader data buffer.
- void BL_ApplicationStart (void)
  Performs actions to jump the MCU program counter to the application start address.

- bool BL_CheckForcedEntry (void)
  Checks the software entry flags for a forced entry into Boot mode.

2.1.1.5. Variables

- static uint32_t writeBuffer [BL_WRITE_BYTE_LENGTH/4U]
  Buffer used for write operations.
- static bool bootloaderCoreUnlocked = false
  Flag for indicating if the meta data has been validated in the update process.

2.1.2. Definition Documentation

2.1.2.1. ASM_VECTOR

#define ASM_VECTOR asm("bx "r" (reset_vector))

Macro defined to jump the program to the application reset location.

2.1.2.2. BL_APPLICATION_END_ADDRESS

#define BL_APPLICATION_END_ADDRESS (0x1FFFF)

End of the application memory space.

2.1.2.3. BL_APPLICATION_IMAGE_COUNT

#define BL_APPLICATION_IMAGE_COUNT (1U)

Number to represent how many image spaces are configured by the bootloader.

2.1.2.4. BL_APPLICATION_START_ADDRESS

#define BL_APPLICATION_START_ADDRESS (0x1000U)

Start of the application memory space.

2.1.2.5. BL_BLOCK_HEADER_SIZE

#define BL_BLOCK_HEADER_SIZE (3U)

Total size of the basic block header part.

2.1.2.6. BL_COMMAND_HEADER_SIZE

#define BL_COMMAND_HEADER_SIZE (4U)

Total size of the operational block header part.

2.1.2.7. BL_DEVICE_ID_START_ADDRESS_U

#define BL_DEVICE_ID_START_ADDRESS_U (0x41002018U)

Device ID address.

2.1.2.8. BL_IMAGE_FORMAT_MAJOR_VERSION

#define BL_IMAGE_FORMAT_MAJOR_VERSION (0x1)

Represents the major version of the image format that is understood by the bootloader core.

.

2.1.2.9. BL_IMAGE_FORMAT_MINOR_VERSION

#define BL_IMAGE_FORMAT_MINOR_VERSION (0x0)

Represents the minor version of the image format that is understood by the bootloader core.

.

2.1.2.10. BL_IMAGE_FORMAT_PATCH_VERSION

#define BL_IMAGE_FORMAT_PATCH_VERSION (0x0)

Represents the patch version of the image format that is understood by the bootloader core.

.

2.1.2.11. BL_IMAGE_PARTITION_SIZE

#define BL_IMAGE_PARTITION_SIZE (0x1F000)

Defined size of the application memory space.

2.1.2.12. BL_MAX_BUFFER_SIZE

#define BL_MAX_BUFFER_SIZE (BL_BLOCK_HEADER_SIZE + BL_COMMAND_HEADER_SIZE + BL_WRITE_BYTE_LENGTH)

Maximum length of data in bytes that the bootloader can receive from the host in each operational block.

2.1.2.13. BL_SOFTWARE_ENTRY_PATTERN

#define BL_SOFTWARE_ENTRY_PATTERN (0x5048434DU)

32-bit pattern used to indicate that a software entry has been requested.

2.1.2.14. BL_SOFTWARE_ENTRY_PATTERN_START

#define BL_SOFTWARE_ENTRY_PATTERN_START (0x20000000)

Start address of the software entry pattern array.

2.1.2.15. BL_STAGING_IMAGE_END

#define BL_STAGING_IMAGE_END (BL_APPLICATION_END_ADDRESS)

End of the application download space.

2.1.2.16. BL_STAGING_IMAGE_ID

#define BL_STAGING_IMAGE_ID (0U)

Image area ID that identifies the download location of the transferred data.

2.1.2.17. BL_STAGING_IMAGE_START

#define BL_STAGING_IMAGE_START (BL_APPLICATION_START_ADDRESS)

Start of the application download space.

2.1.2.18. BL_VECTORED_INTERRUPTS_ENABLED

#define BL_VECTORED_INTERRUPTS_ENABLED (0)

Indicates that the bootloader supports vectored interrupts in the application.

Note:
Not needed by all architectures.

2.1.2.19. BL_WRITE_BYTE_LENGTH

#define BL_WRITE_BYTE_LENGTH (NVMCTRL_FLASH_PAGESIZE)

Maximum number of bytes that the bootloader can hold inside of its process buffer.

2.1.3. Function Documentation

2.1.3.1. BL_ApplicationStart()

void BL_ApplicationStart (void)

Performs actions to jump the MCU program counter to the application start address.

2.1.3.2. BL_BootCommandProcess()

bl_result_t BL_BootCommandProcess (uint8_t * commandBuffer, uint16_t commandLength)

Executes the required action based on the block type received in the bootloader data buffer.

Parameters:

in	commandBuffer	- Pointer to the start of the bootloader operational data
in	commandLength	- Length of the new data received by the FTP

Returns:

BL_PASS - Process cycle finished successfully
BL_FAIL - Process cycle failed unexpectedly
BL_ERROR_UNKNOWN_COMMAND - Process cycle encountered an unknown command
BL_ERROR_VERIFICATION_FAIL - Process cycle failed to verify the application image
BL_ERROR_COMMAND_PROCESSING - Process cycle failed due to a data or processing related issue
BL_ERROR_ADDRESS_OUT_OF_RANGE - Process cycle failed due to an incorrect address

2.1.3.3. BL_CheckForcedEntry()

bool BL_CheckForcedEntry (void)

Checks the software entry flags for a forced entry into Boot mode.

Returns:

True - The first four addresses of RAM contain the BL_SOFTWARE_ENTRY_PATTERN
False - The first four addresses of RAM do not contain the BL_SOFTWARE_ENTRY_PATTERN

Note:
This function can be updated to check any forced entry mechanism. For example, utilizing a switch to enter the bootloader at start-up.

2.1.3.4. BL_ImageVerify()

bl_result_t BL_ImageVerify (void)

Performs a verification sequence on the staging area image memory space.

Parameters:

None

Returns:

BL_PASS - Bootloader verified the application image with no errors
BL_ERROR_VERIFICATION_FAIL - Bootloader image verification failed
BL_ERROR_COMMAND_PROCESSING - Bootloader image verification failed due to incorrect processing data
BL_ERROR_ADDRESS_OUT_OF_RANGE - Bootloader image verification failed due to incorrect addresses
BL_ERROR_ROLLBACK_FAILURE - Bootloader image verification failed due to anti-rollback feature. This is only used when multiple image spaces are present and version roll-back protection is enabled.

2.1.3.5. BL_Initialize()

bl_result_t BL_Initialize (void)

Performs the initialization steps required to configure the bootloader peripherals.

Parameters:

None

Returns:

BL_PASS - Bootloader initialization was successful
BL_ERROR_COMMAND_PROCESSING - Bootloader initialization has failed

2.1.3.6. BootloaderProcessorUnlock()

static bl_result_t BootloaderProcessorUnlock (uint8_t * bufferPtr)[static]

Unlocks the bootloader processor using the provided buffer.

This function attempts to unlock the bootloader processor by processing the meta data found at the data pointer.

Parameters:

in	bufferPtr	Pointer to a buffer containing meta data

Returns:

BL_PASS - Bootloader metadata block has been validated and the core memory functions can now be used
BL_ERROR_VERIFICATION_FAIL - Invalid data was found in the metadata block and core memory functions remain disabled
BL_FAIL - Metadata validation failed unexpectedly

Verify the file format major version. The core must use the exact major version of the file format.

- If the file has a lower major version then there are likely missing data elements that are required by the running version of the core.

If the file has a larger major version then the data elements in the new file format have likely shifted around and may not function as intended, so it is more stable to reject it, in this case.

Note: We must always increase the major version of the file anytime the metadata block changes or a new block is added to the file definition, that is a requirement of the core firmware in order to perform an update.

2.1.3.7. CRC32_Calculate()

static void CRC32_Calculate (uint32_t startAddress, uint32_t length, uint32_t * crc)[static]

Calculates the CRC32 checksum for a specified memory region.

This function computes the CRC32 checksum for given range of memory starting at the given address and spanning the specified length. The result is stored in the provided CRC seed pointer. This function utilized the DSU peripheral.

Parameters:

in	startAddress	- The starting address of the memory block to calculate the CRC for
in	length	- The length of the memory block in bytes
in,out	crc	- Pointer to a variable where the calculated CRC32 checksum will be stored. This variable must be passed to the function with the CRC seed value set at the pointer.

Returns:

None

2.1.3.8. CRC32_Validate()

static bl_result_t CRC32_Validate (uint32_t startAddress, uint32_t length, uint32_t crcAddress)[static]

Validates the CRC32 checksum for a specified memory region.

This function checks the CRC32 checksum of a memory block against a stored CRC value to verify data integrity and then returns a value indicating whether the validation was successful or not.

Parameters:

in	startAddress	- The starting address of the memory block to validate
in	length	- The length of the memory block in bytes
in	crcAddress	- The address where the expected CRC32 checksum is stored

Returns:

BL_PASS - Bootloader verified the application image with no errors
BL_FAIL - Bootloader encountered an error and failed unexpectedly
BL_ERROR_VERIFICATION_FAIL - Bootloader image verification failed

2.1.3.9. DownloadAreaErase()

static void DownloadAreaErase (uint32_t startAddress)[static]

Erases the entire area used to download the image data.

This function will erase the entire image area if there is only one image. In build configurations where there is a staging area this function will erase the staging area only.

Parameters:

in	startAddress	- Start address of the area used to download the image data

Returns:

None

2.1.4. Enumeration Type Documentation

2.1.4.1. bl_block_type_t

enum bl_block_type_t

Contains codes corresponding to the various types of data blocks that the bootloader supports.

UNLOCK_BOOTLOADER	0x01U - Unlock Bootloader Block - Identifies an operational block that holds precondition metadata to be checked and validated before any memory-changing actions occur in the bootloader
WRITE_FLASH	0x02U - Flash Data Block - Identifies operational blocks that need to be written into the Flash section of memory

2.1.4.2. bl_result_t

enum bl_result_t

Enumeration of bootloader API return codes.

This enumeration defines the various return codes used by the bootloader APIs. Each code represents a specific status or error condition that can be returned by the bootloader and FTP functions.

BL_PASS	(0b10000001) (dec 129) Operation completed successfully
BL_BUSY	(0b00111100) (dec 60) Bootloader is busy processing another request
BL_FAIL	(0b11000011) (dec 195) Operation failed
BL_ERROR_COMMUNICATION_FAIL	(0b00011000) (dec 24) Communication failure occurred
BL_ERROR_FRAME_VALIDATION_FAIL	(0b11111111) (dec 255) Frame validation failed
BL_ERROR_BUFFER_OVERLOAD	(0b10111101) (dec 190) Buffer overload detected
BL_ERROR_INVALID_ARGUMENTS	(0b11100111) (dec 231) Invalid arguments provided
BL_ERROR_UNKNOWN_COMMAND	(0b01000010) (dec 66) Unknown command received
BL_ERROR_ADDRESS_OUT_OF_RANGE	(0b00100100) (dec 36) Address out of range
BL_ERROR_COMMAND_PROCESSING	(0b01111110) (dec 126) Error occurred during command processing
BL_ERROR_VERIFICATION_FAIL	(0b11011011) (dec 219) Verification failed
BL_ERROR_BUFFER_UNDERLOAD	(0b10101010) (dec 170) Buffer underload detected
BL_ERROR_ROLLBACK_FAILURE	(0b11110000) (dec 240) Version validation failure

2.1.5. Variable Documentation

2.1.5.1. bootloaderCoreUnlocked

bool bootloaderCoreUnlocked = false[static]

Flag for indicating if the meta data has been validated in the update process.

2.1.5.2. writeBuffer

uint32_t writeBuffer[BL_WRITE_BYTE_LENGTH/4U][static]

Buffer used for write operations.

This static buffer is allocated to hold data for write operations. The size of the buffer is determined by BL_WRITE_BYTE_LENGTH divided by 4, to accommodate 32-bit (uint32_t) data elements.

2.2. File Transfer Protocol (FTP) Client Handler

File Transfer Protocol (FTP) Client Handler. This version of the FTP Handler supports version 1.0.0 of the MDFU Protocol (https://onlinedocs.microchip.com/oxy/GUID-58904FDA-338A-488F-A88D-766D29B27E37-en-US-1/)

2.2.1. Module description

File Transfer Protocol (FTP) Client Handler. This version of the FTP Handler supports version 1.0.0 of the MDFU Protocol (https://onlinedocs.microchip.com/oxy/GUID-58904FDA-338A-488F-A88D-766D29B27E37-en-US-1/)

2.2.1.1. Data structures

- struct ftp_parser_helper_t

  A structure to help manage the reception of commands, sending responses, and frame validation logic of the FTP Handler.
- struct ftp_tlv_t

  A structure to help manage the Type-Length-Value (TLV) data payloads used during the Get Client Info stage.

2.2.1.2. Definitions

- #define COMMAND_DATA_SIZE (1U)

  Length of the command data field in bytes.
- #define SEQUENCE_DATA_SIZE (1U)

  Length of the sequence data field in bytes.
- #define MAX_RESPONSE_SIZE (25U)

  Length of the largest possible response in bytes.
- #define TLV_HEADER_SIZE (2U)

  Length of a Type-Length-Value object header in bytes.
- #define MAX_TRANSFER_SIZE (BL_MAX_BUFFER_SIZE + SEQUENCE_DATA_SIZE + COMMAND_DATA_SIZE + COM_FRAME_BYTE_COUNT)

  Length of the largest possible data transfer in bytes.
- #define MIN_TRANSFER_SIZE (2U)

  Length of the smallest possible transfer in bytes.
- #define PACKET_BUFFER_COUNT (1U)

  Number of buffers supported for reception.
- #define RETRY_TRANSFER_bm (0x40U)

  Mask of the Retry bit.

- #define SYNC_TRANSFER_bm (0x80U)
  Mask of the Sync bit.
- #define SEQUENCE_NUMBER_bm (0x3FU)
  Mask of the sequence number field.
- #define MAX_SEQUENCE_VALUE (31U)
  Maximum value of the sequence field.
- #define FTP_BYTE_INDEX (1U)
  Index of the status or command byte in the receive buffer.
- #define SEQUENCE_BYTE_INDEX (0U)
  Index of the sequence byte in the receive buffer.
- #define FILE_DATA_INDEX (COMMAND_DATA_SIZE + SEQUENCE_DATA_SIZE)
  Index of the start of the file transfer data in the receive buffer.

2.2.1.3. Enumerations

- enum ftp_command_t { FTP_GET_CLIENT_INFO = 0x01U, FTP_START_TRANSFER = 0x02U, FTP_WRITE_CHUNK = 0x03U, FTP_GET_IMAGE_STATE = 0x04U, FTP_END_TRANSFER = 0x05U }
  Enumeration of file transfer command codes defined by the MDFU Protocol.
- enum ftp_response_status_t { FTP_COMMAND_SUCCESS = 0x01U, FTP_COMMAND_NOT_SUPPORTED = 0x02U, FTP_COMMAND_NOT_AUTHORIZED = 0x03U, FTP_COMMAND_NOT_EXECUTED = 0x04U, FTP_ABORT_TRANSFER = 0x05U }
  Enumeration of file transfer status codes defined by the MDFU Protocol.
- enum ftp_abort_code_t { FTP_GENERIC_ERROR = 0x00U, FTP_INVALID_FILE_ERROR = 0x01U, FTP_INVALID_DEVICE_ID_ERROR = 0x02U, FTP_ADDRESS_ERROR = 0x03U, FTP_ERASE_ERROR = 0x04U, FTP_WRITE_ERROR = 0x05U, FTP_READ_ERROR = 0x06U, FTP_APP_VERSION_ERROR = 0x07U }
  Enumeration of response codes used to communicate the client abort cause defined by the MDFU Protocol.
- enum ftp_transport_failure_code_t { FTP_INTEGRITY_CHECK_ERROR = 0x00U, FTP_COMMAND_TOO_LONG_ERROR = 0x01U, FTP_COMMAND_TOO_SHORT_ERROR = 0x02U, FTP_INVALID_SEQUENCE_NUMBER_ERROR = 0x03U }
  Enumeration of response codes used to communicate the client transport failure cause defined by the MDFU Protocol.
- enum ftp_image_state_t { FTP_IMAGE_VALID = 0x01U, FTP_IMAGE_INVALID = 0x02U }
  Enumeration of get image state response codes.
- enum tlv_type_code_t { FTP_PROTOCOL_VERSION = 0x01U, FTP_TRANSFER_PARAMETERS = 0x02U, FTP_TIMEOUT_INFO = 0x03U }
  Enumeration of discovery data type codes.

2.2.1.4. Functions

- static void DeviceResetCheck (void)
  Checks and performs a reset when required.
- static void ParserDataReset (void)
  Resets parser data use for command reception.
- static bool SequenceNumberValidate (void)
  Validates the sequence number of the incoming command.
- static void ClientInfoResponseSet (void)
  Sets the Get Client Info data in the response buffer.

- static bl_result_t OperationalBlockExecute (void)
  Processes and executes the FTP command data received by the host.
- static void ResponseSet (uint8_t *buffer, uint8_t *responsePayload, ftp_response_status_t responseStatus, uint8_t sequenceByte, uint16_t responsePayloadLength)
  Sets the response in the FTP process frame.
- static uint8_t TLVAppend (uint8_t *dataBufferStart, ftp_tlv_t *tlvData)
  Appends a TLV (Type-Length-Value) structure to a data buffer.
- static ftp_abort_code_t AbortCodeGet (bl_result_t targetStatus)
  Converts the given result codes into MDFU Protocol defined data values.
- bl_result_t FTP_Task (void)
  Acts as the main task runner of the FTP process. This function will be called in a loop to receive commands from the host and make calls to the responsible software layers to facilitate the device firmware update.
- bl_result_t FTP_Initialize (void)
  Performs the initialization actions required to set up the FTP and dependent layers.

2.2.1.5. Variables

- static uint8_t FTP_RECEIVE_BUFFER [MAX_TRANSFER_SIZE]
  Buffer for receiving FTP data.
- static uint8_t FTP_RESPONSE_BUFFER [MAX_RESPONSE_SIZE]
  Buffer for storing FTP response data.
- static uint8_t FTP_RETRY_BUFFER [MAX_RESPONSE_SIZE]
  Buffer for retrying FTP responses.
- static ftp_parser_helper_t ftpHelper
  Structure manages the FTP parser data.

2.2.2. Definition Documentation

2.2.2.1. COMMAND_DATA_SIZE

#define COMMAND_DATA_SIZE (1U)

Length of the command data field in bytes.

2.2.2.2. FILE_DATA_INDEX

#define FILE_DATA_INDEX (COMMAND_DATA_SIZE + SEQUENCE_DATA_SIZE)

Index of the start of the file transfer data in the receive buffer.

2.2.2.3. FTP_BYTE_INDEX

#define FTP_BYTE_INDEX (1U)

Index of the status or command byte in the receive buffer.

Note:
This index is valid for both the command byte of the receive buffer and the status byte of the response buffer.

2.2.2.4. MAX_RESPONSE_SIZE

#define MAX_RESPONSE_SIZE (25U)

Length of the largest possible response in bytes.

2.2.2.5. MAX_SEQUENCE_VALUE

#define MAX_SEQUENCE_VALUE (31U)

Maximum value of the sequence field.

2.2.2.6. MAX_TRANSFER_SIZE

#define MAX_TRANSFER_SIZE (BL_MAX_BUFFER_SIZE + SEQUENCE_DATA_SIZE + COMMAND_DATA_SIZE + COM_FRAME_BYTE_COUNT)

Length of the largest possible data transfer in bytes.

2.2.2.7. MIN_TRANSFER_SIZE

#define MIN_TRANSFER_SIZE (2U)

Length of the smallest possible transfer in bytes.

2.2.2.8. PACKET_BUFFER_COUNT

#define PACKET_BUFFER_COUNT (1U)

Number of buffers supported for reception.

2.2.2.9. RETRY_TRANSFER_bm

#define RETRY_TRANSFER_bm (0x40U)

Mask of the Retry bit.

2.2.2.10. SEQUENCE_BYTE_INDEX

#define SEQUENCE_BYTE_INDEX (0U)

Index of the sequence byte in the receive buffer.

2.2.2.11. SEQUENCE_DATA_SIZE

#define SEQUENCE_DATA_SIZE (1U)

Length of the sequence data field in bytes.

2.2.2.12. SEQUENCE_NUMBER_bm

#define SEQUENCE_NUMBER_bm (0x3FU)

Mask of the sequence number field.

2.2.2.13. SYNC_TRANSFER_bm

#define SYNC_TRANSFER_bm (0x80U)

Mask of the Sync bit.

2.2.2.14. TLV_HEADER_SIZE

#define TLV_HEADER_SIZE (2U)

Length of a Type-Length-Value object header in bytes.

2.2.3. Function Documentation

2.2.3.1. AbortCodeGet()

static ftp_abort_code_t AbortCodeGet (bl_result_t targetStatus)[static]

Converts the given result codes into MDFU Protocol defined data values.

This function converts the bootloader core result codes into codes that are defined by the MDFU Protocol.

Parameters:

in	targetStatus	- Bootloader result code that needs to be mapped to one of the defined protocol codes

Returns:

FTP_INVALID_FILE_ERROR - The bootloader failed during file verification, either during the metadata block validation or a failed image verification
FTP_ADDRESS_ERROR - The bootloader processed a block with an invalid address
FTP_WRITE_ERROR - The bootloader encountered an error while trying to process the data. Likely be due to NVM errors
FTP_GENERIC_ERROR - The bootloader code received has not been mapped to a specific FTP abort code

2.2.3.2. ClientInfoResponseSet()

static void ClientInfoResponseSet (void)[static]

Sets the Get Client Info data in the response buffer.

This function defines and sets the response data to the Get Client Info command. The Get Client Info command data payload used in this function is defined by the MDFU Protocol.

Parameters:

None

Returns:

None

This solution is utilizing the FTP UART implementation defined in the MDFU Protocol version 1.0.0

2.2.3.3. DeviceResetCheck()

static void DeviceResetCheck (void)[static]

Checks and performs a reset when required.

This function checks the static reset flag and performs the reset operation. This controls the reset logic after the update has completed.

Parameters:

None

Returns:

None

2.2.3.4. FTP_Initialize()

bl_result_t FTP_Initialize (void)

Performs the initialization actions required to set up the FTP and dependent layers.

Parameters:

None.

Returns:

BL_PASS - FTP initialization finished successfully
BL_FAIL - FTP initialization failed unexpectedly

2.2.3.5. FTP_Task()

bl_result_t FTP_Task (void)

Acts as the main task runner of the FTP process. This function will be called in a loop to receive commands from the host and make calls to the responsible software layers to facilitate the device firmware update.

Parameters:

None.

Returns:

BL_PASS - FTP process cycle finished successfully
BL_FAIL - FTP process cycle failed unexpectedly
BL_ERROR_COMMUNICATION_FAIL - FTP process cycle failed to communicate with the host
BL_ERROR_FRAME_VALIDATION_FAIL - FTP process cycle failed at the frame check stage
BL_ERROR_BUFFER_OVERLOAD - FTP process cycle failed due to a communication buffer overflow
BL_ERROR_UNKNOWN_COMMAND - FTP process cycle encountered an unknown command from the host
BL_ERROR_VERIFICATION_FAIL - FTP process cycle failed because the core's image verification process failed
BL_ERROR_COMMAND_PROCESSING - FTP process cycle failed due to a core data or processing related issue
BL_ERROR_ADDRESS_OUT_OF_RANGE - FTP process cycle failed due to the core encountering an incorrect address
BL_ERROR_BUFFER_UNDERLOAD - FTP process cycle failed due the FTP command being too short

2.2.3.6. OperationalBlockExecute()

static bl_result_t OperationalBlockExecute (void)[static]

Processes and executes the FTP command data received by the host.

This function processes the FTP receive buffer and calls the required operational layer or performs the steps needed to execute the FTP command. The Get Client Info Command data payload used in this function is defined by the MDFU Protocol.

Parameters:

None

Returns:

BL_PASS - FTP process cycle finished successfully
BL_ERROR_UNKNOWN_COMMAND - FTP process failed due to an unknown command code or unknown file data block type
BL_ERROR_VERIFICATION_FAIL - FTP process failed due to a data verification failed. Could be returned when an image failure occurs or when a meta-data validation fails
BL_ERROR_ADDRESS_OUT_OF_RANGE - FTP process failed due to an address error
BL_ERROR_COMMAND_PROCESSING - FTP process failed due to a general memory process error

2.2.3.7. ParserDataReset()

static void ParserDataReset (void)[static]

Resets parser data use for command reception.

This function resets all flags, buffers, and counters used when receiving FTP commands.

Parameters:

None

Returns:

None

2.2.3.8. ResponseSet()

static void ResponseSet (uint8_t * buffer, uint8_t * responsePayload, ftp_response_status_t responseStatus, uint8_t sequenceByte, uint16_t responsePayloadLength)[static]

Sets the response in the FTP process frame.

This function configures the response for a given FTP buffer by setting the response payload, status, sequence byte, and response length.

Parameters:

in,out	buffer	- Pointer to the FTP response buffer where the frame must be set
in	responsePayload	- Pointer to the response payload data
in	responseStatus	- Status of the FTP response
in	sequenceByte	- Sequence byte for the response
in	responsePayloadLength	- Length of the response payload

Returns:

None

2.2.3.9. SequenceNumberValidate()

static bool SequenceNumberValidate (void)[static]

Validates the sequence number of the incoming command.

This function checks the validity of the sequence number based on past operations and the next expected number.

Parameters:

None

Returns:

Returns true if the sequence number is valid, false otherwise

2.2.3.10. TLVAppend()

static uint8_t TLVAppend (uint8_t * dataBufferStart, ftp_tlv_t * tlvData)[static]

Appends a TLV (Type-Length-Value) structure to a data buffer.

This function appends a given TLV structure to the specified data buffer. It updates the buffer with the TLV data, ensuring that the data is correctly formatted and aligned within the response buffer.

Parameters:

in,out	dataBufferStart	- Pointer to the start of the data buffer where the TLV will be appended
in	tlvData	- Pointer to the TLV structure containing the data to append

Returns:

The number of bytes appended to the buffer

Note:
Ensure that the data buffer has sufficient space to accommodate the TLV data.

2.2.4. Enumeration Type Documentation

2.2.4.1. ftp_abort_code_t

enum ftp_abort_code_t

Enumeration of response codes used to communicate the client abort cause defined by the MDFU Protocol.

FTP_GENERIC_ERROR
FTP_INVALID_FILE_ERROR
FTP_INVALID_DEVICE_ID_ERROR
FTP_ADDRESS_ERROR

FTP_ERASE_ERROR
FTP_WRITE_ERROR
FTP_READ_ERROR
FTP_APP_VERSION_ERROR

2.2.4.2. ftp_command_t

enum ftp_command_t

Enumeration of file transfer command codes defined by the MDFU Protocol.

FTP_GET_CLIENT_INFO
FTP_START_TRANSFER
FTP_WRITE_CHUNK
FTP_GET_IMAGE_STATE
FTP_END_TRANSFER

2.2.4.3. ftp_image_state_t

enum ftp_image_state_t

Enumeration of get image state response codes.

FTP_IMAGE_VALID
FTP_IMAGE_INVALID

2.2.4.4. ftp_response_status_t

enum ftp_response_status_t

Enumeration of file transfer status codes defined by the MDFU Protocol.

FTP_COMMAND_SUCCESS
FTP_COMMAND_NOT_SUPPORTED
FTP_COMMAND_NOT_AUTHORIZED
FTP_COMMAND_NOT_EXECUTED
FTP_ABORT_TRANSFER

2.2.4.5. ftp_transport_failure_code_t

enum ftp_transport_failure_code_t

Enumeration of response codes used to communicate the client transport failure cause defined by the MDFU Protocol.

FTP_INTEGRITY_CHECK_ERROR
FTP_COMMAND_TOO_LONG_ERROR
FTP_COMMAND_TOO_SHORT_ERROR
FTP_INVALID_SEQUENCE_NUMBER_ERROR

2.2.4.6. tlv_type_code_t

enum tlv_type_code_t

Enumeration of discovery data type codes.

FTP_PROTOCOL_VERSION
FTP_TRANSFER_PARAMETERS
FTP_TIMEOUT_INFO

2.2.5. Variable Documentation

2.2.5.1. FTP_RECEIVE_BUFFER

uint8_t FTP_RECEIVE_BUFFER[MAX_TRANSFER_SIZE][static]

Buffer for receiving FTP data.

This buffer is used to store incoming FTP data packets. The size of the buffer is defined by MAX_TRANSFER_SIZE which is based on the bootloader's write size.

2.2.5.2. FTP_RESPONSE_BUFFER

uint8_t FTP_RESPONSE_BUFFER[MAX_RESPONSE_SIZE][static]

Buffer for storing FTP response data.

This buffer holds the data that will be sent as a response frame to each FTP command. The size of the buffer is defined by MAX_RESPONSE_SIZE.

2.2.5.3. FTP_RETRY_BUFFER

uint8_t FTP_RETRY_BUFFER[MAX_RESPONSE_SIZE][static]

Buffer for retrying FTP responses.

This buffer is used to store FTP response data that needs to be resent in case of transmission failures. The size of the buffer is defined by MAX_RESPONSE_SIZE.

2.2.5.4. ftpHelper

ftp_parser_helper_t ftpHelper[static]

Initial value:

```
= {
    .lastSequenceNumber = 0U,
    .currentSequenceNumber = 0U,
    .nextSequenceNumber = 1U,
    .resendRequired = false,
    .responseRequired = false,
    .isComBusy = false,
    .resetPending = false,
    .ftpReceiveCount = 0U,
    .ftpResponseLength = 0U
}
```

Structure manages the FTP parser data.

2.3. Communication Adapter

This layer implements the custom transport layer that is defined by the MDFU Protocol.

2.3.1. Module description

This layer implements the custom transport layer that is defined by the MDFU Protocol.

2.3.1.1. Definitions

- #define START_OF_PACKET_BYTE (0x56U)
  Special character for identifying the start of the frame.
- #define END_OF_PACKET_BYTE (0x9EU)
  Special character for identifying the end of the frame.
- #define ESCAPE_BYTE (0xCCU)
  Special character for identifying an escaped byte in the command data.
- #define FRAME_CHECK_SIZE (2U)
  Length of the frame check field in bytes.

2.3.1.2. Enumerations

- enum com_adapter_result_t { COM_PASS = 0xE7U, COM_FAIL = 0xC3U, COM_INVALID_ARG = 0x96U, COM_BUFFER_ERROR = 0x69U, COM_BUSY = 0x18U, COM_TRANSPORT_FAILURE = 0x3CU, COM_SEND_COMPLETE = 0x7EU }

  Contains codes for the return values of the bootloader communication adapter layer APIs.

2.3.1.3. Functions

- static com_adapter_result_t DataSend (uint8_t data)

  Abstracted UART write function for sending a single byte.
- static uint16_t FrameCheckCalculate (uint8_t *ftpData, uint16_t bufferLength)

  Calculate the frame check on the given data buffer.
- com_adapter_result_t COM_FrameTransfer (uint8_t *receiveBufferPtr, uint16_t *receiveIndexPtr)

  Receive or send byte over SERCOM.
- com_adapter_result_t COM_FrameSet (uint8_t *responseBufferPtr, uint16_t responseLength)

  Copy and format bytes from the given buffer into the static send buffer using the defined framing format.
- com_adapter_result_t COM_Initialize (uint16_t maximumBufferLength)

  Performs initialization actions for the communication peripheral and adapter code.

2.3.2. Definition Documentation

2.3.2.1. END_OF_PACKET_BYTE

#define END_OF_PACKET_BYTE (0x9EU)

Special character for identifying the end of the frame.

2.3.2.2. ESCAPE_BYTE

#define ESCAPE_BYTE (0xCCU)

Special character for identifying an escaped byte in the command data.

Note:
For more information about the framing operations used for UART, refer to the MDFU protocol documentation for version 1.0.0.

2.3.2.3. FRAME_CHECK_SIZE

#define FRAME_CHECK_SIZE (2U)

Length of the frame check field in bytes.

Length of the frame bytes needed.

MISRA C:2012 Deviation **Advisory:** misra-c2012- 2.5
Justification: This is a false positive.

This is the length of bytes that must be defined in the FTP Handler buffer in order to properly implement the current transport layer.

2.3.2.4. START_OF_PACKET_BYTE

#define START_OF_PACKET_BYTE (0x56U)

Special character for identifying the start of the frame.

2.3.3. Function Documentation

2.3.3.1. COM_FrameSet()

com_adapter_result_t COM_FrameSet (uint8_t * responseBufferPtr, uint16_t responseLength)

Copy and format bytes from the given buffer into the static send buffer using the defined framing format.

Note:
For UART, this function will simply send the bytes out of the peripheral because the communication layer does not need to wait for the host to initiate any transfer. Doing this makes it so we do not need to define a static buffer in the communication code.

Parameters:

in	responseBufferPtr	- Pointer to the buffer that needs to be sent
in	responseLength	- Length of the response that needs to be sent

Returns:

COM_PASS - Buffer was transferred without error

COM_FAIL - An error occurred in SERCOM while transferring the buffer

2.3.3.2. COM_FrameTransfer()

com_adapter_result_t COM_FrameTransfer (uint8_t * receiveBufferPtr, uint16_t * receiveIndexPtr)

Receive or send byte over SERCOM.

When receiving this function will push data bytes into the buffer provided until a complete frame is received. When sending this function uses the static send buffer defined in communication adapter file to send out bytes until it is complete.

Note:
For UART, this function does not send data out because it is asynchronous, but for other host driven protocols this function controls the transfer in both directions.

Parameters:

in,out	receiveBufferPtr	- Pointer to the buffer provided to SERCOM
in,out	receiveIndexPtr	- Pointer to the number of bytes successfully received by SERCOM

Returns:

COM_PASS - SERCOM has received a complete frame and is ready for further processing

COM_BUSY - SERCOM still loading the buffer

COM_BUFFER_ERROR - SERCOM received too many or encountered a data error

COM_FAIL - An error occurred in SERCOM

2.3.3.3. COM_Initialize()

com_adapter_result_t COM_Initialize (uint16_t maximumBufferLength)

Performs initialization actions for the communication peripheral and adapter code.

Note:
This function takes the maximum buffer length of the FTP handler so that it knows when to signal an overflow to the FTP code.

Parameters:

in	maximumBufferLength	- Maximum length that the COM adapter is allowed to read

Returns:

COM_PASS - Specified arguments are valid and initialization was successful
COM_INVALID_ARG - Specified arguments were invalid
COM_FAIL - Error occurred in the SERCOM initialization

2.3.3.4. DataSend()

static com_adapter_result_t DataSend (uint8_t data)[static]

Abstracted UART write function for sending a single byte.

Parameters:

in	data	- Data byte to be transferred over UART

Returns:

COM_PASS - Data transfer did not encounter any errors
COM_FAIL - Data transfer encounter an errors. The peripheral returned an error after attempting to transmit data

2.3.3.5. FrameCheckCalculate()

static uint16_t FrameCheckCalculate (uint8_t * ftpData, uint16_t bufferLength)[static]

Calculate the frame check on the given data buffer.

Note:
For more information on the frame check used by the FTP refer to the MDFU Protocol document version 1.0.0.

Parameters:

in	ftpData	Data buffer to be used for the frame check calculation
in	bufferLength	Length of data objects in the buffer

Returns:

Calculated frame check

2.3.4. Enumeration Type Documentation

2.3.4.1. com_adapter_result_t

enum com_adapter_result_t

Contains codes for the return values of the bootloader communication adapter layer APIs.

COM_PASS	0xE7U - com_adapter transfer has full completed as is ready for processing
COM_FAIL	0xC3U - com_adapter transfer has failed
COM_INVALID_ARG	0x96U - com_adapter transfer has an invalid argument
COM_BUFFER_ERROR	0x69U - com_adapter transfer has encountered an overflow
COM_BUSY	0x18U - com_adapter transfer is not finished yet
COM_TRANSPORT_FAILURE	0x3CU - com_adapter transfer has encountered a transport error
COM_SEND_COMPLETE	0x7EU - com_adapter transfer is done sending

3. Data Structure Documentation

3.1. bl_block_header_t Struct Reference

Header data orientation for each block.

3.1.1. Detailed Description

Header data orientation for each block.

#include <bl_core.h>

3.1.1.1. Data Fields

- uint16_t blockLength
- bl_block_type_t blockType

3.1.2. Field Documentation

The documentation for this struct was generated from the following file:

source/

bl_core.h

3.1.2.1. blockLength

bl_block_header_t::blockLength

Member 'blockLength' contains the total length of data bytes in the block

3.1.2.2. blockType

bl_block_header_t::blockType

Member 'blockType' contains the code that corresponds to the type of data inside the payload buffer

3.2. bl_command_header_t Struct Reference

Operational data orientation for each operation.

3.2.1. Detailed Description

Operational data orientation for each operation.

#include <bl_core.h>

3.2.1.1. Data Fields

- uint32_t startAddress

3.2.2. Field Documentation

The documentation for this struct was generated from the following file:

source/

bl_core.h

3.2.2.1. startAddress

bl_command_header_t::startAddress

Member 'startAddress' contains the start address of the data payload.

3.3. bl_unlock_boot_metadata_t Struct Reference

Structure containing metadata required to unlock the bootloader.

3.3.1. Detailed Description

Structure containing metadata required to unlock the bootloader.

This structure holds information about the bootloader image version, device identification, payload size, and associated command header. This data is processed as the first block of file data and will not allow memory operations to occur in the core until this data has been validated.

3.3.1.1. Data Fields

- bl_block_header_t blockHeader
- uint8_t imageVersionPatch
- uint8_t imageVersionMinor
- uint8_t imageVersionMajor
- uint32_t deviceId
- uint16_t maxPayloadSize
- bl_command_header_t commandHeader

3.3.2. Field Documentation

The documentation for this struct was generated from the following file:

source/

bl_core.c

3.3.2.1. blockHeader

bl_unlock_boot_metadata_t::blockHeader

Block header information for the bootloader metadata.

3.3.2.2. commandHeader

bl_unlock_boot_metadata_t::commandHeader

Command header information for the bootloader write commands.

3.3.2.3. deviceId

bl_unlock_boot_metadata_t::deviceId

Unique identifier for the target device.

3.3.2.4. imageVersionMajor

bl_unlock_boot_metadata_t::imageVersionMajor

Major version of the bootloader image.

3.3.2.5. imageVersionMinor

bl_unlock_boot_metadata_t::imageVersionMinor

Minor version of the bootloader image.

3.3.2.6. imageVersionPatch

bl_unlock_boot_metadata_t::imageVersionPatch

Patch version of the bootloader image.

3.3.2.7. maxPayloadSize

bl_unlock_boot_metadata_t::maxPayloadSize

Maximum allowed payload size for bootloader write operations.

3.4. ftp_parser_helper_t Struct Reference

A structure to help manage the reception of commands, sending responses, and frame validation logic of the FTP Handler.

3.4.1. Detailed Description

A structure to help manage the reception of commands, sending responses, and frame validation logic of the FTP Handler.

This structure is used to manage indexes and track the various sequence numbers involved in FTP command/response logic, maintain the flags indicating whether a response or resend is required as well as maintain any additional data flags needed.

3.4.1.1. Data Fields

- uint8_t lastSequenceNumber
- uint8_t currentSequenceNumber
- uint8_t nextSequenceNumber
- bool responseRequired
- bool resendRequired
- bool resetPending
- bool isComBusy
- uint16_t ftpReceiveCount
- uint16_t ftpResponseLength

3.4.2. Field Documentation

The documentation for this struct was generated from the following file:

source/

bl_ftp.c

3.4.2.1. currentSequenceNumber

uint8_t currentSequenceNumber

The current sequence number being processed

3.4.2.2. ftpReceiveCount

uint16_t ftpReceiveCount

Counter to indicate the amount of file data bytes in the receive buffer

3.4.2.3. ftpResponseLength

uint16_t ftpResponseLength

The length of data ready to be transferred back to the host

3.4.2.4. isComBusy

bool isComBusy

Flag indicating if the communication is busy an needs additional cycles

3.4.2.5. lastSequenceNumber

uint8_t lastSequenceNumber

The last sequence number processed

3.4.2.6. nextSequenceNumber

uint8_t nextSequenceNumber

The next expected sequence number

3.4.2.7. resendRequired

bool resendRequired

Flag indicating if a resend is required

3.4.2.8. resetPending

bool resetPending

Flag indicating if a reset is required

3.4.2.9. responseRequired

bool responseRequired

Flag indicating if a response is required

3.5. ftp_tlv_t Struct Reference

A structure to help manage the Type-Length-Value (TLV) data payloads used during the Get Client Info stage.

3.5.1. Detailed Description

A structure to help manage the Type-Length-Value (TLV) data payloads used during the Get Client Info stage.

This structure is used to manage the definition of the TLV data in the Get Client Info response. This type is also used as an input to a function that helps append new data to the response buffer in a more dynamic way.

3.5.1.1. Data Fields

- uint8_t dataType
- uint8_t dataLength
- uint8_t * valueBuffer

3.5.2. Field Documentation

The documentation for this struct was generated from the following file:

source/

bl_ftp.c

3.5.2.1. dataLength

uint8_t dataLength

The length of data held in the data buffer

3.5.2.2. dataType

uint8_t dataType

The type of data held in the data buffer

3.5.2.3. valueBuffer

uint8_t* valueBuffer

The data buffer to be transferred to the host

4. File Documentation

4.1. source/bl_app_verify.c File Reference

Contains APIs to support verification of the application image space.

```
#include <stdint.h>
#include <stdbool.h>
#include "bl_app_verify.h"
#include "bl_config.h"
#include "../../../peripheral/nvmctrl/plib_nvmctrl.h"
#include "../../../peripheral/dsu/plib_dsu.h"
#include "../../../peripheral/pac/plib_pac.h"
```

4.1.1. Functions

- static void CRC32_Calculate (uint32_t startAddress, uint32_t length, uint32_t *crc)
  Calculates the CRC32 checksum for a specified memory region.
- static bl_result_t CRC32_Validate (uint32_t startAddress, uint32_t length, uint32_t crcAddress)
  Validates the CRC32 checksum for a specified memory region.
- bl_result_t BL_ImageVerify (void)
  Performs a verification sequence on the staging area image memory space.

4.1.2. Detailed Description

Contains APIs to support verification of the application image space.

4.2. source/bl_app_verify.h File Reference

Contains API prototypes to perform application image verification.

```
#include "bl_result_type.h"
#include "bl_config.h"
```

4.2.1. Functions

- bl_result_t BL_ImageVerify (void)
  Performs a verification sequence on the staging area image memory space.

4.2.2. Detailed Description

Contains API prototypes to perform application image verification.

4.3. source/bl_config.h File Reference

Contains macros and type definitions related to the bootloader client device configuration and bootloader settings.

4.3.1. Macros

- #define BL_IMAGE_FORMAT_MAJOR_VERSION (0x1)
  Represents the major version of the image format that is understood by the bootloader core.
  .
- #define BL_IMAGE_FORMAT_MINOR_VERSION (0x0)
  Represents the minor version of the image format that is understood by the bootloader core.
  .
- #define BL_IMAGE_FORMAT_PATCH_VERSION (0x0)
  Represents the patch version of the image format that is understood by the bootloader core.
  .
- #define BL_VECTORED_INTERRUPTS_ENABLED (0)
  Indicates that the bootloader supports vectored interrupts in the application.
- #define BL_APPLICATION_START_ADDRESS (0x1000U)
  Start of the application memory space.
- #define BL_DEVICE_ID_START_ADDRESS_U (0x41002018U)
  Device ID address.
- #define BL_APPLICATION_END_ADDRESS (0x1FFFF)
  End of the application memory space.
- #define BL_IMAGE_PARTITION_SIZE (0x1F000)
  Defined size of the application memory space.
- #define BL_STAGING_IMAGE_START (BL_APPLICATION_START_ADDRESS)
  Start of the application download space.
- #define BL_STAGING_IMAGE_END (BL_APPLICATION_END_ADDRESS)
  End of the application download space.
- #define BL_STAGING_IMAGE_ID (0U)
  Image area ID that identifies the download location of the transferred data.
- #define BL_APPLICATION_IMAGE_COUNT (1U)

Number to represent how many image spaces are configured by the bootloader.

- #define BL_SOFTWARE_ENTRY_PATTERN_START (0x20000000)
  Start address of the software entry pattern array.
- #define BL_SOFTWARE_ENTRY_PATTERN (0x5048434DU)
  32-bit pattern used to indicate that a software entry has been requested.
- #define BL_HASH_DATA_SIZE (4U)
- #define ASM_VECTOR asm("bx %0"::"r" (reset_vector))
  Macro defined to jump the program to the application reset location.

4.3.2. Detailed Description

Contains macros and type definitions related to the bootloader client device configuration and bootloader settings.

4.3.3. Macro Definition Documentation

4.3.3.1. BL_HASH_DATA_SIZE

#define BL_HASH_DATA_SIZE (4U)

4.4. source/bl_core.c File Reference

Contains APIs to support file transfer-based bootloader operations, including an FTP module and all bootloader core firmware.

```
#include "bl_core.h"
#include "bl_config.h"
#include "ftp/bl_ftp.h"
```

4.4.1. Data structures

- struct bl_unlock_boot_metadata_t
  Structure containing metadata required to unlock the bootloader.

4.4.2. Functions

- static bl_result_t BootloaderProcessorUnlock (uint8_t *bufferPtr)
  Unlocks the bootloader processor using the provided buffer.
- static void DownloadAreaErase (uint32_t startAddress)
  Erases the entire area used to download the image data.

- bl_result_t BL_BootCommandProcess (uint8_t *bootDataPtr, uint16_t commandLength)
  Executes the required action based on the block type received in the bootloader data buffer.
- void BL_ApplicationStart (void)
  Performs actions to jump the MCU program counter to the application start address.
- bl_result_t BL_Initialize (void)
  Performs the initialization steps required to configure the bootloader peripherals.
- bool BL_CheckForcedEntry (void)
  Checks the software entry flags for a forced entry into Boot mode.

4.4.3. Variables

- static uint32_t writeBuffer [BL_WRITE_BYTE_LENGTH/4U]
  Buffer used for write operations.
- static bool bootloaderCoreUnlocked = false
  Flag for indicating if the meta data has been validated in the update process.

4.4.4. Detailed Description

Contains APIs to support file transfer-based bootloader operations, including an FTP module and all bootloader core firmware.

See also:

File Transfer Protocol (FTP) Client Handler

4.5. source/bl_core.h File Reference

Contains API prototypes to perform bootloader operations.

```
#include <stdint.h>
#include <stdbool.h>
#include "bl_result_type.h"
#include "bl_config.h"
#include "../../../peripheral/port/plib_port.h"
#include "../../../peripheral/nvmctrl/plib_nvmctrl.h"
```

4.5.1. Data structures

- struct bl_command_header_t
  Operational data orientation for each operation.
- struct bl_block_header_t

Header data orientation for each block.

4.5.2. Functions

- bl_result_t BL_Initialize (void)

  Performs the initialization steps required to configure the bootloader peripherals.
- bl_result_t BL_BootCommandProcess (uint8_t *commandBuffer, uint16_t commandLength)

  Executes the required action based on the block type received in the bootloader data buffer.
- void BL_ApplicationStart (void)

  Performs actions to jump the MCU program counter to the application start address.
- bool BL_CheckForcedEntry (void)

  Checks the software entry flags for a forced entry into Boot mode.

4.5.3. Macros

- #define BL_COMMAND_HEADER_SIZE (4U)
- #define BL_BLOCK_HEADER_SIZE (3U)

  Total size of the basic block header part.
- #define BL_WRITE_BYTE_LENGTH (NVMCTRL_FLASH_PAGESIZE)

  Maximum number of bytes that the bootloader can hold inside of its process buffer.
- #define BL_MAX_BUFFER_SIZE (BL_BLOCK_HEADER_SIZE + BL_COMMAND_HEADER_SIZE + BL_WRITE_BYTE_LENGTH)

  Maximum length of data in bytes that the bootloader can receive from the host in each operational block.

4.5.4. Enumerations

- enum bl_block_type_t { UNLOCK_BOOTLOADER = 0x01U, WRITE_FLASH = 0x02U }

  Contains codes corresponding to the various types of data blocks that the bootloader supports.

4.5.5. Detailed Description

Contains API prototypes to perform bootloader operations.

4.6. source/bl_ftp.c File Reference

This file is the implementation file of the file transfer protocol layer.

```
#include "bl_ftp.h"
#include "../bl_core.h"
```

```
#include "../../com_adapter/com_adapter.h"
#include "../bl_app_verify.h"
```

4.6.1. Data structures

- struct ftp_parser_helper_t

  A structure to help manage the reception of commands, sending responses, and frame validation logic of the FTP Handler.
- struct ftp_tlv_t

  A structure to help manage the Type-Length-Value (TLV) data payloads used during the Get Client Info stage.

4.6.2. Functions

- static void DeviceResetCheck (void)

  Checks and performs a reset when required.
- static void ParserDataReset (void)

  Resets parser data use for command reception.
- static bool SequenceNumberValidate (void)

  Validates the sequence number of the incoming command.
- static void ClientInfoResponseSet (void)

  Sets the Get Client Info data in the response buffer.
- static bl_result_t OperationalBlockExecute (void)

  Processes and executes the FTP command data received by the host.
- static void ResponseSet (uint8_t *buffer, uint8_t *responsePayload, ftp_response_status_t responseStatus, uint8_t sequenceByte, uint16_t responsePayloadLength)

  Sets the response in the FTP process frame.
- static uint8_t TLVAppend (uint8_t *dataBufferStart, ftp_tlv_t *tlvData)

  Appends a TLV (Type-Length-Value) structure to a data buffer.
- static ftp_abort_code_t AbortCodeGet (bl_result_t targetStatus)

  Converts the given result codes into MDFU Protocol defined data values.
- bl_result_t FTP_Task (void)

  Acts as the main task runner of the FTP process. This function will be called in a loop to receive commands from the host and make calls to the responsible software layers to facilitate the device firmware update.
- bl_result_t FTP_Initialize (void)

  Performs the initialization actions required to set up the FTP and dependent layers.

4.6.3. Macros

- #define COMMAND_DATA_SIZE (1U)

  Length of the command data field in bytes.
- #define SEQUENCE_DATA_SIZE (1U)

  Length of the sequence data field in bytes.
- #define MAX_RESPONSE_SIZE (25U)

  Length of the largest possible response in bytes.
- #define TLV_HEADER_SIZE (2U)

  Length of a Type-Length-Value object header in bytes.

- #define MAX_TRANSFER_SIZE (BL_MAX_BUFFER_SIZE + SEQUENCE_DATA_SIZE + COMMAND_DATA_SIZE + COM_FRAME_BYTE_COUNT)
  Length of the largest possible data transfer in bytes.
- #define MIN_TRANSFER_SIZE (2U)
  Length of the smallest possible transfer in bytes.
- #define PACKET_BUFFER_COUNT (1U)
  Number of buffers supported for reception.
- #define RETRY_TRANSFER_bm (0x40U)
  Mask of the Retry bit.
- #define SYNC_TRANSFER_bm (0x80U)
  Mask of the Sync bit.
- #define SEQUENCE_NUMBER_bm (0x3FU)
  Mask of the sequence number field.
- #define MAX_SEQUENCE_VALUE (31U)
  Maximum value of the sequence field.
- #define FTP_BYTE_INDEX (1U)
  Index of the status or command byte in the receive buffer.
- #define SEQUENCE_BYTE_INDEX (0U)
  Index of the sequence byte in the receive buffer.
- #define FILE_DATA_INDEX (COMMAND_DATA_SIZE + SEQUENCE_DATA_SIZE)
  Index of the start of the file transfer data in the receive buffer.

4.6.4. Enumerations

- enum ftp_command_t { FTP_GET_CLIENT_INFO = 0x01U, FTP_START_TRANSFER = 0x02U, FTP_WRITE_CHUNK = 0x03U, FTP_GET_IMAGE_STATE = 0x04U, FTP_END_TRANSFER = 0x05U }
  Enumeration of file transfer command codes defined by the MDFU Protocol.
- enum ftp_response_status_t { FTP_COMMAND_SUCCESS = 0x01U, FTP_COMMAND_NOT_SUPPORTED = 0x02U, FTP_COMMAND_NOT_AUTHORIZED = 0x03U, FTP_COMMAND_NOT_EXECUTED = 0x04U, FTP_ABORT_TRANSFER = 0x05U }
  Enumeration of file transfer status codes defined by the MDFU Protocol.
- enum ftp_abort_code_t { FTP_GENERIC_ERROR = 0x00U, FTP_INVALID_FILE_ERROR = 0x01U, FTP_INVALID_DEVICE_ID_ERROR = 0x02U, FTP_ADDRESS_ERROR = 0x03U, FTP_ERASE_ERROR = 0x04U, FTP_WRITE_ERROR = 0x05U, FTP_READ_ERROR = 0x06U, FTP_APP_VERSION_ERROR = 0x07U }
  Enumeration of response codes used to communicate the client abort cause defined by the MDFU Protocol.
- enum ftp_transport_failure_code_t { FTP_INTEGRITY_CHECK_ERROR = 0x00U, FTP_COMMAND_TOO_LONG_ERROR = 0x01U, FTP_COMMAND_TOO_SHORT_ERROR = 0x02U, FTP_INVALID_SEQUENCE_NUMBER_ERROR = 0x03U }
  Enumeration of response codes used to communicate the client transport failure cause defined by the MDFU Protocol.
- enum ftp_image_state_t { FTP_IMAGE_VALID = 0x01U, FTP_IMAGE_INVALID = 0x02U }
  Enumeration of get image state response codes.
- enum tlv_type_code_t { FTP_PROTOCOL_VERSION = 0x01U, FTP_TRANSFER_PARAMETERS = 0x02U, FTP_TIMEOUT_INFO = 0x03U }
  Enumeration of discovery data type codes.

4.6.5. Variables

- static uint8_t FTP_RECEIVE_BUFFER [MAX_TRANSFER_SIZE]
  Buffer for receiving FTP data.
- static uint8_t FTP_RESPONSE_BUFFER [MAX_RESPONSE_SIZE]
  Buffer for storing FTP response data.
- static uint8_t FTP_RETRY_BUFFER [MAX_RESPONSE_SIZE]
  Buffer for retrying FTP responses.
- static ftp_parser_helper_t ftpHelper
  Structure manages the FTP parser data.

4.6.6. Detailed Description

This file is the implementation file of the file transfer protocol layer.

4.7. source/bl_ftp.h File Reference

Contains prototypes to transfer binary file blocks using the MDFU Protocol version 1.0.0.

```
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include "../bl_result_type.h"
```

4.7.1. Functions

- bl_result_t FTP_Task (void)
  Acts as the main task runner of the FTP process. This function will be called in a loop to receive commands from the host and make calls to the responsible software layers to facilitate the device firmware update.
- bl_result_t FTP_Initialize (void)
  Performs the initialization actions required to set up the FTP and dependent layers.

4.7.2. Detailed Description

Contains prototypes to transfer binary file blocks using the MDFU Protocol version 1.0.0.

4.8. source/bl_result_type.h File Reference

Contains an enumeration of all the bootloader status codes used by the core.

4.8.1. Enumerations

- enum bl_result_t { BL_PASS = 0x81U, BL_BUSY = 0x3CU, BL_FAIL = 0xC3U, BL_ERROR_COMMUNICATION_FAIL = 0x18U, BL_ERROR_FRAME_VALIDATION_FAIL = 0xFFU, BL_ERROR_BUFFER_OVERLOAD = 0xBDU, BL_ERROR_INVALID_ARGUMENTS = 0xE7U, BL_ERROR_UNKNOWN_COMMAND = 0x42U, BL_ERROR_ADDRESS_OUT_OF_RANGE = 0x24U, BL_ERROR_COMMAND_PROCESSING = 0x7EU, BL_ERROR_VERIFICATION_FAIL = 0xDBU, BL_ERROR_BUFFER_UNDERLOAD = 0xAAU, BL_ERROR_ROLLBACK_FAILURE = 0xF0 }

  Enumeration of bootloader API return codes.

4.8.2. Detailed Description

Contains an enumeration of all the bootloader status codes used by the core.

4.9. source/com_adapter.c File Reference

This is the implementation file for the communication adapter layer using UART.

```
#include "com_adapter.h"
#include "peripheral/sercom/usart/plib_sercom1_usart.h"
#include <stdbool.h>
```

4.9.1. Functions

- static com_adapter_result_t DataSend (uint8_t data)

  Abstracted UART write function for sending a single byte.

- static uint16_t FrameCheckCalculate (uint8_t *ftpData, uint16_t bufferLength)
  Calculate the frame check on the given data buffer.
- com_adapter_result_t COM_FrameTransfer (uint8_t *receiveBufferPtr, uint16_t *receiveIndexPtr)
  Receive or send byte over SERCOM.
- com_adapter_result_t COM_FrameSet (uint8_t *responseBufferPtr, uint16_t responseLength)
  Copy and format bytes from the given buffer into the static send buffer using the defined framing format.
- com_adapter_result_t COM_Initialize (uint16_t maximumBufferLength)
  Performs initialization actions for the communication peripheral and adapter code.

4.9.2. Macros

- #define START_OF_PACKET_BYTE (0x56U)
  Special character for identifying the start of the frame.
- #define END_OF_PACKET_BYTE (0x9EU)
  Special character for identifying the end of the frame.
- #define ESCAPE_BYTE (0xCCU)
  Special character for identifying an escaped byte in the command data.

4.9.3. Variables

- static uint16_t MaxBufferLength = 0U
- static bool isReceiveWindowOpen = false
- static bool isEscapedByte = false

4.9.4. Detailed Description

This is the implementation file for the communication adapter layer using UART.

4.9.5. Variable Documentation

4.9.5.1. isEscapedByte

bool isEscapedByte = false[static]

4.9.5.2. isReceiveWindowOpen

bool isReceiveWindowOpen = false[static]

4.9.5.3. MaxBufferLength

uint16_t MaxBufferLength = 0U[static]

4.10. source/com_adapter.h File Reference

Contains prototypes and other data types for communication adapter layer.

```
#include <stdint.h>
```

4.10.1. Functions

- com_adapter_result_t COM_FrameTransfer (uint8_t *receiveBufferPtr, uint16_t *receiveIndexPtr)
  Receive or send byte over SERCOM.
- com_adapter_result_t COM_FrameSet (uint8_t *responseBufferPtr, uint16_t responseLength)
  Copy and format bytes from the given buffer into the static send buffer using the defined framing format.
- com_adapter_result_t COM_Initialize (uint16_t maximumBufferLength)
  Performs initialization actions for the communication peripheral and adapter code.

4.10.2. Macros

- #define FRAME_CHECK_SIZE (2U)
  Length of the frame check field in bytes.
- #define COM_FRAME_BYTE_COUNT (FRAME_CHECK_SIZE)

4.10.3. Enumerations

- enum com_adapter_result_t { COM_PASS = 0xE7U, COM_FAIL = 0xC3U, COM_INVALID_ARG = 0x96U, COM_BUFFER_ERROR = 0x69U, COM_BUSY = 0x18U, COM_TRANSPORT_FAILURE = 0x3CU, COM_SEND_COMPLETE = 0x7EU }
  Contains codes for the return values of the bootloader communication adapter layer APIs.

4.10.4. Detailed Description

Contains prototypes and other data types for communication adapter layer.

4.10.5. Macro Definition Documentation

4.10.5.1. COM_FRAME_BYTE_COUNT

#define COM_FRAME_BYTE_COUNT (FRAME_CHECK_SIZE)

Microchip Information

Trademarks

ISBN:

Legal Notice

Microchip Devices Code Protection Feature