Microchip Device Firmware Update (MDFU) Protocol Specification 1.0.0

Notice to Development Tools Customers

Important:
All documentation becomes dated, and Development Tools manuals are no exception. Our tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our website (www.microchip.com/) to obtain the latest version of the PDF document.

Documents are identified with a DS number located on the bottom of each page. The DS format is DS<DocumentNumber><Version>, where <DocumentNumber> is an 8-digit number and <Version> is an uppercase letter.

For the most up-to-date information, find help for your tool at onlinedocs.microchip.com/.

© 2024 Microchip Technology Inc. and its subsidiaries

Table of Contents

Notice to Development Tools Customers......1

1. Microchip Device Firmware Update (MDFU) Protocol Overview......4
   1.1. MDFU Protocol Characteristics and Benefits......5
   1.2. MDFU Protocol Layers......6
   1.3. Microchip Device Firmware Update Protocol Version......6
   1.4. Definition of Terms......7

2. File Transfer Based Approach to Firmware Updates......8
   2.1. MDFU File Abstraction Paradigms......8
   2.2. MDFU File Abstraction Benefits......9

3. Firmware Update Protocol Layer......10
   3.1. Firmware Update Algorithm......10
      3.1.1. Client Setup Prior to Update......11
      3.1.2. Discovery Stage......11
      3.1.3. Start Transfer Stage......12
      3.1.4. File Transfer Stage......12
      3.1.5. Verification Stage......13
      3.1.6. End Transfer Stage......13
   3.2. Command-Response Pairs......13
      3.2.1. Commands......14
      3.2.2. Responses......15
      3.2.3. Command/Response Byte Order......16
      3.2.4. Data Types......17
      3.2.5. Command-Response Pair Definitions......17
      3.2.6. Error Response Details......26
   3.3. Host Command Generation and Response Processing Algorithm......29
      3.3.1. Host Command Generation......29
      3.3.2. Host Response Processing......29
      3.3.3. Host Command Generation and Response Processing Algorithm Diagram......30
      3.3.4. Host Command and Response Buffer Sizing......33
   3.4. Client Command Processing and Response Generation......33
      3.4.1. Client Command Processing and Response Generation Algorithm......33
   3.5. Flow Control......35
      3.5.1. Stop and Wait Flow Control with Time-outs......35
      3.5.2. Flow Control Algorithm Diagram......36
   3.6. Definition of a Successful Update......38
   3.7. Error Classification and Handling......38
      3.7.1. Limitations of the MDFU Error Detection and Recovery......38
      3.7.2. Recoverable Errors......38
      3.7.3. Unrecoverable Errors......45
   3.8. Sequence Numbers......47
      3.8.1. Command Sequence Field Definition......48
      3.8.2. Response Sequence Field Definition......49
      3.8.3. Host Command Sequence Number Selection......49
      3.8.4. Client Sequence Number Processing......50

© 2024 Microchip Technology Inc. and its subsidiaries

4. Transport Layer........55
4.1. Transport Layer Responsibilities........55
4.1.1. Command Transport from Host to Client........55
4.1.2. Response Retrieval from the Client........55
4.1.3. Command and Response Integrity Checks........56
4.1.4. Host Transport Layer Response Retrieval Time-out........56
4.2. UART Command/Response Transport Layer........56
4.2.1. UART Configuration........57
4.2.2. Host Response Retrieval........57
4.2.3. UART Command/Response Framing........57
4.2.4. UART Command/Response Frame Creation........60
4.2.5. UART Command/Response Frame Transmission........60
4.2.6. UART Command/Response Frame Reception........61
4.2.7. UART Command/Response Integrity Check........66
4.2.8. Maximum UART Command/Response Frame Size........68

5. Revision History........70

Microchip Information........71
The Microchip Website........71
Product Change Notification Service........71
Customer Support........71
Product Identification System........72
Microchip Devices Code Protection Feature........72
Legal Notice........72
Trademarks........73
Quality Management System........74
Worldwide Sales and Service........75

© 2024 Microchip Technology Inc. and its subsidiaries

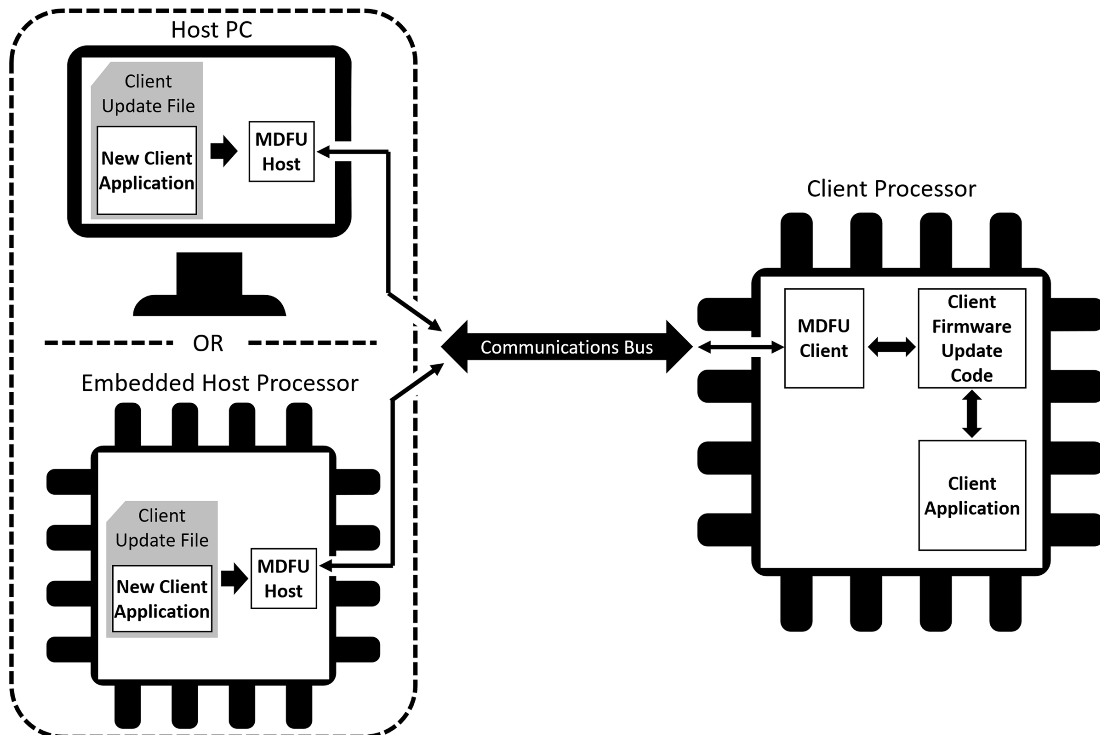
1. Microchip Device Firmware Update (MDFU) Protocol Overview

Many embedded system designs require a mechanism to update a processor's firmware using a standard communications bus (UART, I2C, SPI, etc.).

The Microchip Device Firmware Update Protocol defines a mechanism for updating the firmware of a client processor, using a host processor over different communications busses.

The figure below shows how an MDFU host (either a PC or an embedded processor) is used to transfer a new client application to an MDFU client, running on a client processor. The client firmware update code updates the client application to the new client application that it receives from the host.

More information on terms used in this specification can be found in the Definition of Terms section.



© 2024 Microchip Technology Inc. and its subsidiaries

1.1 MDFU Protocol Characteristics and Benefits

The MDFU Protocol provides a flexible mechanism for updating client firmware that is able to accommodate the needs of many embedded system designs while maintaining a small client memory footprint. It is intended to be capable of supporting the needs of a wide range of client microcontrollers, from small 8-bit microcontrollers to large 32-bit MCUs and beyond.

The MDFU Protocol updates client processor firmware using a file transfer approach that is simple, stable, and client processor independent. Client processor specific memory details are contained in the Client Update File and the client firmware update code. The MDFU Protocol commands and responses are client processor agnostic. Hosts discover all necessary client information from the client as part of the update process. These MDFU Protocol characteristics lead to the following benefits:

- MDFU hosts can update any attached MDFU client processor, if the following conditions are met:
  a. The host supports the version of the protocol the client has implemented.
  b. The host has allocated enough command/response buffer space to support the attached client.
- Client processor specific features can be supported/added by appropriately defining/modifying the Client Update File contents without modifying the MDFU Protocol or host
- The simplicity and stability of the protocol reduces version compatibility issues between MDFU hosts and clients

The layering of the MDFU Protocol makes adding support for new communications busses straightforward. It maximizes the reuse of valuable MDFU features across multiple communication interfaces and clearly calls out which details need to be defined for each unique communications interface. Hosts and clients can be written in a way which maximizes code reuse across different communications interfaces and minimizes the effort of adding support for a new communications interface.

The MDFU Protocol provides mechanisms for detecting command/response corruption and contains mechanisms for automatically recovering from detected corruption.

Clients have the option to abort the update process immediately, upon detection of an unrecoverable error, rather than waiting for the entire update attempt to complete before the host discovers the update failure. A mechanism is also provided for clients to report a reason for terminating the update to help developers or users debug any issues.

Clients can also determine if their new firmware is valid and report that status back to the host. Firmware verification algorithms are a tradeoff of complexity, client resources (Flash/RAM/Processing), and corruption detection capability. Each individual MDFU client can select the verification algorithm that is the best fit for their design.

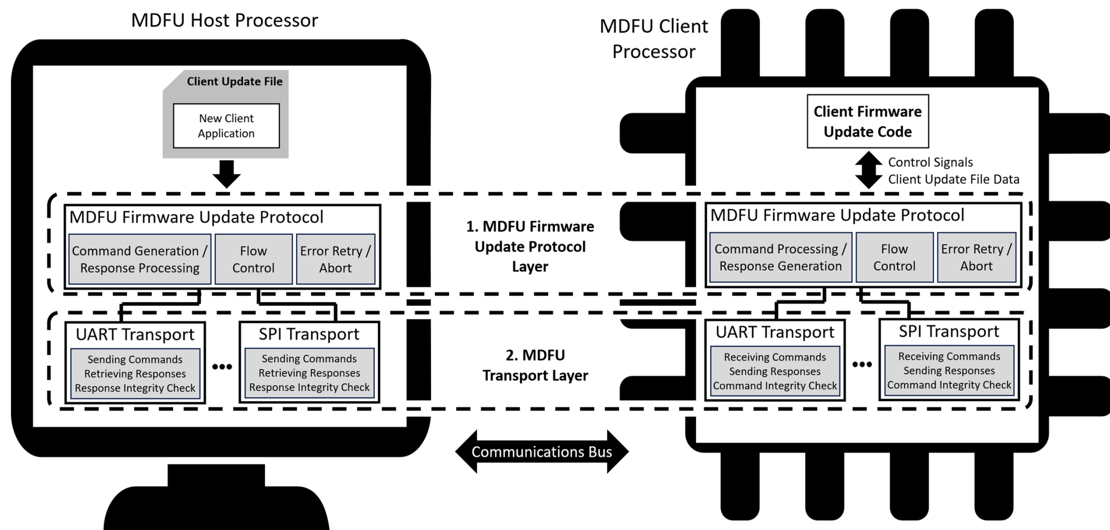
The file transport approach to firmware updates, used by the MDFU Protocol, provides an opportunity to use the same Client Update File format and Client Update File processing code for both hosted firmware update use cases and non-hosted firmware update use cases, i.e. Over The Air (OTA) updates.

© 2024 Microchip Technology Inc. and its subsidiaries

1.2 MDFU Protocol Layers

The MDFU Protocol allows updates to be performed over different physical communication busses. To help facilitate the definition and operation of the MDFU Protocol over these various communication busses, the MDFU Protocol specifications are organized into two layers.

- Firmware Update Protocol Layer
  - The Firmware Update Protocol Layer defines protocol details that are reused by every communications interface.
- Transport Layer
  - The MDFU Transport Layer definitions specify how MDFU commands and responses are sent over a specific communication bus.
  - Each unique communication bus may have a unique specification of how commands and responses are sent over that communication bus.



1.3 Microchip Device Firmware Update Protocol Version

This document describes the following version of the Microchip Device Firmware Update Protocol.

MajorVersion	0x01
MinorVersion	0x00
Patch	0x00
PreRelease	N/A

Any protocol changes that are not backward compatible with the prior version of the protocol increment the MajorVersion byte. Any protocol changes that introduce new capabilities without breaking backwards compatibility with the prior version of the protocol increment the MinorVersion byte. Updates to the specification which do not change the protocol (such as fixing bugs, improving/ clarifying the documentation, fixing typographical errors, etc.) increment the Patch byte. All public releases will omit the PreRelease byte.

© 2024 Microchip Technology Inc. and its subsidiaries

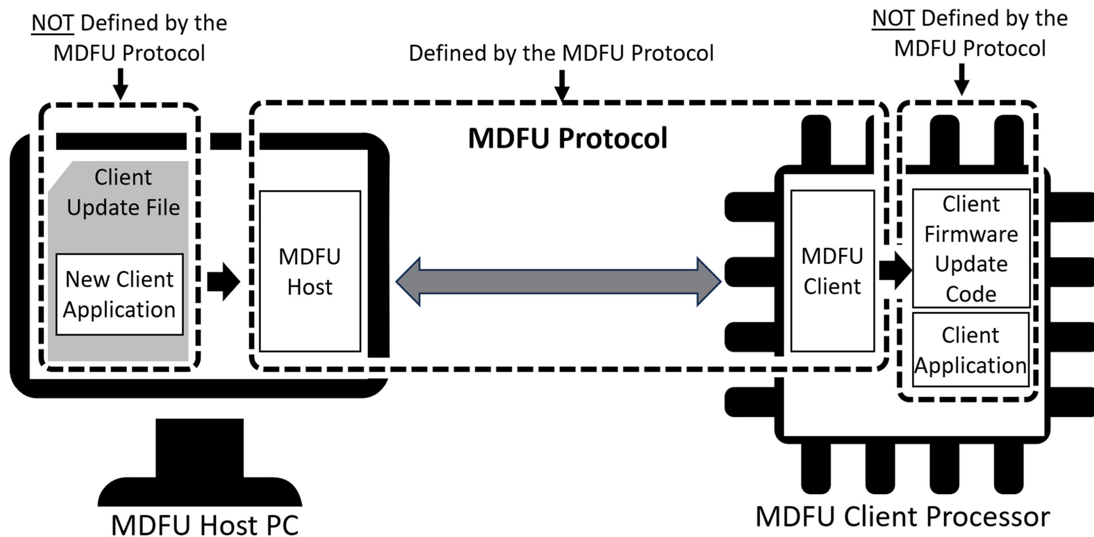
1.4 Definition of Terms

- Host: A processing unit which is used to control the transfer of a Client Update File to a client being updated. This host could be a PC, embedded microprocessor, or microcontroller.
- Client: The processor whose firmware (memory) is being updated.
- Client Update File: A file that the host transfers to the client containing information necessary to update the client firmware (memory).
- MDFU: Microchip Device Firmware Update.

© 2024 Microchip Technology Inc. and its subsidiaries

2. File Transfer Based Approach to Firmware Updates

The MDFU Protocol uses a Client Update File abstraction to separate out the client application transport details from the client memory update details. The MDFU Protocol fully specifies the process (update algorithms, commands/responses, error handling, etc.) to transfer a Client Update File from a host to the client.

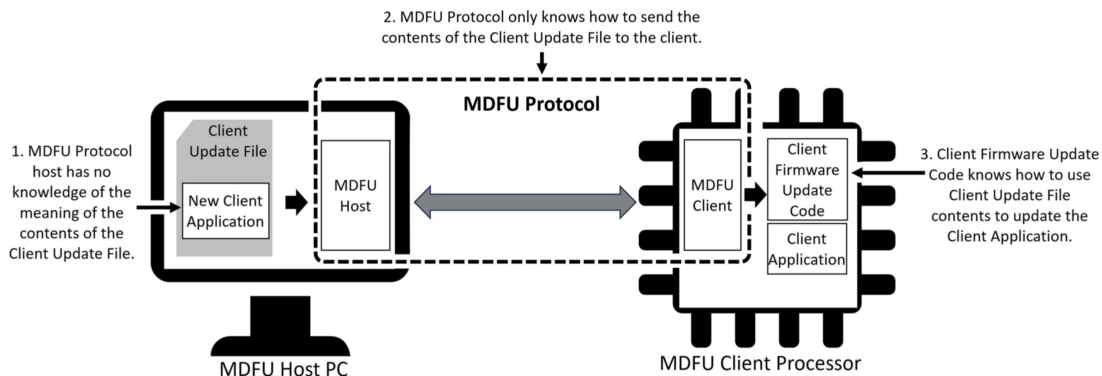


2.1 MDFU File Abstraction Paradigms

The File Transfer Approach to firmware updates used by the MDFU Protocol has a few key paradigms:

1. The MDFU Protocol and MDFU host have no knowledge of the meaning of the Client Update File contents.
2. The MDFU host only knows how to transfer the Client Update File to the client.
3. The Client Firmware Update Code knows how to use the Client Update File contents to update the client firmware.

The format of the Client Update File is an agreement that must exist between the Client Firmware Update Code and the Client Update File. Client Firmware Update Code solutions are responsible for providing a specification for the file format that is used by those Client Firmware Update Code implementations.



© 2024 Microchip Technology Inc. and its subsidiaries

2.2 MDFU File Abstraction Benefits

The paradigms listed in the previous section result in the following desirable MDFU Protocol characteristics:

1. Hosts have universal client device processor support:
   - All client processor specific details (i.e. memory addresses, Flash controller unlock keys, etc.) are kept out of the MDFU Protocol and placed in the Client Update File and Client Firmware Update Code. This allows new client processors to be supported by a host without any modifications to the host. Clients simply need to implement the protocol and then immediately have host support, provided the host has allocated sufficient command/response buffer space to support the client implementation.
2. Host Simplicity and Stability:
   - The process of transporting a file is well-defined, simple, stable, and client independent.
   - New client specific features can be added by modifying the Client Update File contents without modifying the MDFU protocol or host.
   - Embedded host support is easier to achieve with a simple transport protocol where the host is only responsible for Client Update File transport.
   - Host simplicity and stability reduces version compatibility issues between hosts and clients.

© 2024 Microchip Technology Inc. and its subsidiaries

3. Firmware Update Protocol Layer

The MDFU Protocol is divided into two layers. This section specifies the Firmware Update Protocol Layer.

The Firmware Update Protocol Layer defines the following protocol details that are reused by every communications interface:

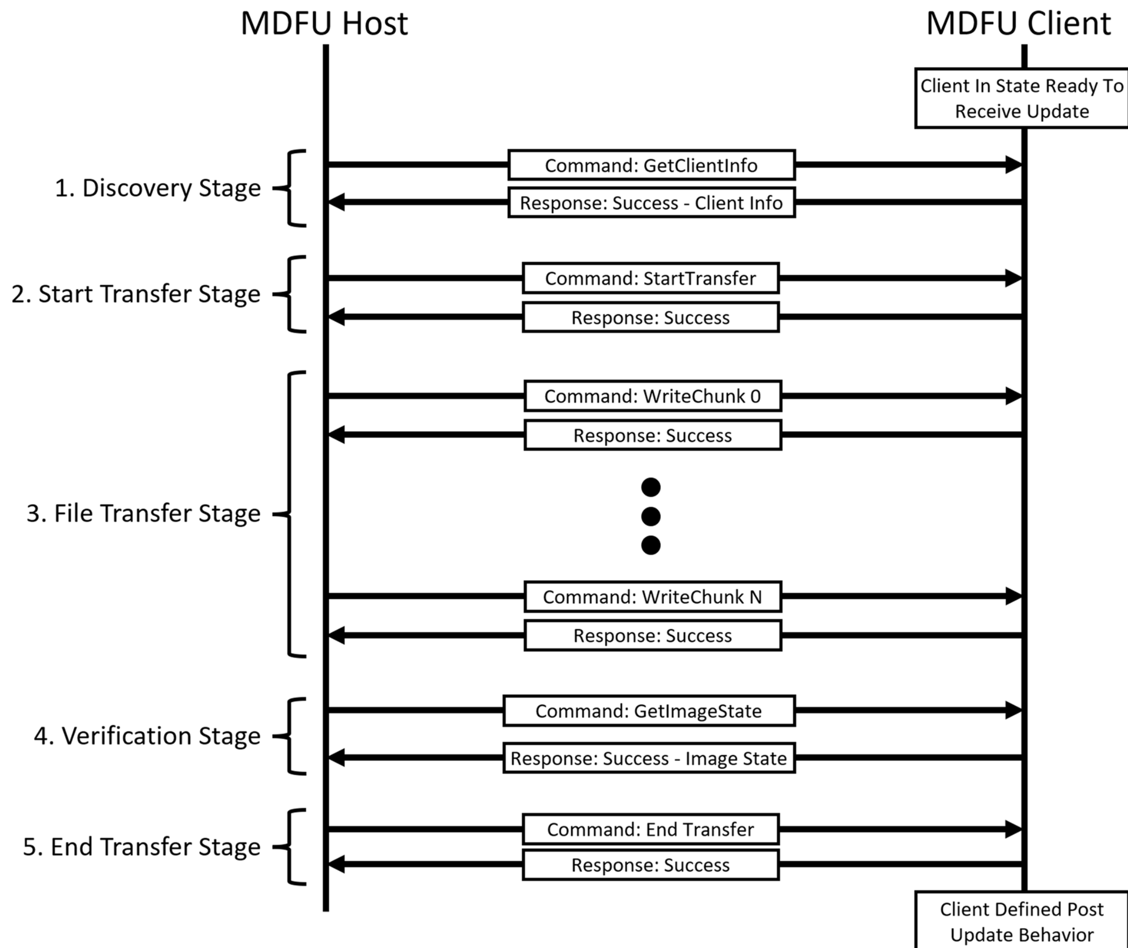
- Firmware Update Algorithm
- File Transfer Stage
- Command-Response Pairs
- Host Command Generation and Response Processing Algorithm
- Client Command Processing and Response Generation
- Commands
- Responses
- Verification Stage
- Flow Control
- Definition of a Successful Update
- Error Classification and Handling
- Sequence Numbers

The Firmware Update Protocol Layer does not define how the commands and responses are sent over different communication busses. Please see the Transport Layer specification section of this document for information on how commands and responses are sent over different physical busses.

3.1 Firmware Update Algorithm

The Microchip Device Firmware Update Protocol uses five different stages that occur sequentially for a host to update a client's firmware. The host uses Command-Response Pairs to accomplish each of these stages. The diagram below and the following sections provide information on each of these stages.

© 2024 Microchip Technology Inc. and its subsidiaries



3.1.1 Client Setup Prior to Update

Prior to initiating an update from the MDFU host, the user must place the client in a mode where it is ready to receive and process MDFU Protocol commands. Different clients may have different mechanisms for entering a firmware update mode such as:

- Pressing/holding a button.
- Power cycling while holding a button.
- Resetting while holding a button.
- Sending an update trigger.
- Accepting an incoming update from a running application (live update).

The mechanism used to enter a firmware update mode is selected by the client firmware developer.

3.1.2 Discovery Stage

Each update starts with a Discovery stage. During this stage, the host discovers the client parameters necessary for updating the client.

Discovery is accomplished through a single *GetClientInfo* command (see Get Client Info Command) sent from the host to the client. The client must respond to this command with a list of client parameter values. The parameter list includes the MDFU Protocol Version implemented by the client, the Client Buffer Info, and the Client Command Time-out Info.

If the client fails to report a mandatory parameter, the host must terminate the update and provide a mechanism for reporting the cause for terminating the update to the user.

© 2024 Microchip Technology Inc. and its subsidiaries

The *GetClientInfo* command definition in the specification contains a complete description of the parameters.

3.1.3 Start Transfer Stage

After discovering the client capabilities, the host sends a *StartTransfer* command (see Start Transfer Command).

The purpose of this signal is to allow the client to initialize any state necessary to prepare for incoming Client Update File data.

If an update fails part way through for any reason, this command must result in the client being reset to a state where it is ready to receive the contents of the entire Client Update File starting from the beginning of the file.

Additional details of actions taken when receiving this command are client dependent.

3.1.4 File Transfer Stage

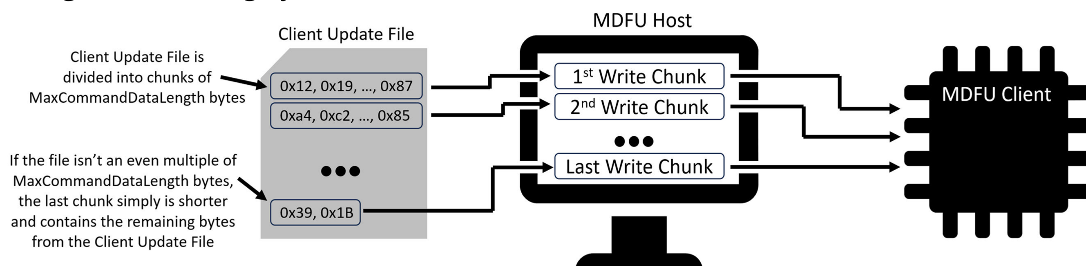
The client firmware is updated by transferring all the data in the Client Update File from the host to the client.

The Client Update File is divided into chunks and sent to the client in smaller pieces for two reasons:

1. Flow Control
   - The MDFU Protocol uses a stop and wait flow control technique, where commands with a known fixed maximum size are sent from the host to the client and the file chunks must fit inside these command data payloads, to use this form of flow control.
   - See the Flow Control section of the specification for more detailed information.
2. Quickly Detecting and Handling Errors
   - Integrity checks on individual commands with small chunks of the file enables faster error detection and faster error recovery from errors due to transmission corruption.
   - Sending file data in small chunks allows the client to abort a transfer as soon as an unrecoverable error is detected.

3.1.4.1 File Chunking Details

- The host divides the file into fixed size chunks and sends the chunks sequentially to the client using successive *WriteChunk* commands (see Write Chunk Command).
- The chunk size is obtained during the Discovery stage from the client using the *GetClientInfo* command.
- The Client Buffer Info Parameter contains the MaxCommandDataLength field which defines the maximum number of bytes of the file that a client can receive in a single command.
- The Client Update File is divided into chunks of MaxCommandDataLength bytes.
- If the file is not an integer multiple of the chunk size, the last file chunk is a smaller chunk containing the remaining bytes from the file.



© 2024 Microchip Technology Inc. and its subsidiaries

3.1.5 Verification Stage

The host sends a *GetImageState* command (see Get Image State Command) after the file has been completely transferred. This command tells the client to run its firmware verification algorithm and respond with information about whether the firmware is valid or not. The response to this command lets the host know if the client has determined that its new firmware is valid.

The mechanism the client uses to verify its memory is selectable by the client and not specified by the MDFU Protocol. Clients could choose to use a checksum, Cyclic Redundancy Check (CRC), or any other algorithm that meets the requirements of the product being designed.

Hosts must provide a mechanism for raising an error, logging the error, or notifying the user of the invalid image on the client so that appropriate actions can be taken to recover from this invalid image.

3.1.6 End Transfer Stage

After verification is complete, the host sends the *EndTransfer* command (see End Transfer Command) to the client to signal that the update is complete.

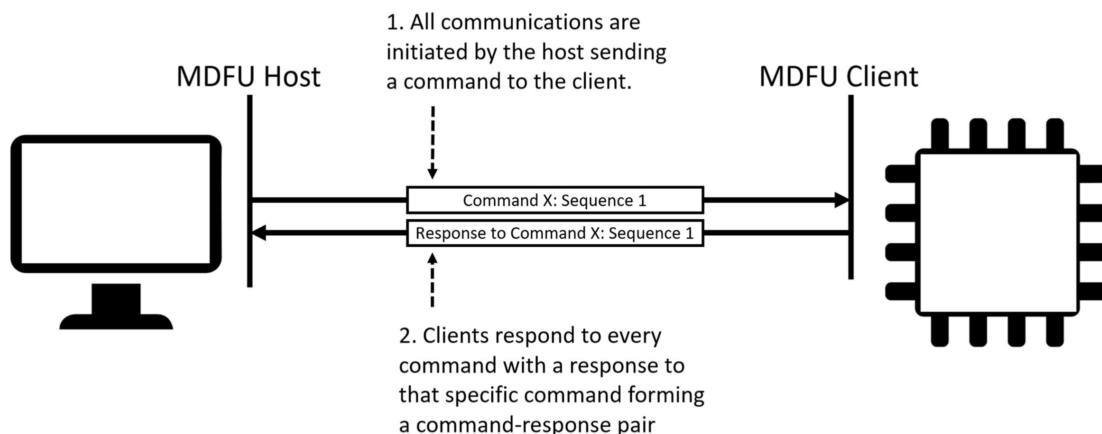
The Client Firmware Update Code can decide what action to take at this point (e.g., launching updated firmware).

3.2 Command-Response Pairs

The Microchip Device Firmware Update Protocol is a host driven protocol. All communications follow three sequential steps:

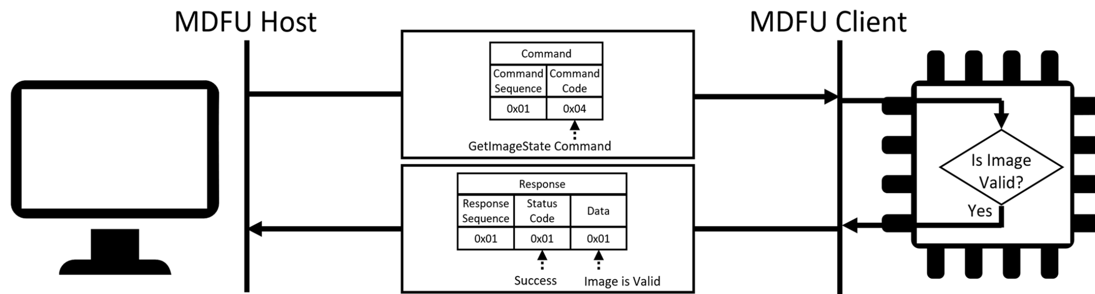
1. Host Command Generation
2. Client Command Processing and Response Generation
3. Host Response Processing

The combination of a command with its response is called a command-response pair. Commands and responses can be identified as being associated as members of the same command-response pair through their sequence number.



The following is an example showing how a command can be used to trigger an operation on the client and retrieve data associated with that command. This example shows the host sending the *GetImageState* command (see Get Image State Command) to the client. The client determines whether its firmware image is valid or not and then responds to that command with information about the state of its firmware image.

© 2024 Microchip Technology Inc. and its subsidiaries



3.2.1 Commands

The MDFU commands provide the host with a mechanism for sending signals to the client, sending Client Update File data to the client, and triggering firmware verification algorithms on the client.

A client only needs to support five mandatory commands in order to be successfully updated by a host. However, the protocol provides a mechanism for adding optional commands in the future to support new use cases.

3.2.1.1 Command List

Each command-response pair has a subsection in this specification which defines the details for that command including the command code, command data payload, client action upon receiving the command, and the client successful response data payload definition.

Command	Command Code	Client Implementation Required
Reserved - Never to Be Used[1]	0x00	N/A
GetClientInfo	0x01	Mandatory
StartTransfer	0x02	Mandatory
WriteChunk	0x03	Mandatory
GetImageState	0x04	Mandatory
EndTransfer	0x05	Mandatory
Reserved for Future Use	0x06-0xFF	Optional
Note 1: The command code 0x00 must never be used by any host/client. Using the command code 0x00 may interfere with other aspects of the protocol. Any host/client that implements a command code of 0x00 is not compliant with the MDFU Protocol specification.		

3.2.1.2 Command Format

Commands are formed by concatenating several different fields together. Commands consist of a single byte Command Sequence Field, a single byte Command Code, and may also contain a data payload depending on the command being sent. Information on the contents/meaning of the data payload for each individual command can be found in the detailed definitions of the commands.

The maximum number of bytes that can be placed into the command data payload (N_C) is client dependent and equal to MaxCommandDataLength. Clients report MaxCommandDataLength to the host using the Client Buffer Info Parameter during the Discovery phase of the update.

© 2024 Microchip Technology Inc. and its subsidiaries

Command Format		
Command Sequence	Command Code	Command Data Payload
1 byte	1 byte	0 – N_C Bytes[1]

Note 1: N_C = MaxCommandDataLength Client Parameter

3.2.2 Responses

Responses in the MDFU Protocol serve multiple purposes. Responses enable the following:

- Hosts can confirm that commands have executed successfully in the client.
- Hosts can retrieve data related to commands that have executed successfully.
- Hosts can know when the client is ready for the next command based on the responses and protocol flow control rules.
- Responses allow the host to understand when recoverable errors have been encountered and trigger the host's error recovery mechanism.
- When unrecoverable errors are encountered, hosts can learn information about the error to help with diagnosing and debugging problems.
- Clients can abort an update as soon as an unrecoverable error is detected without having to wait until the end of the update for the host to know the update has failed.

3.2.2.1 Response Format

Responses are constructed using a 1-byte Response Sequence Field, a 1-byte Status Code, and a flexible length response data payload. These fields are concatenated together to form a response as shown in the following figure.

Response Format		
Response Sequence	Status Code	Response Data Payload
1 byte	1 byte	0 – N_R Bytes

The maximum number of bytes that are placed into the response data payload (N_R) is client dependent. In version 1.0.0 of the protocol, the *GetClientInfo* response is the longest response defined by the protocol. If all optional *GetClientInfo* response parameter values defined by the protocol are included in the response, the data payload to the *GetClientInfo* response is 28 bytes (6-byte Protocol Version Parameter + 5-byte Client Buffer Info Parameter + 17-byte Client Command Time-out Parameter with four SpecificCommandTimeOuts). Adding more optional parameters to the *GetClientInfo* response would make this response longer. Future versions of the protocol may define longer responses.

© 2024 Microchip Technology Inc. and its subsidiaries

3.2.2.1.1 Response Status

The table below lists the available response status codes. Response status codes allow the client to tell the host if the command completed successfully, there was an issue/error that prevented the command from executing, or if the client wants to abort the file transfer firmware update process.

Supporting some of these codes are mandatory and must be present in every client implemented. Others are optional and the client developer can choose whether or not to implement them in a specific client.

Response Status	Response Status Code	Client Implementation Required
Reserved - Never Used to Be Used[1]	0x00	N/A
SUCCESS	0x01	Mandatory
COMMAND_NOT_SUPPORTED	0x02	Mandatory
Reserved for Future Use	0x03	Optional
COMMAND_NOT_EXECUTED	0x04	Mandatory
ABORT_FILE_TRANSFER	0x05	Optional
Reserved for Future Use	0x06 - 0xFF	Optional
Note 1: The response status code 0x00 must never be used by any host/client. Using the response status code 0x00 may interfere with other aspects of the protocol. Any host/client that implements a response status code of 0x00 is not compliant with the MDFU Protocol specification.		

3.2.2.1.2 Response Data Payload

The contents of the response data payload depend on the status code that is contained in the response.

- When the status code is SUCCESS, this indicates that the command executed successfully and the response data payload contains data related to the execution of the command. The SUCCESS response data payload definitions are provided in the detailed definitions of each individual command-response pair.
- When the status code is not SUCCESS, this indicates that an error was detected by the client. In these cases, the response data payload may contain additional information which can be used to help diagnose and debug the issue. See the Error Response Details section of this specification for details on the meaning of the error response status codes and the data payload definitions for these status responses.

Response		
Response Sequence	Status Code	Response Data Payload
0xYY	0x01 SUCCESS	Data Related to the Successful Command Execution
	Other Status Codes	Information about Errors Encountered

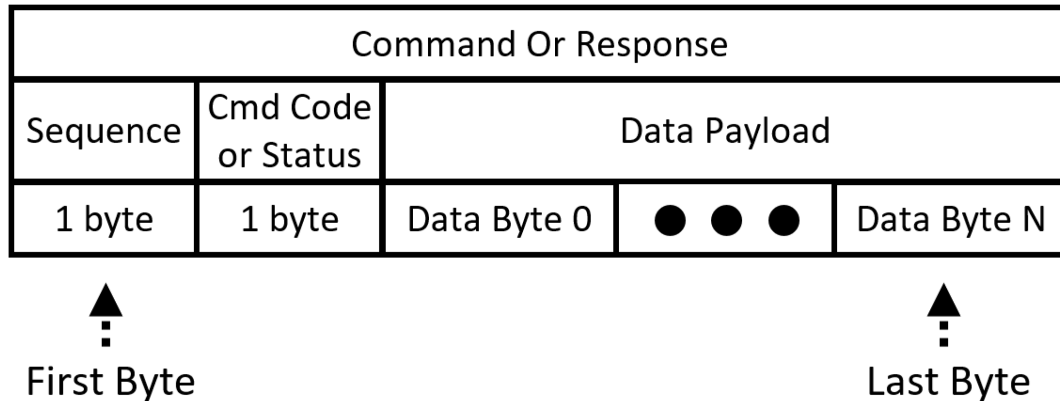
3.2.3 Command/Response Byte Order

To enable interoperability between hosts and clients, the MDFU Protocol specifies the order of bytes within commands/responses and the order of bytes for multi-byte fields.

Command/Response Byte Order

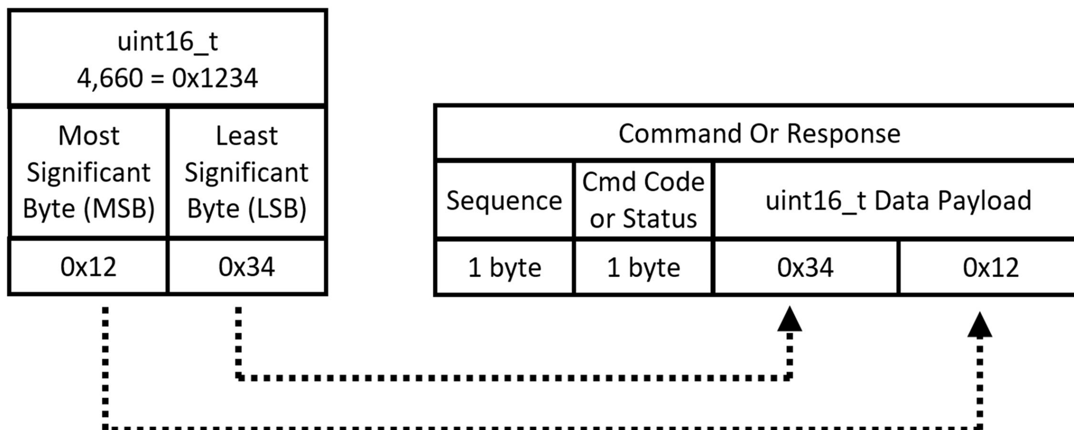
© 2024 Microchip Technology Inc. and its subsidiaries

The specification contains a number of figures which are used to define the command and response fields. These figures show the order of bytes within commands/responses. The bytes are ordered from left to right as shown below.



Multi-Byte Field Byte Order

Multi-byte data fields are ordered in little endian format within a command or response. Less significant bytes are placed before more significant bytes. The figure below shows an example of how an unsigned 16-bit integer value of 4,660(0x1234) would be encoded in a command/response with a uint16_t data payload field.



3.2.4 Data Types

The MDFU Protocol uses the following data types:

- uint8_t: Unsigned 8-bit integer with a range from 0 to 255
- uint16_t: Unsigned 16-bit integer with a range from 0 to 65,535

3.2.5 Command-Response Pair Definitions

The following sections provide the detailed definitions of the Command-Response pairs.

3.2.5.1 Get Client Info Command

When creating a Microchip Device Firmware Update client, there are several flexible protocol parameters whose values can be chosen to meet the requirements of a specific design, such as the size of the client buffers and the command time-out values.

© 2024 Microchip Technology Inc. and its subsidiaries

The host needs to know the values of these parameters in order to successfully update the client and learns these details during the client discovery process.

The host sends the *GetClientInfo* command to the client during the Discovery Stage of the update. The client must respond with the details of its implementation in the form of a list of Client Parameters.

Since the *GetClientInfo* command is used to learn the command time-outs, there is no way to customize the time-out duration of the *GetClientInfo* command. The *GetClientInfo* command has a fixed time-out of one second.

3.2.5.1.1 Command Details

Get Client Info Command Details and Success Response Data Payload	
Command Code	0x01
Command Data Payload	None
Successful Response Data Payload	Concatenated list of Client Parameters [Parameter1, Parameter2, ..., ParameterN] Parameters are encoded in Type, Length, Value (TLV) Format

Client Parameters

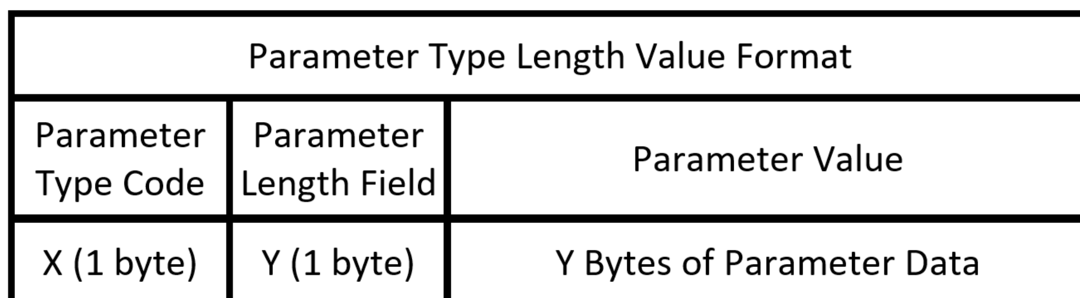
There are several mandatory client parameters which must be returned by all clients. Additionally, there is a mechanism for supporting optional parameters which clients may choose to implement, if desired.

All client parameters are concatenated together into a list and returned by the client in the response to the *GetClientInfo* command. A list of the parameters is given below and the details of each parameter are given in the following subsections of this specification.

Parameter Name	Parameter Type Code	Mandatory/Optional
Reserved for Future Use	0x00	Optional
Protocol Version Parameter	0x01	Mandatory
Client Buffer Info Parameter	0x02	Mandatory
Client Command Timeout Parameter	0x03	Mandatory
Reserved for Future Use	0x04-0xFF	Optional

Parameter Type, Length, Value (TLV) Format Definition

The client parameters are encoded in a type, length, value (TLV) format with 1-byte defining the Parameter Type, 1-byte defining the Parameter Length, and the Parameter Length Field defining the number of bytes in the Parameter Value Field as shown in the following figure.

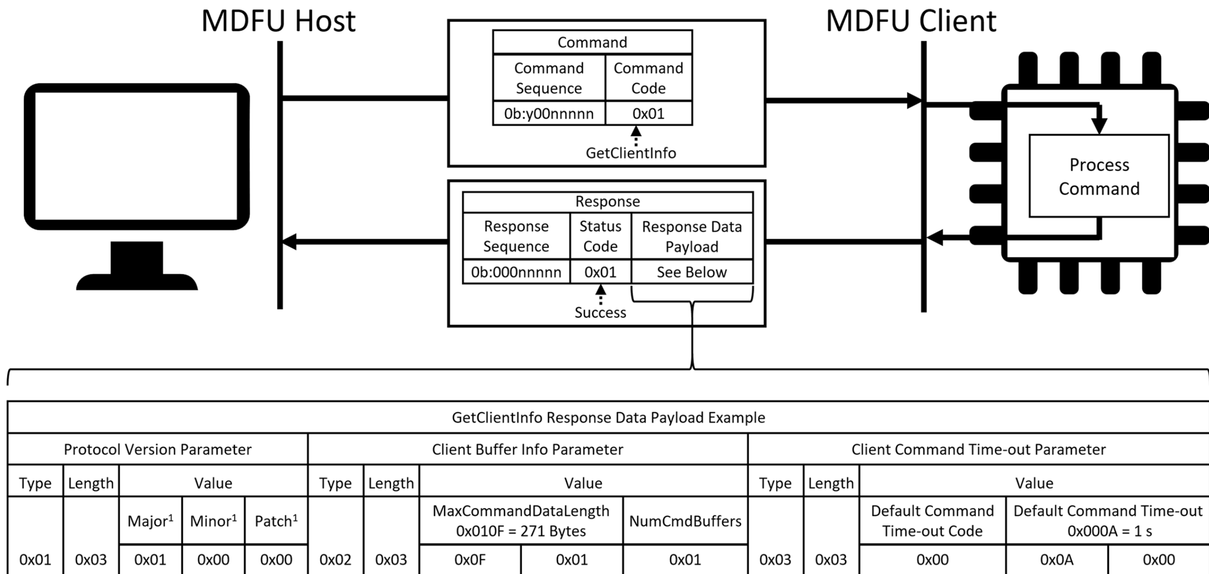


© 2024 Microchip Technology Inc. and its subsidiaries

3.2.5.1.2 Command Response Pair Example

The following figure shows an example of a host sending the *GetClientInfo* command to a client and the client responding with an example set of the mandatory parameters.

Figure 3-1. Command Response Pair Example



GetClientInfo Response Data Payload Example														
Protocol Version Parameter					Client Buffer Info Parameter					Client Command Time-out Parameter				
Type	Length	Value			Type	Length	Value			Type	Length	Value		
		Major[1]	Minor[1]	Patch[1]			MaxCommandDataLength 0x010F = 271 Bytes		NumCmdBuffers			Default Command Time-out Code	Default Command Time-out 0x000A = 1 s	
0x01	0x03	0x01	0x00	0x00	0x02	0x03	0x0F	0x01	0x01	0x03	0x03	0x00	0x0A	0x00

Figure Notes

Note 1: This example contains version 1.0.0 of the MDFU Protocol for the Protocol Version Parameter. When implementing a different version of the protocol, this version must be updated to be the version of the protocol being implemented.

3.2.5.1.3 Protocol Version Parameter

The Protocol Version parameter allows the host to discover what version of the Microchip Device Firmware Update Protocol the client has implemented. Hosts use this version to verify that they are capable of updating the client processor and providing helpful information to the user in circumstances where they do not support the version of the protocol implemented on the client.

For a host to be capable of updating a client, the host must meet the following requirements.

1. The host must support the MajorVersion of the protocol implemented by the client.
2. The host must support the MinorVersion of the protocol greater than or equal to the MinorVersion implemented by the client.

If the host does not support updating the client, the host must terminate the update and provide a mechanism for alerting the user to obtain a host which supports the version of the protocol implemented by the client.

Parameter Details

Public release versions are 3-bytes long containing major, minor, and patch details (one byte each). Microchip internal pre-release versions may append a fourth pre-release byte that is not present for public release versions.

Protocol Version Parameter Details	
Parameter Type Code	0x01
Parameter Length	3 for Public Releases, 4 for Internal Pre-Release Builds

© 2024 Microchip Technology Inc. and its subsidiaries

...........continued

Protocol Version Parameter Details	
Parameter Value	<MajorVersionByte><MinorVersionByte><PatchByte>[<PreReleaseByte>]

Parameter Example

The following figure shows an example of the Protocol Version parameter.

Protocol Version Parameter				
Type	Length	Value		
		Major[1]	Minor[1]	Patch[1]
0x01	0x03	0x01	0x00	0x00

Note 1: This example contains version 1.0.0 of the MDFU Protocol for the Protocol Version Parameter. When implementing a different version of the protocol, this version must be updated to be the version of the protocol being implemented.

3.2.5.1.4 Client Buffer Info Parameter

To support the needs of small and large clients in a flexible way, clients are allowed to select the details of the buffers they implement. Clients must report these buffer details to the host including the maximum number of bytes of data that can be sent in a single command data payload and the number of command buffers that have been implemented in the client. The MDFU Protocol specification currently does not support more than one command buffer so clients must implement and report one command buffer.

Parameter Details

Client Buffer Info Parameter Details	
Parameter Type Code	0x02
Parameter Length	3
Parameter Value	<MaxCommandDataLength><NumCmdBuffers>
MaxCommandDataLength	uint16_t: The maximum number of bytes of data that the client can support in the command data payload.
NumCmdBuffers	uint8_t: The number of command buffers implemented on the client. Currently, this must be 1.

Parameter Example

The following figure shows an example of the Client Buffer Info parameter.

Client Buffer Info Parameter				
Type	Length	Value		
		MaxCommandDataLength 0x010F = 271 Bytes		NumCmdBuffers
0x02	0x03	0x0F	0x01	0x01

© 2024 Microchip Technology Inc. and its subsidiaries

3.2.5.1.5 Client Command Time-out Parameter

Clients are required to report the maximum amount of time that commands take to execute so that the host knows when to safely time-out and retry commands. To balance simplicity and flexibility, clients can report two types of time-outs. The client command time-out parameter value must begin with a Default Command Time-out which applies to all commands which do not have a Specific Command Time-out. Clients can optionally report one or more Specific Command Time-outs which apply only to specific commands.

Since the Get Client Info Command (*GetClientInfo*) is used to learn the command time-outs, there is no way to customize the time-out duration of the *GetClientInfo* command. The *GetClientInfo* command has a fixed time-out of one second.

Parameter Details

Client Command Time-out Parameter Details	
Parameter Type Code	0x03
Parameter Length	(3 Bytes per time-out) x Number of Time-outs Reported
Parameter Value	<DefaultCmdTimeOutCode><DefaultCommandTimeOut>[<cmdCode_1><uint16_t SpecificCommandTimeOut_1>]... [<cmdCode_N><SpecificCommandTimeOut_N>]
DefaultCmdTimeOutCode	uint8_t: 0x00
DefaultCommandTimeOut	uint16_t: The default time-out applied to all commands which don’t have a specific command time-out
cmdCode_x	uint8_t: The command code of a specific command that is being assigned a unique time-out value
SpecificCommandTimeOut_x	uint16_t: A time-out which applies specifically to the command identified by the preceding cmdCode_x

Time-out Units and Calculation

- DefaultCommandTimeOut and SpecificCommandTimeOut_x are uint16_t values.
- The LSB of each of these parameters is 0.1s.
- The minimum time-out is 0.1 seconds.
- The maximum time-out is 65,535 x 0.1 s = 6,553.5 seconds = 109.225 minutes.
- TimeOutInSeconds = TimeOutParameterValue x (0.1 Seconds / LSB).
- TimeOutParameterValue = TimeOutInSeconds / (0.1 Seconds / LSB).

Parameter Examples

The following figures show examples of how to create the Client Command Time-out Parameter for two different scenarios.

The first example shows the simplest case where only the Default Time-out is provided. This time-out applies to all commands.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-2. Example 1: One Second Default Time-out Response

Client Command Time-out Parameter Example				
Type	Length	Value		
		Default Command Time-out Code	Default Command Time-out = 0x000A = 1 s	
0x03	0x03	0x00	0x0A	0x00

Default Command Time-out = 1 s

The second example shows how to use a specific command time-out to modify the time-out of a command. It shows a ten second time-out that applies only to the *GetImageState* command. The Default Command Time-out of one second applies to all other commands.

Figure 3-3. Example 2: One Second Default Time-out and Ten Second GetImageState Time-out

Client Command Time-out Parameter Example							
Type	Length	Value					
		Default Command Time-out Code	Default Command Time-out = 0x000A = 1s		GetImageState Command Code	GetImageState Time-out = 0x0064 = 10 s	
0x03	0x06	0x00	0x0A	0x00	0x04	0x64	0x00

Default Command Time-out = 1 s

GetImageState Time-out = 10 s

3.2.5.2 Start Transfer Command

The host sends the *StartTransfer* command to the client during the Start Transfer Stage to signal to the client that the Client Update File transfer is about to begin. The client must prepare to receive the contents of an entire incoming Client Update File starting from the beginning of the file. Once the client is ready to start receiving the Client Update File, it must respond to to the host with a success status to inform the host that it is ready for a new incoming Client Update File.

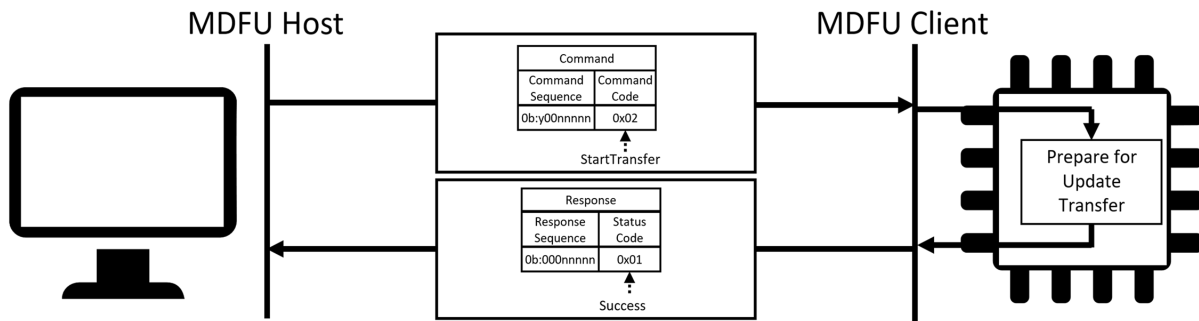
3.2.5.2.1 Command Details

Start Transfer Command Details and Success Response Data Payload	
Command Code	0x02
Command Data Payload	None
Successful Response Data Payload	None

3.2.5.2.2 Command Response Pair Example

The following figure shows an example of a host sending the *StartTransfer* command to a client and the client responding to indicate that it is ready to begin an update.

© 2024 Microchip Technology Inc. and its subsidiaries



3.2.5.3 Write Chunk Command

During the Start Transfer Stage of the update, the host sends a series of *WriteChunk* commands to the client. These commands contain sequential chunks of data from the Client Update File as described in the File Chunking Details description.

3.2.5.3.1 Command Details

Write Chunk Command Details and Success Response Data Payload	
Command Code	0x03
Command Data Payload	<N bytes of data from the Client Update File>
Successful Response Data Payload	None
N	If NumberRemainingBytesInFile >= MaxCommandDataLength: N = MaxCommandDataLength If NumberRemainingBytesInFile < MaxCommandDataLength: N = NumberRemainingBytesInFile

3.2.5.3.2 Command Response Pair Example

The following figure shows an example of a host using *WriteChunk* commands to divide a Client Update File into pieces that are sent to the client. This example breaks up the file into 4-byte chunks and sends them to the client. The number of bytes in each chunk is discovered from the client during the Discovery stage. The Client Buffer Info Parameter contains a MaxCommandDataLength field. The Client Update File is divided into MaxCommandDataLength size chunks and sent over to the client. If the last chunk isn't exactly MaxCommandDataLength bytes long, the last chunk is simply equal in size to the number of bytes remaining in the Client Update File.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-4. Command Response Pair Example

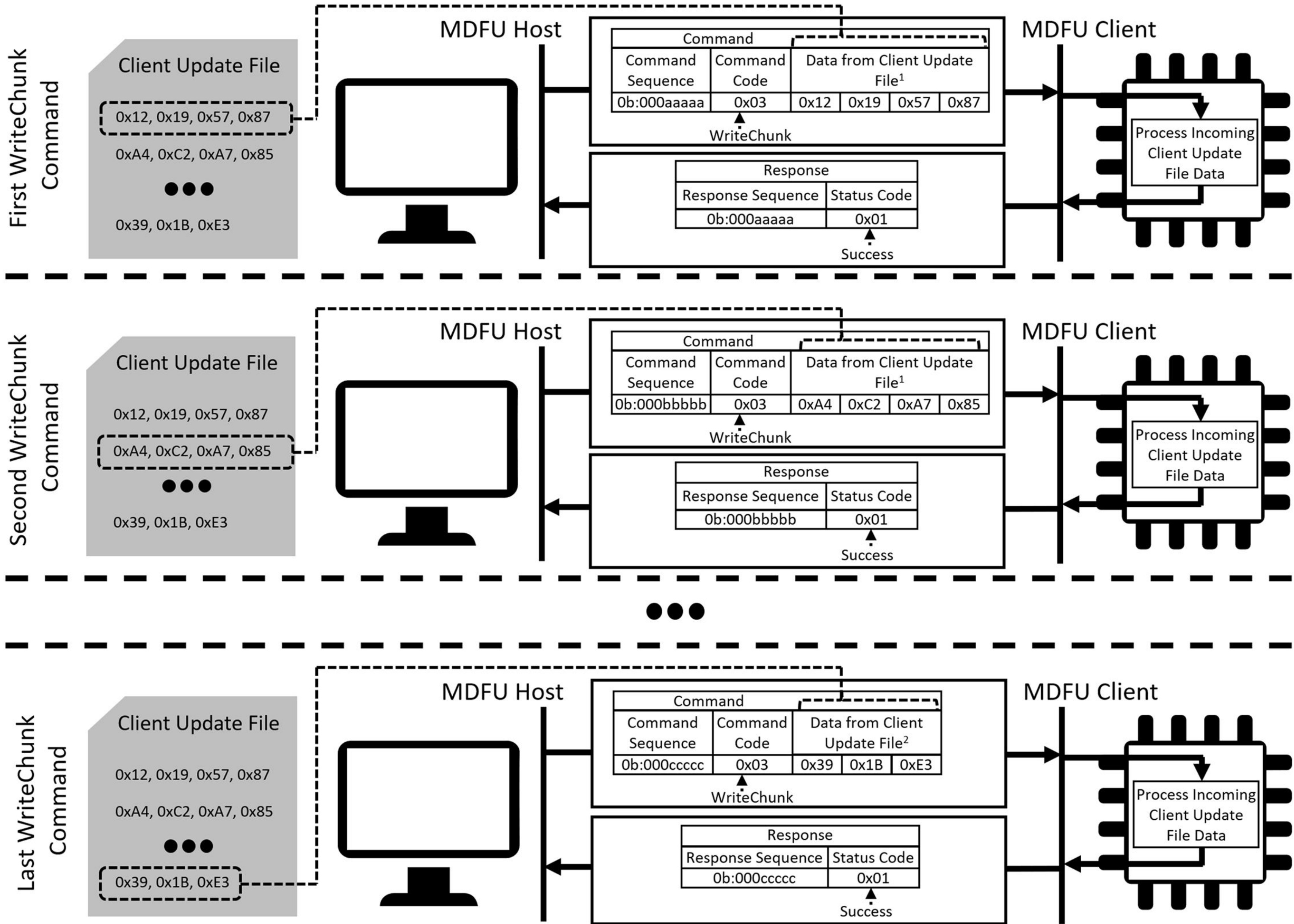


Figure Notes

Note 1: This example assumes the ClientBufferInfo parameter MaxCommandDataLength obtained during the discovery stage is 4 bytes and divides the file into 4-byte chunks for the *WriteChunk* commands.

Note 2: Since the last chunk of the Client Update File only contains 3 bytes, only 3 bytes of the file data are sent to the client in the last *WriteChunk* command.

© 2024 Microchip Technology Inc. and its subsidiaries

3.2.5.4 Get Image State Command

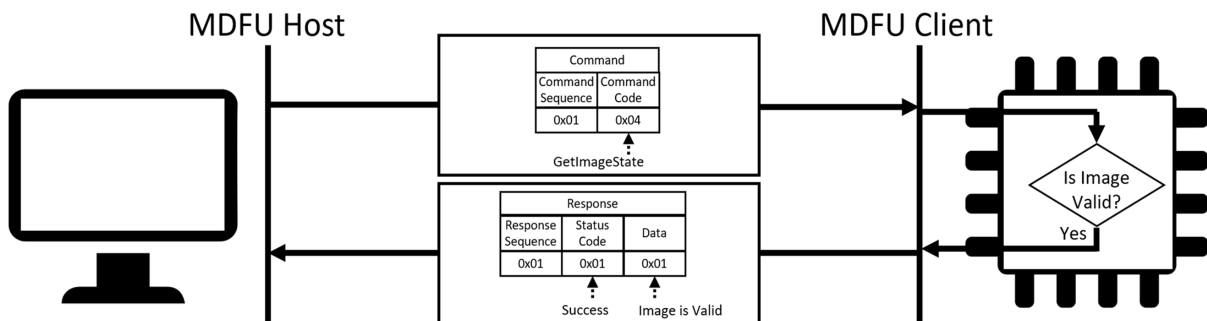
The host sends the *GetImageState* command to the client during the Verification Stage of the update. The client runs an image verification algorithm to determine the state of its firmware image and responds to the host indicating whether it has a valid or invalid image.

3.2.5.4.1 Command Details

Get Image State Command Details and Success Response Data Payload	
Command Code	0x04
Command Data Payload	None
Successful Response Data Payload	<uint8_t: ImageState>
ImageState Values	IMAGE_VALID = 0x01 IMAGE_INVALID = 0x02

3.2.5.4.2 Command Response Pair Example

The following figure shows an example of a host sending the *GetImageState* command to a client and the client responding that its image is valid.



3.2.5.5 End Transfer Command

The host sends the *EndTransfer* command to the client during the End Transfer Stage of the update to signal to the client that the Client Update File transfer is complete. The client acknowledges that it has successfully received and processed this command with a success status response.

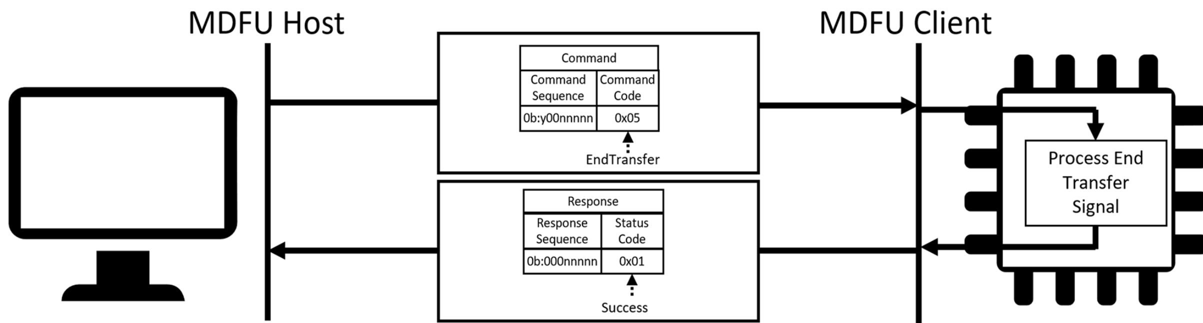
3.2.5.5.1 Command Details

End Transfer Command Details and Success Response Data Payload	
Command Code	0x05
Command Data Payload	None
Successful Response Data Payload	None

3.2.5.5.2 Command Response Pair Example

The following figure shows an example of a host sending the *EndTransfer* command to a client and the client responding with a success response.

© 2024 Microchip Technology Inc. and its subsidiaries



3.2.6 Error Response Details

When a MDFU client detects an error during the update, it is important that the client is able to communicate information about the error to the host.

The mechanism that the client uses to report error conditions to the host is the response (see Responses). When an error is detected, the client can report back a Response Status which indicates that an error has occurred along with a response data payload that contains more information about the error.

The following sections provide detailed definitions of the error responses including their status codes and data payload definitions.

3.2.6.1 COMMAND_NOT_SUPPORTED Response

When a client executes a command, the client is responsible for making sure that it has implemented the command. The execution of an unimplemented command by a client is defined to produce a response with the COMMAND_NOT_SUPPORTED status. This response tells the host that that the client has not implemented the command.

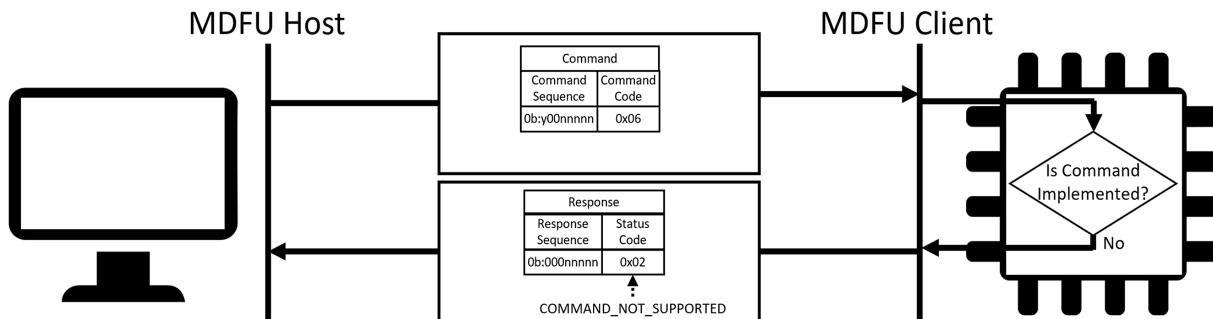
See Client Command Processing and Response Generation for more details on when this response is generated by a client.

3.2.6.1.1 Response Details

COMMAND_NOT_SUPPORTED Response Details	
Status Code	0x02
Response Data Payload	None

3.2.6.1.2 Response Example

The following figure shows an example of the COMMAND_NOT_SUPPORTED response.



3.2.6.2 COMMAND_NOT_EXECUTED Response

When a client receives a command, there are a number of error checks it performs. Several of those checks result in the client concluding that there is a problem detected with an incoming command.

© 2024 Microchip Technology Inc. and its subsidiaries

These can include command corruption errors, framing errors, and sequence number errors. When these errors are detected, the client does not execute the command and responds to the host with the COMMAND_NOT_EXECUTED response.

See Client Command Processing and Response Generation for more details on when this response is generated by a client.

3.2.6.2.1 Response Details

COMMAND_NOT_EXECUTED Response Details	
Status Code	0x04
Response Data Payload	[uint8_t CommandNotExecutedCause][1]
Note 1: When reporting that a command was not executed, it is optional to report the CommandNotExecutedCause.	

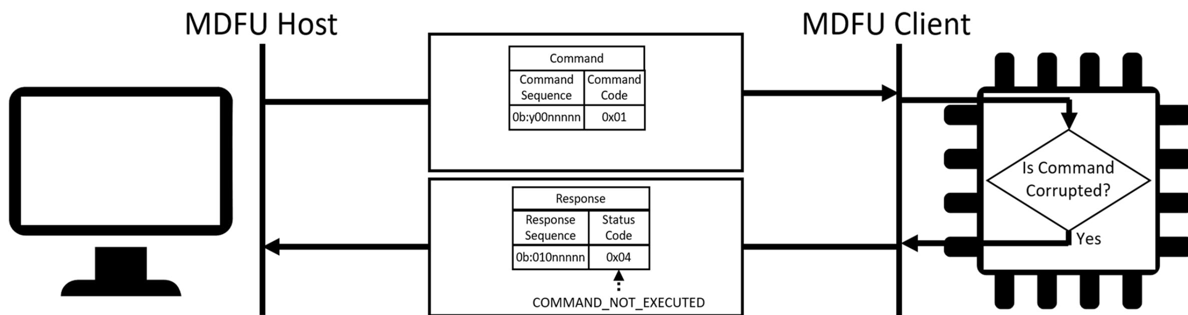
CommandNotExecutedCause Definition

Reporting a CommandNotExecutedCause is optional but encouraged. The smallest clients are allowed to report a COMMAND_NOT_EXECUTED status without reporting a CommandNotExecutedCause and the protocol will function properly. However, when errors are encountered, the reason the command was not executed will not be known by the host and can't be communicated to the developer or user. It is encouraged for clients to report a CommandNotExecutedCause to improve the debugging experience when errors are encountered.

CommandNotExecutedCause Name	CommandNotExecutedCause Value	Detailed Description
TRANSPORT_INTEGRITY_CHECK_ERROR	0x00	Code for reporting that the command received failed the Transport Integrity Check indicating that the command was corrupted during transportation from the host to the client
COMMAND_TOO_LONG	0x01	Code for reporting that the received command exceeded the size of the client buffer
COMMAND_TOO_SHORT	0x02	Code for reporting that the received command was too short
SEQUENCE_NUMBER_INVALID	0x03	Code for reporting that the sequence number of the received command is invalid
Reserved for Future Use	0x04-0xFF	Reserved for future use

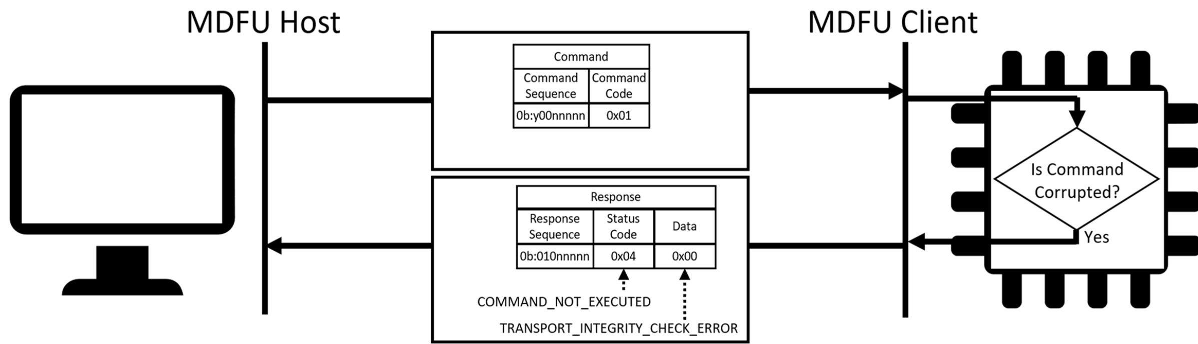
3.2.6.2.2 Response Examples

The following figure shows an example of the COMMAND_NOT_EXECUTED response where no CommandNotExecutedCause is reported.



The following figure shows an example of the COMMAND_NOT_EXECUTED response where the TRANSPORT_INTEGRITY_CHECK_ERROR CommandNotExecutedCause is reported.

© 2024 Microchip Technology Inc. and its subsidiaries



3.2.6.3 ABORT_FILE_TRANSFER Response

When clients detect an unrecoverable error, the MDFU Protocol provides clients with a mechanism to terminate the update immediately using the ABORT_FILE_TRANSFER response. Clients can respond to any command with the ABORT_FILE_TRANSFER response to terminate the update. Additionally, clients can communicate information about the reason why the update was terminated.

See the section on Unrecoverable Errors and Client Command Processing and Response Generation for more details on when this response is generated by a client.

3.2.6.3.1 Response Details

ABORT_FILE_TRANSFER Response Details	
Status Code	0x05
Response Data Payload	[uint8_t FileAbortCause][1]
Note 1: When aborting a file transfer, it is optional to report one of the following causes for debugging purposes.	

FileAbortCause Definition

Reporting a FileAbortCause is optional but encouraged. The smallest clients are allowed to report a ABORT_FILE_TRANSFER status without reporting a FileAbortCause and the protocol will function properly. However, when errors are encountered, the reason for encountering the ABORT_FILE_TRANSFER will not be known and can't be communicated to the user or developer. It is encouraged for clients to report a FileAbortCause to improve the debugging experience when errors are encountered.

FileAbortCause Name	FileAbortCause Value	Reason Client is Rejecting the Update
GENERIC_CLIENT_ERROR	0x00	Code for reporting a generic problem encountered by the client
INVALID_FILE	0x01	Code for reporting a generic problem detected in the Client Update File
INVALID_CLIENT_DEVICEID	0x02	The Client Update File is not compatible with the client device ID
ADDRESS_ERROR	0x03	An invalid address is present in the Client Update File
ERASE_ERROR	0x04	Client memory did not properly erase
WRITE_ERROR	0x05	Client memory did not properly write
READ_ERROR	0x06	Client memory did not properly read
APPLICATION_VERSION_ERROR	0x07	Client won't allow changing to the application version in the Client Update File (Anti-rollback)

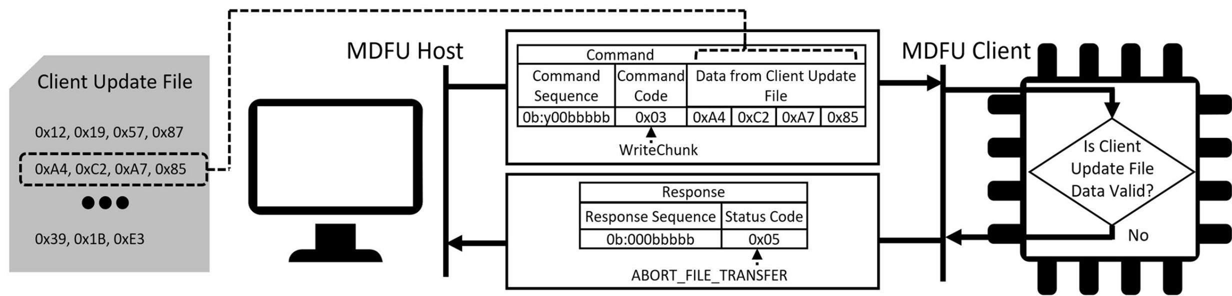
© 2024 Microchip Technology Inc. and its subsidiaries

...........continued

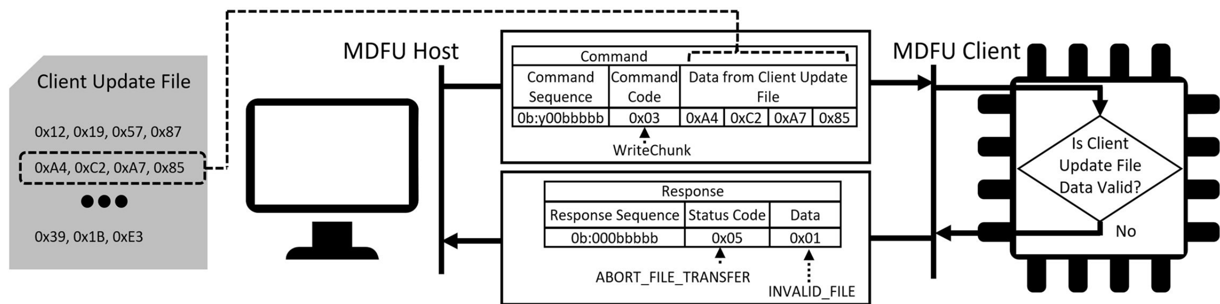
FileAbortCause Name	FileAbortCause Value	Reason Client is Rejecting the Update
Reserved for Future Use	0x08-0xFF	Reserved for future use

3.2.6.3.2 Response Examples

The following figure shows an example of the ABORT_FILE_TRANSFER response where no FileAbortCause is reported.



The following figure shows an example of the ABORT_FILE_TRANSFER response where the INVALID_FILE FileAbortCause is reported.



3.3 Host Command Generation and Response Processing Algorithm

The following sections detail the process the host goes through to generate/send commands and process responses from the client.

3.3.1 Host Command Generation

All communications in the Microchip Device Firmware Update Protocol are initiated by the host. The host uses the Firmware Update Algorithm to determine the commands it needs to generate and send to walk a client through the process of a firmware update.

The host forms these commands and uses a Transport Layer definition for one of the physical communications busses supported to send the commands to the client.

The host follows the Flow Control rules to know when it is ok to send the next command.

3.3.2 Host Response Processing

Each command that the host sends should result in a response that is returned from the client.

The response from the client is received by the Transport Layer.

When the response arrives, the host checks the response for a number of different conditions. These response conditions allow the host to determine if the command was executed successfully and the response data to the command is available. It also allows the host to determine if there were errors which occurred and what to do about those errors. Detected Recoverable Errors

© 2024 Microchip Technology Inc. and its subsidiaries

activate the host's automatic recovery mechanism. Detected Unrecoverable Errors terminate the update process and deliver error messages with debug information to the user.

The host checks for a number of different types of errors in the response including:

- Transport Errors: Response corruption or framing errors (i.e., noise bit flips)
- Sequence Number Errors
- Status Errors such as ABORT_FILE_TRANSFER

If the response does not contain any detectable errors and has the SUCCESS status, this status and data payload are used by the host to complete the processing of the successfully transmitted and executed command-response pair. The current command-response pair has now been completed, and the host uses the firmware update algorithm to begin working on the next command-response pair in the update.

3.3.3 Host Command Generation and Response Processing Algorithm Diagram

The following diagrams show the process the host goes through to generate/send commands and process responses from the client.

A host developer can decide to place the flow control time-out mechanism in either the host Firmware Update Protocol Layer or in the host Transport Layer. The first figure shows the time-out implemented in the host Firmware Update Protocol Layer. The second figure shows the time-out implemented in the host Transport Layer.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-5. Host Command Generation and Response Processing (Time-out in Firmware Update Protocol Layer)

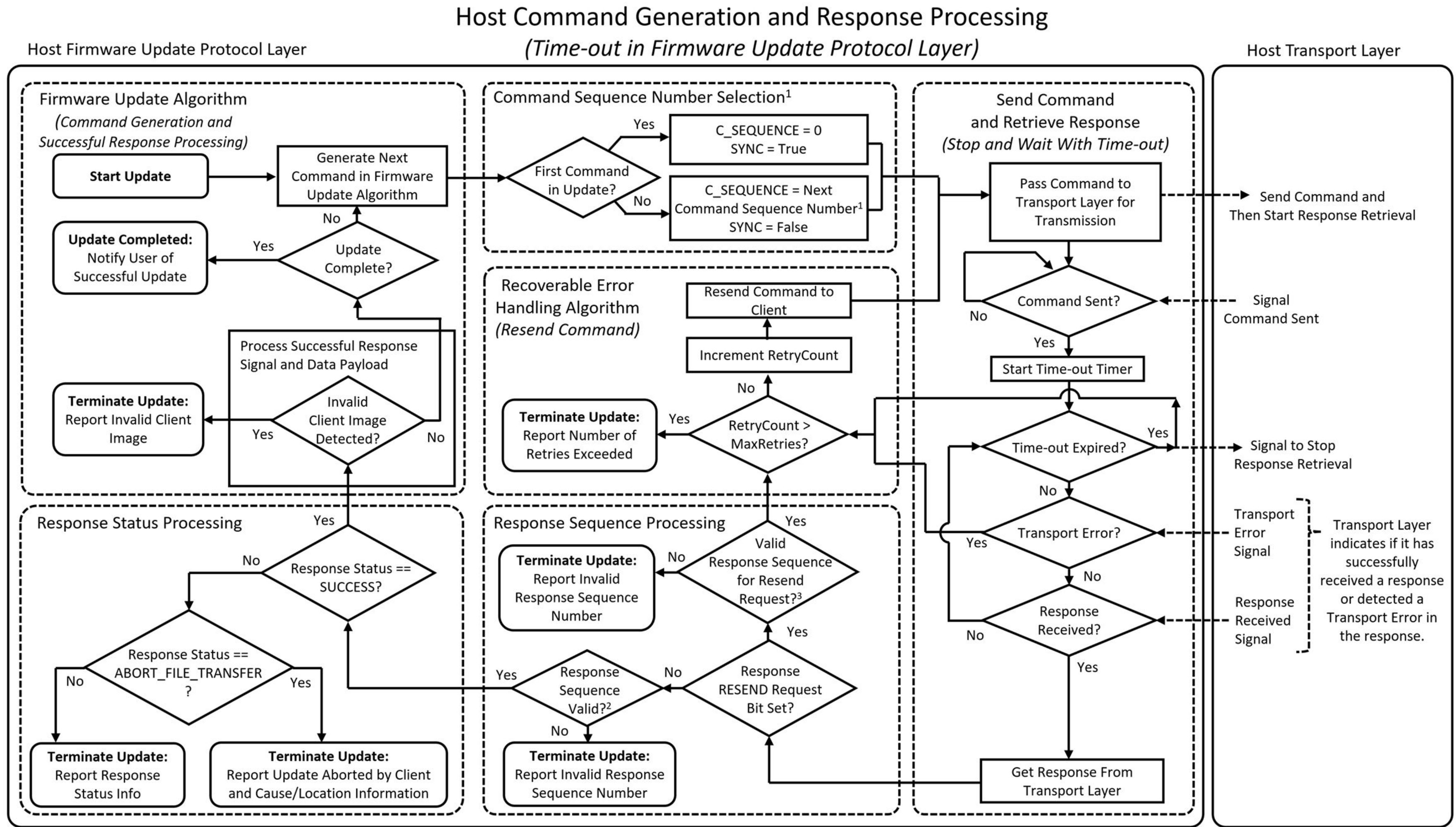


Figure Notes

Note 1: See Host Command Sequence Number Selection for more details.

Note 2: The response sequence number is valid if R_SEQUENCE is equal to C_SEQUENCE.

Note 3: The response sequence number is valid for a resend request if R_SEQUENCE is equal to C_SEQUENCE or if R_SEQUENCE is equal to (C_SEQUENCE + 1) % 32.

MICROCHIP
Manual
© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-6. Host Command Generation and Response Processing (Time-out in Transport Layer)

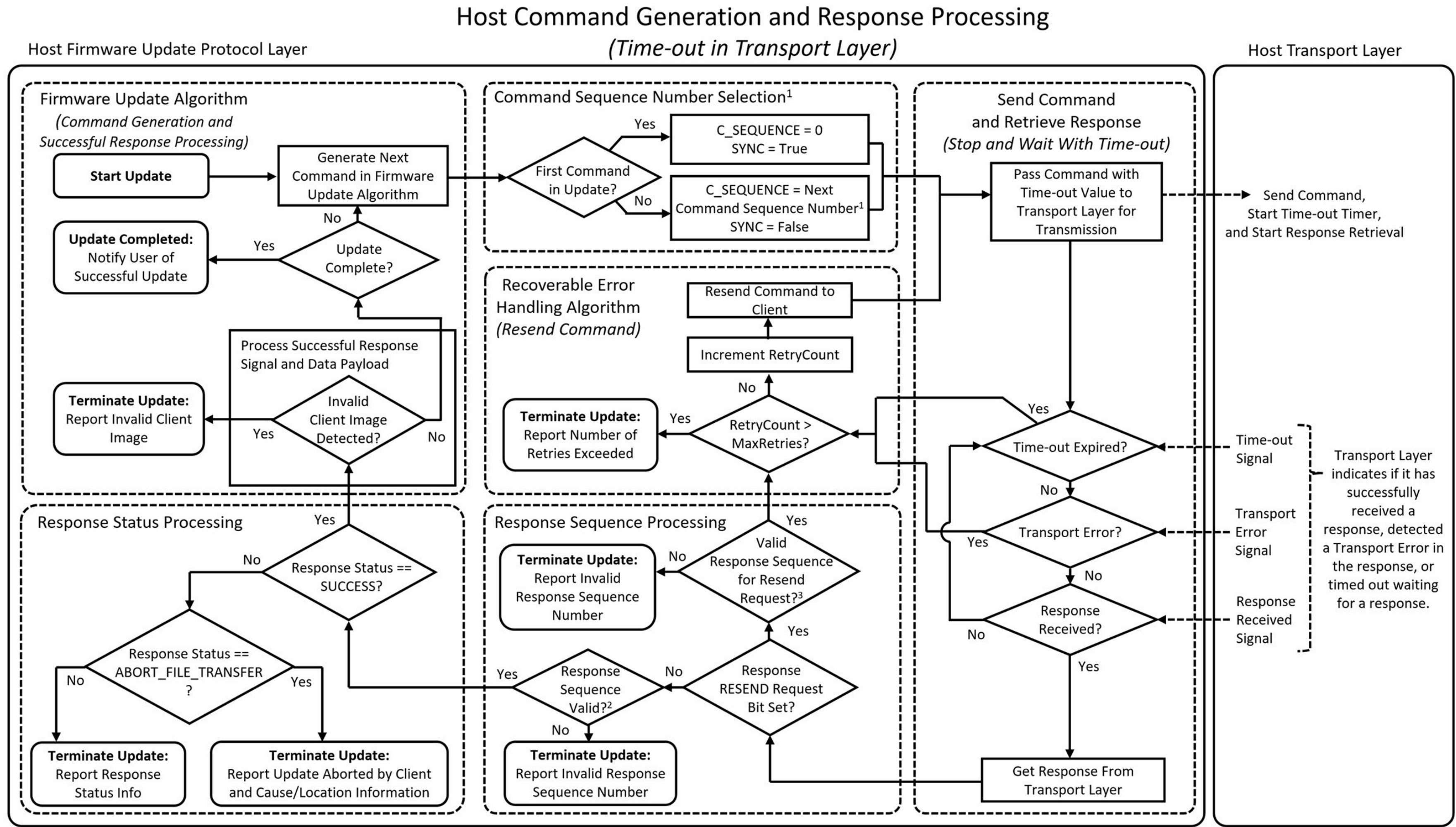


Figure Notes

Note 1: See Host Command Sequence Number Selection for more details.

Note 2: The response sequence number is valid if R_SEQUENCE is equal to C_SEQUENCE.

Note 3: The response sequence number is valid for a resend request if R_SEQUENCE is equal to C_SEQUENCE or if R_SEQUENCE is equal to (C_SEQUENCE + 1) % 32.

© 2024 Microchip Technology Inc. and its subsidiaries

3.3.4 Host Command and Response Buffer Sizing

The MDFU Protocol enables the development of hosts which are capable of updating any attached client provided that the host has allocated enough buffer memory to store the largest command and largest response of the attached client.

Hosts must have enough command buffer space to send *WriteChunk* commands containing MaxCommandDataLength byte chunks of the file. MaxCommandDataLength is a client specific parameter which can have different values for different clients. Additionally, different clients may also have different maximum response data payload sizes. See Command-Response Pairs for more information on command and response sizes.

Since the maximum command and response sizes are client dependent, ensuring that the host has large enough command and response buffers is an integration task that must be performed to make sure that a specific host has enough memory allocated to update the attached clients.

PC hosts with plenty of memory and dynamic memory allocation can be written in a way that is unlikely to require modification of the host to support most clients. However, embedded hosts with fixed size statically allocated buffers may need to be rebuilt to allocate more memory to the command and response buffers to accommodate the needs of specific clients. It is the MDFU user's responsibility to ensure that their host has enough memory allocated to update their attached clients.

3.4 Client Command Processing and Response Generation

The Microchip Device Firmware Update Protocol specifies that all communications are initiated by the host sending a command to the client. The client only takes action and generates a response after receiving a command from the host. Responses are generated for a specific command as part of the command-response pair and the host can identify the command associated with a response using the Response Sequence Field.

Responses serve multiple purposes:

- Responses provide an opportunity for the client to inform the host that a command was executed successfully and return data associated with that command
- Responses provide the client with an opportunity to inform the host about errors detected by the client and include important details such as:
  - The type of error detected: Response Status
  - If recovery from the error is requested: Resend Request
  - If the error is unrecoverable and the client wants to stop the update: ABORT_FILE_TRANSFER Response
  - Useful debugging information in the form of response data payloads for responses with an error status
- Responses from the client are a key part of the protocol Flow Control mechanism

3.4.1 Client Command Processing and Response Generation Algorithm

The following figure shows the process a client goes through to receive an incoming command, check the command for errors, process/execute the command, generate a response, and send the response to the host.

Important: After executing a command, a client must retain the response to that command until the client can conclude that the host will not request that specific response be resent. See the Recoverable Errors section of the specification for more information on the error recovery algorithms. See the Client Sequence Number Processing section of the specification for more information on when a client can discard a response to an executed command.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-7. MDFU Protocol Client

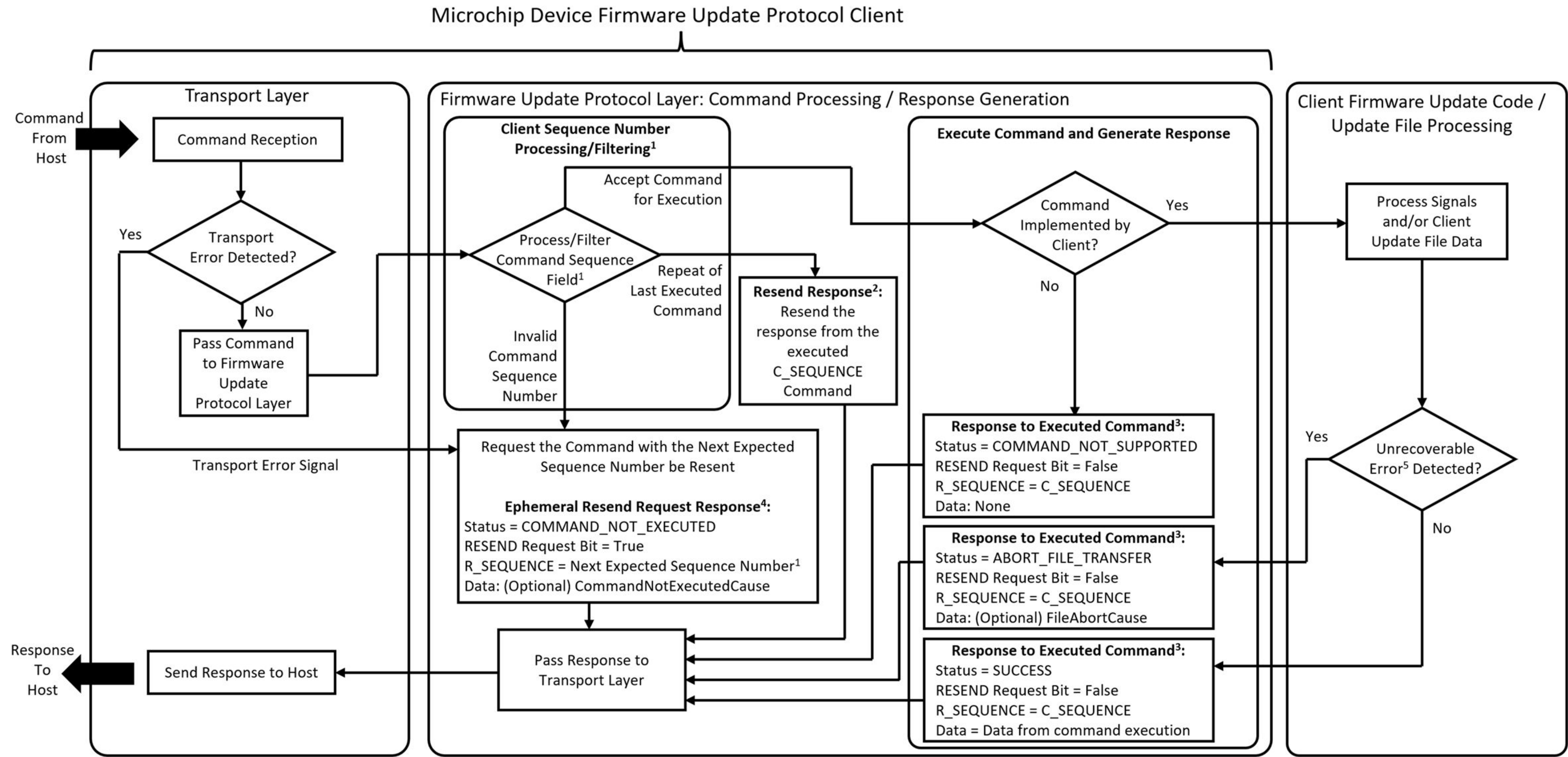


Figure Notes

Note 1: See Client Sequence Number Processing section of the specification for more information.

Note 2: The only responses resent by the client are responses to executed commands[3]. Ephemeral Resend Request Responses[4] are never resent by the client.

Note 3: Responses to executed commands must be retained by the client until the client can conclude that the host will not request for the response to be resent.

Note 4: Ephemeral Resend Request Responses are never resent by the client, and the client can discard them as soon as they have been sent to the host.

Note 5: See Unrecoverable Errors section of the specification for more information on these errors.

© 2024 Microchip Technology Inc. and its subsidiaries

3.5 Flow Control

During an update, a host uses commands to send the contents of a Client Update File to the client. There are several potential reasons why a client processor might not be able to process incoming commands or Client Update File data as fast as the host could theoretically transmit the Client Update File data to the client.

- Some microcontrollers don't support executing code while erasing/writing Flash.
- In some scenarios the communication data rate may be greater than the Flash erase and write speed.
- Client firmware update routines may be a low priority background task with limited processor bandwidth allocation.

The MDFU Protocol uses the flow control mechanism described in this section to allow the client to slow down the update to a speed it is capable of supporting without command/data loss. This flow control mechanism allows the host to avoid sending commands and Client Update File data to the client when the client does not have adequate resources available (processing cycles, RAM buffer space, etc.) to process them.

3.5.1 Stop and Wait Flow Control with Time-outs

Host Stop and Wait

MDFU hosts know exactly how much data a client can receive at a time.

- MDFU clients have a single command reception buffer.
- MDFU hosts learn the size of this buffer during the Discovery phase of the update.

After sending a command to the client, hosts must stop and wait for a response to that command before sending additional commands/data.

The response from the client is an indication that the client is ready to receive another command.

If a response is corrupted, the response can produce a Transport Error instead of a correctly received response. Therefore, detecting a response Transport Error is another indication that the client has finished processing the prior command and is ready to receive a the next command.

Using this process, MDFU hosts avoid sending commands/data when MDFU clients are not ready to receive them, and hosts know when clients are ready to receive additional commands/data.

Host Time-outs

Host time-outs are necessary to avoid situations where command/response corruption could cause the host to wait forever for a response that will never come. Hosts will never see a response in a situation where a command/response is lost during transport:

- Some forms of corruption during transport result in commands/responses being lost and never fully received by a client/host.
- Lost commands are never processed by the client and do not generate a response from the client.
- Lost responses are never processed by the host.

Host time-outs provide the host with an opportunity to recover from commands/responses which have been completely lost. Hosts recover from lost commands/responses by resending those commands.

In order for hosts to honor the stop and wait flow control mechanism described above, hosts are not allowed to resend a command before clients have had enough time to process that command.

In the update Discovery phase, the host learns the maximum amount of time that the client takes to execute each command through the Client Command Time-out Parameter.

© 2024 Microchip Technology Inc. and its subsidiaries

The host uses the command time-out parameter information to make sure that it does not time out and retry a command before the client has had sufficient time to complete the command execution.

Since the Get Client Info Command (*GetClientInfo*) is used to learn the command time-outs, there is no way to customize the time-out duration of the *GetClientInfo* command. The *GetClientInfo* command has a fixed time-out of one second.

3.5.2 Flow Control Algorithm Diagram

The following figure shows the MDFU Protocol flow control algorithm.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-8. Flow Control Algorithm

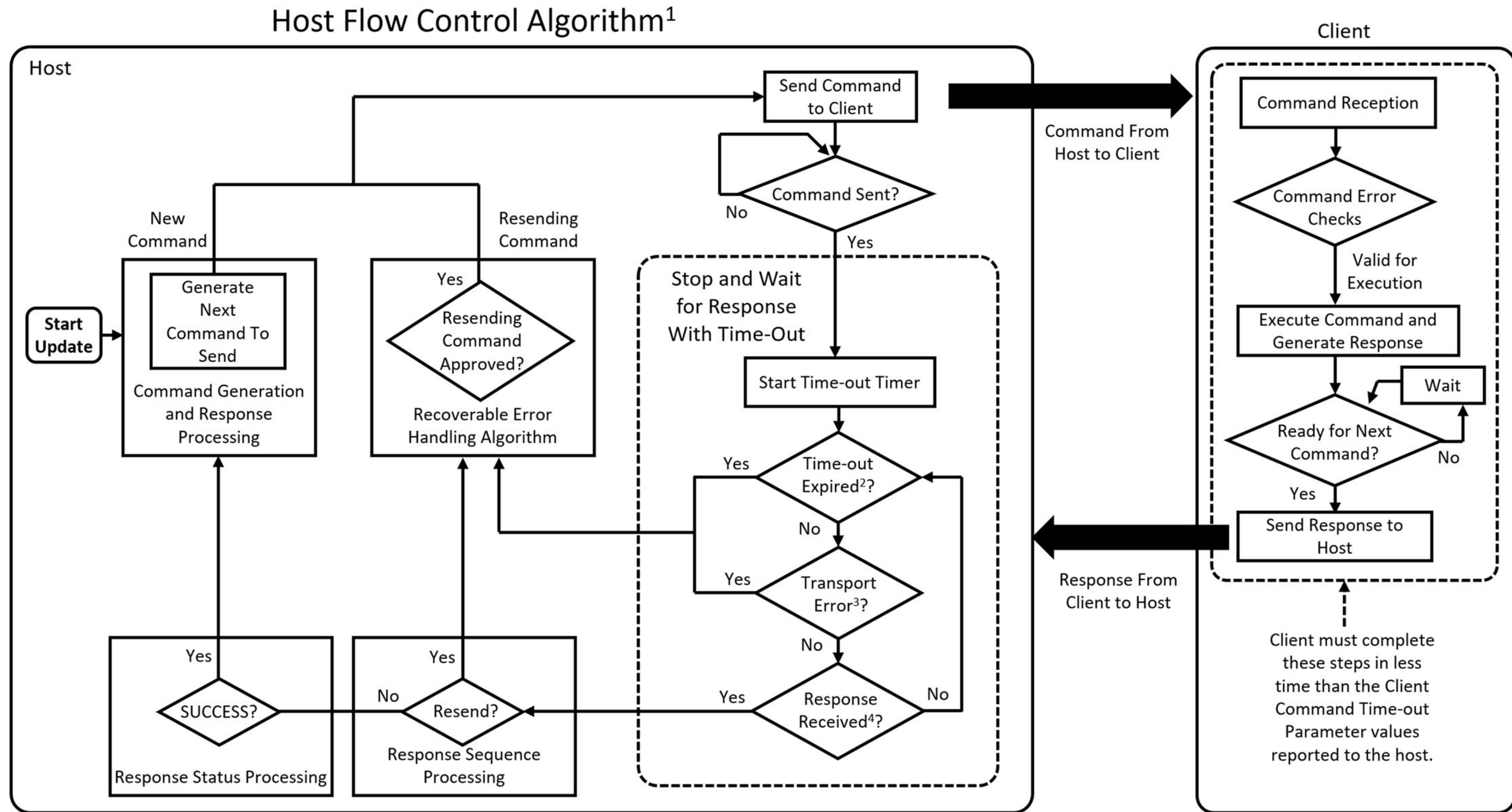


Figure Notes

Note 1: This diagram is a simplification of the host processing algorithms intended to show the MDFU flow control mechanism. Please refer to the Host Command Generation and Response Processing section of the specification for the more detailed algorithms.

Note 2: If a command or response is corrupted in a way that results in the command or response being lost, no response will arrive and the time-out will expire. See the Client Command Time-out Parameter section for more details on the time-out values.

Note 3: Corruption detected in a response during reception produces a Transport Error.

Note 4: Responses which are received without any corruption detected are considered received responses.

© 2024 Microchip Technology Inc. and its subsidiaries

3.6 Definition of a Successful Update

A successful update requires the host to walk the client through all five stages of the update process successfully. The host accomplishes this using a sequence of command-response pairs that need to be executed by the client. In order to complete the update successfully, the following conditions must be met:

1. Each command in the update must be successfully delivered to the client without corruption.
2. Each command in the update must be executed by the client once in the proper order.
3. Each response to an executed command must be successfully delivered to the host without corruption.

Ideally no commands or responses will get corrupted during an update, but corrupted commands and responses will not necessarily cause an update to fail if the MDFU error recovery algorithms can recover from the command/response corruption.

The MDFU Protocol defines a number of algorithms to help detect command/response corruption and automatically recover if possible. The MDFU Protocol defines procedures for resending corrupted commands/responses as well as filtering mechanisms to ensure commands are executed only once and in the proper order. To understand the mechanisms that the MDFU provides to assist with detecting and recovering from command/response corruption, refer to the following sections of the specification.

- Error Classification and Handling
- Sequence Numbers
- Command and Response Integrity Checks
- Flow Control

3.7 Error Classification and Handling

At a high level the MDFU Protocol classifies errors as either Recoverable Errors or Unrecoverable Errors. When recoverable errors are detected, they trigger the MDFU Protocol recovery mechanism. When unrecoverable errors are detected, they trigger the file transfer abort mechanism which terminates the update immediately and provides the ability to report debugging information to the user.

3.7.1 Limitations of the MDFU Error Detection and Recovery

The error detection and recovery mechanisms are meant to enhance the reliability and robustness of the update process. However, there are limitations to the effectiveness of these mechanisms and they cannot detect and/or recover from every type of conceivable error. Users of the protocol must assess whether or not the mechanisms provided in this protocol are sufficient for their application.

3.7.2 Recoverable Errors

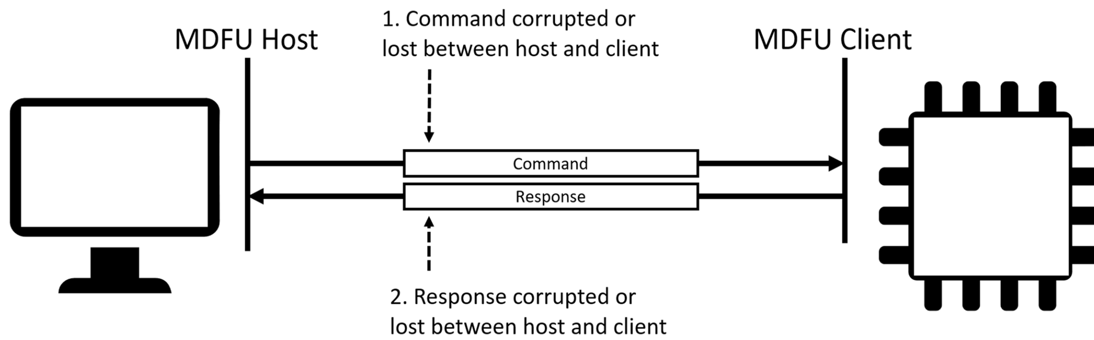
The MDFU Protocol defines mechanisms for detecting and recovering from errors. When a recoverable error occurs in an update, a proper series of subsequent operations can be used to correct this error. Once the error has been corrected, the update can resume and be successfully completed.

The following errors are considered recoverable errors:

1. Commands that have been corrupted during transmission from the host to the client.
2. Responses that have been corrupted during transmission from the client to the host.
3. Commands that are completely lost due to corruption on the physical communication bus.
4. Responses that are completely lost due to corruption on the physical communication bus.

These errors may be caused by sources such as noise bit flips which occur as the commands and responses are transported across a physical bus.

© 2024 Microchip Technology Inc. and its subsidiaries



3.7.2.1 Detection

One of the responsibilities of each Transport Layer definition is to provide a command-response integrity check mechanism for determining if commands and responses have been corrupted when they are transmitted across the communications bus.

Commands and responses that are completely lost are detected using the command time-outs which are learned during the Discovery phase and incorporated as part of the flow control algorithm.

3.7.2.2 Host Recovery Algorithm

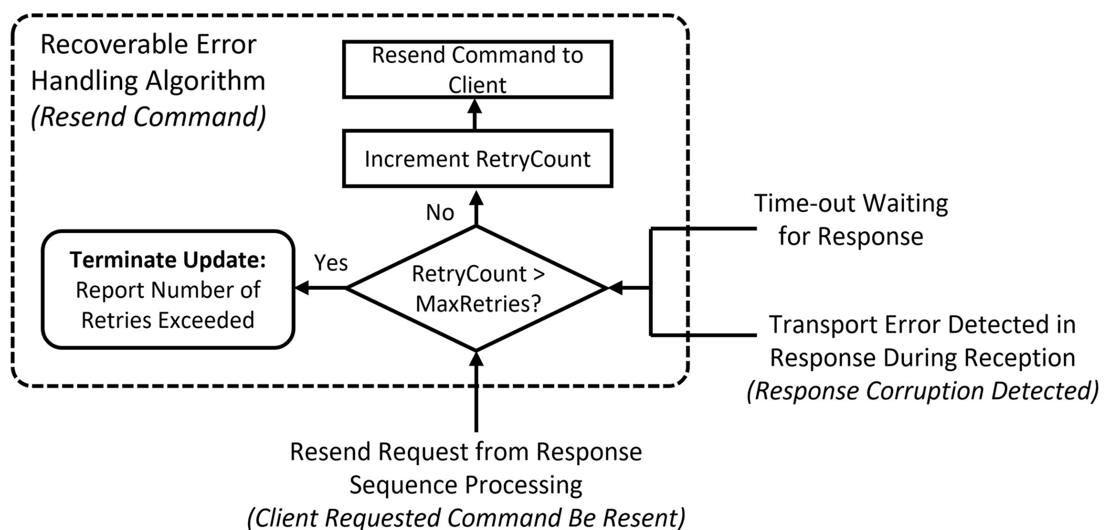
In order to recover from corruption, hosts are responsible for:

1. Resending commands when requested by the client.
2. Resending commands when responses are not received in the command time-out time period.
3. Detecting response corruption and resending the command to recover from this corruption.

The host recovery algorithm is very simple and it is shown in the diagram below.

The host must provide a mechanism to log all the errors detected and notify a use of the errors. The host must also have a configurable maximum retry count (MaxRetries) before terminating the update to help make link problems obvious to the user.

Figure 3-9. Host Error Recovery Algorithm

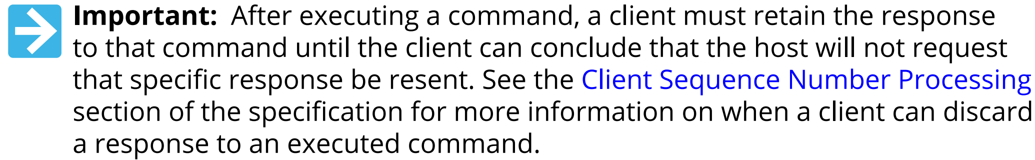


3.7.2.3 Client Recovery Algorithm

In order to recover from corruption, clients are responsible for:

© 2024 Microchip Technology Inc. and its subsidiaries

1. Detecting command corruption and requesting the host resend corrupted commands.
2. Implementing the receive sequence number filtering to ensure that commands are only executed once and are executed in the proper order.
3. Resending a response to the host without triggering a second execution of a command when the host is recovering from a corrupted response.



Important: After executing a command, a client must retain the response to that command until the client can conclude that the host will not request that specific response be resent. See the Client Sequence Number Processing section of the specification for more information on when a client can discard a response to an executed command.

The figure below shows how the client fulfills the three error recovery responsibilities described above.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-10. Client Recovery Algorithm

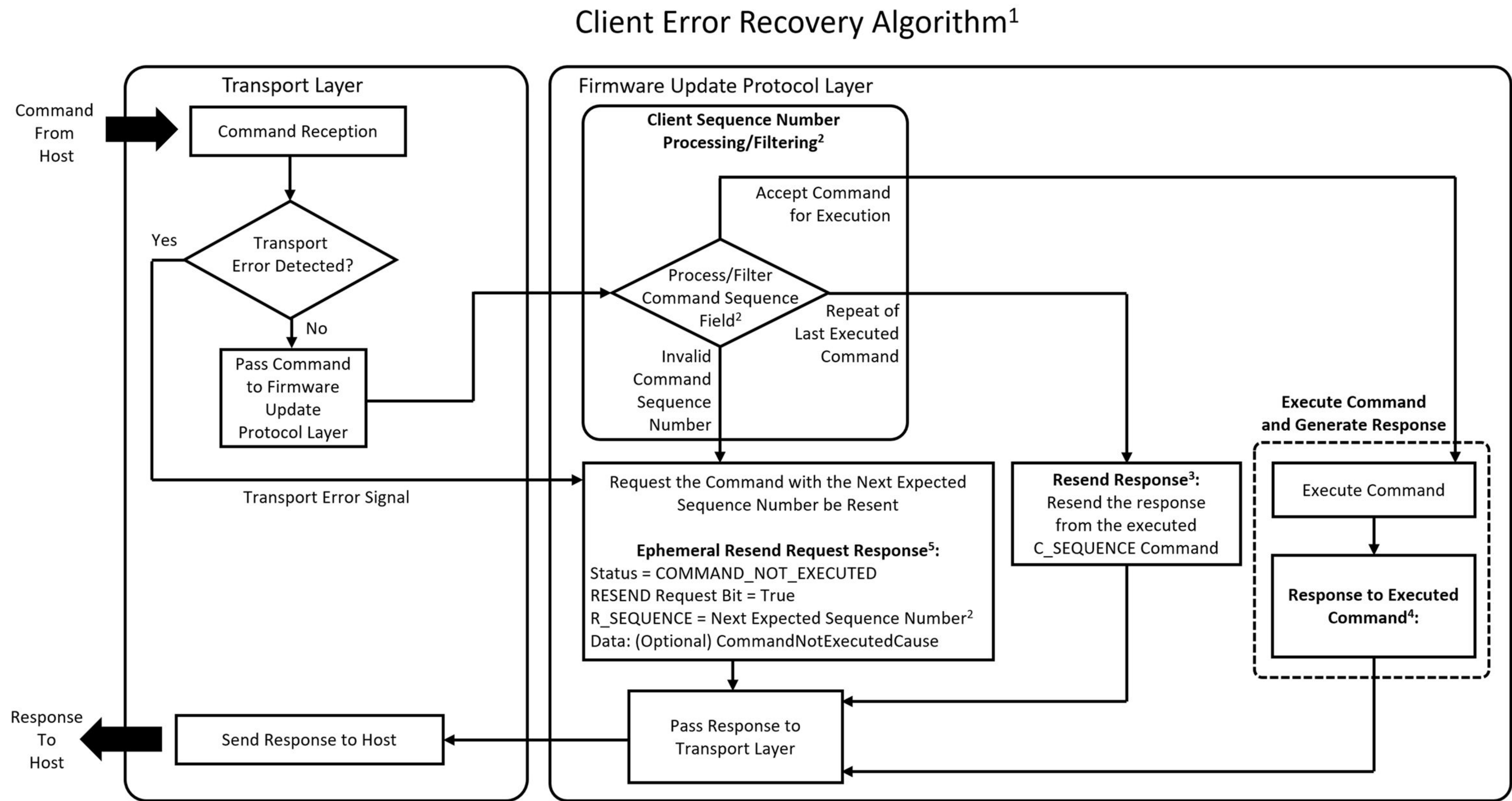


Figure Notes

Note 1: This figure only highlights the client algorithm details that allow clients to recover from errors. See the Client Command Processing and Response Generation section for more details on the complete client command processing and response generation algorithm.

Note 2: See Client Sequence Number Processing section of the specification for more information.

Note 3: The only responses resent by the client are responses to executed commands[4]. Ephemeral Resend Request Responses[5] are never resent by the client.

Note 4: Responses to executed commands must be retained by the client until the client can conclude that the host will not request for the response to be resent.

Note 5: Ephemeral Resend Request Responses are never resent by the client, and the client can discard them as soon as they have been sent to the host.

© 2024 Microchip Technology Inc. and its subsidiaries

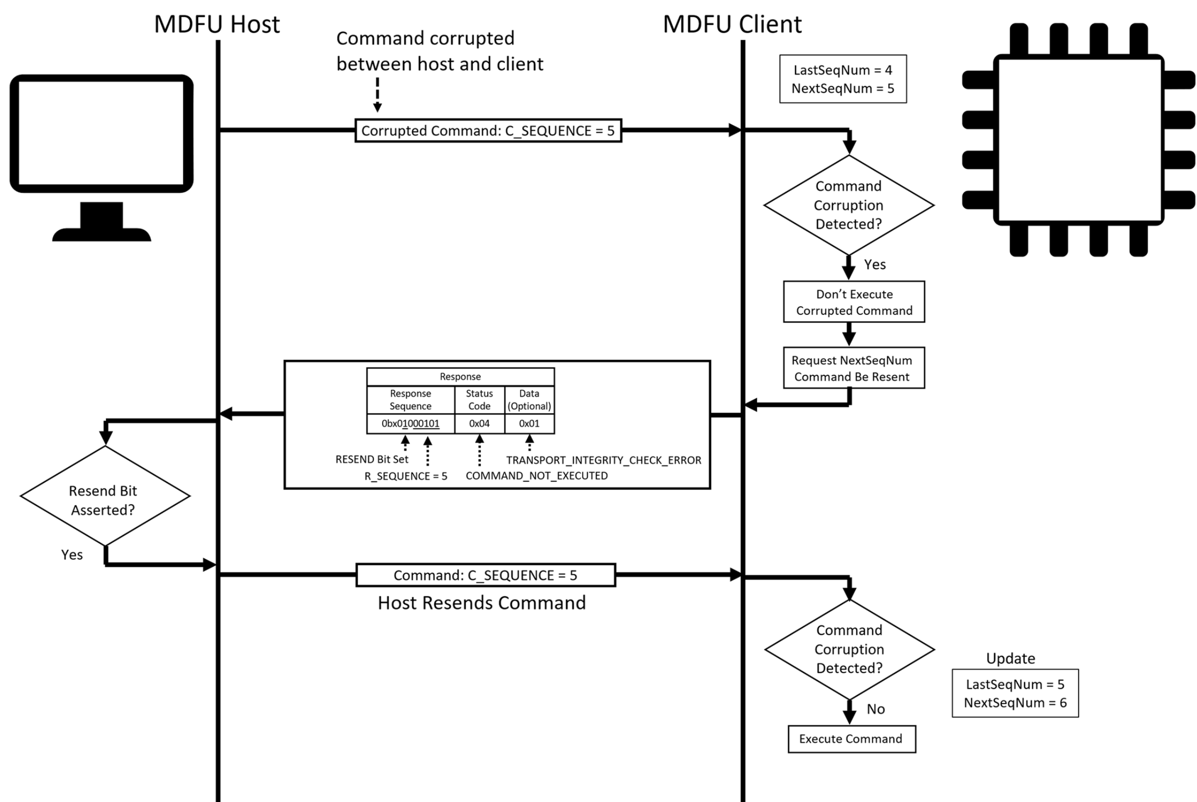
3.7.2.4 Detailed Recovery Diagrams

The following sections illustrate detailed sequence diagrams which show how the host and client recovery algorithms can recover from the following types of recoverable errors:

- Corrupted Command
- Corrupted Response
- Corrupted Command followed by Corrupted Response
- Corrupted Response followed by Corrupted Command
- Lost Command
- Lost Response

3.7.2.4.1 Recovering From a Corrupted Command

Upon detecting a corrupted command, the client sends a response with status COMMAND_NOT_EXECUTED and asserts the response sequence number RESEND bit. When the host sees the resend bit asserted, it resends the command to the client.

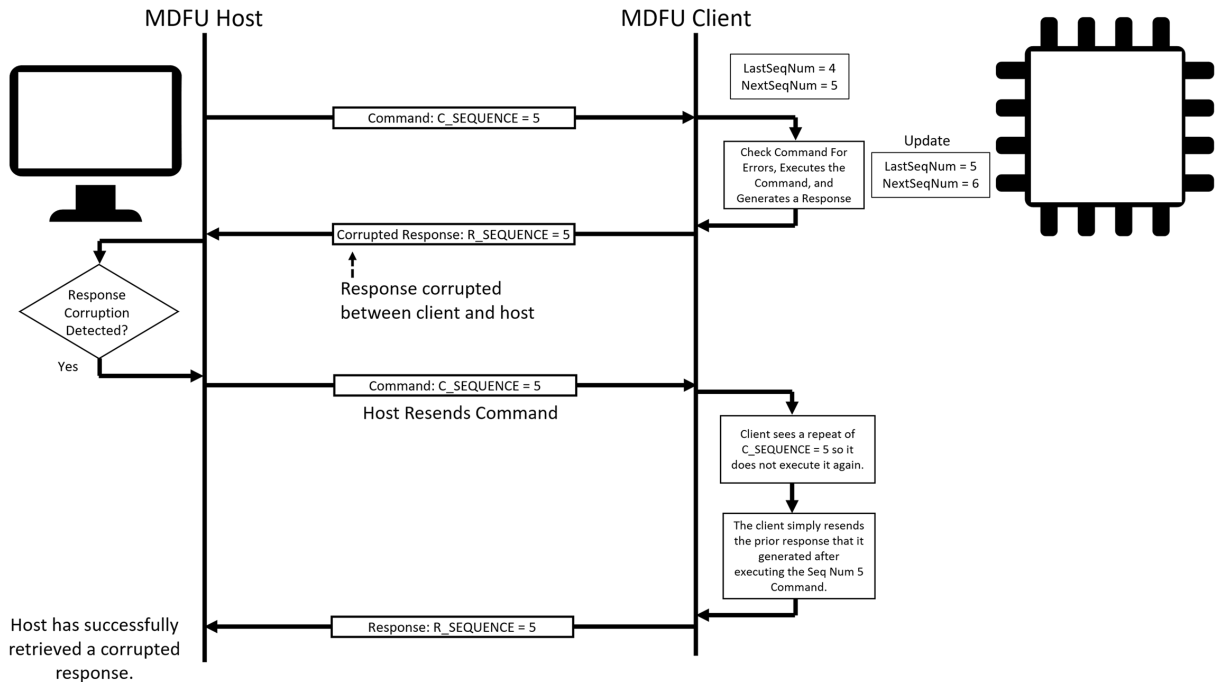


3.7.2.4.2 Recovering From a Corrupted Response

When the host detects a corrupted response, the host resends the last command sent.

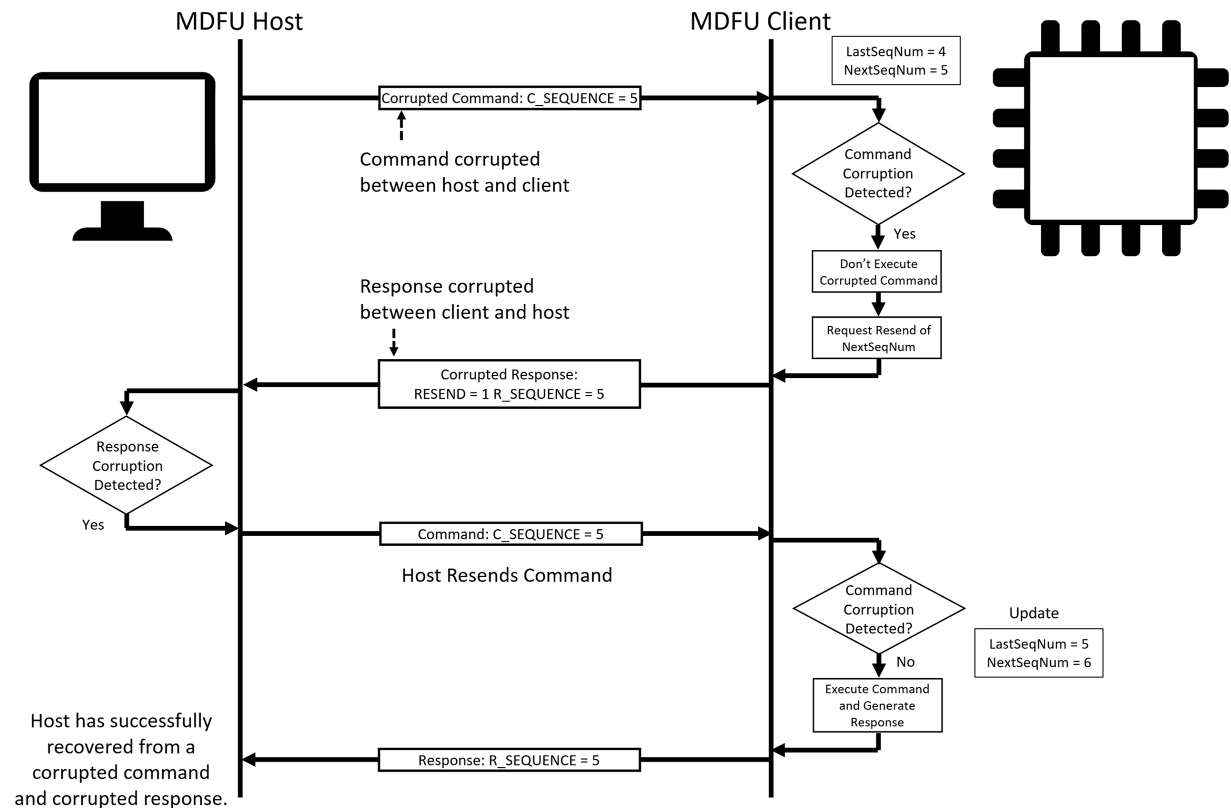
The client uses the repeated sequence number of the repeated command to avoid repeated execution and simply resends the corresponding response.

© 2024 Microchip Technology Inc. and its subsidiaries



3.7.2.4.3 Recovering From a Corrupted Command Followed by a Corrupted Response

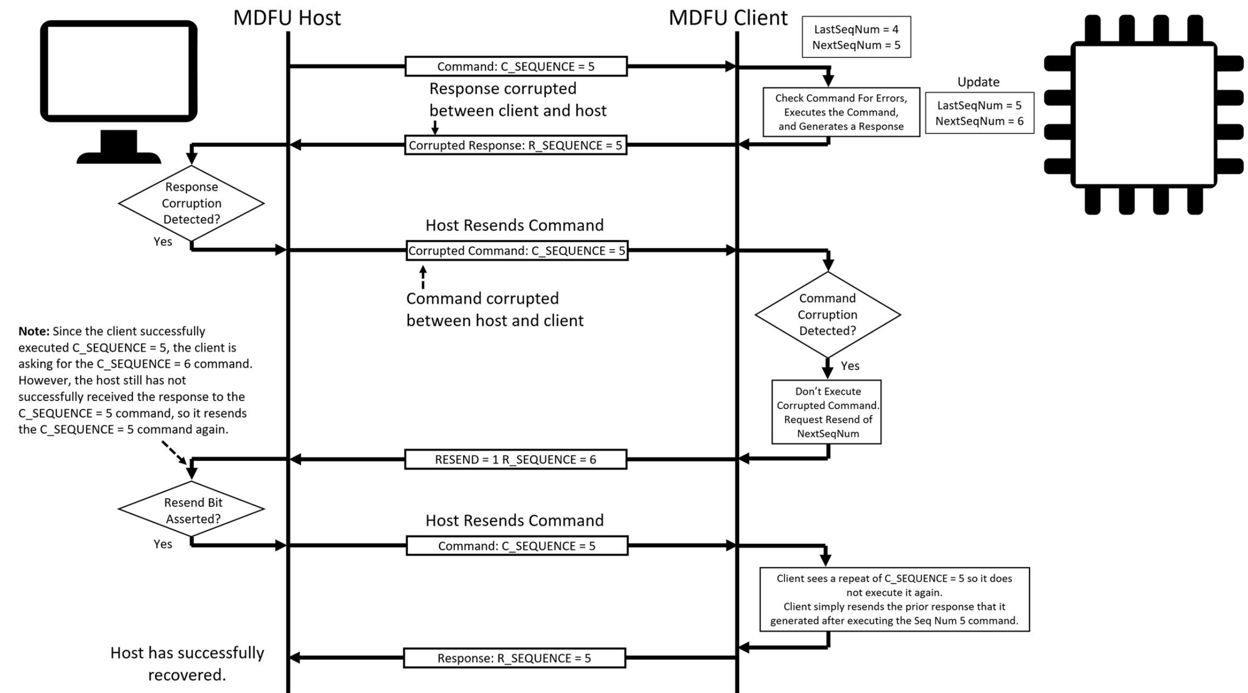
In a situation where a command is corrupted and the response to that command is also corrupted, the host resends the command. Upon receiving the command, the client executes the command and sends the response to the command allowing successful recovery from a corrupted command and corrupted response.



© 2024 Microchip Technology Inc. and its subsidiaries

3.7.2.4.4 Recovering From a Corrupted Response Followed by a Corrupted Command

The figure below illustrates how a host recovers from a situation where a response is corrupted and the command that is resent by the host also gets corrupted. The client detects the command corruption and sends a resend request for the NextSeqNum command. The host still has not successfully received a response for the last command it sent so it resends the command a second time. The client sees it has already executed this command and simply resends the response to that command without executing the command again. This process enables recovery from a situation where a response and subsequent command are both corrupted.

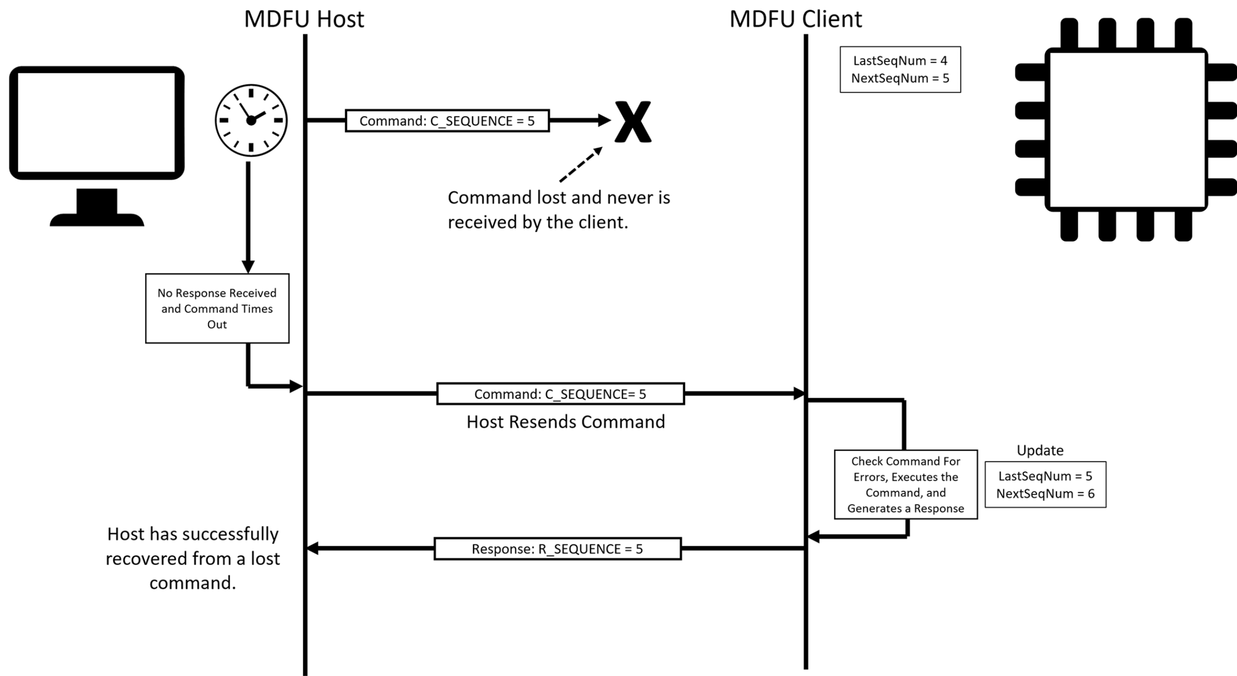


3.7.2.4.5 Recovering From a Lost Command

There are certain forms of command corruption which can occur on certain transport interfaces that can result in the client failing to completely receive a command. In these scenarios, the command is lost and the client never generates a response to the lost command.

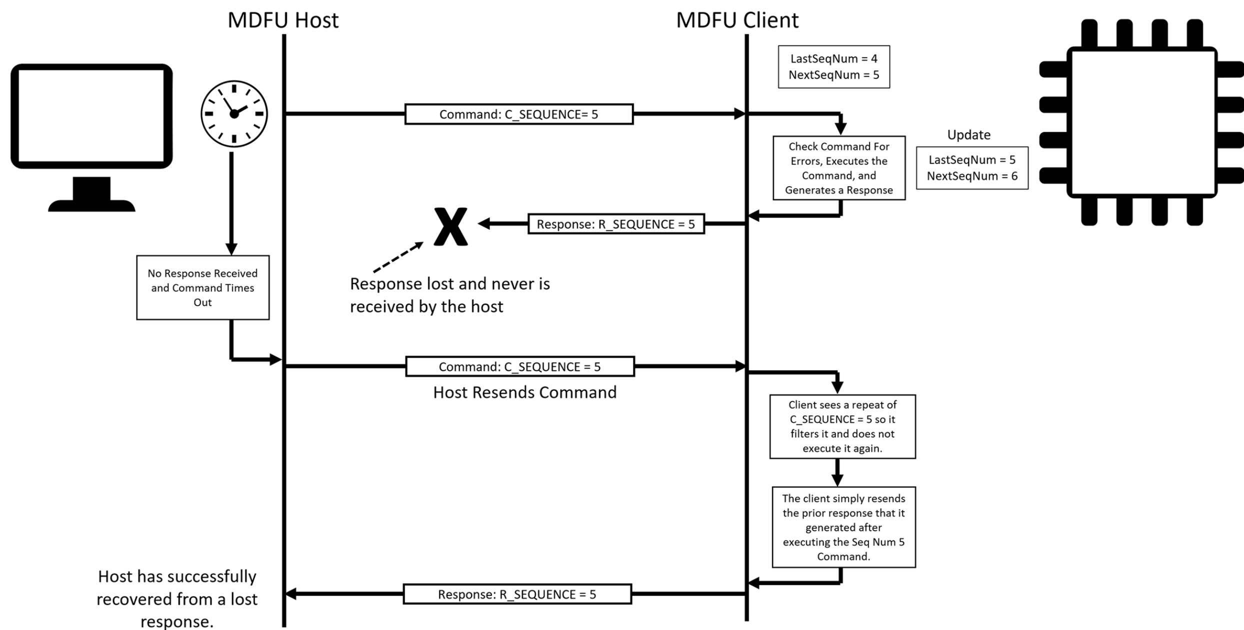
The host detects command loss when it times out waiting for a response from the client. At this point, the host can recover by resending the command to the client.

© 2024 Microchip Technology Inc. and its subsidiaries



3.7.2.4.6 Recovering From a Lost Response

The host detects response loss when it times out waiting for a response from the client. At this point, the host can recover by resending the command to the client.



3.7.3 Unrecoverable Errors

There are a number of different errors which can be encountered during a firmware update which can't be recovered by an automatic recovery mechanism. These errors are classified by the MDFU Protocol as unrecoverable errors.

© 2024 Microchip Technology Inc. and its subsidiaries

Since there is no automatic recovery mechanism possible for an unrecoverable error, the MDFU Protocol provides a mechanism for stopping the update immediately when an unrecoverable error has been detected by the client and reporting back information on why the update was terminated.

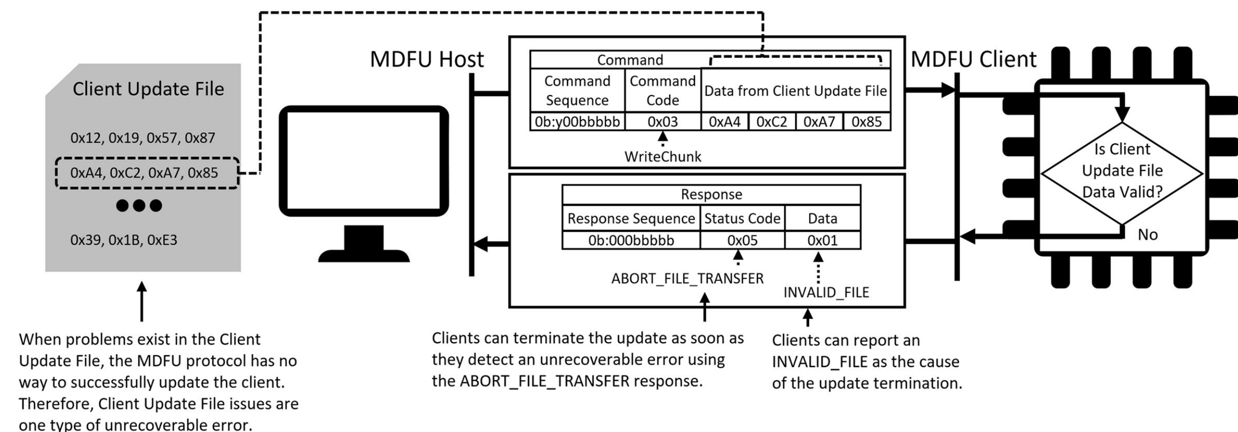
Unrecoverable errors are detected by the MDFU client and reported back to the host using the abort file transfer mechanism defined in this section of the specification. There are many different types of unrecoverable errors which clients could potentially detect. Checking for unrecoverable errors is an optional MDFU Protocol feature for clients to implement. Clients can choose to implement as many or as few checks for unrecoverable errors and report them as desired. Adding client checks for unrecoverable errors is a tradeoff that can be selected by the client developer. Adding more checks consumes more memory for the client implementation but has the following benefits:

1. Improved Troubleshooting and Debugging
   – The process of aborting a file transfer allows the client to report a cause that the update failed which helps debug the issue.
2. Time Savings
   – Aborting an update as soon as an error is detected eliminates wasted time which would otherwise occur if the protocol waited until the end of the file transfer to learn that the update failed. This can help eliminate wasted time in production line update environments.

3.7.3.1 Example 1: Invalid Update File

The process of performing an update involves the host transferring the Client Update File to the client. The Client Update File needs to contain data that has a valid application for the client in a valid format for the client to use the data in the Client Update File to successfully update the client.

The MDFU Protocol is responsible for reliably transporting the data in the Client Update File to the client. However, an invalid Client Update File will fail to produce a successful client update, and there is nothing the MDFU Protocol or MDFU host can do to fix or recover from a problem that exists with the contents of the Client Update File. This is an example of an unrecoverable error.



Clients may optionally choose to implement checks for various problems with the Client Update File. If a client detects a problem with the data in the Client Update File, it can abort the file transfer as soon as an error is detected and report a reason why it aborted the transfer.

There are a number of types of problems that could exist with the data in the Client Update File. For example, a Client Update File could contain invalid memory addresses or have an invalid application version. See the FileAbortClause section for a list of the available file problems a client can report.

3.7.3.2 Example 2: Client Flash Memory Issues

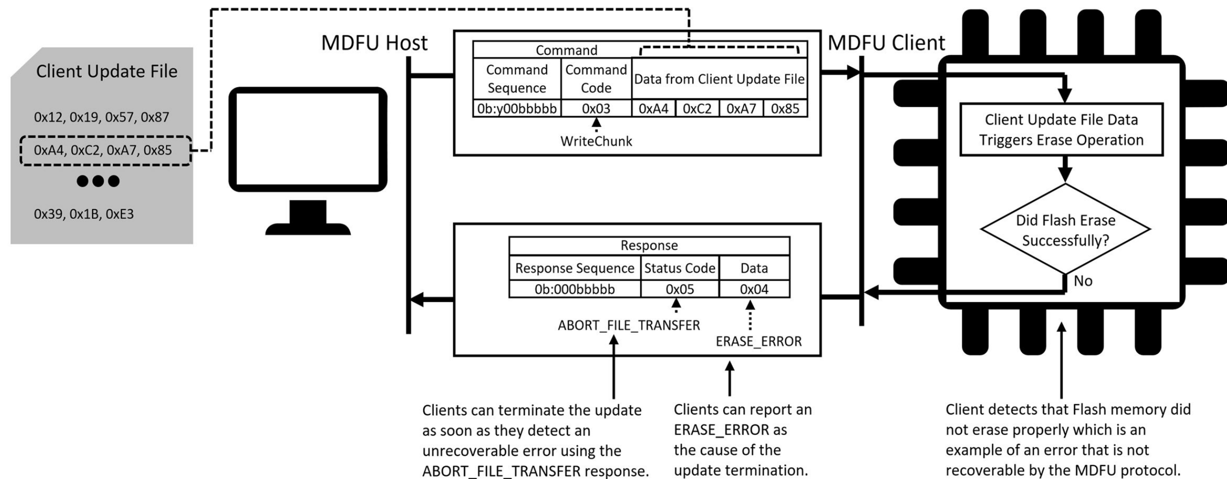
Performing a firmware update involves erasing and writing nonvolatile memory (NVM) on a client processor. There are scenarios where a client's Flash memory may fail to fully erase properly or fully write to the proper values for various reasons. Clients may optionally choose to verify that the result

© 2024 Microchip Technology Inc. and its subsidiaries

of erase/write operations matches the desired values. When a client has detected that Flash fails to erase/write properly, the MDFU Protocol recommends that the client abort the update process for two reasons:

1. Users will want to be aware of client Flash issues as opposed to trying to silently recover from them.
2. Placing automatic retry loops on re-executing erase/write operations could result in a large number of erases/writes to client Flash which could wear it out without the user being aware.

For these reasons, the MDFU Protocol doesn't include any automatic recovery mechanisms for Flash memory issues, and they are considered unrecoverable errors.



3.7.3.3 Detection

The responsibility of detecting unrecoverable errors is placed on the client.

Clients can choose to check for as many or few unrecoverable errors as are useful for the end application. This allows for the client developer to choose the best balance of client code size vs troubleshooting and debug experience for a particular design.

3.7.3.4 Aborting A Transfer

At any point in the update after an unrecoverable error has been detected by the client, the client can respond to a command with a response containing the ABORT_FILE_TRANSFER status and report any of the available FileAbortCause reasons for aborting the update.

3.8 Sequence Numbers

The MDFU Protocol utilizes command-response pairs to transfer signals and data between the host and client. Sequence numbers provide a way for the host/client to specifically identify commands and responses as being part of a command-response pair. The sequence fields also provide some additional control information.

Sequence numbers allow the following functionality:

1. Hosts can properly associate responses from the client with specific commands that were sent from the hosts.
2. Clients can ensure that all commands are executed in the proper order and are only executed once.
3. Clients can request a specific corrupted command be resent by the host.
4. Hosts can request a corrupted response be resent without triggering a repeated command execution.

© 2024 Microchip Technology Inc. and its subsidiaries

Please refer to the following sections for the host's algorithm for determining the sequence field value for commands and the client's algorithm for processing sequence numbers.

3.8.1 Command Sequence Field Definition

Command Sequence Field:

Bit	7	6	5	4	3	2	1	0
	SYNC	Reserved: 0	Reserved: 0	C_SEQUENCE[4:0]				

Bit 7 – SYNC

The host sets the SYNC bit to 1 to synchronize the client's sequence number to the host's sequence number. The host must set SYNC=1 for the first command of an update. The host must clear SYNC=0 for all subsequent commands in the update.

Value	Description
0x0	The client uses the C_SEQUENCE number to determine if it should accept the command for execution, resend a prior response, or reject the command due to an invalid C_SEQUENCE.
0x1	The client synchronizes its sequence number to the number contained in the C_SEQUENCE bit field. The client accepts this command for execution.

Bit 6 – Reserved

Reserved for future use. Hosts must set this bit to 0x0.

Bit 5 – Reserved

Reserved for future use. Hosts must set this bit to 0x0.

Bits 4:0 – C_SEQUENCE

The C_SEQUENCE bit field contains the sequence number of the command.

© 2024 Microchip Technology Inc. and its subsidiaries

3.8.2 Response Sequence Field Definition

Response Sequence Field:

Bit	7	6	5	4	3	2	1	0
	Reserved: 0	RESEND	Reserved: 0	R_SEQUENCE[4:0]				

Bit 7 – Reserved

Reserved for future use. Clients must set this bit to 0x0.

Bit 6 – RESEND

The client uses the RESEND bit to either request that a specific command be resent by the host or provide a response to the command with C_SEQUENCE[4:0] equal to the response sequence number R_SEQUENCE[4:0].

Value	Description
0x0	The client is responding to the command with C_SEQUENCE[4:0] equal to the response R_SEQUENCE[4:0] of this response.
0x1	The client is requesting the host resend the command with C_SEQUENCE[4:0] equal to the R_SEQUENCE[4:0] of this response.

Bit 5 – Reserved

Reserved for future use. Clients must set this bit to 0x0.

Bits 4:0 – R_SEQUENCE

The R_SEQUENCE bit field contains the sequence number of the response.

3.8.3 Host Command Sequence Number Selection

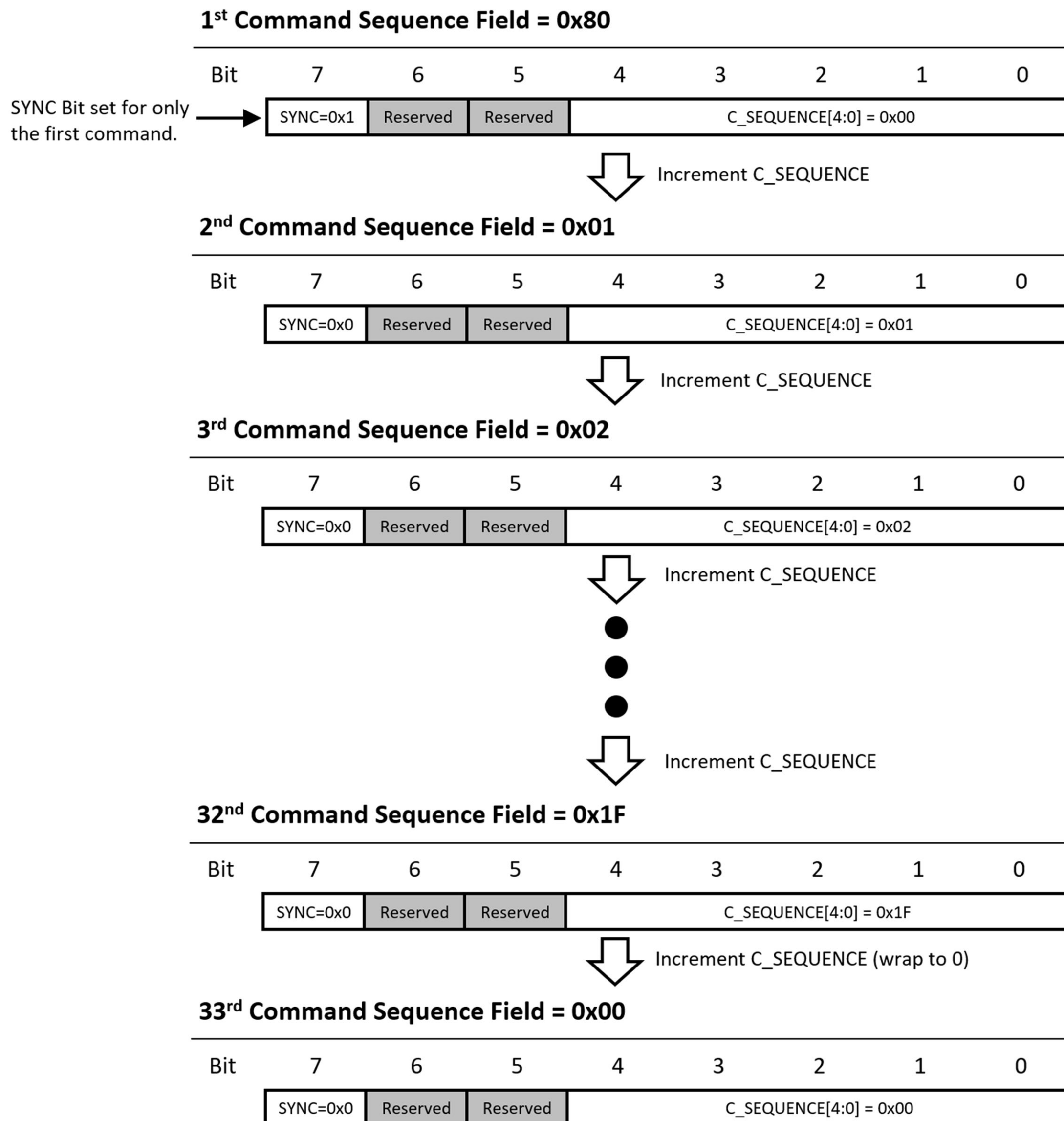
The host follows the following procedure when selecting the sequence numbers for commands.

1. First Command:
   - The sequence number field provides the client with a mechanism to filter incoming commands to ensure commands occur once and in the proper order.
   - When an update starts, the host doesn't know what the client's next expected sequence number is. To resolve this situation, the protocol provides a mechanism for the host to synchronize the client's sequence number.
   - For only the first command from the host to the client of an update attempt, the host sets the SYNC bit of the sequence field to 1 and sets the sequence number to 0.
   - When the client sees that the SYNC bit is set to 1, the client overrides its current sequence number setting with the value of the sequence number in the command from the host which is 0.
2. All Subsequent Commands:
   - After the host synchronizes the client sequence number with the client on its first command. All subsequent new commands must increment the sequence number by one and clear the SYNC bit to 0.
   - The sequence number is five bits and wraps back around to 0 after overflowing the five bits.

3.8.3.1 Command Sequence Number Example

The following figure shows an example of the sequence numbers that are selected by the host for commands.

© 2024 Microchip Technology Inc. and its subsidiaries



3.8.4 Client Sequence Number Processing

The client implements filtering logic to make sure that each command is executed in order and only executed once.

The client keeps track of the sequence number of the last command executed (LastSeqNum) and the expected sequence number of the next command (NextSeqNum).

Prior to the client performing any command sequence number processing/filtering, the command must have already passed the Transport Layer integrity checks. See the Client Command Processing and Response Generation section of the specification for more details on when the sequence number processing/filtering occurs.

The client uses the following checks/logic to accomplish the following tasks.

- Client Sequence Number Synchronization:
  - When an update starts, the host doesn't know what the client's LastSeqNum and NextSeqNum value are. Since clients reject incoming commands that don't have

© 2024 Microchip Technology Inc. and its subsidiaries

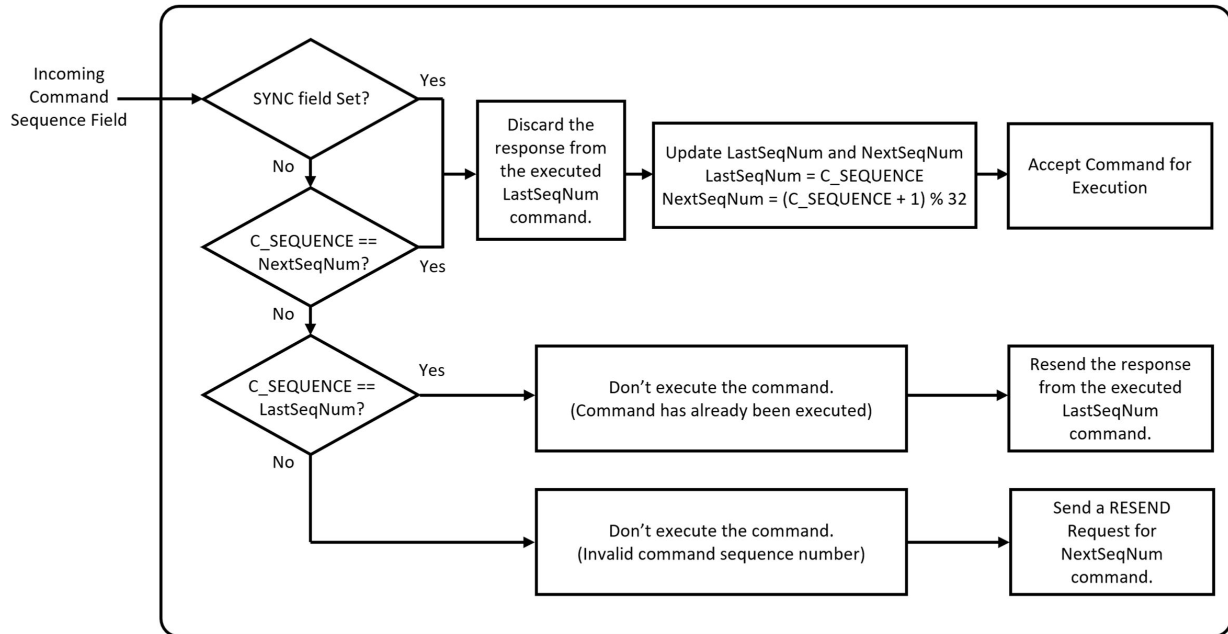
C_SEQUENCE equal to NextSeqNum, the host uses the synchronization mechanism to set the client's sequence number.
  - The first command (and only the first command) of the update that the host sends to the client has the SYNC bit set to 1 which tells the client to synchronize its sequence number to the host sequence number.
  - The client accepts this command for execution.
- Valid Sequence Number Command Processing:
  - When the command from the host has C_SEQUENCE equal to NextSeqNum, the client accepts this command for execution.
- Corrupted Response Retrieval:
  - When a response is corrupted, the host resends the command to the client. If the client has already executed this command (C_SEQUENCE equals LastSeqNum), the client must prevent repeated execution of the command and simply resend the response from the first time that command was executed.
  - **IMPORTANT**: After executing a command, a client must retain the response to that command until the client can conclude that the host will not request that specific response be resent. Clients can discard the response to the LastSeqNum command only when they receive a new command with C_SEQUENCE equal to the next expected sequence number (NextSeqNum) or when the host synchronizes the sequence number.
- Invalid Sequence Number Signaling:
  - If the host ever sends a command with the SYNC bit cleared to 0 and with a sequence number different from NextSeqNum or LastSeqNum, this sequence number is invalid and the command must not be executed.
  - Clients must reply to the host and request that the command with the next expected sequence number (NextSeqNum) be resent by the host. This is accomplished by creating an ephemeral resend request response:
    - Response Sequence
      - RESEND = 1
      - R_SEQUENCE = NextSeqNum
    - Response Status = COMMAND_NOT_EXECUTED
    - Response data can either be empty or contain the SEQUENCE_NUMBER_INVALID cause code
  - This ephemeral resend request response is not the result of executing a command, and the client does not need to retain this response after it has been sent as the client will never resend this response.

3.8.4.1 Processing Algorithm Diagram

The following figure defines the client sequence number processing/filtering algorithm.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 3-11. Client Sequence Number Processing/Filtering

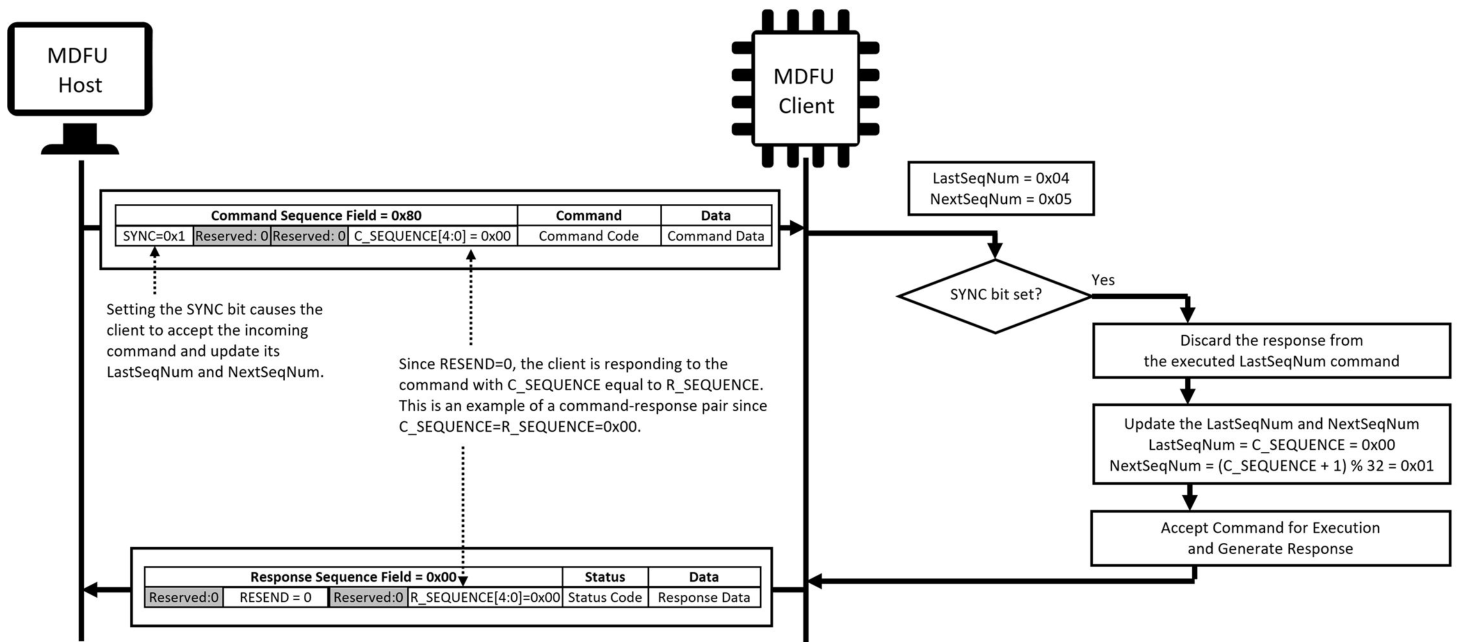


3.8.4.2 Client Sequence Number Processing Examples

The following examples show the details of the command and response sequence fields for the four scenarios discussed above.

3.8.4.2.1 Client Sequence Number Synchronization Example

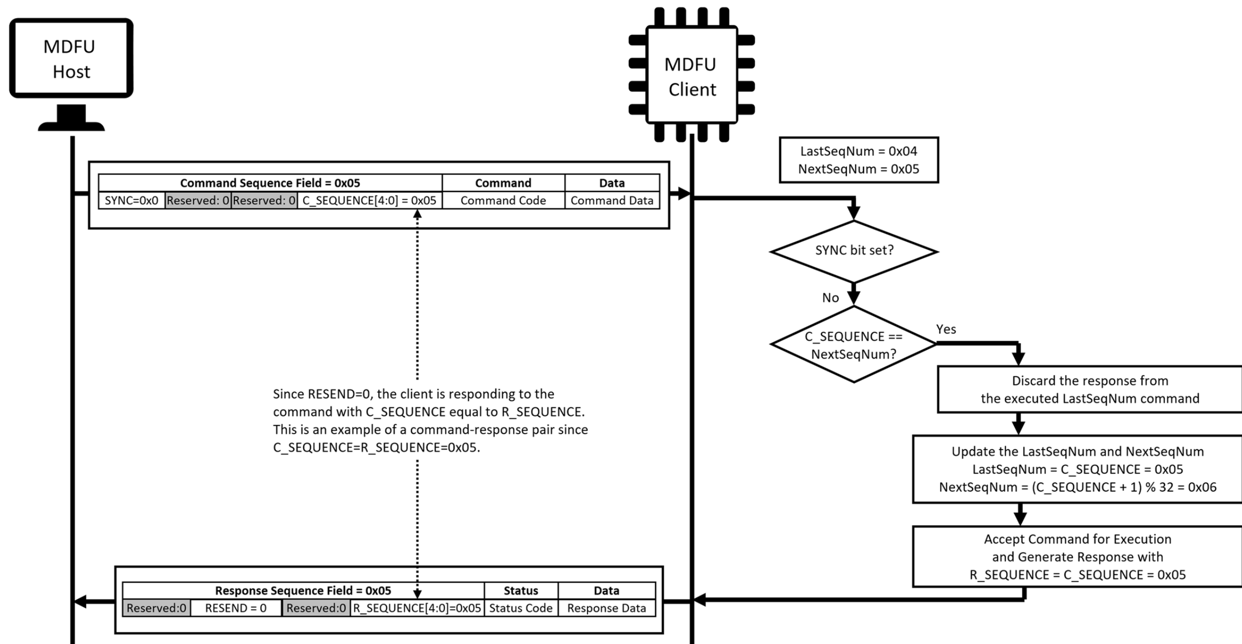
Figure 3-12. Client Sequence Number Synchronization



© 2024 Microchip Technology Inc. and its subsidiaries

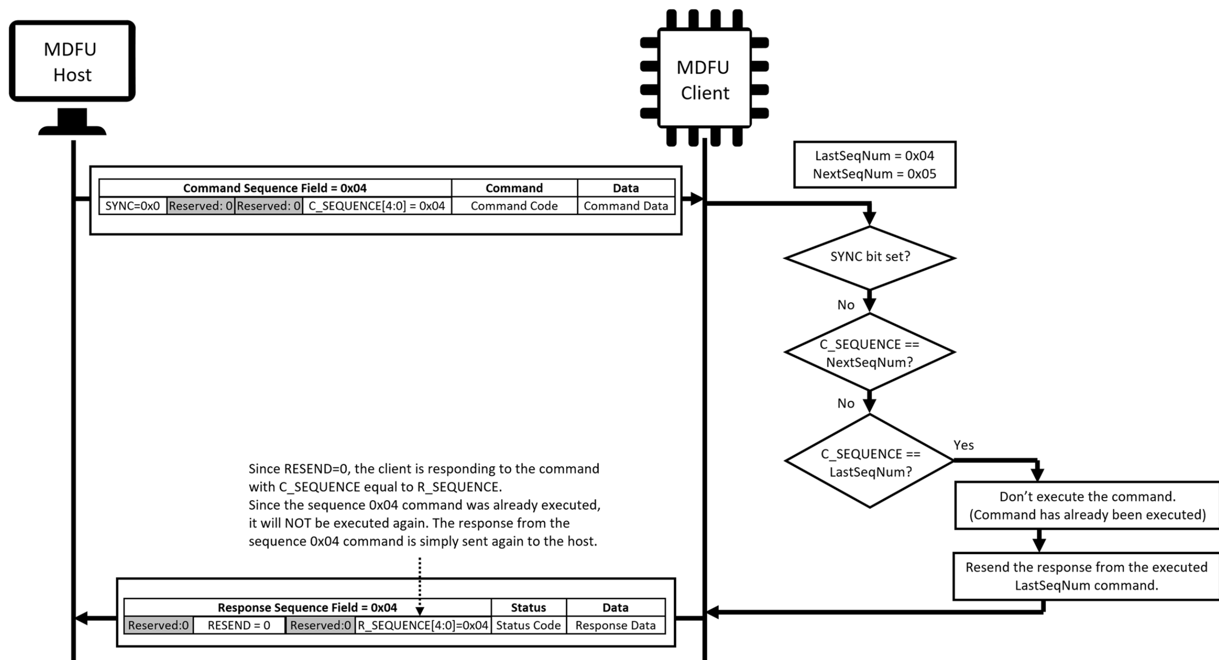
3.8.4.2.2 Valid Sequence Number Command Processing Example

Figure 3-13. Client Receiving Next Expected Sequence Number



3.8.4.2.3 Corrupted Response Retrieval Example

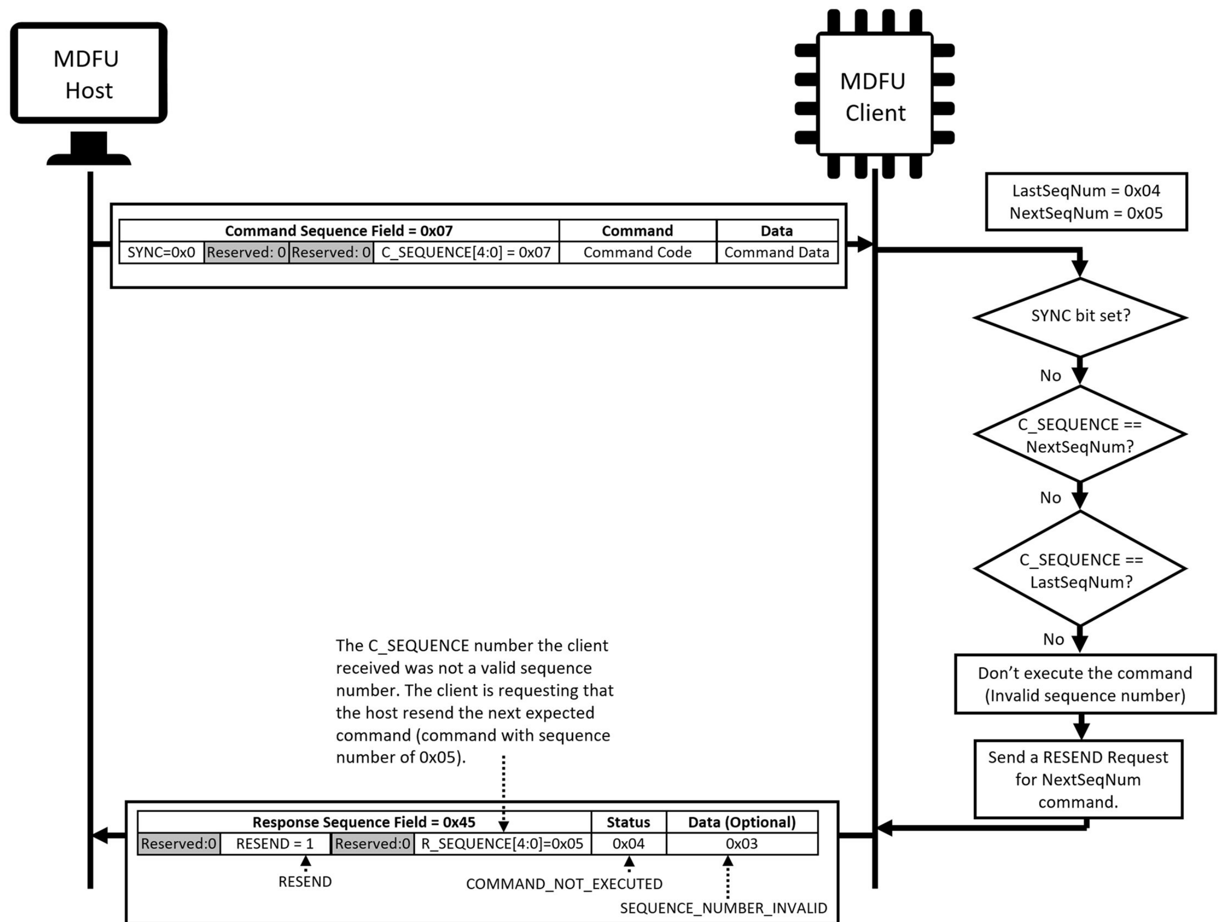
Figure 3-14. Client Resends Response from LastSeqNum Command



© 2024 Microchip Technology Inc. and its subsidiaries

3.8.4.2.4 Invalid Sequence Number and Resend Request

Figure 3-15. Client Requests the Next Expected Command from Host



© 2024 Microchip Technology Inc. and its subsidiaries

4. Transport Layer

The MDFU Protocol is divided into two layers (see MDFU Protocol Layers.) The Firmware Update Protocol Layer defines all the protocol details that are identical for every physical communication interface. The Transport Layer defines the protocol details which may be unique to a particular physical communications bus.

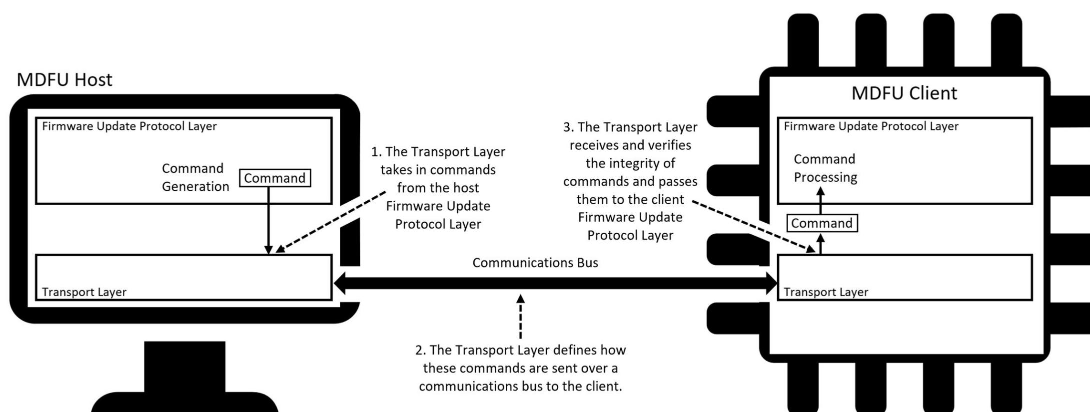
This section of the MDFU specification provides the Transport Layer definitions for different communications busses. These Transport Layer definitions specify how commands and responses can be sent/received over different communications busses.

4.1 Transport Layer Responsibilities

Each individual communications bus transport definition specifies details shown in the following sections.

4.1.1 Command Transport from Host to Client

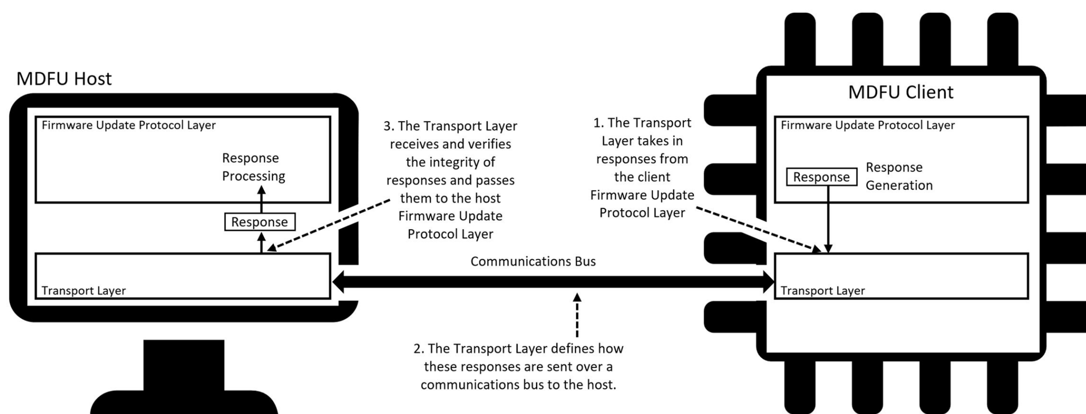
1. Commands are generated by the Firmware Update Protocol Layer and passed to the Transport Layer.
2. Commands are sent over the communications bus from the host to the client.
3. The client Transport Layer receives the commands and verifies their integrity before passing them to the Firmware Update Protocol Layer.



4.1.2 Response Retrieval from the Client

1. The client's Firmware Update Protocol Layer generates responses and passes them to the Transport Layer.
2. Responses are sent over the communications bus from the client to the host.
3. The host's Transport Layer retrieves responses and verifies their integrity before passing them up to the host's Firmware Update Protocol Layer.

© 2024 Microchip Technology Inc. and its subsidiaries



4.1.3 Command and Response Integrity Checks

- The MDFU Protocol places the responsibility of detecting corruption in commands/responses (i.e. bit flips due to noise) on the Transport Layer
- When corruption is detected, the Firmware Update Protocol Layer is notified so that the error recovery mechanisms can be utilized to recover from these errors

4.1.4 Host Transport Layer Response Retrieval Time-out

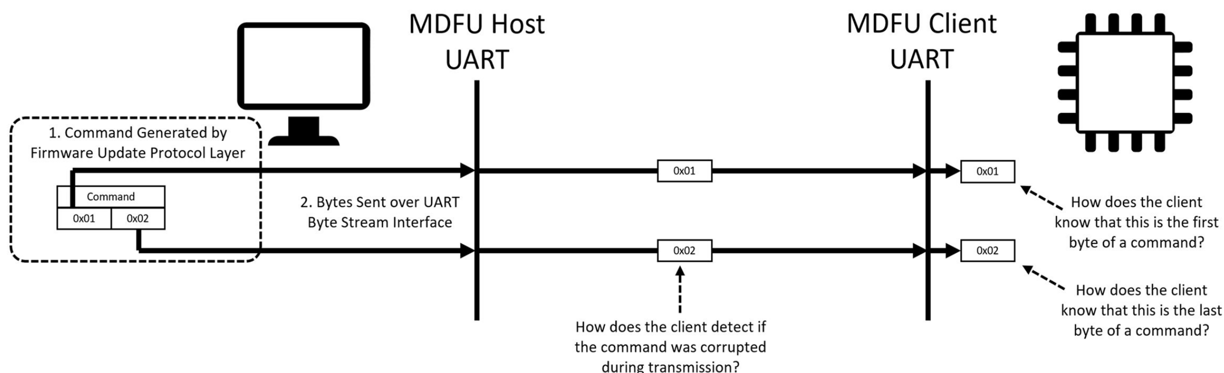
Hosts use a time-out to recover from commands or responses that are lost. Hosts can choose whether they want to implement that time-out in the host Firmware Update Protocol Layer or the host Transport Layer. See Host Command Generation and Response Processing for more details.

4.2 UART Command/Response Transport Layer

The MDFU UART Transport Layer is responsible for reliably transporting commands and responses across a UART byte stream interface.

Transmitting a command across a UART byte stream interface results in several challenges when receiving a command. There's a similar set of challenges when the host is receiving responses.

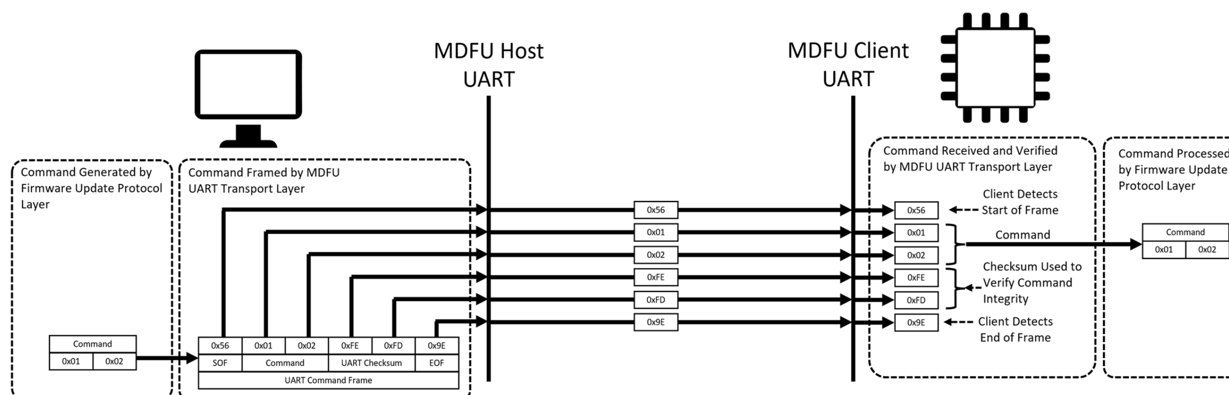
- How does the client know where the first byte of the command is in the byte stream?
- How does the client know where the last byte of a command is in the byte stream?
- How does the client detect if the command was corrupted during transmission?



The following sections of the MDFU UART Transport Layer describe a UART command-response framing mechanism which allows the detection of the start and end of a command/response and also allows for the detection of corruption of commands and responses.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 4-1. MDFU UART Command Framing Example



4.2.1 UART Configuration

The MDFU UART Transport Layer has been designed to operate over a generic byte stream of 8-bit data transfers without any parity field as defined below.

Minimum UART Configuration Requirements:

- Bidirectional
- Full Duplex
- Asynchronous
- Data Rate: Any Data Rate
- Data Width: 8 Bits
- Parity: None
- Stop Bits: 1 or more

4.2.2 Host Response Retrieval

When clients receive a command, they execute it and generate a response. Since the UART Configuration is an asynchronous, bidirectional, full duplex link, clients can transmit a response back to the host as soon as the response is ready.

The host simply waits for a response to come back from the client after it has sent a command to the client.

Hosts use a time-out to recover from commands or responses that are lost. Hosts can choose whether they want to implement that time-out in the host Firmware Update Protocol Layer or the host Transport Layer. See Host Command Generation and Response Processing for more details.

4.2.3 UART Command/Response Framing

This section defines the MDFU UART framing mechanism which accomplishes the following goals:

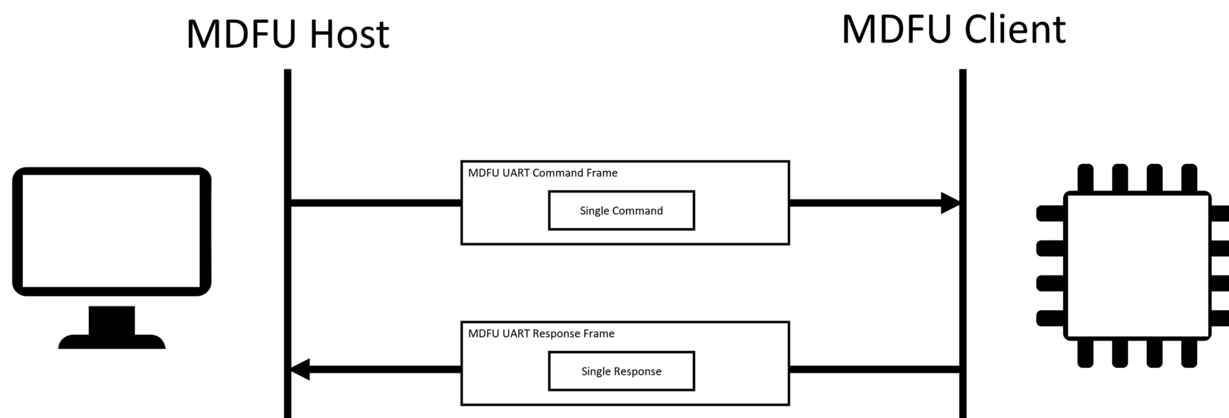
1. Identifies the start of a command or response in the UART byte stream.
2. Identifies the end of a command or response in the UART byte stream.
3. Provides a mechanism for detecting corruption of commands and responses.

4.2.3.1 One Command or Response per UART Command/Response Frame

MDFU UART command/response frames are used to transport commands from the host to the client and responses from the client to the host. MDFU UART command/response frames always contain a single complete command or single complete response in its entirety. Commands and responses are never split across multiple MDFU UART command/response frames. Multiple

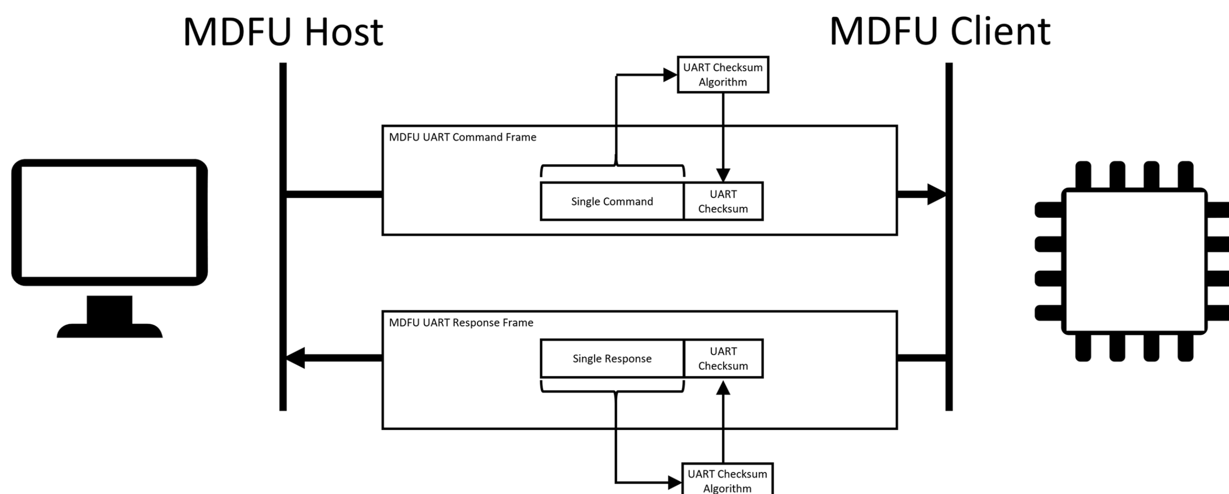
© 2024 Microchip Technology Inc. and its subsidiaries

commands and multiple responses are never put into a single MDFU UART command/response frame.



4.2.3.2 UART Checksum for Command/Response Frame Corruption Detection

The MDFU UART Transport Layer has been designed to operate over a UART Configuration which does not provide any error detection included as part of the UART physical layer. The MDFU UART Transport Layer appends a 16-bit checksum to the end of the command/response to provide a mechanism for detecting command/response corruption.

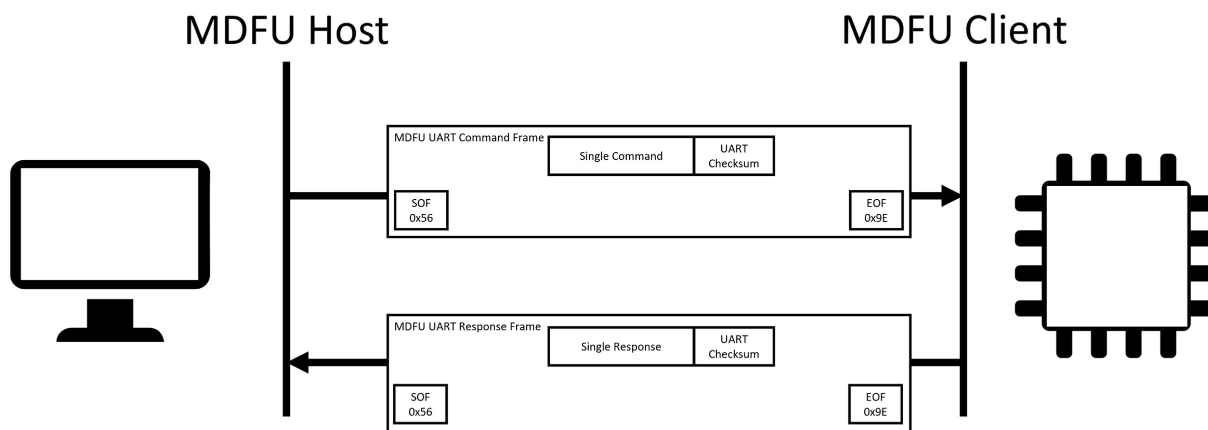


4.2.3.3 Start-Of-Frame and End-Of-Frame

The MDFU UART Transport Layer reserves the byte code 0x56 to mark the start of a UART command/response frame. Any and every time a 0x56 is detected in the UART byte stream, this means that a UART command or response frame is starting. The code 0x56 is not allowed to exist anywhere else in the byte stream.

The MDFU UART Transport Layer reserves the byte code 0x9E to mark the end of a UART command/response frame. Any and every time a 0x9E is detected in the UART byte stream, this means that a UART command or response frame is ending. The code 0x9E is not allowed to exist anywhere else in the byte stream.

© 2024 Microchip Technology Inc. and its subsidiaries



4.2.3.4 Reserved Code Substitutions

Commands, responses, and UART checksums may be composed of bytes containing the codes 0x56 and 0x9E, but those codes are not allowed to be transmitted in the byte stream except to mark the start or end of a command/response frame. Whenever a 0x56 or 0x9E are present in the commands, responses, and UART checksums, those bytes must be substituted with an alternative representation of those reserved byte codes.

The UART Transport Layer framing provides a mechanism for representing reserved byte codes which appear in the commands, responses, and checksums by substituting the reserved byte code with a two byte sequence.

- The first byte code in the two byte sequence is the escape code (0xCC).
- The second byte code in the two byte sequence is the one's compliment of the code being represented.

The following figure shows the reserved codes along with the 2-byte substitutions for those reserved codes.

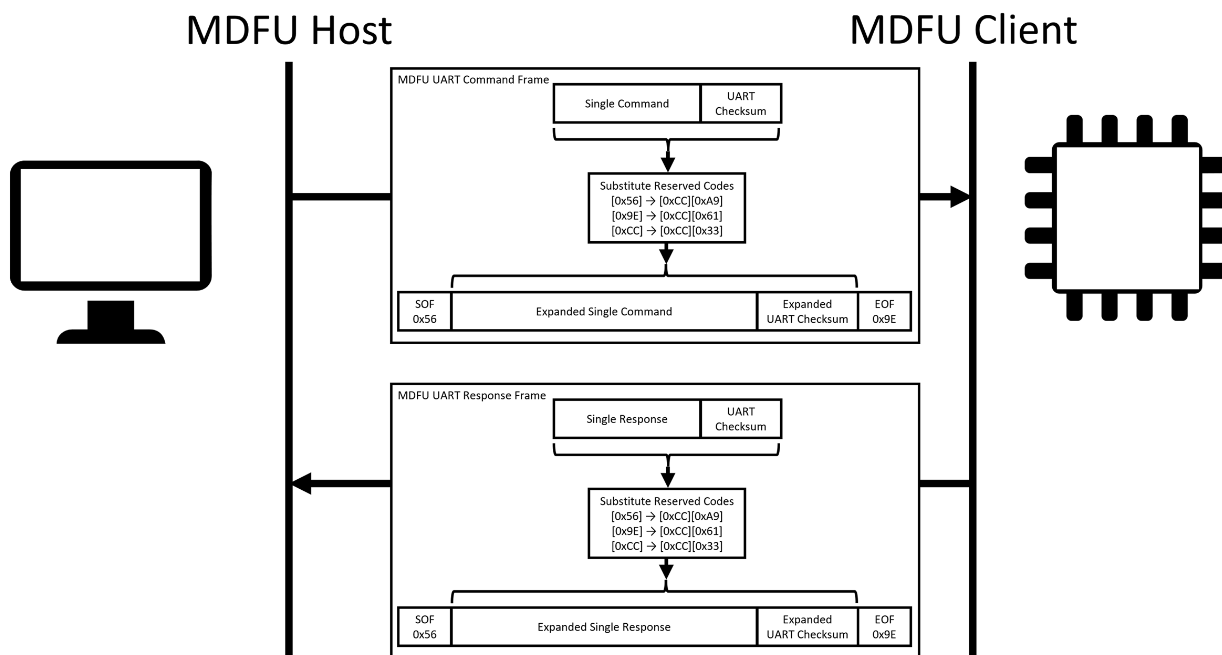
Reserved Codes and 2-Byte Substitutions

Code Name	Reserved Code	2-Byte Substitution	Description
SOF	[0x56]	[0xCC][0xA9]	Start of Frame
EOF	[0x9E]	[0xCC][0x61]	End of Frame
ESC_CODE	[0xCC]	[0xCC][0x33]	Escape Sequence Code

The following figure shows how commands, responses, and checksums go through the reserved code substitutions when being placed in the MDFU UART command/response frame.

This is the final form of the command and response frames which are transmitted over the UART byte stream.

© 2024 Microchip Technology Inc. and its subsidiaries

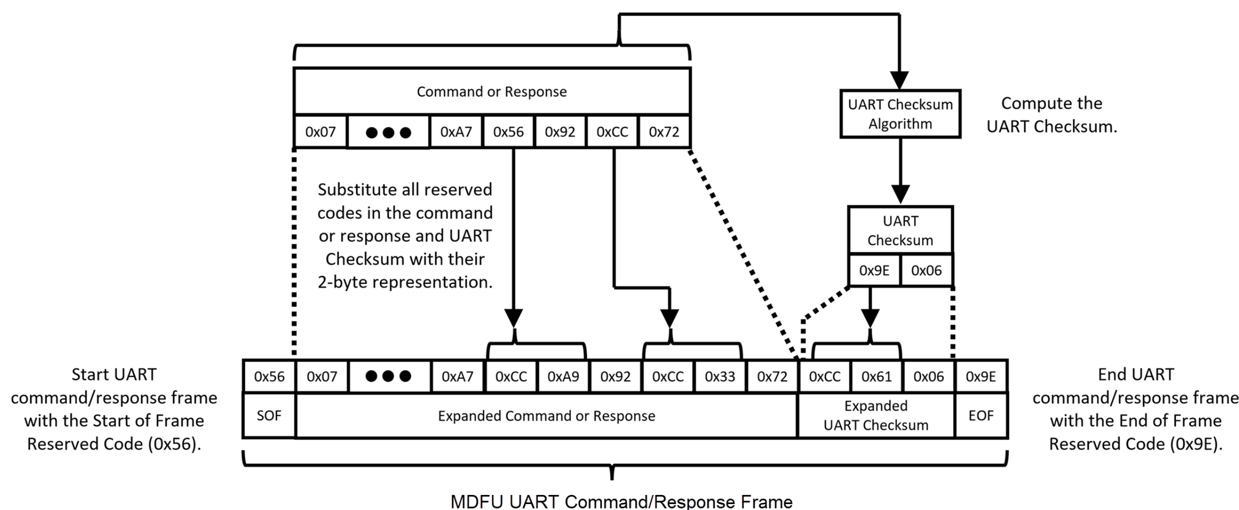


4.2.4 UART Command/Response Frame Creation

Creating a UART command or response frame involves the following steps:

1. Start each command/response frame with the Start-Of-Frame code.
2. Calculate the UART Checksum.
3. Substitute all the reserved codes in the command/response and checksum by replacing them with their 2-byte representation.
4. End the command/response frame with the End-Of-Frame code.

Figure 4-2. MDFU UART Command/Response Frame



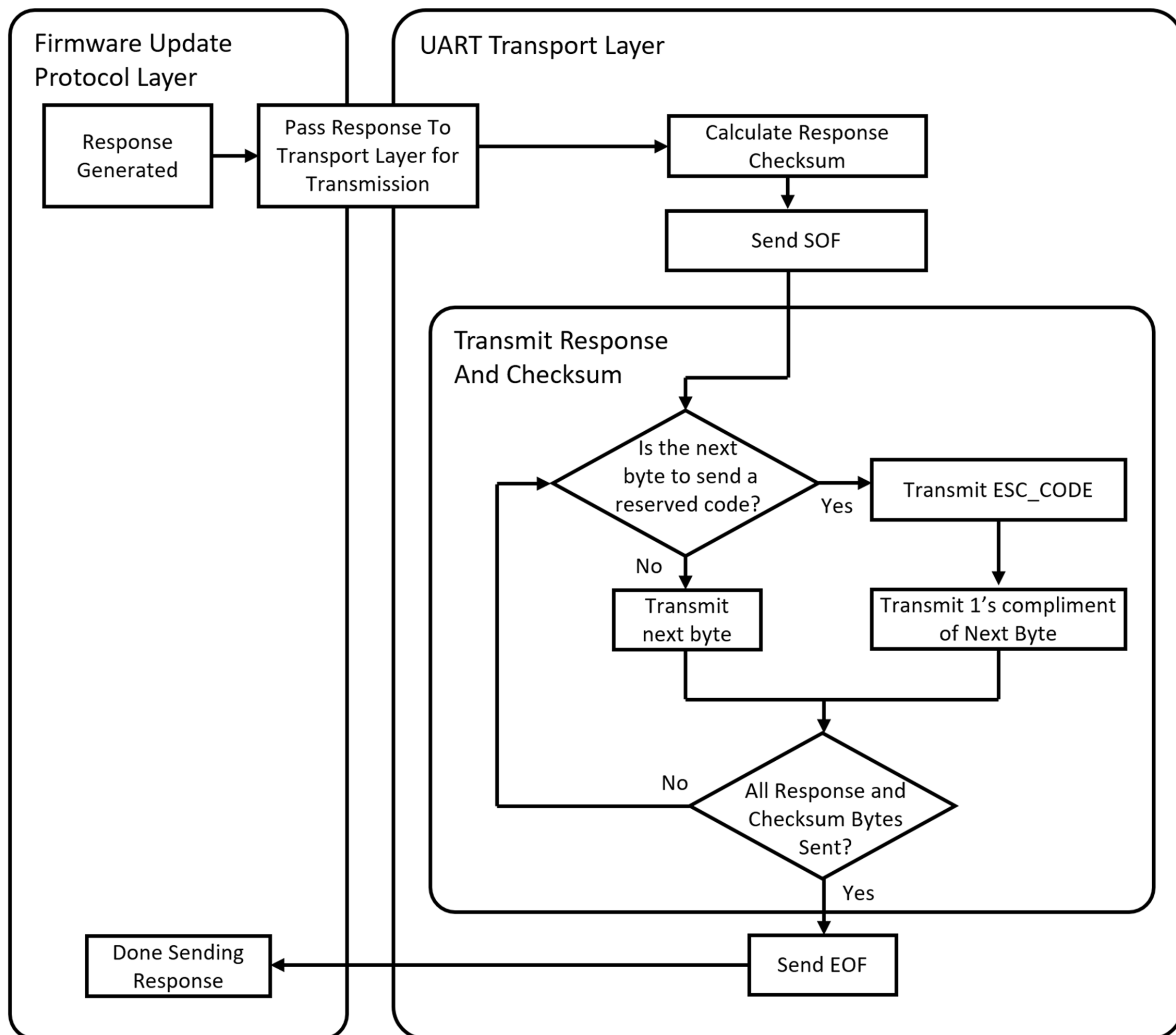
4.2.5 UART Command/Response Frame Transmission

When creating an MDFU host or client, developers make tradeoffs in their implementation. Some designs may prioritize minimizing the RAM buffer size of UART command /response frames. It is

© 2024 Microchip Technology Inc. and its subsidiaries

possible to compose MDFU UART command/response frames on the fly without having to store complete command/response frames in their expanded form (where reserved codes are replaced with 2-byte substitutions). The following figure shows an example algorithm for composing a response frame on the fly without storing it in its expanded form.

Figure 4-3. Client Composing MDFU UART Response Frame on the Fly



Other designs may want to pre-process commands and responses into UART command/response frames prior to beginning their transmission. See the section about Maximum UART Command/ Response Frame Size to understand the memory requirements for storing complete expanded UART command/response frames.

4.2.6 UART Command/Response Frame Reception

Receiving UART command/response frames involves the following capabilities:

1. Detecting the start of UART command/response frames.
2. Detecting the end of UART command/response frames.
3. Verifying the command/response frame checksum.

© 2024 Microchip Technology Inc. and its subsidiaries

4. Notifying the Firmware Update Protocol Layer of certain types of command/response frame reception errors (Transport Errors).

Additionally, this section defines a command-response frame reception algorithm with the following goals:

1. Allow resynchronization of command/response frame reception to the UART byte stream after some form of data corruption.
2. Limit the number of error responses to invalid or corrupted commands.

4.2.6.1 Details of Frame Reception

The following sections detail detection of the Start-Of-Frame and End-Of-Frame, as well as the related Receive Window.

4.2.6.1.1 Start-Of-Frame

Since all UART command/response frames start with the Start-Of-Frame (SOF) reserved code, this code is used as a synchronization mechanism. When a SOF is received, the following actions are taken:

1. Any partially received UART command/response frame is discarded.
2. The receive window is opened to allow incoming bytes to be processed as part of a command or response frame.

4.2.6.1.2 End-Of-Frame

All UART command/response frames end with the End-Of-Frame (EOF) reserved code. The EOF signals that a complete command or response has arrived and is ready for further processing. When an EOF is received, the following actions are taken:

1. The receive window is closed.
2. The command or response checksum is verified.
   - If the checksum is valid, the command/response is passed on for further processing.
   - If the checksum is invalid, the command/response frame is discarded and a Transport Error is signaled to the Firmware Update Protocol Layer.

4.2.6.1.3 The Receive Window

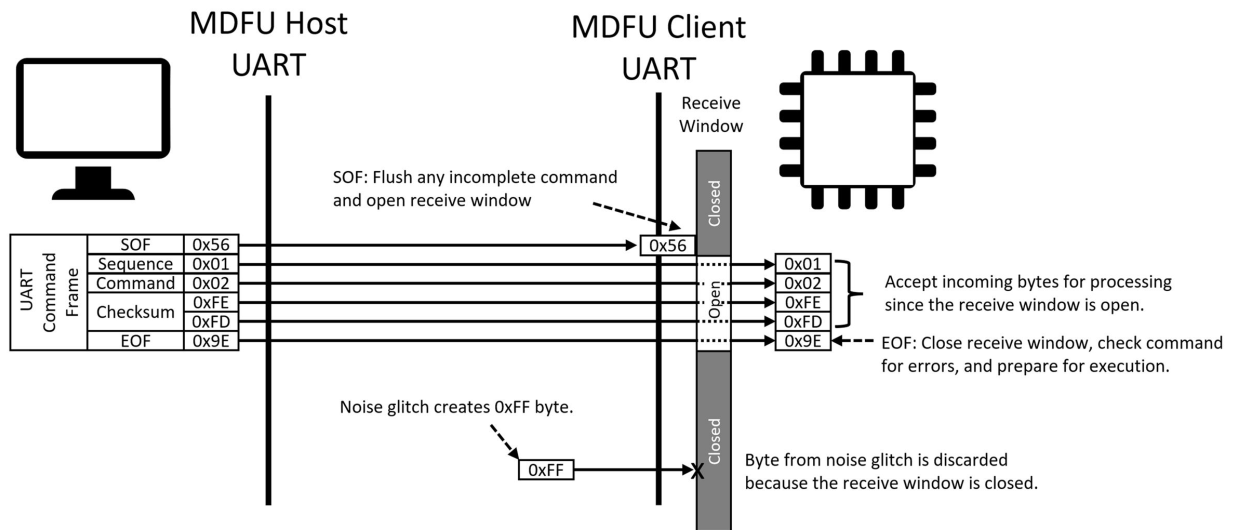
When byte streams get corrupted and command/response frame synchronization is lost, there is the potential to place invalid data into the receive processing algorithm.

To help reduce the chances of processing corrupted data streams, the MDFU UART Transport Layer implements a receive window which gates when received data is processed.

When the receive window is closed, incoming bytes are discarded and not processed. When the window is open, incoming bytes are accepted. The Start-Of-Frame code is the only code that is accepted when the window is closed and it is used to open the window. The End-Of-Frame code and certain types of detected errors are used to close the window.

The following figure shows an example command frame where the Start-Of-Frame opens the receive window. Incoming bytes are then accepted as part of the MDFU UART command/response frame. The End-Of-Frame code closes the window and prepares to process the command. The figure also shows how the closed receive window can reject a noise glitch which causes a 0xFF byte.

© 2024 Microchip Technology Inc. and its subsidiaries



4.2.6.2 Incoming Byte Processing Algorithm

The following algorithm specifies how the incoming byte stream must be processed to properly receive commands/responses, verify their integrity, and react to various error conditions.

© 2024 Microchip Technology Inc. and its subsidiaries

Figure 4-4. MDFU UART Transport Layer Incoming Byte Processing Algorithm

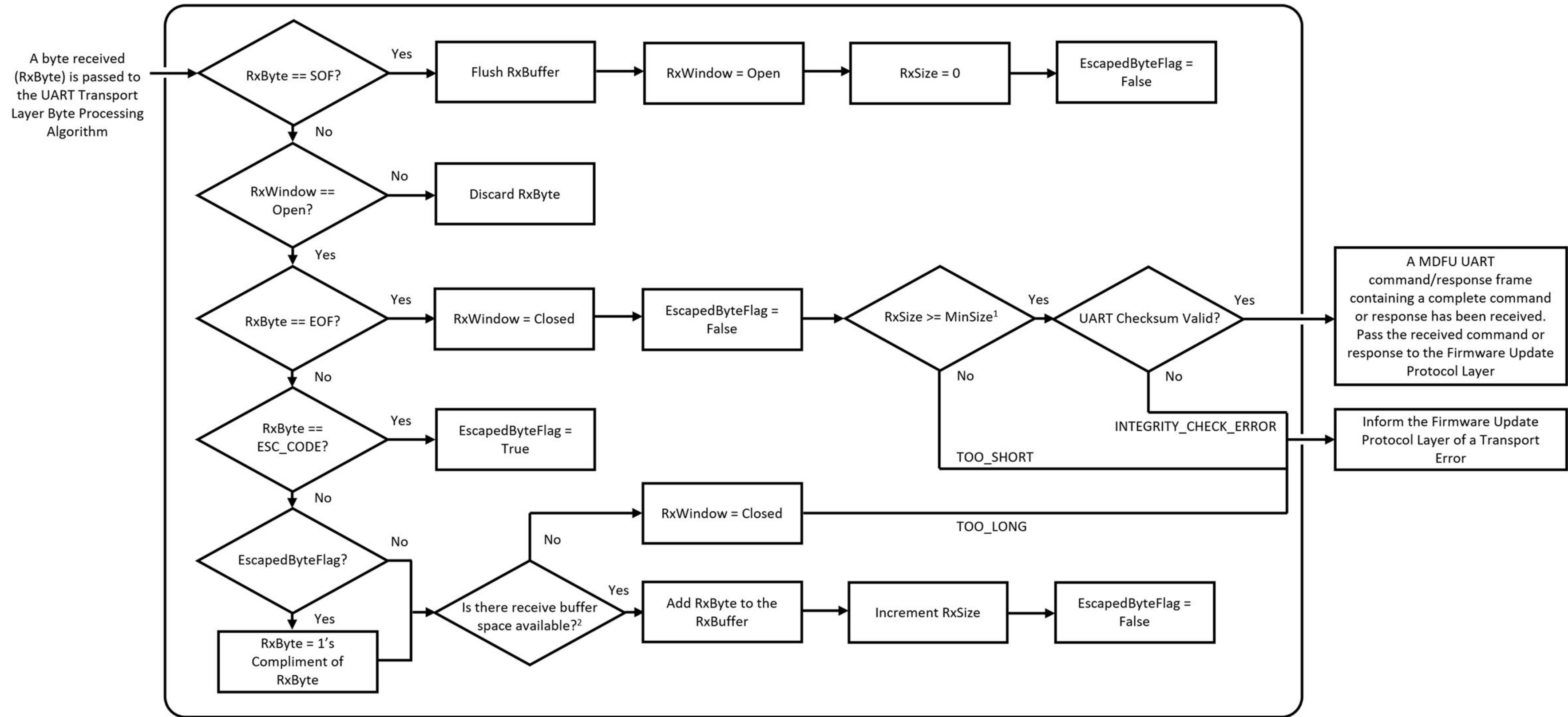


Figure Notes

Note 1: MinSize = 4; MinSize = 1-byte Sequence + 1-byte Command Code or 1-byte Status + 2-byte checksum = 4 bytes

Note 2: For **commands**, there is space available if RxSize < (MaxCommandDataLength + 4):

1-byte Sequence + 1-byte Command Code + MaxCommandDataLength Data Payload + 2-byte checksum = MaxCommandDataLength + 4

For **responses**, hosts must allocate sufficient size response buffers for the clients they are updating and protect from overflowing their response buffers.

See the Command-Response Pairs section of the specification for more information on response sizes.

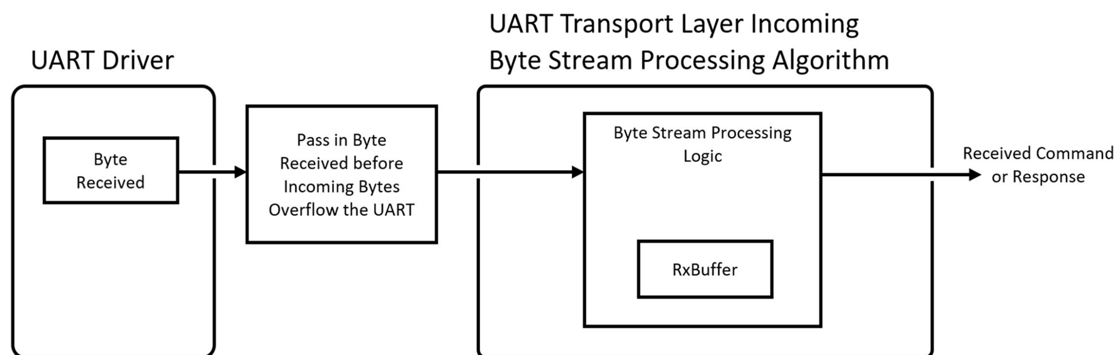
© 2024 Microchip Technology Inc. and its subsidiaries

4.2.6.3 UART Driver Buffer Sizing

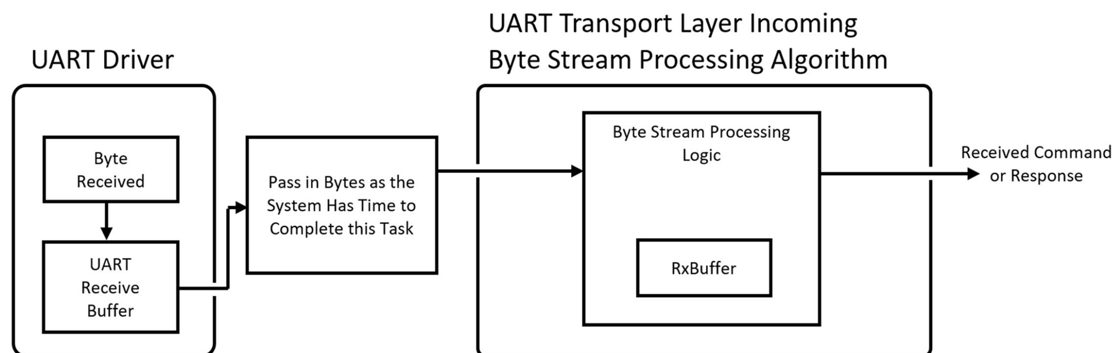
The incoming byte processing algorithm operates on one byte at a time. Incoming bytes can be processed fairly quickly by this algorithm as they come in.

The timing requirements associated with pulling bytes out of the UART Driver and feeding them into the Transport Layer Incoming Byte Stream Processing Algorithm is a function of how much buffering is present in the UART driver. Individual system designers can decide how much buffering inside the UART driver makes sense based on the system resources (Flash/RAM/Processing) available in a design.

Implementations that are memory constrained and can allocate enough processing priority to process incoming bytes on the fly as they arrive may elect to have little to no buffering in the UART driver, as shown below.



Other systems may have higher priority tasks that are running at the same time as the MDFU tasks and may only be able to provide limited timing priority and/or processing cycles to the UART Driver and MDFU tasks. In these systems, adding buffering inside the UART driver as shown in the figure below can give the system more time to do other things before it needs to come back and get bytes from the UART buffer to pass them into the Transport Layer incoming byte stream processing algorithm. Sizing this UART reception buffer is left to the individual host and client designer.



Please consider the following two points when sizing the client UART reception buffer:

1. If the client UART reception buffer is smaller than the size of any command, the time frame associated with having to remove bytes out of the UART buffer to prevent it from overflowing is related to the size of the buffer and the data rate of the UART.
2. If the client UART reception buffer is sized to be able to hold the largest MDFU UART command frame size, then the timing requirement to remove bytes from the UART buffer is no longer determined by the size of the buffer and the data rate of the UART. Rather, the timing requirement is now set by the need to satisfy the command time-out parameter requirement. This time-out parameter can be configured to be a long period of time which can allow the MDFU tasks to run as very low-priority background tasks with very slow timing requirements.

© 2024 Microchip Technology Inc. and its subsidiaries

See the Maximum UART Command/Response Frame Size section for details on how large the UART buffer needs to be sized to receive an entire UART command frame.

Similarly, please consider the following two points when sizing the host UART reception buffer:

1. If the host UART reception buffer is smaller than the size of any response, the time frame associated with having to remove bytes out of the UART buffer to prevent it from overflowing is related to the size of the buffer and the data rate of the UART.
2. If the host UART reception buffer is sized to be able to hold the largest UART response frame size that the attached client sends, then bytes don't need to be removed from the UART receive buffer as a response is arriving to avoid overflowing the buffer. This increases the amount of time that the host can work on other things before needing to service the MDFU tasks which can be helpful for hosts that are running other high priority tasks.

4.2.7 UART Command/Response Integrity Check

The MDFU UART Transport Layer definition uses a one's compliment 16-bit checksum as a mechanism for detecting corruption in commands and responses. The details of the checksum calculation are provided below.

4.2.7.1 Checksum Computation Details

The following steps are used to calculate the one's complement checksum over the entire command or response.

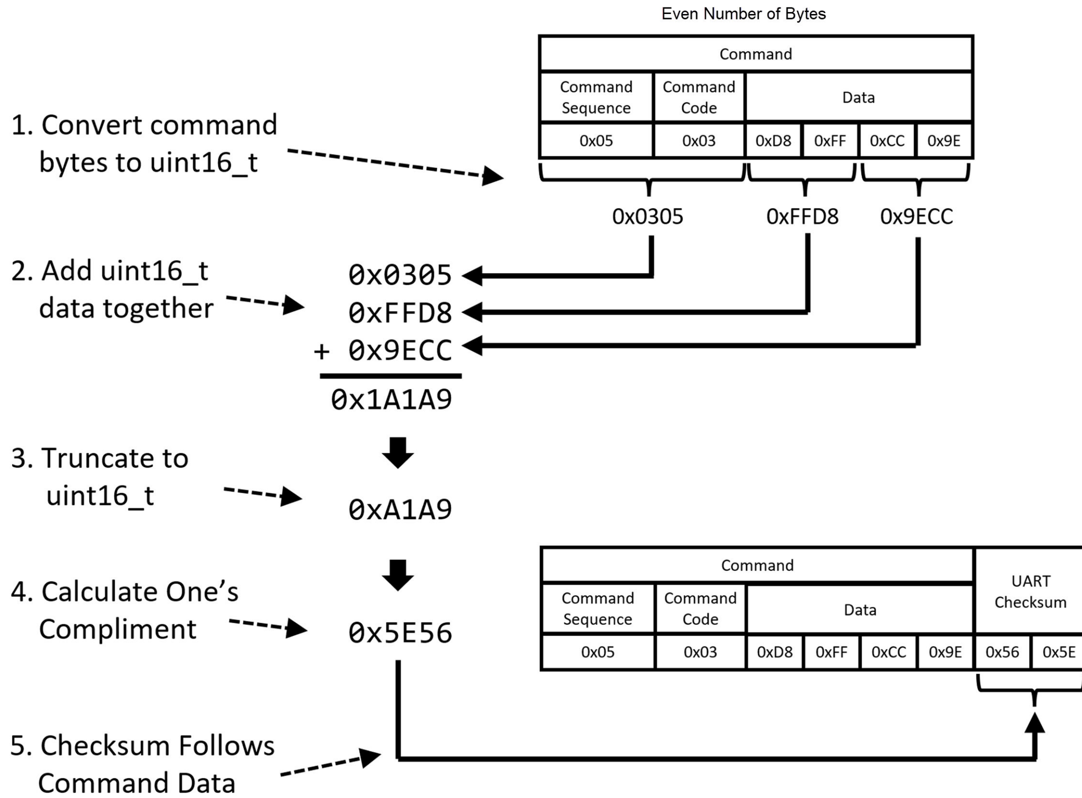
1. Convert the command or response into a series of uint16_t values.
   - When the command or response contains an odd number of bytes, the upper byte of the last uint16_t value is set to 0x00.
   - The added 0x00 is NOT placed into the command or response for transmission. It is only used in the checksum calculation.
2. Add the uint16_t values together.
3. Truncate the result of the addition to a uint16_t value.
4. Calculate the one's compliment of the truncated result.

4.2.7.2 Examples

The following example shows the checksum calculation for a command with an even number of bytes.

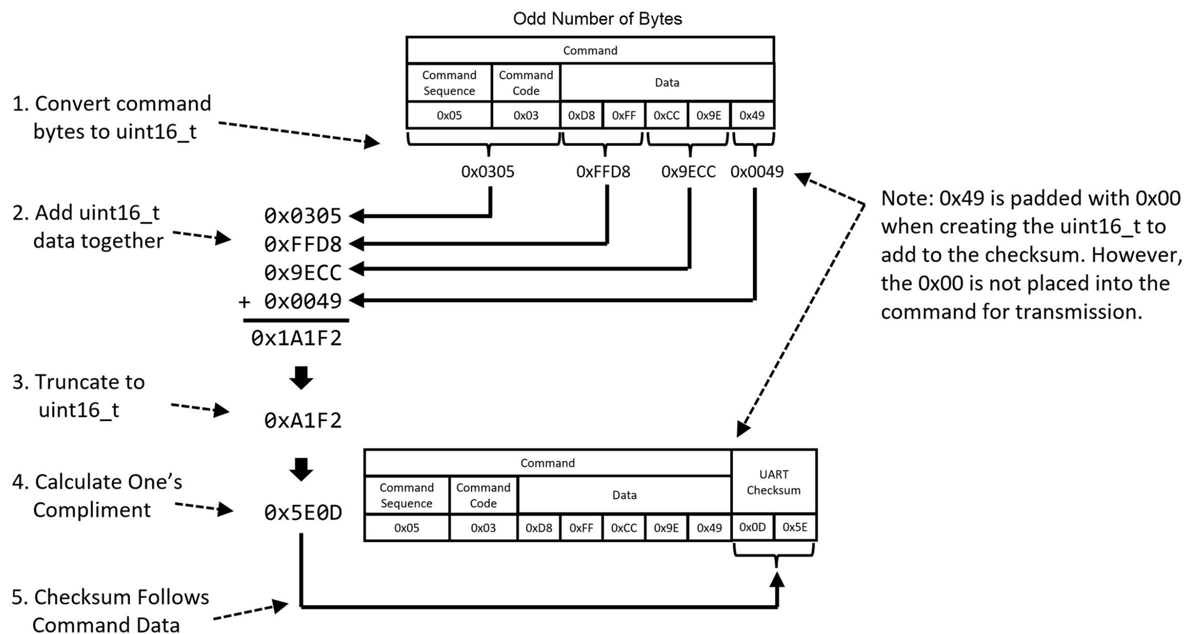
© 2024 Microchip Technology Inc. and its subsidiaries

Figure 4-5. Checksum of Even Number of Bytes



The following example shows the checksum calculation for a command with an odd number of bytes.

Figure 4-6. Checksum of Odd Number of Bytes



© 2024 Microchip Technology Inc. and its subsidiaries

4.2.8 Maximum UART Command/Response Frame Size

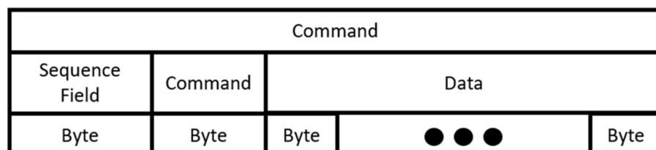
This section details the calculation of the worst case size a UART command/response frame can be.

4.2.8.1 Maximum Size of Command/Response

The maximum command and response data payload sizes are client dependent. The MaxCommandDataLength parameter defines the maximum number of bytes contained in the command data payload. See Command-Response Pairs for more information on command and response sizes.

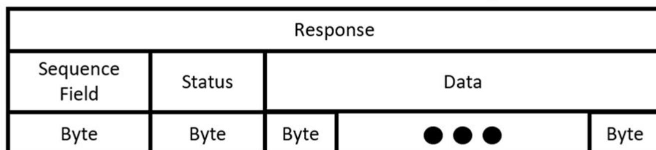
The figure below shows the maximum size of a command and response.

Maximum Size of Command



1 Byte + 1 Byte + MaxCommandDataLength Bytes = 2 + MaxCommandDataLength Bytes

Maximum Size of Response

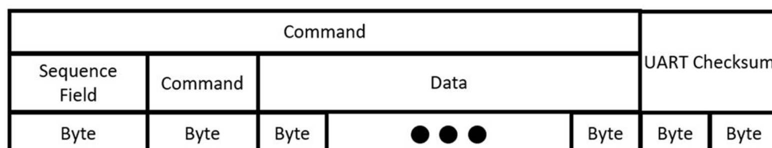


1 Byte + 1 Byte + N_R = 2 + N_R Bytes

4.2.8.2 Maximum Size of Command/Response With Checksum

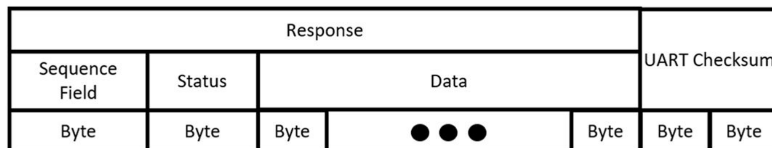
The UART transport layer adds on a 2-byte checksum to commands and responses as shown in the figure below.

Maximum Size of Command With Checksum



1 Byte + 1 Byte + MaxCommandDataLength Bytes + 2 Bytes = 4 + MaxCommandDataLength Bytes

Maximum Size of Response With Checksum



1 Byte + 1 Byte + N_R Bytes + 2 Bytes = 4 + N_R Bytes

© 2024 Microchip Technology Inc. and its subsidiaries

4.2.8.3 Worst Case Size of Expanded Command/Response With Expanded Checksum

The worst case scenario is if every byte in the command or response is equal to a reserved code and has to be replaced with a 2-byte substitution in the expanded command/response and checksum as shown in the figure below.

Worst Case Size of Expanded Command With Expanded Checksum

Expanded Command										Expanded UART Checksum			
Sequence Field		Command		Data									
ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	●●●	ESC_CODE	1's Complement of Byte		ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte

2 x (1 Byte) + 2 x (1 Byte) + 2 x (MaxCommandDataLength Bytes) + 2 x (2 Bytes) = 2 x (4 + MaxCommandDataLength Bytes)

Worst Case Size of Expanded Response With Expanded Checksum

Expanded Response										Expanded UART Checksum			
Sequence Field		Status		Data									
ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	●●●	ESC_CODE	1's Complement of Byte		ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte

2 x (1 Byte) + 2 x (1 Byte) + 2 x (N_R Bytes) + 2 x (2 Bytes) = 2 x (4 + N_R Bytes)

4.2.8.4 Worst Case Size of MDFU UART Command/Response Frames

Finally, the Start-Of-Frame and End-Of-Frame codes are added to the MDFU UART command/ response frames. This results in a final formula for the worst case maximum size MDFU UART command frames and response frames.

Worst Case Command Frame Size

	Expanded Command									Expanded UART Checksum				
	Sequence Field		Command		Data									
SOF 0x56	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	●●●	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	EOF 0x9E

1 Byte + 2 x (1 Byte) + 2 x (1 Byte) + 2 x (MaxCommandDataLength Bytes) + 2 x (2 Bytes) + 1 Byte = 2 x (4 + MaxCommandDataLength Bytes) + 2

Worst Case Response Frame Size

	Expanded Response									Expanded UART Checksum				
	Sequence Field		Status		Data									
SOF 0x56	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	●●●	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	ESC_CODE	1's Complement of Byte	EOF 0x9E

1 Byte + 2 x (1 Byte) + 2 x (1 Byte) + 2 x (N_R Bytes) + 2 x (2 Bytes) + 1 Byte = 2 x (4 + N_R Bytes) + 2

© 2024 Microchip Technology Inc. and its subsidiaries

5. Revision History

The following is a list of changes by version to this document.

Note: Some revision letters are not used - the letters `I` and `O` - as they can be confused for numbers in some fonts.

PDF Rev A, WebHelp Ver 1, Protocol 1.0.0 (September 2024)
Initial release of the document.

© 2024 Microchip Technology Inc. and its subsidiaries

Microchip Information

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

© 2024 Microchip Technology Inc. and its subsidiaries

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

Device:	PIC16F18313, PIC16LF18313, PIC16F18323, PIC16LF18323	
Tape and Reel Option:	Blank	= Standard packaging (tube or tray)
	T	= Tape and Reel(1)
Temperature Range:	I	= -40°C to +85°C (Industrial)
	E	= -40°C to +125°C (Extended)
Package:(2)	JQ	= UQFN
	P	= PDIP
	ST	= TSSOP
	SL	= SOIC-14
	SN	= SOIC-8
	RF	= UDFN
Pattern:	QTP, SQTP, Code or Special Requirements (blank otherwise)	

Examples:

- PIC16LF18313- I/P Industrial temperature, PDIP package
- PIC16F18313- E/SS Extended temperature, SSOP package

Notes:

1. Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.
2. Small form-factor packaging options may be available. Please check www.microchip.com/packaging for small-form factor package availability, or contact your local Sales Office.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for

© 2024 Microchip Technology Inc. and its subsidiaries

additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, TimeCesium, TimeHub, TimePictra, TimeProvider, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, Clockstudio, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, EyeOpen, GridTime, IdealBridge, IGaT, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, IntelliMOS, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, MarginLink, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mSiC, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, Power MOS IV, Power MOS 7, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, Trusted Time, TSHARC, Turing, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2024, Microchip Technology Incorporated and its subsidiaries. All Rights Reserved.

ISBN: 978-1-6683-0280-4

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Hod Hasharon
Tel: 972-9-775-5100

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

© 2024 Microchip Technology Inc. and its subsidiaries