注意 : この日本語版文書は参考資料としてご利用ください。
最新情報は必ずオリジナルの英語版をご参照願います。

Microchip デバイス ファームウェア更新(MDFU) プロトコル仕様 1.0.0

開発ツールをご使用のお客様へ

重要:
どのような文書でも内容は時間が経つにつれ古くなります。本書も例外ではありません。Microchip 社のツールとマニュアルは、お客様のニーズを満たすために常に改良を重ねており、実際のダイアログやツールの内容が本書の説明とは異なる場合があります。最新の PDF 文書については、Microchip 社のウェブサイト(www.microchip.com/)をご覧ください。

文書は各ページのフッタに表記している「DS」番号で識別します。DS 番号のフォーマットは DS<文書番号><リビジョン>です。<文書番号>は 8 桁の番号、<リビジョン>はアルファベットの大文字です。

最新情報はツールのヘルプ(onlinedocs.microchip.com/)をご覧ください。

目次

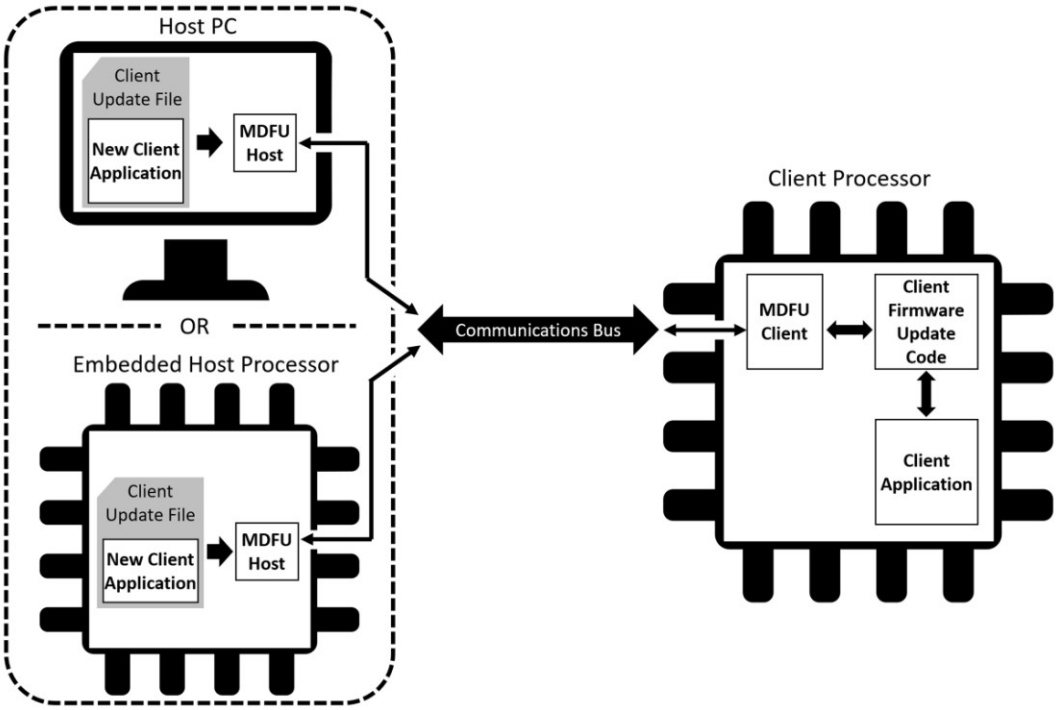
1. Microchip デバイス ファームウェア更新(MDFU)プロトコルの概要

多くの組み込みシステムでは、標準的な通信バス(UART、I2C、SPI 等)を使ってプロセッサのファームウェアを更新する機能が求められます。

Microchip デバイス ファームウェア更新プロトコル(以下 MDFU プロトコルと呼ぶ)は、各種の通信バスを介してホスト プロセッサからクライアント プロセッサのファームウェアを更新するための仕組みを定義します。

下の図は、MDFU ホスト(PC または組み込みプロセッサ)からクライアント プロセッサ上の MDFU クライアントへ新しいクライアント アプリケーションを転送する方法を示しています。クライアント プロセッサ上のクライアント ファームウェア更新コードは、クライアント アプリケーションをホストから受信した新しい内容へ更新します。

本書で使われる用語の詳細は「用語の定義」に記載しています。



1.1 MDFU プロトコルの長所

MDFU プロトコルは、クライアント ファームウェアを更新するための柔軟な仕組みを提供する事で、クライアントの必要メモリ容量を抑えながら多くの組み込みシステムの要求に応える事を可能にします。MDFU プロトコルは、小型の 8 ビット マイクロコントローラから 32 ビット以上の大型 MCU まで広範囲のクライアント マイクロコントローラの要求に対応します。

MDFU プロトコルは、クライアントに依存しないシンプルで安定したファイル転送方式を使ってクライアント プロセッサのファームウェアを更新します。クライアント プロセッサに固有のメモリ情報は、クライアント更新ファイルとクライアント ファームウェア更新コード内に格納されます。MDFU プロトコルのコマンドとレスポンスはクライアント プロセッサに依存しません。ホストは、更新プロセス中にクライアントから必要な情報を取得します。これらの MDFU プロトコルの特長から以下の利点が得られます。

- 下記の条件が満たされていれば、MDFU ホストは接続される全ての MDFU クライアント プロセッサを更新できます。
  a. ホストはクライアントが実装しているプロトコルのバージョンをサポートしている事
  b. ホストにはクライアントをサポートするのに十分なコマンド/レスポンス バッファ空間を割り当て済みである事
- クライアント プロセッサに固有の機能は、クライアント更新ファイルの内容を適切に定義/変更する事によりサポート/追加でき、MDFU クライアントプロトコルまたはホストを変更する必要はありません。
- プロトコルはシンプルで信頼性が高いため、MDFU ホスト-クライアント間のバージョン互換性の問題が緩和されます。

MDFU プロトコルは階層化されているため、新しい通信バス規格に対するサポートを容易に追加できます。階層化する事で、規格が異なる通信インターフェイスの間で MDFU 機能を最大限に再利用できると共に、個々の通信インターフェイスに対して定義する必要がある詳細が明確に示されます。ホストとクライアントのコードは、異なる通信インターフェイス間でのコード再利用性が最大となるよう、また新しい通信インターフェイス向けのサポートを追加する手間が最小となるように書く事ができます。

MDFU プロトコルはコマンド/レスポンスの破損を検出する機能と、検出された破損から自動的に回復する機能を備えています。

クライアントは、回復不可能なエラーを検出した時に即座に更新プロセスを中止するオプション機能を備えています。これにより、ホストが更新の失敗を検出する前に(更新の試みが完了するのを待たずに)プロセスを直ちに中止できます。さらに、ユーザのデバッグ作業を助けるため、クライアントは更新を中止した理由を報告する事ができます。

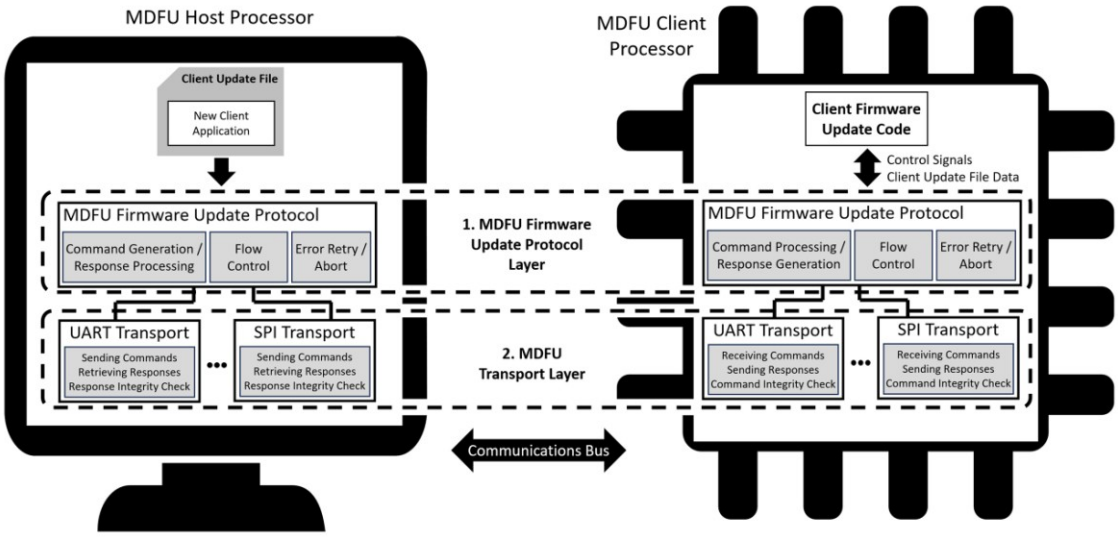
クライアントは新しいファームウェアが有効かどうかを判定し、そのステータスをホストに報告する事もできます。ファームウェア検証アルゴリズムは複雑さ、クライアントのリソース(フラッシュ/RAM 容量と処理能力)および破損検出能力とのトレードオフとなります。このため、MDFU クライアントごとの要件に応じた最適な検証アルゴリズムが選択可能となっています。

MDFU プロトコルが採用するファイル転送方式のファームウェア更新では、ホスト側ファームウェア更新のユースケースと、OTA(Over The Air)に代表されるような非ホスト型ファームウェア更新のユースケースの双方で、クライアント更新時に使うファイルのフォーマットと処理のコードが共通に利用できます。

1.2 MDFU プロトコルの階層

MDFU プロトコルは、各種の物理的通信バスを介する更新をサポートします。各種の通信バスでの MDFU プロトコルの定義と運用を容易にするため、MDFU プロトコル仕様は以下の 2 つの階層で構成されます。

- ファームウェア更新プロトコル層
  - この階層は、全ての通信インターフェイスで再利用されるプロトコルの詳細を定義します。
- トランスポート層
  - この階層は、通信バスごとに固有の MDFU コマンド/レスポンスの送信方法を定義します。
  - 個々の通信バスは、コマンド/レスポンスの送信方法に関する独自の仕様を持つ事ができます。



1.3 MDFU プロトコルのバージョン

本書では、以下のバージョンの MDFU プロトコルについて説明します。

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

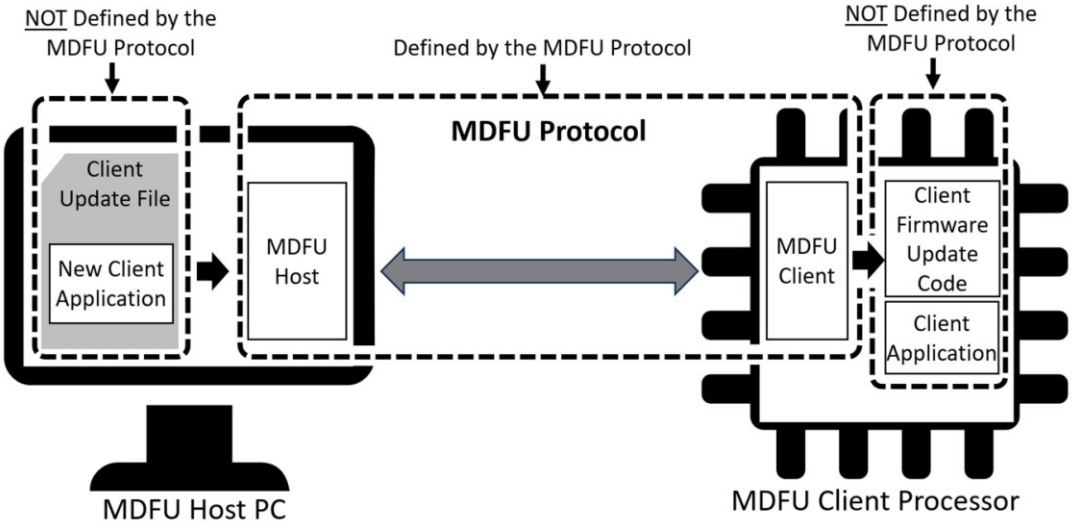
以前のバージョンに対して下位互換性を持たない全てのプロトコル変更は、MajorVersion バイトの値をインクリメントします。以前のバージョンに対して下位互換性を維持したまま新しい機能を追加するプロトコル変更は、MinorVersion バイトの値をインクリメントします。プロトコルそのものを変更しない仕様の更新(バグ修正、文書の改善/明瞭化と誤字/誤植の訂正等)は、Patch バイトの値をインクリメントします。全ての一般公開バージョンでは PreRelease バイトは除外されます。

1.4 用語の定義

- **ホスト:** 更新対象のクライアントに対するクライアント更新ファイルの転送を制御する処理ユニットです。PC、組み込みマイクロ プロセッサ、マイクロコントローラがホストになれます。
- **クライアント:** ファームウェア(メモリ)が更新される側のプロセッサです。
- **クライアント更新ファイル:** このファイルはホストからクライアントへ送信され、クライアントのファームウェア(メモリ)を更新するために必要な情報を格納します。
- **MDFU:** Microchip Device Firmware Update

2. ファイル転送方式のファームウェア更新

MDFU プロトコルは、クライアント メモリ更新プロセスからクライアント アプリケーション転送プロセスを分離するために、クライアント更新ファイルを抽象化します。MDFU プロトコルは、クライアント更新ファイルをホストからクライアントへ転送するためのプロセス(更新アルゴリズム、コマンド/レスポンス、エラー処理等)を完全に定義します。

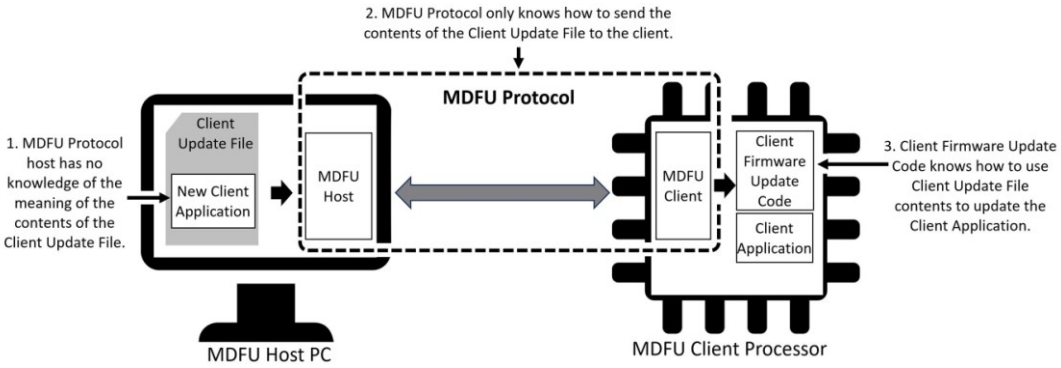


2.1 MDFU ファイル抽象化の基本概念

以下は MDFU プロトコルが採用するファイル転送方式ファームウェア更新の基本概念です。

1. MDFU プロトコルと MDFU ホストは、クライアント更新ファイルの内容(意味)に関与しない
2. MDFU ホストは、クライアント更新ファイルをクライアントへ送信する方法にのみ関与する
3. クライアント ファームウェア更新コードは、クライアント更新ファイルの内容を使ってクライアント ファームウェアを更新する方法に関与する

クライアント更新ファイルのフォーマットは、クライアント ファームウェア更新コードが要求するフォーマットと一致する必要があります。クライアント ファームウェア更新コード ソリューションは、クライアント ファームウェア更新コード実装によって使われるファイル フォーマットの仕様を提供する責任があります。



2.2 MDFU ファイル抽象化の利点

前述の基本概念により、以下の利点が得られます。

1. ホストは汎用的にクライアント デバイス プロセッサをサポート可能
   - クライアント プロセッサに固有の情報(メモリアドレス、フラッシュ コントローラのロック解除キー等)は MDFU プロトコルとは無関係であり、それらの情報はクライアント更新ファイルとクライアント ファームウェア更新コード内に置かれます。このため、ホストに一切変更を加える事なく新しいクライアント プロセッサをサポート可能です。ホストがクライアントをサポートするのに十分なコマンド/レスポンス バッファ空間を割り当て済みであれば、クライアントは単純にプロトコルを実装するだけで直ちにホストによってサポートされます。
2. ホストはシンプルで安定性に優れる
   - ファイル転送プロセスは明確に定義されシンプルで安定しており、クライアントに依存しません。
   - クライアントに固有の新機能は、クライアント更新ファイルの内容を変更するだけで、MDFU プロトコルまたはホストを変更する事なく追加できます。
   - ホストがクライアント更新ファイルの転送にのみ責任を持つシンプルな転送プロトコルを使う事で、組み込みホストサポートをより容易に実現できます。
   - ホストがシンプルで安定している事により、ホストとクライアントの間のバージョン互換性問題が緩和されます。

3. ファームウェア更新プロトコル層

MDFU プロトコルは 2 つの階層に分割されます。以下では、ファームウェア更新プロトコル層について説明します。

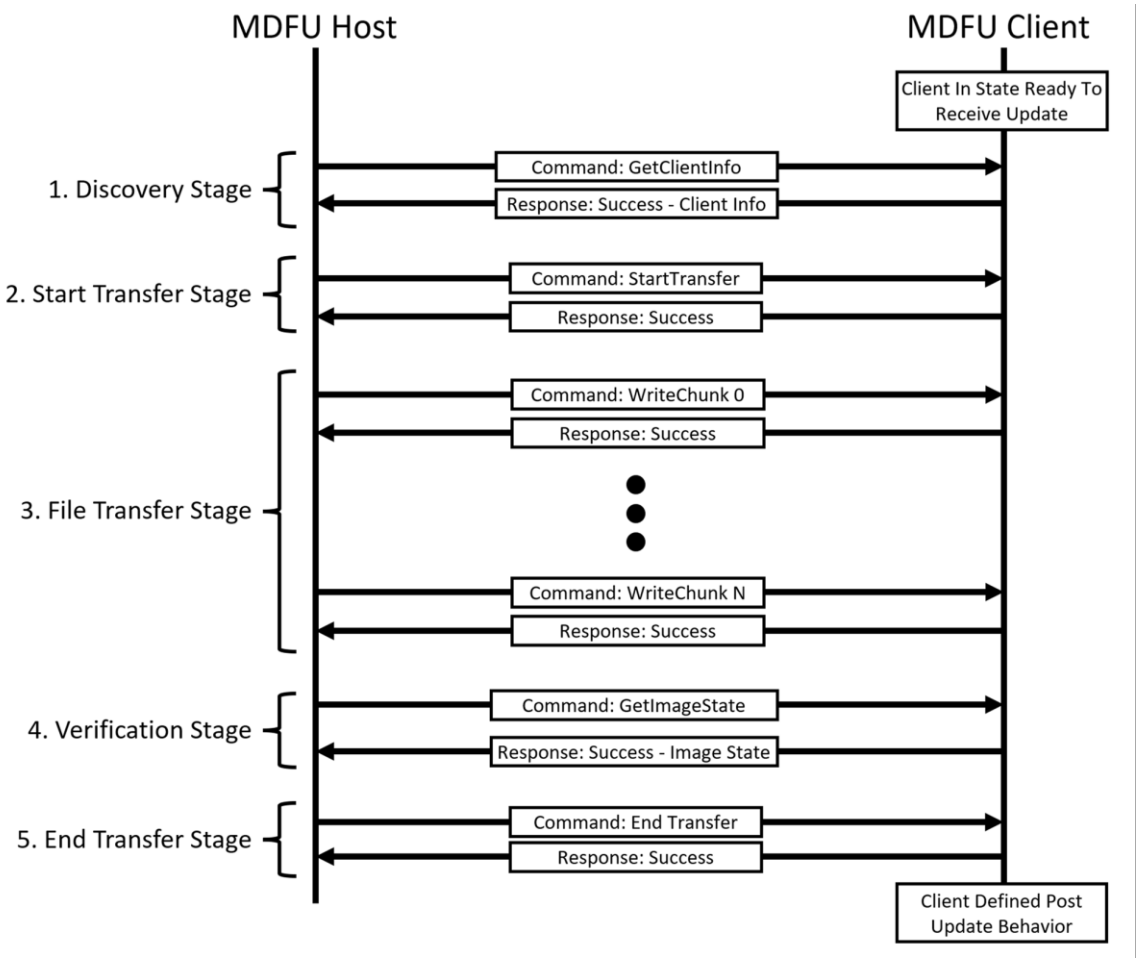
この階層は、全ての通信インターフェイスによって再利用される以下のプロトコルの詳細を定義します。

- ファームウェア更新アルゴリズム
- ファイル転送ステージ
- コマンド/レスポンス ペア
- ホストによるコマンド生成/レスポンス処理
- クライアントによるコマンド処理/レスポンス生成
- コマンド
- レスポンス
- 検証ステージ
- フロー制御
- 更新の正常な完了
- エラーの分類と処理
- シーケンス番号

ファームウェア更新プロトコル層は、各種の通信バスを介してコマンド/レスポンスを送信する方法を定義しません。各種の物理バスを介してコマンド/レスポンスを送信する方法については、「トランスポート層」を参照してください。

3.1 ファームウェア更新アルゴリズム

MDFU プロトコルは 5 段の処理ステージを使います。これらのステージは、ホストがクライアントのファームウェアを更新する際に順番に発生します。ホストはコマンド/レスポンス ペアを使ってこれらの各ステージを実行します。これらの各ステージを次ページの図に示します。図の後に各ステージの説明を記載します。



3.1.1 更新前のクライアント セットアップ

MDFU ホストから更新を開始する前に、クライアントが MDFU クライアントプロトコル コマンドを受信して処理できるようにするために、ユーザはクライアントを適切なモードに設定する必要があります。クライアントをファームウェア更新モードへ移行させるための方法は以下の通りクライアントごとに異なります。

- ボタンのプレス/ホールド
- ボタンを押しながら電源再投入
- ボタンを押しながらリセット
- 更新トリガの送信
- アプリケーションの実行中に受信した更新を受け付ける(ライブ更新)

ファームウェア更新モードへ移行するための方法は、クライアント ファームウェアの開発者が選択します。

3.1.2 探索ステージ

各更新プロセスは探索ステージで始まります。このステージ中に、ホストはクライアントの更新に必要なクライアント パラメータを取得します。

探索ステージは、ホストからクライアントへ送信される 1 つの *GetClientInfo* コマンド(「クライアント情報取得コマンド」参照)により完了します。クライアントは、このコマンドに対してクライアント パラメータ値のリストを返す必要があります。パラメータ リストにはクライアントが実装している MDFU プロトコルのバージョン、クライアント バッファ情報、クライアント コマンド タイムアウト情報が含まれます。

クライアントが必須パラメータの報告に失敗した場合、ホストは更新を終了し、終了した原因をユーザに報告する必要があります。

パラメータの詳細な説明は、本書内の *GetClientInfo* コマンドの定義に記載しています。

3.1.3 転送開始ステージ

クライアントの情報を探索した後に、ホストは *StartTransfer* コマンド(「転送開始コマンド」参照)を送信します。

このコマンドの目的は、クライアントが更新ファイルデータの受信準備を整えるために必要な全ての状態を初期化できるようにする事です。

更新が何らかの理由により途中で失敗した場合、このコマンドによってクライアント更新ファイルを最初から完全に受信できる状態へクライアントをリセットする必要があります。

このコマンドを受信した時に取られるアクションの詳細はクライアントに依存して異なります。

3.1.4 ファイル転送ステージ

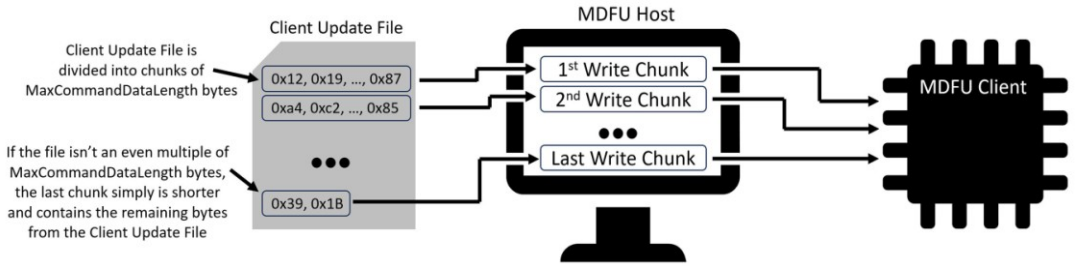
クライアント ファームウェアを更新するには、ホストからクライアントへクライアント更新ファイル内の全てのデータを転送する必要があります。

以下の理由により、クライアント更新ファイルは複数の小さなチャンクに分割されて送信されます。

1. フロー制御
   - MDFU プロトコルは Stop-and-Wait フロー制御を使います。この形態のフロー制御を使う場合、ホストからクライアントへ送信されるコマンドの最大サイズが既知で固定されており、かつファイル チャンクがこれらのコマンド データペイロードに収まる必要があります。
   - 詳細は「フロー制御」を参照してください。
2. 迅速なエラーの検出と対応
   - ファイルの小さなチャンクを送信するたびにコマンドの整合性を確認する事により、送信データの破損を迅速に検出してエラーから迅速に回復できます。
   - ファイルデータを小さなチャンクに分割して送信する事により、クライアントは回復不可能なエラーを検出した時に直ちに転送を中止できます。

3.1.4.1 ファイルのチャンク化

- ホストはファイルを固定サイズのチャンクへと分割し、*WriteChunk* コマンド(「チャンク書き込みコマンド」参照)を連続して使う事により、それらのチャンクをクライアントへ連続的に送信します。
- チャンクのサイズは、*GetClientInfo* コマンドを使って探索ステージ中にクライアントから取得されます。
- クライアント バッファ情報パラメータには、1 つのコマンドでクライアントが受信可能なファイルの最大バイト数を定義する MaxCommandDataLength フィールドが含まれます。
- クライアント更新ファイルは、バイト数が MaxCommandDataLength のチャンクへ分割されます。
- ファイルのサイズがチャンクサイズの整数倍ではない場合、最後のファイルチャンクはファイルからの残りのバイトを格納した通常より小さなチャンクとなります。



3.1.5 検証ステージ

ファイルが完全に転送された後にホストは *GetImageState* コマンド(「イメージ状態取得コマンド」参照)を送信します。このコマンドは、ファームウェア検証アルゴリズムを実行してファームウェアが有効かどうかの情報を返すようクライアントに指示します。このコマンドに対するレスポンにより、ホストはクライアントが新しいファームウェアを有効であると判定したかどうかを知る事ができます。

メモリを検証する方法はクライアントによって選択されます(MDFU プロトコルが指定するのではありません)。クライアントは、その製品の要件に応じて任意のアルゴリズム(チェックサム、巡回冗長検査(CRC)等)を選択できます。

クライアントで無効なイメージが検出された時に適切な対応が取れるよう、ホストはエラーの生成(検知)、エラーのロギング、ユーザへのエラーの通知を行う機能を提供する必要があります。

3.1.6 転送終了ステージ

検証が完了した後に、ホストは *EndTransfer* コマンド(「転送終了コマンド」参照)をクライアントへ送信する事で、更新が完了した事をクライアントに知らせます。

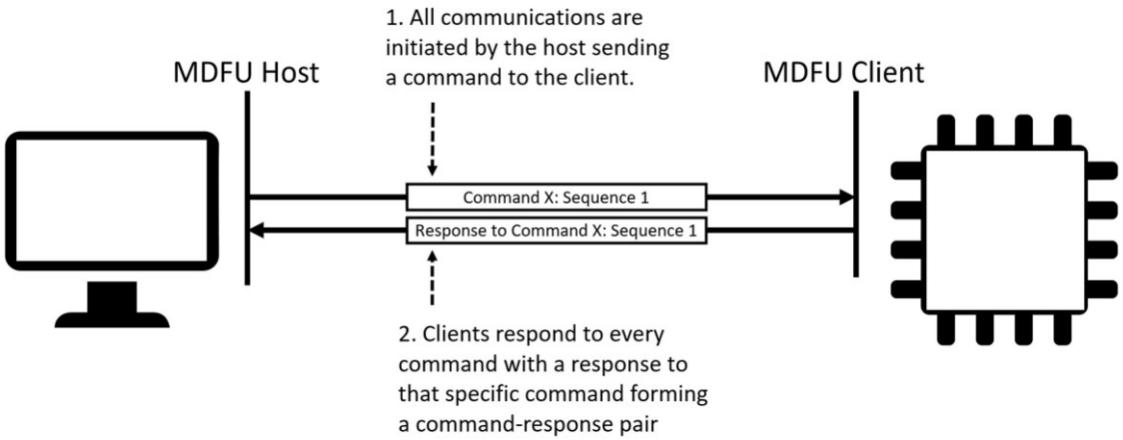
この時点で、クライアント ファームウェア更新コードは次のアクション(例: ファームウェアの起動等)を選択できます。

3.2 コマンド/レスポンス ペア

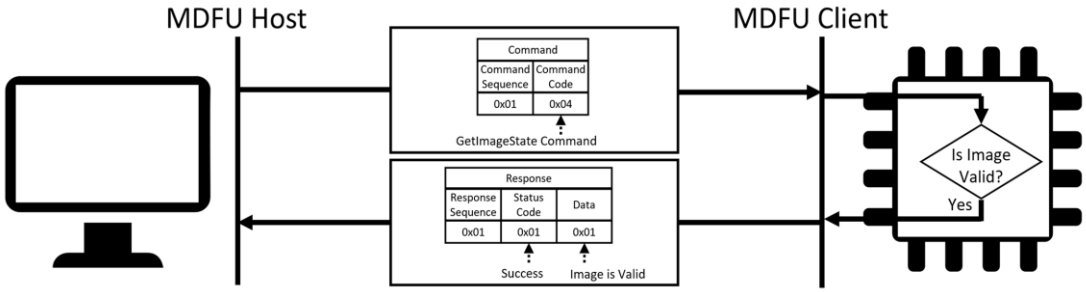
MDFU プロトコルはホスト駆動型プロトコルです。全ての通信は以下の 3 つの手順に従います。

1. ホストによるコマンド生成
2. クライアントによるコマンド処理/レスポンス生成
3. ホストによるレスポンス処理

コマンドとそのコマンドに対応するレスポンスの組み合わせをコマンド/レスポンス ペアと呼びます。コマンドとレスポンスは、シーケンス番号によって同じコマンド/レスポンス ペアのメンバーとして識別されます。



コマンドを使ってクライアント上で動作をトリガし、対応するデータをクライアントから取得する例を次のページの図に示します。この例では、ホストは *GetImageState* コマンド(「イメージ状態取得コマンド」参照)をクライアントへ送信します。クライアントは、そのファームウェア イメージが有効かどうかを判定し、ファームウェア イメージの状態に関する情報をコマンドに対して返します。



3.2.1 コマンド

MDFU コマンドにより、ホストからクライアントに対する信号の送信、クライアント更新ファイルデータの送信、クライアント上のファームウェア検証アルゴリズムの起動が行えます。

ホストからクライアントを正常に更新するために、クライアントは 5 つの必須コマンドをサポートする必要があります。加えて、将来の新しいユースケースをサポートするために、オプション コマンド用のコードが MDFU プロトコルによって予約されています。

3.2.1.1 コマンドの一覧

各コマンド/レスポンス ペアの詳細な定義は、本書内の対応する項目に記載しています。これにはコマンドコード、コマンド データペイロード、コマンド受信時のクライアント動作、更新成功時のクライアントからのレスポンス データペイロードに関する定義が含まれます。

コマンド	コマンドコード	クライアントへの実装
予約済み - 使用禁止 [1]	0x00	N/A
GetClientInfo	0x01	必須
StartTransfer	0x02	必須
WriteChunk	0x03	必須
GetImageState	0x04	必須
EndTransfer	0x05	必須
将来用に予約済み	0x06-0xFF	オプション

Note 1: どのホスト/クライアントもコマンドコード 0x00 を使ってはいけません。コマンドコード 0x00 を使うと、プロトコルの他の局面と干渉する可能性があります。コマンドコード 0x00 を実装した全てのホスト/クライアントは MDFU プロトコル仕様に準拠しません。

3.2.1.2 コマンドの書式

コマンドは複数のフィールドにより構成されます。コマンドは 1 バイトのコマンド シーケンス フィールドと 1 バイトのコマンドコードを含み、一部のコマンドはデータ ペイロードも含みます。個々のコマンドのデータペイロードの内容と意味は、各コマンドの詳細定義に記載しています。

コマンド データペイロードに格納可能な最大バイト数 (N_C) はクライアントに依存し、MaxCommandDataLength の値と一致します。クライアントは、更新の探索フェイズ中にクライアント バッファ情報パラメータを使って MaxCommandDataLength の値をホストに報告します。

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 レスポンス

MDFU プロトコルはレスポンスを各種の目的で使います。レスポンスにより以下が可能となります。

- ホストはコマンドがクライアントによって正常に実行された事を確認できます。
- ホストは正常に実行されたコマンドに関連するデータを取得できます。
- ホストは、レスポンスとプロトコル フロー制御規則に基づいて、クライアントが次のコマンドを受信可能かどうか確認できます。
- ホストは、クライアントが回復可能なエラーに遭遇した事を検知してホストのエラー回復機能をトリガする事ができます。
- クライアントが回復不可能なエラーに遭遇した場合、ホストはエラーに関する情報をクライアントから取得して問題の診断とデバッグに役立てる事ができます。
- 回復不可能なエラーが検出された場合、クライアントは更新プロセスの最後まで待つ事なく直ちに更新を中止し、ホストに更新が失敗した事を知らせる事ができます。

3.2.2.1 レスポンスのフォーマット

レスポンスは 1 バイトのレスポンス シーケンス フィールド、1 バイトのステータスコード、可変長のレスポンス データペイロードで構成されます。これらのフィールドが互いに連結されてレスポンスが形成されます(下図参照)。

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

レスポンス データペイロードに格納可能な最大バイト数(N_R)はクライアントに依存します。MDFU プロトコル バージョン 1.0.0 で定義されている最も長いレスポンスは *GetClientInfo* レスポンスです。このプロトコルによって定義されているクライアント パラメータ (全てのオプション パラメータを含む)を格納した *GetClientInfo* レスポンスのデータペイロードは 28 バイトです(6 バイトのプロトコル バージョン パラメータ + 5 バイトのクライアント バッファ情報パラメータ + 17 バイトのクライアント コマンド タイムアウト パラメータ(4 つの SpecificCommandTimeOut を含む) = 28 バイト)。*GetClientInfo* レスポンスにオプションのパラメータが追加されるとレスポンスの長さは増加します。将来のプロトコル バージョンでは、これより長いレスポンスが定義される可能性があります。

3.2.2.1.1 レスポンス ステータス

下の表に、利用可能なレスポンス ステータスコードを示します。クライアントはレスポンス ステータスコードを使って、コマンド実行が成功したかどうか、コマンド実行を妨げる問題/エラーが発生したかどうか、ファームウェア更新プロセスの中止が必要かどうかをホストに知らせる事ができます。

これらのコードの一部は必須であるため全てのクライアントに実装する必要があります。その他のコードを実装するかどうかはクライアントの開発者が選択できます。

レスポンス ステータス	レスポンス ステータスコード	クライアントへの実装
予約済み - 使用禁止 [1]	0x00	N/A
SUCCESS	0x01	必須
COMMAND_NOT_SUPPORTED	0x02	必須
将来用に予約済み	0x03	オプション
COMMAND_NOT_EXECUTED	0x04	必須
ABORT_FILE_TRANSFER	0x05	オプション
将来用に予約済み	0x06 - 0xFF	オプション
Note 1: どのホスト/クライアントもレスポンス ステータスコード 0x00 を使ってはいけません。コード 0x00 を使うと、プロトコルの他の局面と干渉する可能性があります。レスポンス ステータスコード 0x00 を実装した全てのホスト/クライアントは MDFU プロトコル仕様に準拠しません。		

3.2.2.1.2 レスポンス データペイロード

レスポンス データペイロードの内容は、レスポンスに格納されるステータスコードに応じて異なります。

- ステータスコードが SUCCESS である場合、そのレスポンスはコマンドが正常に実行された事を示し、データペイロードはコマンド実行に関連するデータを格納します。SUCCESS レスポンス データペイロードの定義は、各コマンド/レスポンス ペアの詳細定義に記載しています。
- ステータスコードが SUCCESS ではない場合、クライアントがエラーを検出した事を意味します。この場合、レスポンス データペイロードには問題の診断とデバッグに役立つ追加の情報を格納できます。エラーレスポンス ステータスコードの詳細な意味と、それらのステータス レスポンスのデータペイロードの定義については、「エラーレスポンスの詳細」を参照してください。

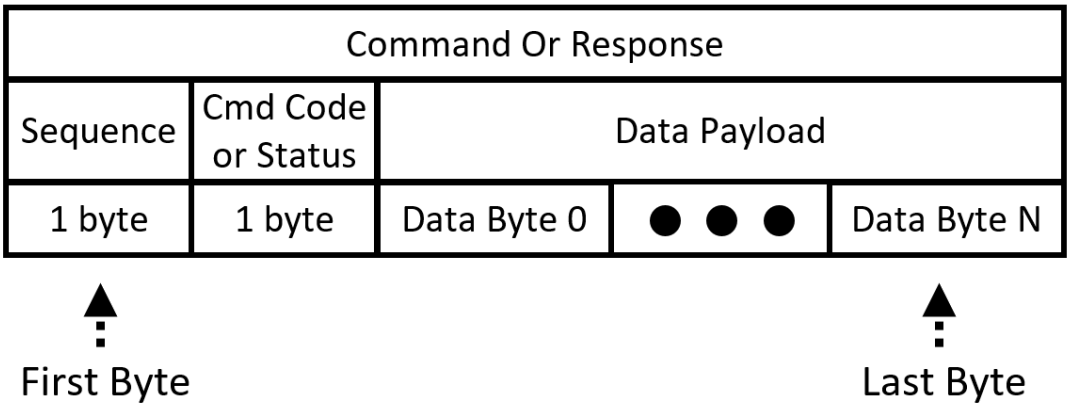
Response		
Response Sequence	Status Code	Response Data Payload
0xYY	0x01 SUCCESS	Data Related to the Successful Command Execution
	Other Status Codes	Information about Errors Encountered

3.2.3 コマンド/レスポンス内のバイト順

ホストとクライアントの相互運用を可能にするため、MDFU プロトコルはコマンド/レスポンス内のバイトの並び順とマルチバイト フィールド内のバイトの並び順を規定しています。

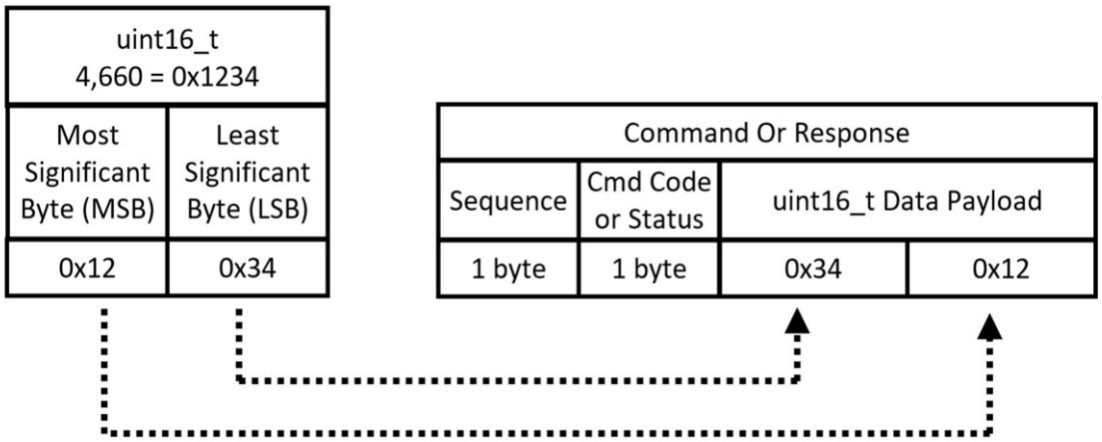
コマンド/レスポンス内のバイト順

本書では、図を使って各コマンドおよびレスポンス フィールドの定義を示します。これらの図には、コマンド/レスポンス内のバイトの並び順が示されます。下図に示す通り、バイトは左から右へ順番に示されます。



マルチバイト フィールド内のバイト順

コマンドまたはレスポンス内のマルチバイト データフィールドは、リトル エンディアン形式でバイトを格納します。従って、最下位バイトから順番に格納されます。下の例は、コマンド/レスポンス内の uint16_t データペイロード フィールドに符号なし 16 ビット整数値 4,660(0x1234)が格納された状態を示しています。



3.2.4 データ型

MDFU プロトコルは以下のデータ型を使います。

- uint8_t: 符号なし 8 ビット整数(0～255)
- uint16_t: 符号なし 16 ビット整数(0～65,535)

3.2.5 コマンド/レスポンス ペアの定義

以下にコマンド/レスポンス ペアの詳細な定義を記載します。

3.2.5.1 クライアント情報取得コマンド

MDFU クライアントを実装する際は、そのクライアントに固有の要件を満たせるように各種のプロトコルパラメータ(例: クライアント バッファのサイズ、コマンド タイムアウト等)の値を柔軟に選択できます。

クライアントを正常に更新するために、ホストは探索ステージ中にこれらのパラメータの値をクライアントから取得する必要があります。

ホストは、更新の探索ステージ中に *GetClientInfo* コマンドをクライアントへ送信します。これに対してクライアントは、実装の詳細情報をクライアント パラメータのリストとしてホストへ返す必要があります。

GetClientInfo コマンドはコマンドのタイムアウト値を取得するために使われるため、*GetClientInfo* コマンド自体のタイムアウト値をカスタマイズする方法はありません。*GetClientInfo* コマンドのタイムアウト値は 1 秒で固定されます。

3.2.5.1.1 コマンドの詳細

クライアント情報取得コマンドの詳細と SUCCESS レスポンスのデータ ペイロード	
コマンドコード	0x01
コマンド データペイロード	なし
SUCCESS レスポンスのデータ ペイロード	クライアント パラメータの連結リスト [Parameter1、Parameter2、…、ParameterN] は TLV (Type/Length/Value)フォーマットでエンコードされます。

クライアント パラメータ

全てのクライアントが返す必要のある必須クライアント パラメータが複数存在します。加えて、クライアントは必要に応じてオプション パラメータを選択して実装できます。

全てのクライアント パラメータは 1 つのリストとして連結され、*GetClientInfo* コマンドに対するレスポンス内でクライアントから返されます。パラメータのリストを下の表に示します。各パラメータの詳細は、本書内の対応する項目に記載しています。

パラメータ名	パラメータ型コード	必須/オプション
将来用に予約済み	0x00	オプション
プロトコル バージョン	0x01	必須
クライアント バッファ情報	0x02	必須
クライアント コマンド タイムアウト	0x03	必須
将来用に予約済み	0x04-0xFF	オプション

TLV (Type/Length/Value)フォーマット

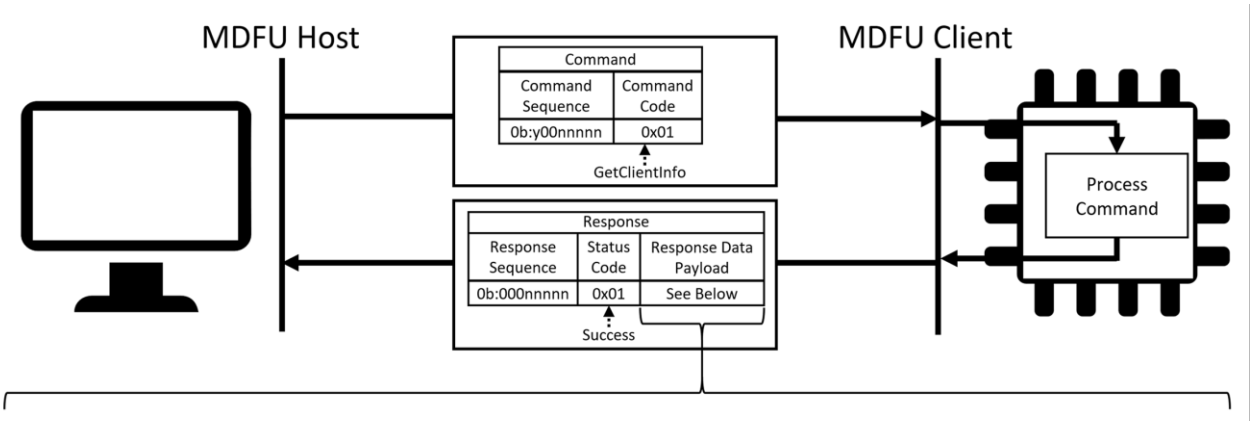
クライアント パラメータは TLV (Type/Length/Value)フォーマットでコード化されます(下図参照)。最初の 1 バイトはパラメータの型を定義し、次の 1 バイトは後続のパラメータ値のバイト数を定義します。

Parameter Type Length Value Format		
Parameter Type Code	Parameter Length Field	Parameter Value
X (1 byte)	Y (1 byte)	Y Bytes of Parameter Data

3.2.5.1.2 コマンド/レスポンス ペアの例

下の図 3-1 に、ホストからクライアントへ送信される *GetClientInfo* コマンドとクライアントからホストへ返されるレスポンス(必須パラメータのセットを格納)の例を示します。

図 3-1. コマンド/レスポンス ペアの例



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

Note 1: この例のプロトコル バージョン パラメータの値は、MDFU プロトコル バージョン 1.0.0 を示します。異なるプロトコル バージョンを実装している場合、この値を変更する必要があります。

3.2.5.1.3 プロトコル バージョン パラメータ

プロトコル バージョン パラメータにより、ホストはクライアントが実装している MDFU プロトコルのバージョンを知る事ができます。ホストは、このバージョン情報に基づいてクライアント プロセッサを更新可能かどうか確認します。このバージョンをサポートしていない場合、ホストはユーザを支援するための情報を提供します。

クライアントを更新するためにホストは以下の要件を満たす必要があります。

1. クライアントが実装しているプロトコルの MajorVersion をサポートしている事
2. クライアントが実装しているプロトコルの MinorVersion 以上をサポートしている事

上記の要件を満たしていない場合、ホストは更新を終了し、クライアントのプロトコル バージョンをサポート可能なバージョンのホストを入手するようユーザに警告する要があります。

パラメータの詳細

一般公開バージョンのプロトコル バージョン パラメータメータは 3 バイト長であり MajorVersion、MinorVersion、Patch(各 1 バイト)を格納します。Microchip 社内向けのプレリリース バージョンでは 4 つ目の PreRelease バイトが追加される場合がありますが、このバイトは一般公開バージョンには存在しません。

プロトコル バージョン パラメータ	
パラメータの型コード	0x01
パラメータの長さ	一般公開バージョンでは 3 バイト Microchip 社内プレリリース ビルドでは 4 バイト

.....続き	
プロトコル バージョン パラメータ	
パラメータの値	<MajorVersionByte><MinorVersionByte><PatchByte>[<PreReleaseByte>]

パラメータの例

下図にプロトコル バージョン パラメータの例を示します。

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

Note 1: この例のプロトコル バージョン パラメータの値は、MDFU プロトコル バージョン 1.0.0 を示します。異なるプロトコル バージョンを実装している場合、この値を変更する必要があります。

3.2.5.1.4 クライアント バッファ情報パラメータ

クライアントのリソースに応じてクライアントに実装するバッファの詳細を選択できます。クライアントは、これらのバッファ情報をホストに報告する必要があります。バッファ情報には 1 つのコマンド データ ペイロードで送信可能な最大データバイト数と、クライアントが実装しているコマンドバッファの数が含まれます。現在の MDFU プロトコルは複数のコマンドバッファをサポートしていないため、クライアントはコマンドバッファを 1 つだけ実装してバッファ数が 1 である事をホストに報告する必要があります。

パラメータの詳細

クライアントバッファ情報パラメータ	
パラメータの型コード	0x02
パラメータの長さ	3
パラメータの値	<MaxCommandDataLength><NumCmdBuffers>
MaxCommandDataLength	uint16_t: クライアントがコマンド データ ペイロードでサポート可能な最大データバイト数
NumCmdBuffers	uint8_t: クライアントが実装しているコマンドバッファの数 (現在この値は 1 である事が必要)

パラメータの例

下図にクライアント バッファ情報パラメータの例を示します。

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

3.2.5.1.5 クライアント コマンド タイムアウト パラメータ

クライアントは、コマンドの実行に要する最大時間をホストに報告する必要があります。これにより、ホストは適切にタイムアウトを判断してコマンドを再送信できます。簡潔さと柔軟性のバランスを取るため、クライアントは 2 種類のタイムアウトを報告できます。クライアント コマンド タイムアウト パラメータ値は、既定値コマンド タイムアウト値(固有のコマンド タイムアウト値が指定されない全てのコマンドに適用)で始まる必要があります。必要に応じ、クライアントは特定コマンドにのみ適用する 1 つまたは複数の固有コマンド タイムアウト値をホストに報告できます。

クライアント情報取得コマンド(*GetClientInfo*)は、コマンド タイムアウト値をクライアントから取得するために使われるため、*GetClientInfo* コマンド自体のタイムアウト値をカスタマイズする方法はありません。*GetClientInfo* コマンドのタイムアウト値は 1 秒に固定されます。

パラメータの詳細

クライアント コマンド タイムアウト パラメータ	
パラメータの型コード	0x03
パラメータの長さ	タイムアウトあたり 3 バイト x 報告するタイムアウトの数
パラメータの値	<DefaultCmdTimeOutCode><DefaultCommandTimeOut>[<cmdCode_1><uint16_t SpecificCommandTimeOut_1>]… [<cmdCode_N><SpecificCommandTimeOut_N>]
DefaultCmdTimeOutCode	uint8_t: 0x00
DefaultCommandTimeOut	uint16_t: 既定値タイムアウト 固有のコマンド タイムアウト値が指定されない全てのコマンドに適用
cmdCode_x	uint8_t: 固有のタイムアウト値が割り当てられるコマンドのコード
SpecificCommandTimeOut_x	uint16_t: cmdCode_x によって指定されたコマンドに割り当てる固有のタイムアウト値

タイムアウト値の単位と計算式

- DefaultCommandTimeOut と SpecificCommandTimeOut_x は uint16_t 値です。
- これらのパラメータの LSB は 0.1 秒です。
- 最小タイムアウト値は 0.1 秒です。
- 最大タイムアウト値は 65,535 x 0.1 秒 = 6,553.5 秒 = 109.225 分です。
- TimeOutInSeconds = TimeOutParameterValue x (0.1 秒 / LSB)
- TimeOutParameterValue = TimeOutInSeconds / (0.1 秒 / LSB)

パラメータの例

次のページに、2 通りのクライアント コマンド タイムアウト パラメータの設定例を示します。

最初の例には、既定値タイムアウトのみを使う最もシンプルなケースを示します。この既定値タイムアウトが全てのコマンドに適用されます。

図 3-2. 例 1: 1 秒間の既定値タイムアウトを全てのコマンドに適用する場合

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

次の例として、特定コマンドのタイムアウトを変更する方法を示します。この例では、*GetImageState* コマンドにのみ 10 秒間のタイムアウトを適用しています。その他の全てのコマンドには 1 秒間の既定値コマンド タイムアウトが適用されます。

図 3-3. 例 2: 1 秒間の既定値タイムアウトを設定し GetImageState コマンドにのみ 10 秒間の固有タイムアウトを適用する場合

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 転送開始コマンド

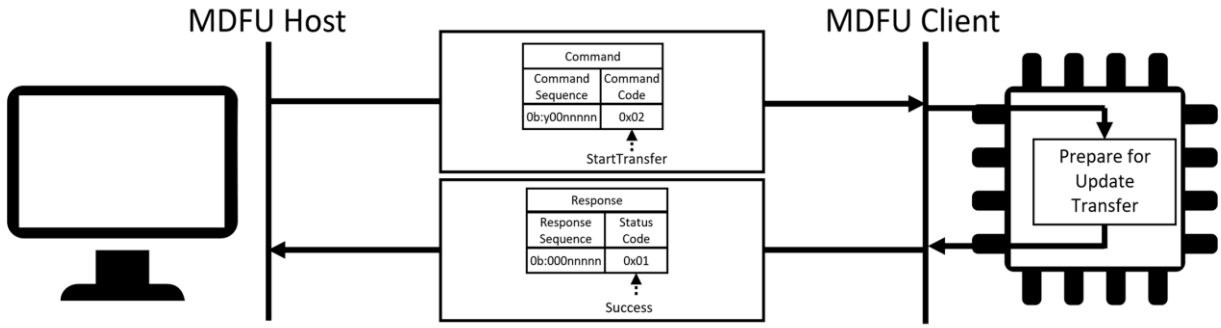
ホストは転送開始ステージ中に転送開始(*StartTransfer*)コマンドをクライアントへ送信する事により、クライアント更新ファイルの転送が間もなく始まる事をクライアントに伝えます。クライアントは、クライアント更新ファイルの内容を先頭から完全に受信できるよう準備する必要があります。クライアントは、クライアント更新ファイルの受信準備が完了した後に SUCCESS ステータスをホストに返す事により、新しいクライアント更新ファイルの受信が可能である事をホストに知らせる必要があります。

3.2.5.2.1 コマンドの詳細

転送開始コマンドと SUCCESS レスポンス データペイロード	
コマンドコード	0x02
コマンド データペイロード	なし
SUCCESS レスポンス データペイロード	なし

3.2.5.2.2 コマンド/レスポンス ペアの例

次ページの図に、ホストからクライアントへ送信される *StartTransfer* コマンドと、更新の開始が可能である事を示すクライアントからのレスポンスの例を示します。



3.2.5.3 チャンク書き込みコマンド

更新の転送開始ステージ中に、ホストは *WriteChunk* コマンドをクライアントへ送信します。これらのコマンドは、クライアント更新ファイル内のデータの一連のチャンクを格納します(「ファイルのチャンク化」参照)。

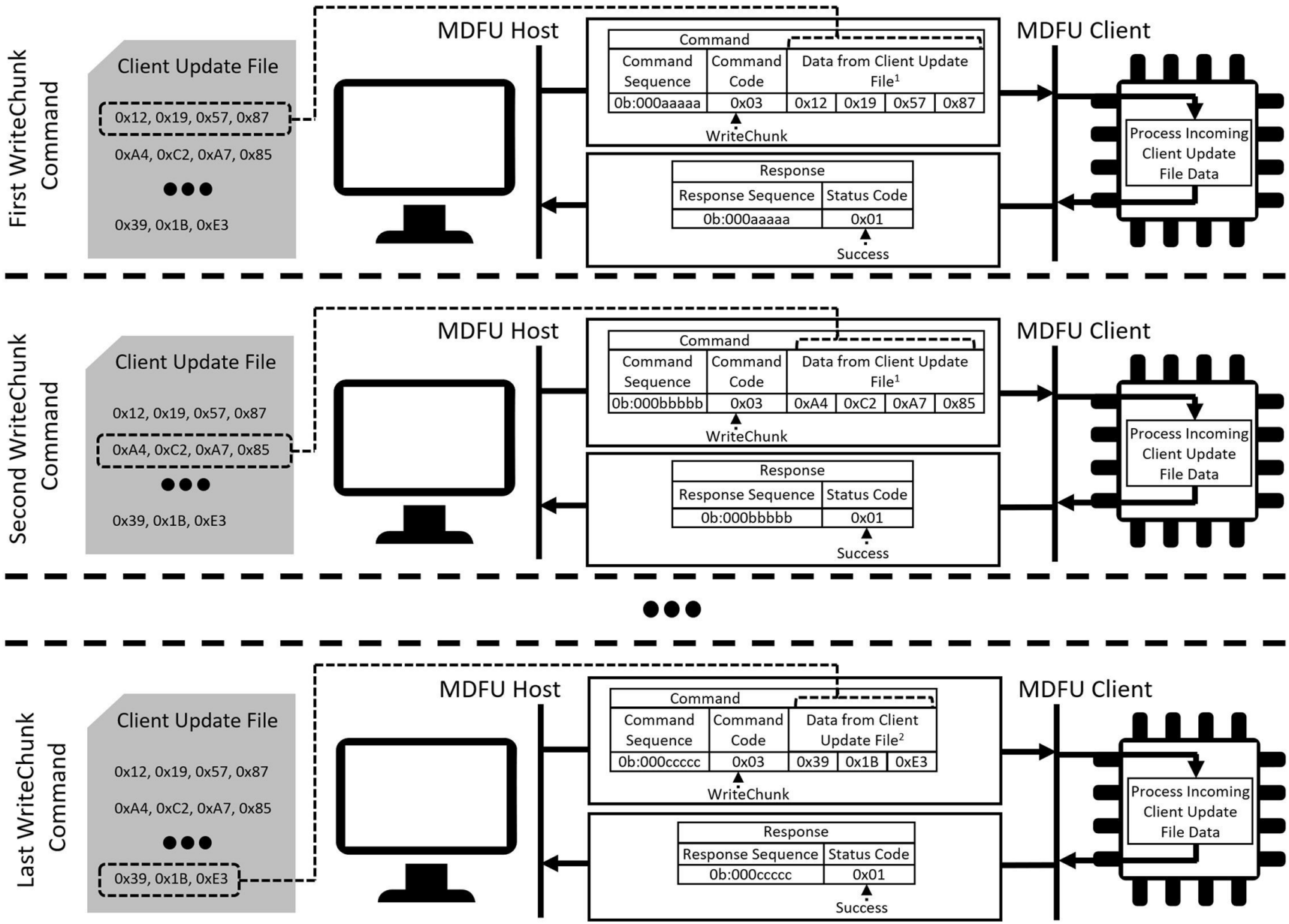
3.2.5.3.1 コマンドの詳細

チャンク書き込みコマンドと SUCCESS レスポンスのデータ ペイロード	
コマンドコード	0x03
コマンド データペイロード	<クライアント更新ファイルからの N バイトのデータ>
SUCCESS レスポンス データペイロード	なし
N	NumberRemainingBytesInFile >= MaxCommandDataLength の場合: N = MaxCommandDataLength NumberRemainingBytesInFile < MaxCommandDataLength の場合: N = NumberRemainingBytesInFile

3.2.5.3.2 コマンド/レスポンス ペアの例

次ページの図に、ホストが *WriteChunk* コマンドを使ってクライアント更新ファイルからの一連のチャンクをクライアントへ送信する例を示します。この例では、ファイルを 4 バイトチャンクに分割してクライアントへ送信しています。各チャンクのバイト数は、探索ステージ中にクライアントから取得されます(クライアント バッファ情報パラメータの MaxCommandDataLength フィールドから取得)。クライアント更新ファイルは、サイズが MaxCommandDataLength のチャンクに分割されてクライアントへ送信されます。最後のチャンクが MaxCommandDataLength バイトに満たない場合、最後のチャンクのサイズは単純にクライアント更新ファイル内の残りのバイト数となります。

図 3-4. コマンド/レスポンス ペアの例



Note 1: この例では、探索ステージ中に取得されたクライアント バッファ情報パラメータ MaxCommandDataLength は 4 バイトを示し、*WriteChunk* コマンド向けにファイルは 4 バイトのチャンクに分割されると想定しています。

Note 2: クライアント更新ファイルの最後のチャンクは 3 バイトしか含まないため、最後の *WriteChunk* コマンドではファイルデータが 3 バイトだけクライアントへ送信されます。

3.2.5.4 イメージ状態取得コマンド

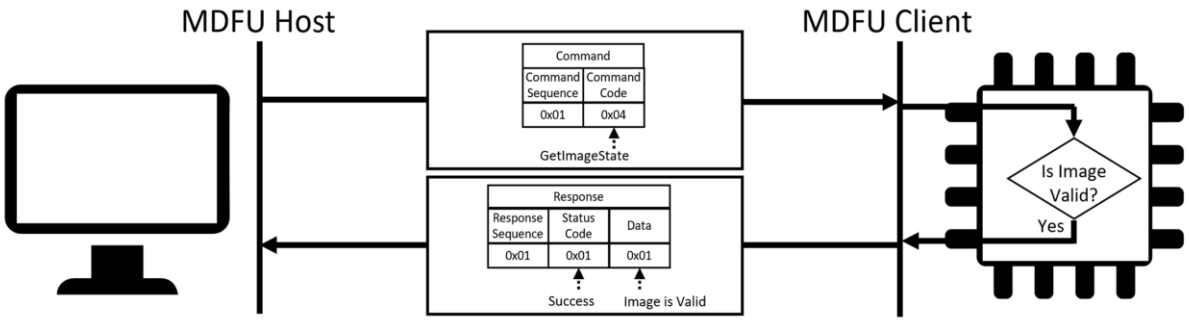
ホストは、更新の検証ステージ中に *GetImageState* コマンドをクライアントへ送信します。クライアントは検証アルゴリズムを実行してファームウェア イメージの状態を判定し、そのイメージが有効であるかどうかを示すレスポンスをホストに返します。

3.2.5.4.1 コマンドの詳細

イメージ状態取得コマンドと SUCCESS レスポンスのデータペイロード	
コマンドコード	0x04
コマンド データペイロード	なし
SUCCESS レスポンス データペイロード	<uint8_t:ImageState>
ImageState の値	IMAGE_VALID = 0x01 IMAGE_INVALID = 0x02

3.2.5.4.2 コマンド/レスポンス ペアの例

下の図に、ホストからクライアントへ送信される *GetImageState* コマンドと、イメージが有効である事を示すクライアントからのレスポンスの例を示します。



3.2.5.5 転送終了コマンド

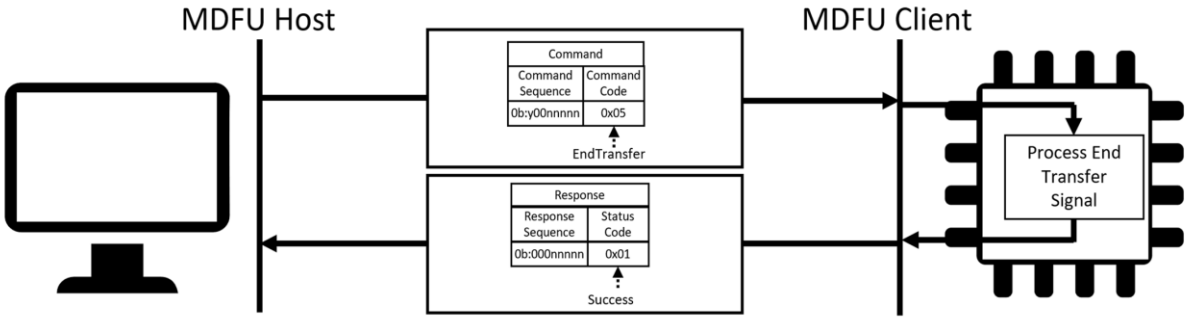
ホストは更新の転送終了ステージ中に *EndTransfer* コマンドをクライアントへ送信する事により、更新ファイルの転送が完了した事をクライアントに示します。クライアントは、SUCCESS ステータス レスポンスにより、このコマンドの受信と処理に成功した事をホストに知らせます。

3.2.5.5.1 コマンドの詳細

転送終了コマンドと SUCCESS レスポンスのデータペイロード	
コマンドコード	0x05
コマンド データペイロード	なし
SUCCESS レスポンス データペイロード	なし

3.2.5.5.2 コマンド/レスポンス ペアの例

次ページの図に、ホストからクライアントへ送信される *EndTransfer* コマンドとクライアントからホストへ返される SUCCESS レスポンスの例を示します。



3.2.6 エラーレスポンスの詳細

更新中に MDFU クライアントがエラーを検出した場合、エラーに関する情報をホストに伝える事が重要です。

クライアントはレスポンスを介してエラー条件をホストに報告します。エラーが検出された時にクライアントは、エラーが発生した事を示すレスポンス ステータスと一緒に、エラーに関する追加情報を格納したレスポンス データペイロードをホストへ返す事ができます。

以下にエラーレスポンスの詳細な定義を記載します。これには、それらのステータスコードとデータペイロードの定義が含まれます。

3.2.6.1 COMMAND_NOT_SUPPORTED レスポンス

ホストからコマンドを受信した時に、クライアントはそのコマンドがクライアントに実装済みであるかどうか確認する必要があります。未実装コマンドを受信した場合、クライアントは COMMAND_NOT_SUPPORTED ステータスを示すレスポンスを生成する必要があります。このレスポンスは、クライアントがそのコマンドを実装していないという事をホストに伝えます。

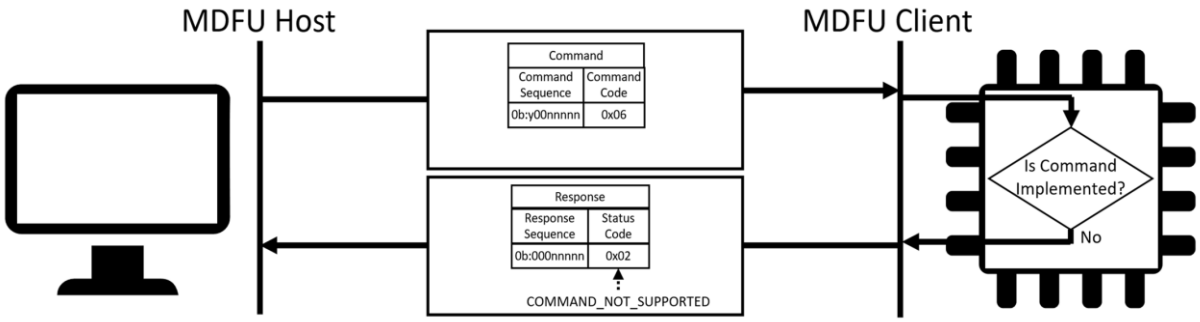
このレスポンスがクライアントによって生成される場合の詳細は、「クライアントによるコマンド処理/レスポンス生成」を参照してください。

3.2.6.1.1 レスポンスの詳細

COMMAND_NOT_SUPPORTED レスポンス	
ステータスコード	0x02
レスポンス データペイロード	なし

3.2.6.1.2 レスポンスの例

下図に COMMAND_NOT_SUPPORTED レスポンスの例を示します。



3.2.6.2 COMMAND_NOT_EXECUTED レスポンス

クライアントはコマンドを受信した時に各種のエラーチェックを実行します。これらのチェックにより、受信コマンドの問題が検出されます。

これらの問題にはコマンド破損エラー、フレーミング エラー、シーケンス番号エラーが含まれます。これらのエラーが検出された場合、クライアントはコマンドを実行せずに COMMAND_NOT_EXECUTED レスポンスをホストへ返します。

このレスポンスがクライアントによって生成される場合の詳細は、「クライアントによるコマンド処理/レスポンス生成」を参照してください。

3.2.6.2.1 レスポンスの詳細

COMMAND_NOT_EXECUTED レスポンス	
ステータスコード	0x04
レスポンス データペイロード	[uint8_t CommandNotExecutedCause][1]

Note 1: COMMAND_NOT_EXECUTED レスポンス内のデータペイロード CommandNotExecutedCause の報告はオプションです(必須ではありません)。

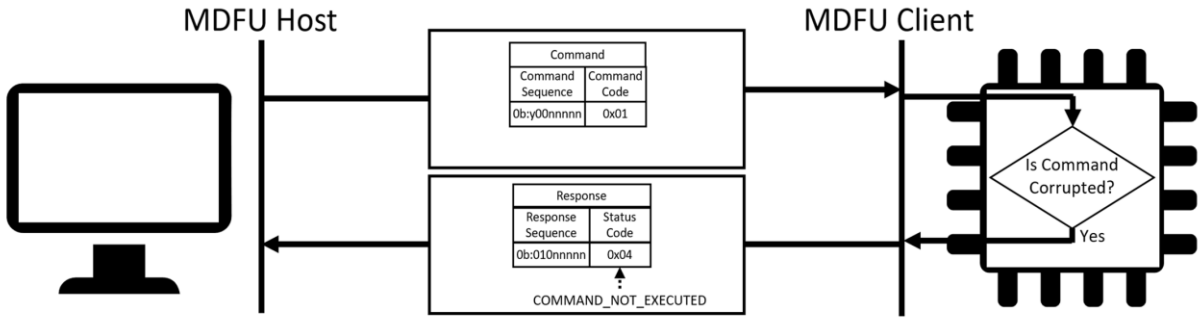
CommandNotExecutedCause の定義

クライアントからの CommandNotExecutedCause の報告は必須ではありませんが、報告する事を推奨します。小型のクライアントでは、CommandNotExecutedCause を示さずに COMMAND_NOT_EXECUTED ステータスのみ報告しても構いません(プロトコルは正しく機能します)。しかしエラーに遭遇した時に、ホストはコマンドが実行されなかった理由を特定してユーザに伝える事ができません。エラーに対するデバッグを容易にするため、クライアントから CommandNotExecutedCause を報告する事を推奨します。

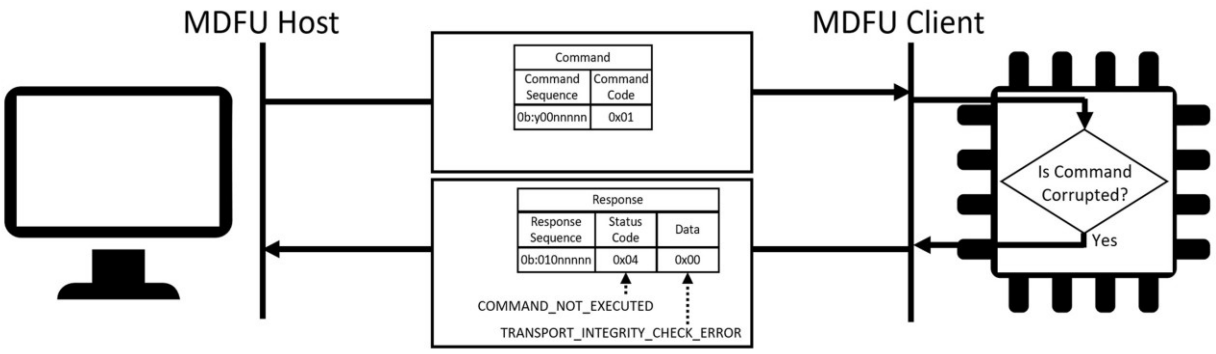
CommandNotExecutedCause の名称	CommandNotExecutedCause の値	意味
TRANSPORT_INTEGRITY_CHECK_ERROR	0x00	このコードは、受信したコマンドが整合性チェックに合格しなかった事を示します。これはホストからクライアントへの転送中にコマンドが破損した場合に発生します。
COMMAND_TOO_LONG	0x01	このコードは、受信したコマンドがクライアント バッファのサイズを超えていた事を示します。
COMMAND_TOO_SHORT	0x02	このコードは、受信したコマンドが短すぎる事を示します。
SEQUENCE_NUMBER_INVALID	0x03	このコードは、受信したコマンドのシーケンス番号が無効である事を示します。
将来用に予約済み	0x04-0xFF	将来用に予約済み

3.2.6.2.2 レスポンスの例

下図に、CommandNotExecutedCause が何も報告されない場合の COMMAND_NOT_EXECUTED レスポンスの例を示します。



次ページの図に、CommandNotExecutedCause として TRANSPORT_INTEGRITY_CHECK_ERROR が報告される場合の COMMAND_NOT_EXECUTED レスポンスの例を示します。



3.2.6.3 ABORT_FILE_TRANSFER レスポンス

MDFU プロトコルは、クライアントが回復不可能なエラーを検出した時に ABORT_FILE_TRANSFER レスポンスを使って更新を即座に中止する機能を提供します。クライアントは、全てのコマンドに対して ABORT_FILE_TRANSFER レスポンスを返す事によって更新を中止できます。加えて、クライアントは更新を中止した理由をホストに伝える事ができます。

このレスポンスがクライアントによって生成される場合の詳細は、「回復不可能なエラー」と「クライアントによるコマンド処理/レスポンス生成」を参照してください。

3.2.6.3.1 レスポンスの詳細

ABORT_FILE_TRANSFER レスポンス	
ステータスコード	0x05
レスポンス データペイロード	[uint8_t FileAbortCause][1]
Note 1: ファイル転送を中止する時に、デバッグ用に原因を FileAbortCause として報告する事ができます(FileAbortCause の報告はオプションであり必須ではありません)。	

FileAbortCause の定義

クライアントからの FileAbortCausee の報告は必須ではありませんが、報告する事を推奨します。非常に小さなクライアントでは、FileAbortCaus を示さずに ABORT_FILE_TRANSFER ステータスのみ報告しても構いません(プロトコルは正しく機能します)。しかしエラーに遭遇した時に、ホストは ABORT_FILE_TRANSFER の発生原因を特定してユーザに伝える事ができません。デバッグを容易にするため、エラー遭遇時にクライアントから FileAbortCause を報告する事を推奨します。

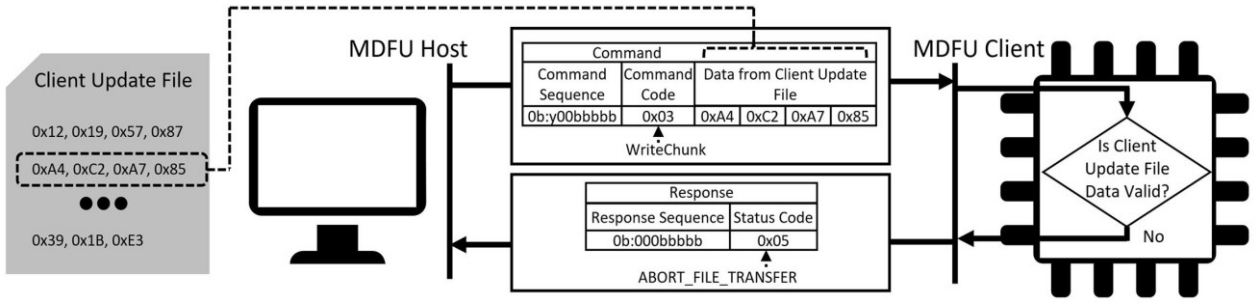
FileAbortCause の名称	FileAbortCause の値	クライアントが更新を中止した理由
GENERIC_CLIENT_ERROR	0x00	このコードは、クライアントが一般的な問題に遭遇した事を示します。
INVALID_FILE	0x01	このコードは、クライアント更新ファイル内で一般的な問題が検出された事を示します。
INVALID_CLIENT_DEVICEID	0x02	クライアント更新ファイルがクライアント デバイス ID に適合しない事を示します。
ADDRESS_ERROR	0x03	クライアント更新ファイル内に無効なアドレスが存在する事を示します。
ERASE_ERROR	0x04	クライアントメモリが正しく消去されなかった事を示します。
WRITE_ERROR	0x05	クライアントメモリが正しく書き込まれなかった事を示します。
READ_ERROR	0x06	クライアントメモリが正しく読み出されなかった事を示します。
APPLICATION_VERSION_ERROR	0x07	クライアントはクライアント更新ファイル内のアプリケーション バージョンの変更を許可しません(Anti-rollback)。

.....続き

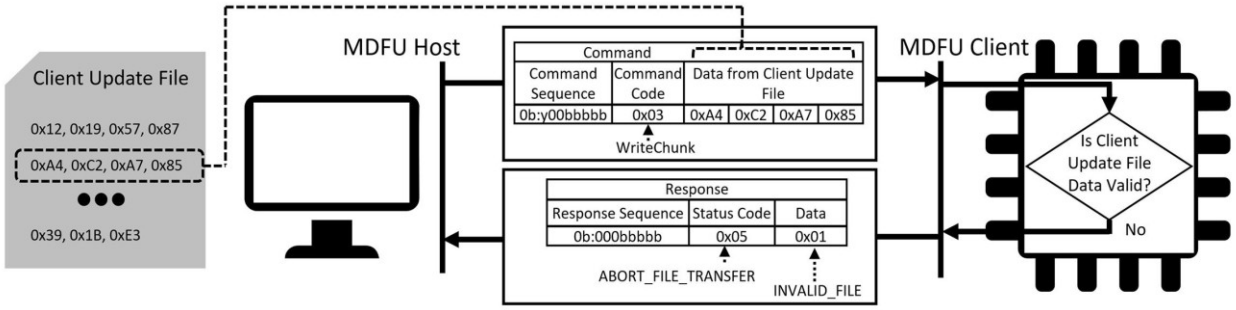
FileAbortCause の名称	FileAbortCause の値	クライアントが更新を中止した理由
将来用に予約済み	0x08-0xFF	将来用に予約済み

3.2.6.3.2 レスポンスの例

下図に、FileAbortCause が報告されない場合の ABORT_FILE_TRANSFER レスポンスの例を示します。



下図に、FileAbortCause として INVALID_FILE が報告される場合の ABORT_FILE_TRANSFER レスポンスの例を示します。



3.3 ホストによるコマンド生成/レスポンス処理

以下では、ホストがコマンドを生成/送信すると共にクライアントからのレスポンスを処理するためのプロセスについて説明します。

3.3.1 ホストによるコマンドの生成

MDFU プロトコル内の全ての通信はホストによって開始されます。ホストは、クライアントにファームウェア更新プロセスを手順通りに実行させるために必要なコマンドを、ファームウェア更新アルゴリズムを使って決定します。

ホストはこれらのコマンドを構成し、サポートされるいずれかの物理的通信バス向けのトランスポート層定義を使って、コマンドをクライアントへ送信します。

ホストはフロー制御規則に従って、次のコマンドが送信可能であるかどうかを判別します。

3.3.2 ホストによるレスポンスの処理

ホストが送信した各コマンドに対してクライアントからレスポンスが返されます。

クライアントからのレスポンスはトランスポート層によって受信されます。

レスポンスが届くと、ホストは各種の条件に対してレスポンスをチェックします。これらのレスポンス条件により、ホストはコマンドが正常に実行されたかどうかと、コマンドに対するレスポンスデータが利用可能かどうかを判断できます。さらに、ホストはエラーが発生したかどうかを判別し、それらのエラーへの対処方法を決定できます。回復可能なエラーが検出された場合、ホストの自動回復機能が起動します。

回復不可能なエラーが検出された場合、更新プロセスは終了し、デバッグ情報を含むエラーメッセージがユーザに提供されます。

ホストはレスポンス内で以下を含む各種エラーをチェックします。

- 転送エラー: レスポンスの破損またはフレーミング エラー(例:ノイズによるビット反転)
- シーケンス番号エラー
- ABORT_FILE_TRANSFER 等のステータスエラー

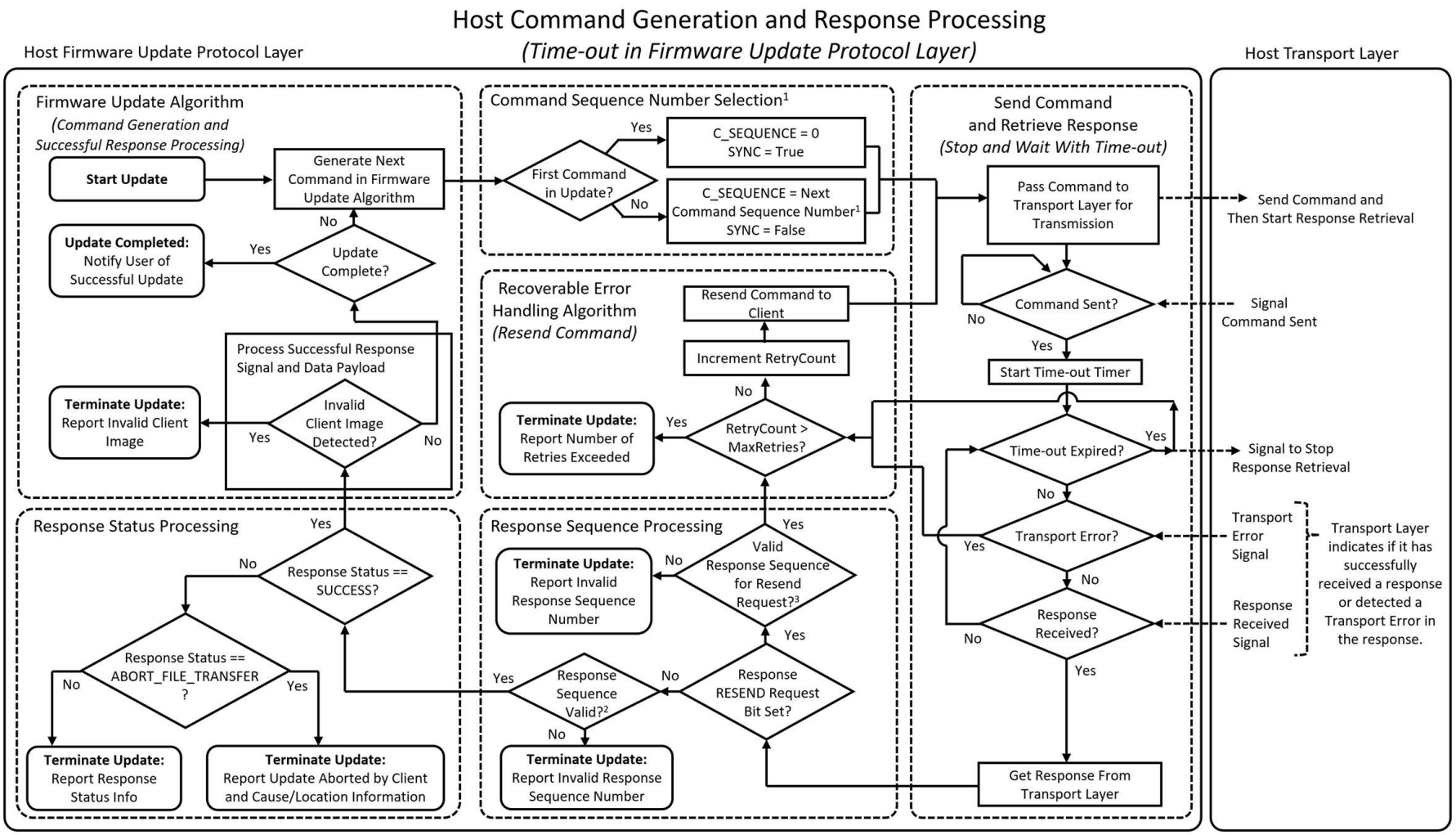
レスポンスが検出可能エラーを含まずに SUCCESS ステータスを持つ場合、ホストはこのステータスとデータペイロードを使って、送信と実行に成功したコマンド/レスポンス ペアの処理を完了します。この時点で現在のコマンド/レスポンス ペアの処理は終了し、ホストはファームウェア更新アルゴリズムを使って、次のコマンド/レスポンス ペアの処理を開始します。

3.3.3 ホストによるコマンド生成/レスポンス処理アルゴリズムのフロー図

図 3-5 と図 3-6 のフロー図に、ホストがコマンドを生成/送信しクライアントからのレスポンスを処理するプロセスを示します。

ホストの開発者は、フロー制御のタイムアウト機能をホスト ファームウェア更新プロトコル層またはホスト トランスポート層のどちらに配置するか選択できます。図 3-5 (p.31)では、タイムアウトをホスト ファームウェア更新プロトコル層に実装しています。図 3-6 (p.32)では、タイムアウトをホストトランスポート層に実装しています。

図 3-5. ホストによるコマンド生成/レスポンス処理(タイムアウトをファームウェア更新プロトコル層に実装)

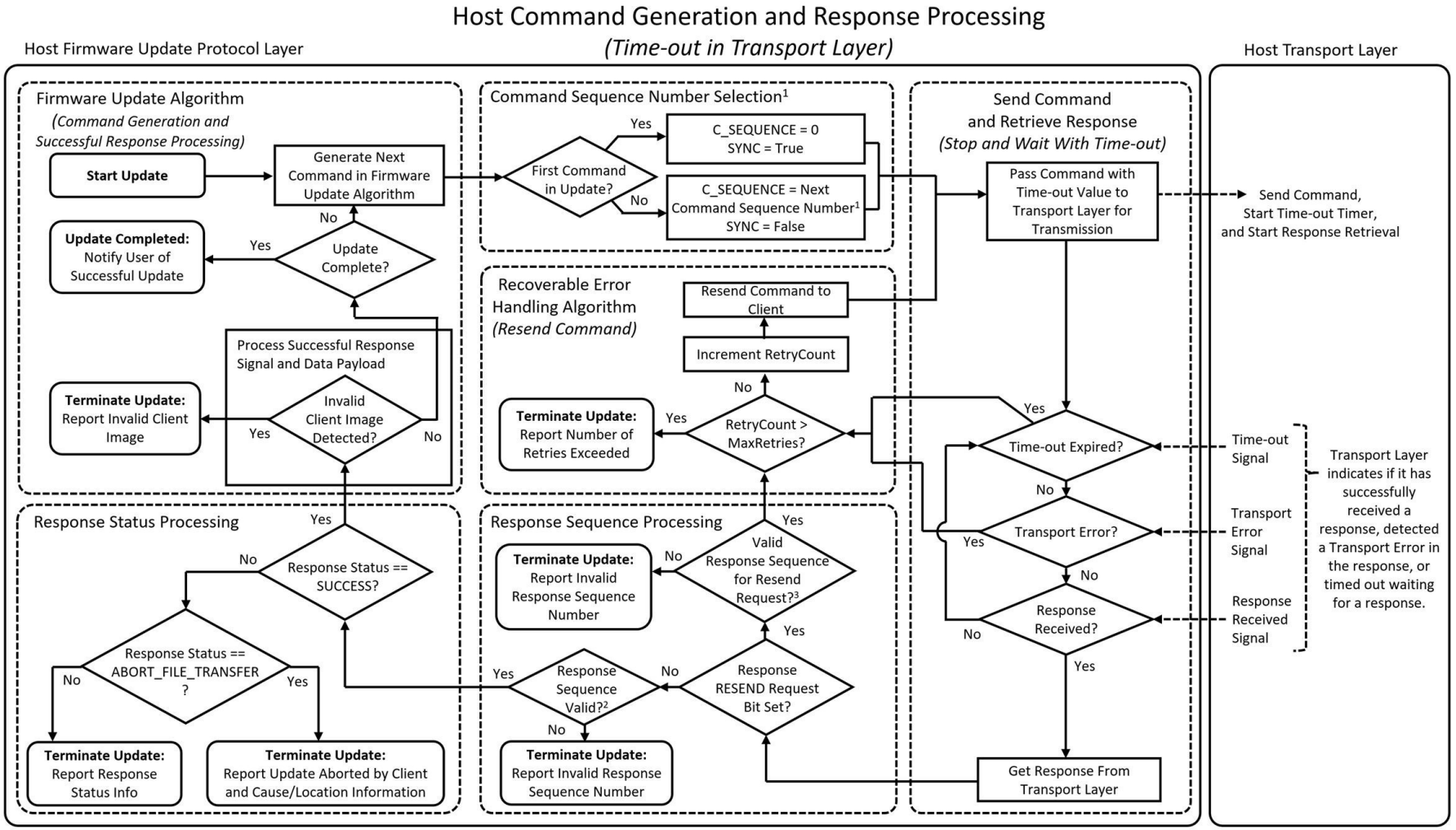


Note 1: 詳細は「ホストによるコマンド シーケンス番号の選択」を参照してください。

Note 2: R_SEQUENCE = C_SEQUENCE の場合にレスポンス シーケンス番号は有効です。

Note 3: R_SEQUENCE = C_SEQUENCE または R_SEQUENCE = (C_SEQUENCE + 1) % 32 の場合にレスポンス シーケンス番号は再送信要求向けに有効です。

図 3-6. ホストによるコマンド生成/レスポンス処理(タイムアウトをトランスポート層に実装)



Note 1： 詳細は「ホストによるコマンド シーケンス番号の選択」を参照してください。

Note 2：R_SEQUENCE = C_SEQUENCE の場合にレスポンス シーケンス番号は有効です。

Note 3： R_SEQUENCE = C_SEQUENCE または R_SEQUENCE = (C_SEQUENCE + 1) % 32 の場合にレスポンス シーケンス番号は再送信要求向けに有効です。

3.3.4 ホストのコマンド/レスポンス用バッファサイズ

MDFU プロトコルにより、最大サイズのコマンドならびに接続されるクライアントからの最大サイズのレスポンスを保存可能なバッファメモリがホストに割り当てられていれば、そのホストは接続される全てのクライアントを更新できます。

ホストは、MaxCommandDataLength バイトのファイルチャンクを格納した *WriteChunk* コマンドを送信するのに十分なバッファ空間を持つ必要があります。MaxCommandDataLength はクライアントに固有のパラメータであり、この値はクライアントごとに異なる可能性があります。加えて、レスポンス データペイロードのサイズもクライアントごとに異なる可能性があります。コマンドとレスポンスのサイズに関する詳細は「コマンド/レスポンス ペア」を参照してください。

コマンドとレスポンスの最大サイズはクライアントに依存するため、ホストに十分なサイズのコマンドおよびレスポンス バッファを持たせる必要があり、これは接続されるクライアントを更新できるよう十分なメモリをその特定のホストに割り当て可能である事を確認するために必要な統合作業です。

大容量のメモリを動的に割り当て可能な PC ホストでは、プログラムの変更なしにほとんど全てのクライアントをサポートできます。しかし、静的に割り当てられた固定サイズのバッファを持つ組み込みホストでは、特定クライアントの要件に応じてコマンドおよびレスポンス バッファへのメモリ割り当てを増やすためにリビルドが必要になる可能性があります。接続されるクライアントを更新するために十分なメモリをホストに割り当てるのは、MDFU ユーザの責任です。

3.4 クライアントによるコマンド処理/レスポンス生成

MDFU プロトコルでは、全ての通信はコマンドをクライアントへ送信するホストによって開始される必要があります。クライアントは、ホストからコマンドを受信した時に対応する処理を実行してレスポンスを生成するだけです。レスポンスはコマンド/レスポンス ペアの一部として(受信した特定コマンドに対応して)生成されます。ホストはレスポンス シーケンス フィールドを使って、そのレスポンスがどのコマンドに対応するのか識別できます。

レスポンスは以下の役目を持ちます。

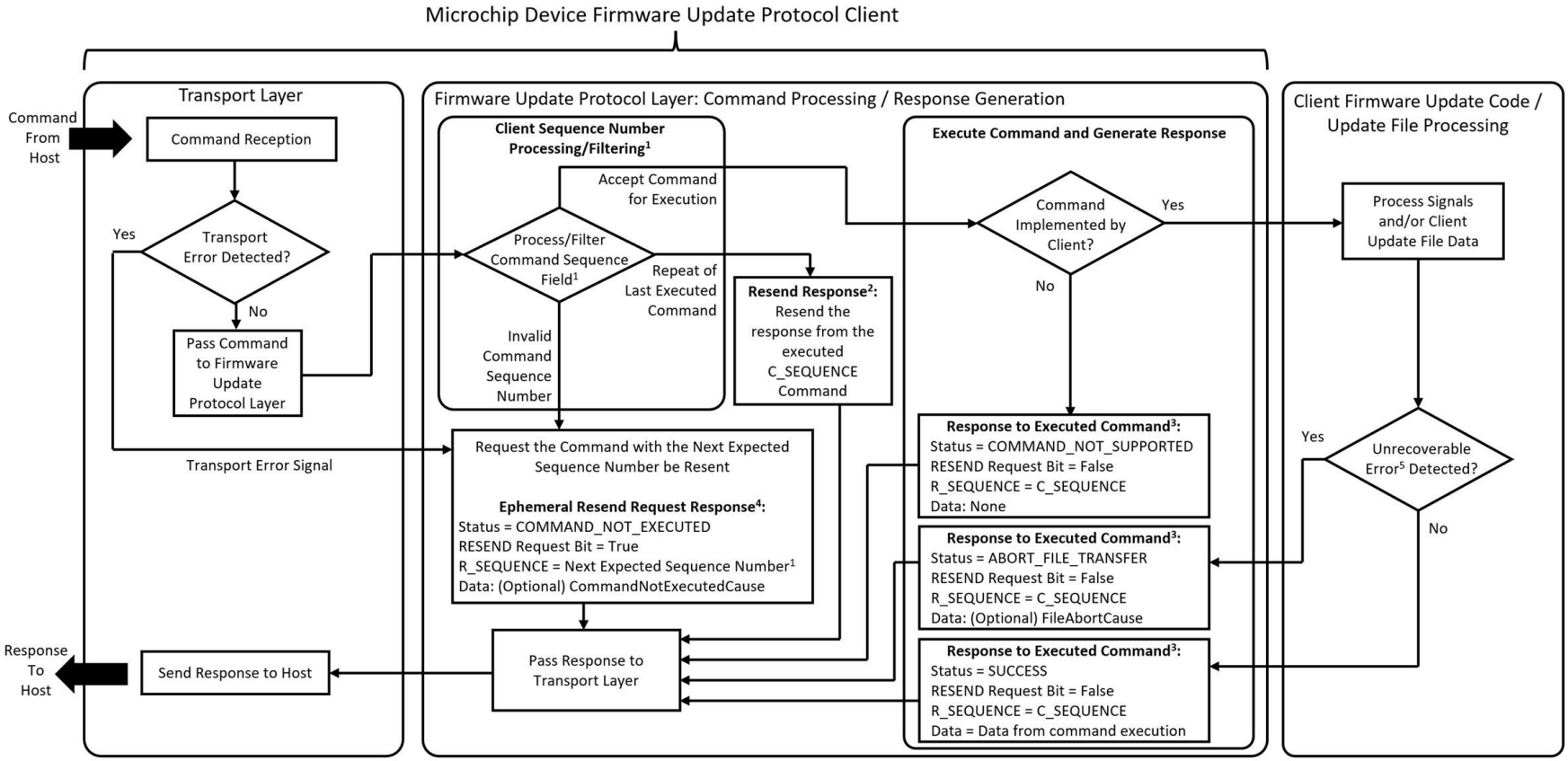
- クライアントがコマンド実行に成功したかどうかを示すステータスと、そのコマンドに関連するデータをホストへ返します。
- クライアントが検出したエラーに関する情報(以下の重要詳細を含む)をホストに報告します。
  - 検出されたエラーのタイプ: レスポンス ステータス
  - エラーからの回復を要求する場合: 再送信要求
  - エラーが回復不可能でありクライアントが更新の停止を要求する場合: ABORT_FILE_TRANSFER レスポンス
  - レスポンスがエラーステータスを含む場合: デバッグに役立つ情報をレスポンス データ ペイロードとして提供
- クライアントからのレスポンスはプロトコル フロー制御で重要な役目を持ちます。

3.4.1 クライアントによるコマンド処理/レスポンス生成アルゴリズム

次ページの図 3-7 に、クライアントの処理フロー (コマンドの受信、コマンドのエラーチェック、コマンドの処理/実行、レスポンスの生成、ホストへのレスポンス送信)を示します。

重要: クライアントは、コマンドの実行後にホストがそのコマンドに対するレスポンスの再送信をもはや要求する必要はないと結論付けるまで、そのレスポンスを保持する必要があります。エラー回復アルゴリズムの詳細は「回復可能エラー」を参照してください。クライアントが実行済みコマンドに対するレスポンスを破棄できるタイミングについては、「クライアントによるシーケンス番号処理」を参照してください。

図 3-7. MDFU プロトコルのクライアント



Note 1: 詳細は「クライアントによるシーケンス番号処理」を参照してください。

Note 2: クライアントは実行済みコマンドに対するレスポンスのみ再送信します [3]。クライアントは 1 回限りの再送信要求レスポンス(Ephemeral Resend Request Response)[4] を決して再送信しません。

Note 3: クライアントは、実行済みコマンドに対するレスポンスの再送信がホストからもはや要求される事はないと結論付けるまで、そのレスポンスを保持する必要があります。

Note 4: クライアントは 1 回限りの再送信要求レスポンス(Ephemeral Resend Request Response)を決して再送信しません。クライアントは、それらのレスポンスをホストへ送信した後直ちに破棄できます。

Note 5: これらのエラーの詳細は「回復不可能なエラー」を参照してください。

3.5 フロー制御

更新プロセス中に、ホストはコマンドを使ってクライアント更新ファイルの内容をクライアントへ送信します。以下を含む各種の理由により、ホストから理論的に可能なはずの速度でクライアント更新ファイルデータを送信しても、クライアント プロセッサは受信したコマンドまたはクライアント更新ファイルデータをその速度で処理できない場合があります。

- 一部のコントローラはフラッシュ消去/書き込み動作中のコード実行をサポートしません。
- 状況によっては、通信データレートがフラッシュ消去/書き込み速度を超える場合があります。
- クライアント ファームウェア更新ルーチンが低優先度のバックグラウンド タスクとして処理され、プロセッサ帯域幅の割り当てが制限される場合があります。

MDFU プロトコルは以下で説明するフロー制御を使う事で、コマンド/データを喪失する事なく対応できるようにクライアントが更新処理を遅らせる事を許容します。これにより、クライアントがコマンドおよび更新ファイルデータの処理に十分なリソース(処理サイクル、RAM バッファ空間等)を利用できない時にホストがそれらを送信してしまう状況を回避する事ができます。

3.5.1 タイムアウトによる停止/待機フロー制御

ホストによる停止/待機

MDFU ホストは、クライアントが一度に受信可能なデータの量を正確に認識します。

- MDFU クライアントは 1 つのコマンド受信バッファを備えます。
- MDFU ホストは、更新の探索ステージ中にこのバッファのサイズをクライアントから取得します。

ホストは、クライアントへ 1 つのコマンドを送信した後、追加のコマンド/データを送信する前に、処理を停止して送信済みコマンドに対するレスポンスを待機する必要があります。

クライアントからのレスポンスにより、クライアントが次のコマンドを受信可能である事が示されます。

レスポンスが破損した場合、そのレスポンスは正常に受信されたレスポンスの代わりに転送エラーを生成できます。従って、レスポンス転送エラーが検出された場合も、クライアントが直前のコマンドの処理を終了して少なくともコマンドの受信待機中であると見なす事ができます。

このプロセスを使う事で、クライアントの受信準備が未完了である時に MDFU ホストがクライアントへコマンド/データを送信してしまう状況を回避できます。

ホスト タイムアウト

コマンド/レスポンスの破損によって永遠に届くはずのないレスポンスをホストが待ち続ける状況を防ぐためにホスト タイムアウトが必要です。コマンド/レスポンスが転送中に喪失した場合、レスポンスは永遠にホストに届きません。

- 転送中の破損の形態によってはコマンド/レスポンスが喪失する可能性があります。この場合、コマンド/レスポンスがクライアント/ホストによって完全に受信される事は決してありません。
- 喪失したコマンドがクライアントによって処理される事は決してなく、従ってクライアントからレスポンスが生成される事もありません。
- 喪失したレスポンスがホストによって処理される事は決してありません。

ホスト タイムアウトは、コマンド/レスポンスが完全に喪失した状態から回復する機会をホストに提供します。ホストは、コマンドを再送信する事によりコマンド/レスポンスの喪失状態から回復します。

前述の停止/待機フロー制御に従い、送信済みのコマンドを処理するために十分な時間をクライアントに与える前にホストからそのコマンドを再送信する事は許可されません。

更新の探索ステージ中に、ホストはクライアントが各コマンドの実行に要する最大時間をクライアント コマンド タイムアウト パラメータから取得します。

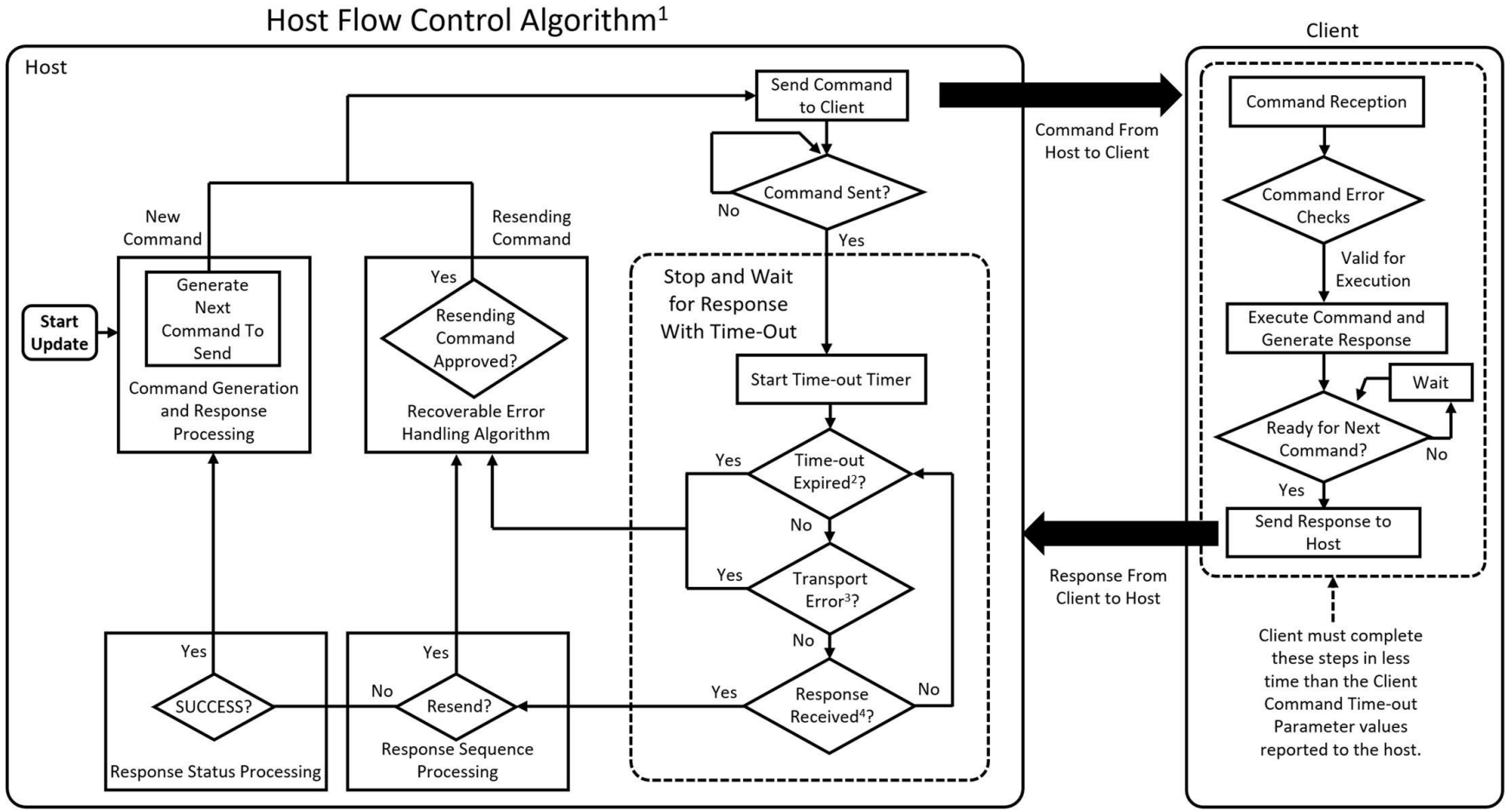
ホストはコマンド タイムアウト パラメータ情報を使って、クライアントによるコマンド実行に十分な時間を与える前にホストがタイムアウトしてコマンドを再送信してしまう事を防ぎます。

クライアント情報取得コマンド(*GetClientInfo*)は、コマンド タイムアウト値をクライアントから取得するために使われます。従って、*GetClientInfo* コマンド自体のタイムアウト値をカスタマイズする方法はありません。*GetClientInfo* コマンドのタイムアウトは 1 秒で固定されます。

3.5.2 フロー制御アルゴリズム図

次ページの図 3-8 に、MDFU プロトコルのフロー制御アルゴリズムを示します。

図 3-8. フロー制御アルゴリズム



Note 1: この図は、MDFU フロー制御の仕組みを示す事を目的とするホスト処理アルゴリズムの概略図です。アルゴリズムの詳細は「ホストによるコマンド生成/レスポンス処理」を参照してください。

Note 2: コマンドまたはレスポンスが破損した結果としてそれらが喪失した場合、レスポンスは届かずにホストはタイムアウトします。タイムアウト値の詳細は「クライアント コマンド タイムアウト パラメータ」 を参照してください。

Note 3: レスポンスの受信中に破損が検出されると転送エラーが生成されます。

Note 4: 破損が検出される事なく受信されたレスポンスは正常に受信されたレスポンスと見なされます。

3.6 更新の正常な完了

更新を正常に完了させるために、ホストはクライアントに更新プロセスの 5 つのステージを正しく実行させる必要があります。このために、ホストは一連のコマンド/レスポンス ペアを使います。更新を正常に完了させるには、以下の条件を満たす必要があります。

1. 更新中に各コマンドが破損する事なくクライアントへ転送される事
2. 更新中に各コマンドが正しい順番でクライアントによって実行される事
3. 実行されたコマンドに対する各レスポンスが破損する事なくホストへ転送される事

更新中に全てのコマンドとレスポンスが破損する事なく転送されるのが理想ですが、それらの破損が発生しても MDFU エラー回復アルゴリズムによってコマンド/レスポンスの破損から回復できれば、更新が失敗するとは限りません。

MDFU プロトコルはコマンド/レスポンスの破損を検出し、可能であれば自動的に回復するための各種アルゴリズムを定義します。MDFU プロトコルは、破損したコマンド/レスポンスを再送信するための手順と、コマンドが正しい順序で 1 度だけ実行される事を確実にするためのフィルタ処理を定義します。コマンド/レスポンスの破損を検出して回復するために MDFU が提供する機能については、本書内の以下の項目を参照してください。

- エラーの分類と処理
- シーケンス番号
- コマンド/レスポンスの整合性チェック
- フロー制御

3.7 エラーの分類と処理

MDFU プロトコルはエラーを回復可能なエラーと回復不可能なエラーに分類します。回復可能なエラーが検出された場合、MDFU プロトコルの回復機能がトリガされます。回復不可能なエラーが検出された場合、ファイル転送中止機能がトリガされます。これにより更新は即座に終了し、ユーザにデバッグ情報が提供されます。

3.7.1 MDFU エラー検出/回復機能の限界

エラー検出/回復機能は、更新プロセスの信頼性と堅牢性の強化を目的とします。しかし、これらの機能の効果には限界があり、想定される全てのタイプのエラーを検出して回復できるわけではありません。ユーザは、プロトコル内で提供される機能がアプリケーションに対して十分であるかどうかを評価する必要があります。

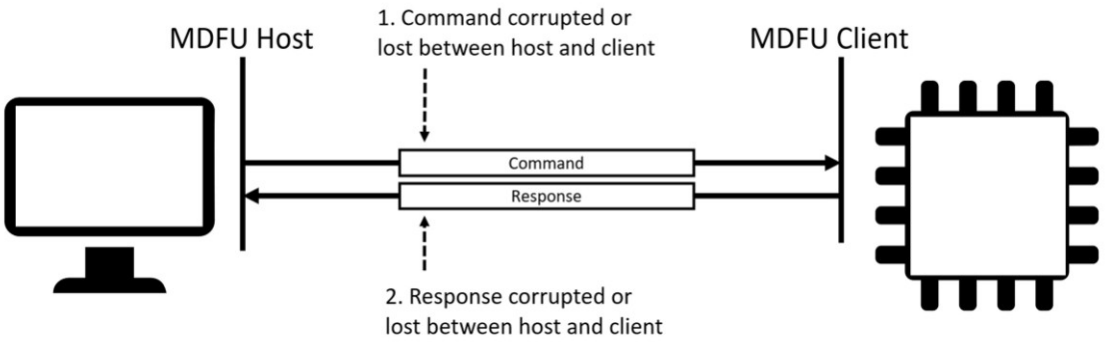
3.7.2 回復可能なエラー

MDFU プロトコルは、エラーを検出してエラーから回復するための機能を定義します。回復可能なエラーが更新中に発生した場合、適切な一連の動作を使ってこのエラーを修正できます。エラーが修正された後に更新を再開して正常に完了できます。

以下のエラーは回復可能なエラーと見なされます。

1. ホストからクライアントへの転送中に破損したコマンド
2. クライアントからホストへの転送中に破損したレスポンス
3. 物理的通信バス上の破損により完全に喪失したコマンド
4. 物理的通信バス上の破損により完全に喪失したレスポンス

コマンドとレスポンスは物理的バスを介して転送されるため、ノイズによるビット反転等が原因でこれらのエラーが発生します。



3.7.2.1 エラーの検出

各トランスポート層定義の役目の 1 つは、コマンドとレスポンスが通信バスを介して送信された時にそれらが破損していないかどうかを判定するためのコマンド/レスポンス整合性チェック機能を提供する事です。

コマンドとレスポンスの完全な喪失は、コマンド タイムアウトを使って検出されます。タイムアウト値は探索ステージ中にクライアントから取得され、フロー制御アルゴリズムの一部として組み込まれます。

3.7.2.2 ホストの回復アルゴリズム

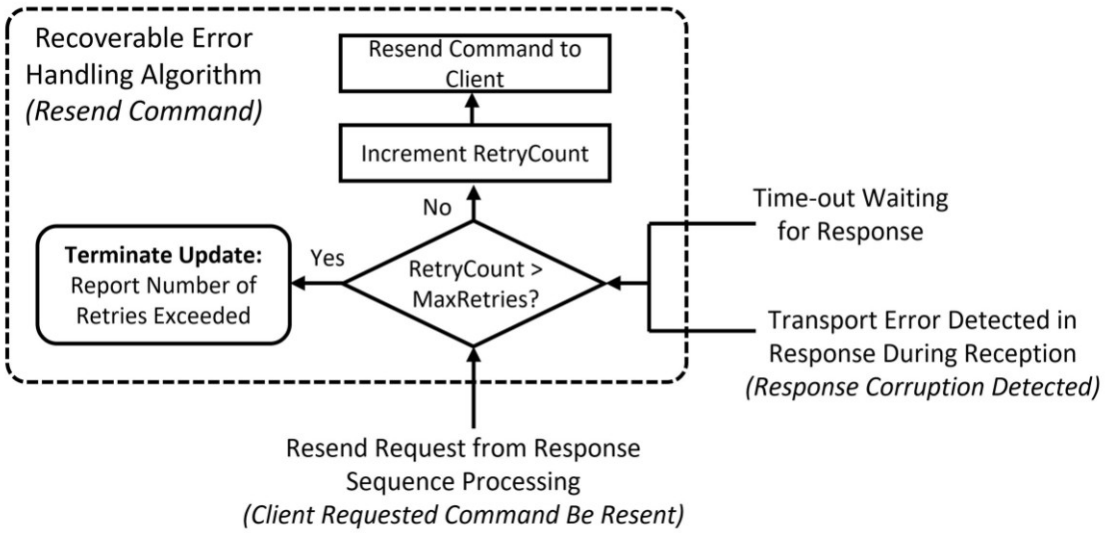
破損から回復するためにホストは以下を行う必要があります。

1. クライアントから要求された時にコマンドを再送信する
2. コマンド タイムアウト期間中にレスポンスを受信しなかった場合にコマンドを再送信する
3. レスポンスの破損を検出しコマンドを再送信する事によりこの破損から回復する

図 3-9 に示す通り、ホスト回復アルゴリズムは非常にシンプルです。

ホストは、検出された全てのエラーを記録すると共にユーザに通知するための機能を提供する必要があります。通信バスの問題をユーザに明確に示すため、ホストは更新中止を判断するための最大再試行回数(MaxRetries)を設定可能なパラメータとして提供する必要があります。

図 3-9. ホストのエラー回復アルゴリズム



3.7.2.3 クライアントの回復アルゴリズム

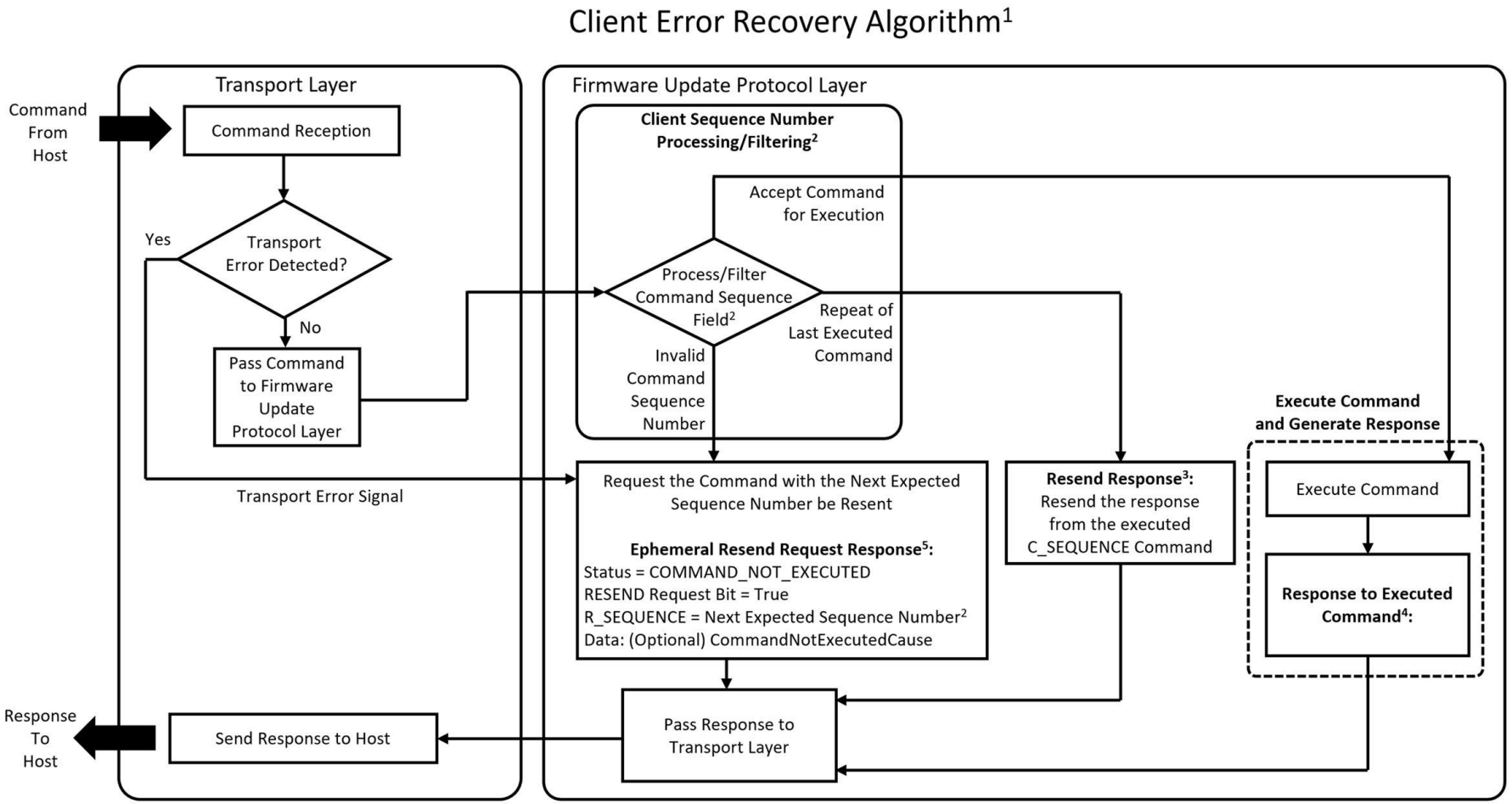
破損から回復するためにクライアントは以下を行う必要があります。

1. コマンドの破損を検出した時にそのコマンドを再送信するようホストに要求する
2. コマンドが正しい順序で 1 度だけ実行される事を確実にするために受信シーケンス番号フィルタ処理を実装する
3. ホストがレスポンスの破損から回復している時に、コマンドを再実行する事なくホストへレスポンスを再送信する

> **重要:** クライアントは、コマンドの実行後にホストがそのコマンドに対するレスポンスの再送信をもはや要求する必要はないと結論付けるまで、そのレスポンスを保持する必要があります。実行済みのコマンドに対するレスポンスをクライアントが破棄できるタイミングについては、「クライアント シーケンス番号処理」を参照してください。

次ページの図 3-10 に、クライアントによる前述の 3 つのエラー回復動作を示します。

図 3-10. クライアントの回復アルゴリズム



Note 1: この図には、クライアントによるエラーからの回復に関係するクライアント アルゴリズムだけを記載しています。クライアントによるコマンド処理およびレスポンス生成アルゴリズムの全体に関する説明は「クライアントによるコマンド処理/レスポンス生成」に記載しています。

Note 2: 詳細は「クライアントによるシーケンス番号処理」を参照してください。

Note 3: クライアントは実行済みコマンドに対するレスポンスのみ再送信します [4]。クライアントは 1 回限りの再送信要求レスポンス(Ephemeral Resend Request Response)[5] を決して再送信しません。

Note 4: クライアントは、実行済みコマンドに対するレスポンスの再送信がホストからもはや要求される事はないと結論付けるまで、そのレスポンスを保持する必要があります。

Note 5: クライアントは 1 回限りの再送信要求レスポンス(Ephemeral Resend Request Response)を決して再送信しません。クライアントは、それらのレスポンスをホストへ送信した後直ちに破棄できます。

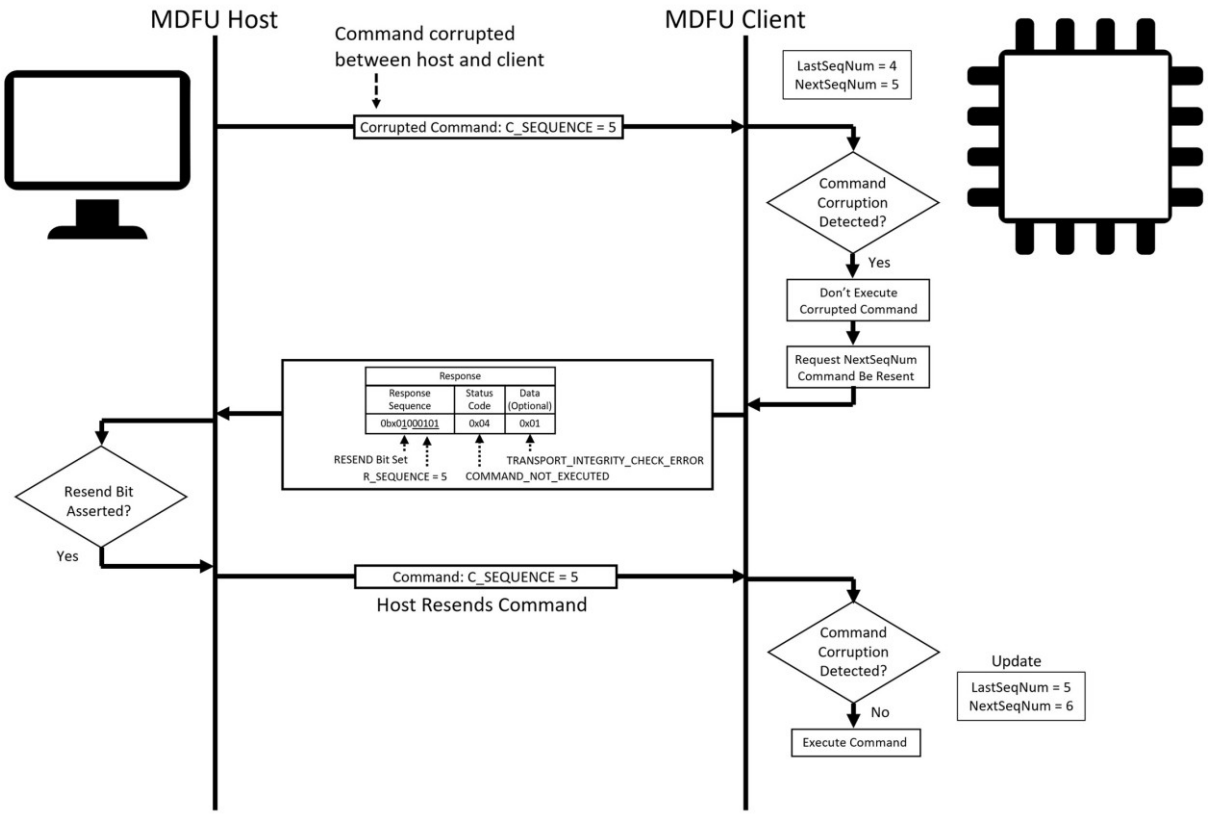
3.7.2.4 回復の詳細フロー図

以下には、ホストとクライアントのリカバリ アルゴリズムが下記の回復可能エラーからどのように回復するのかを示す詳細なシーケンス図を記載しています。

- コマンドの破損
- レスポンスの破損
- コマンドの破損に続くレスポンスの破損
- レスポンスの破損に続くコマンドの破損
- コマンドの喪失
- レスポンスの喪失

3.7.2.4.1 コマンドの破損からの回復

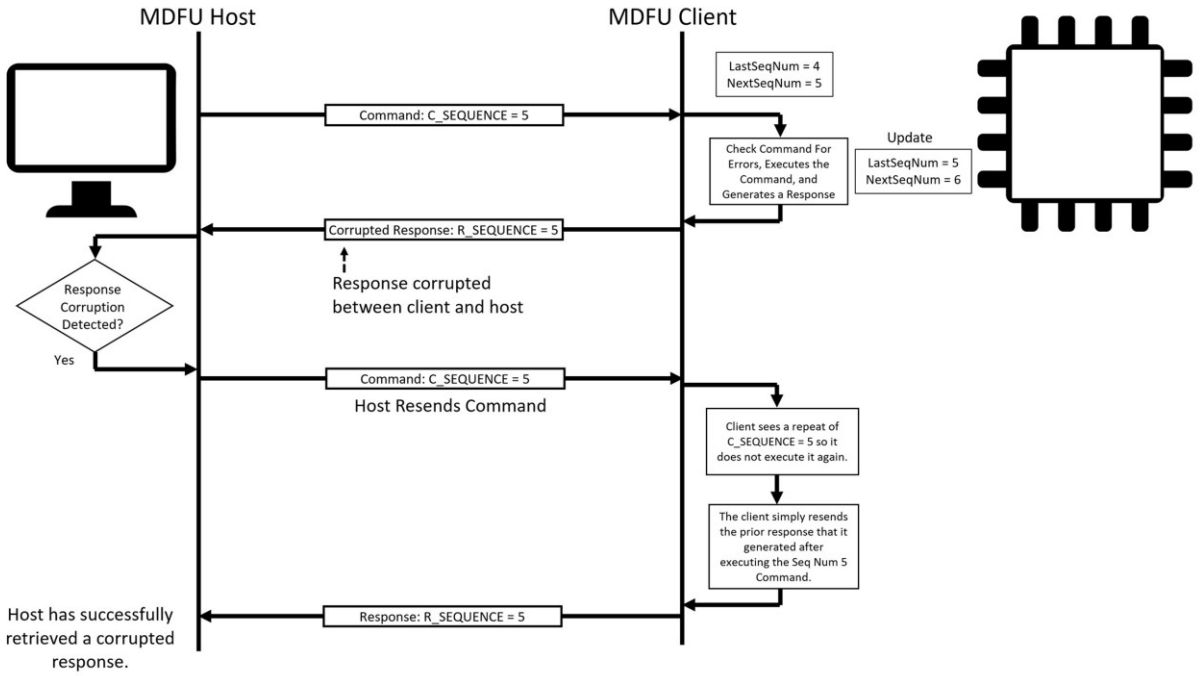
クライアントはコマンドの破損を検出した時に、ステータス COMMAND_NOT_EXECUTED を示すレスポンスを送信し、レスポンス シーケンス番号の RESEND ビットをアサートします。RESEND ビットがアサートされている事を検出したホストは、クライアントへコマンドを再送信します。



3.7.2.4.2 レスポンスの破損からの回復

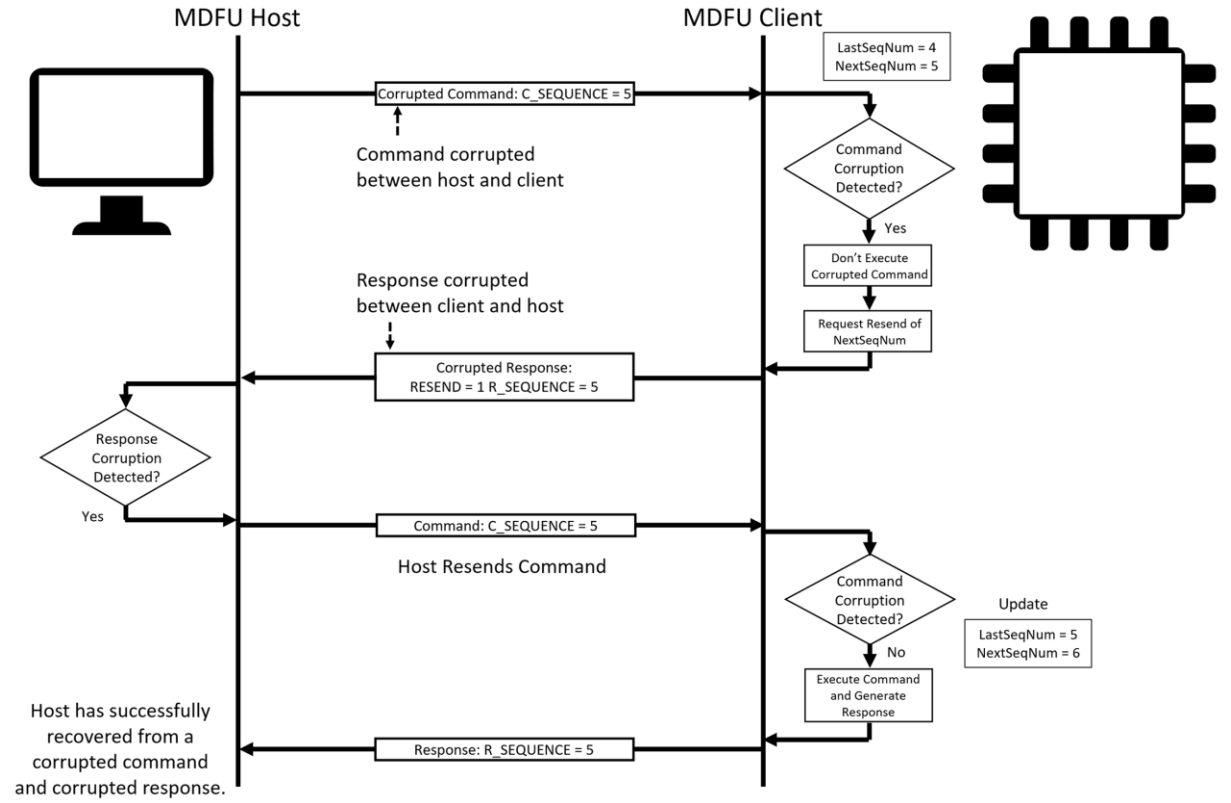
ホストは、レスポンスの破損を検出した時に直前のコマンドを再送信します。

再送信されたコマンドのシーケンス番号はクライアントが直前に実行したコマンドのシーケンス番号と同じであるため、クライアントはそのコマンドを再度実行せずに対応するレスポンスを単純に再送信します。



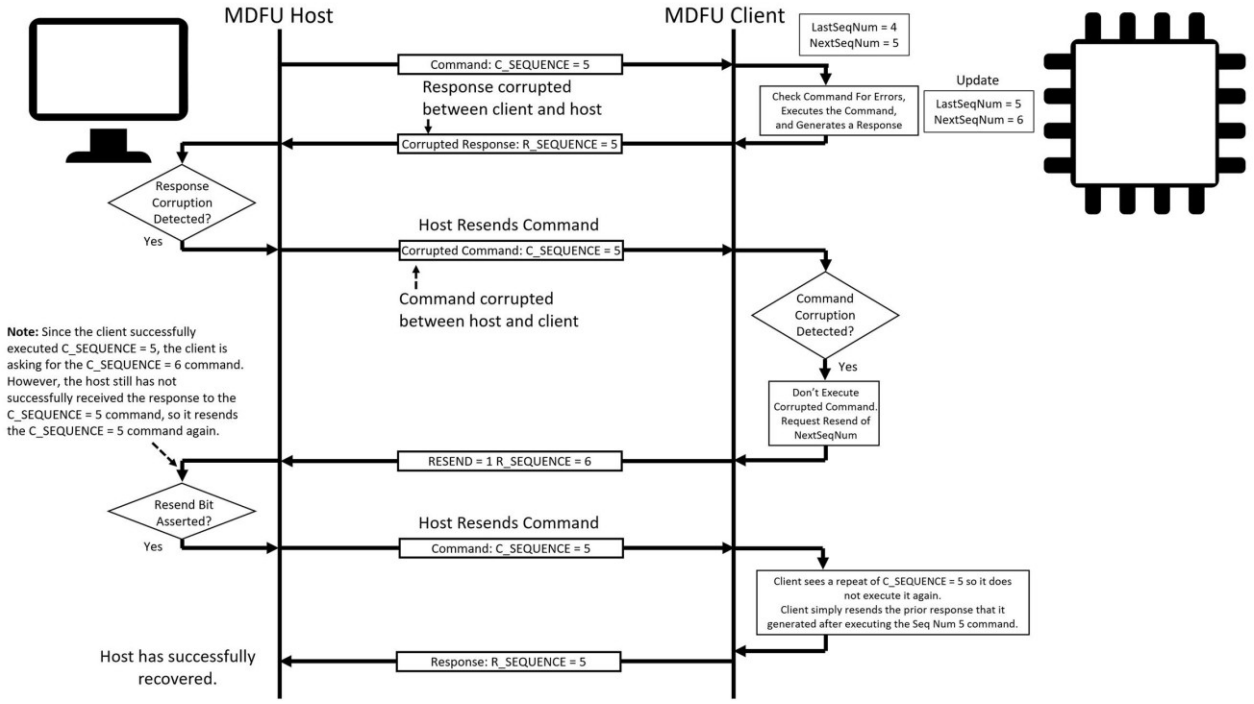
3.7.2.4.3 コマンドの破損に続くレスポンスの破損からの回復

コマンドが破損し、そのコマンドに対するレスポンスも破損した場合、ホストはコマンドを再送信します。再送信されたコマンドをクライアントが実行して対応するレスポンスをホストへ送信する事により、この状況から回復できます。



3.7.2.4.4 レスポンスの破損に続くコマンドの破損からの回復

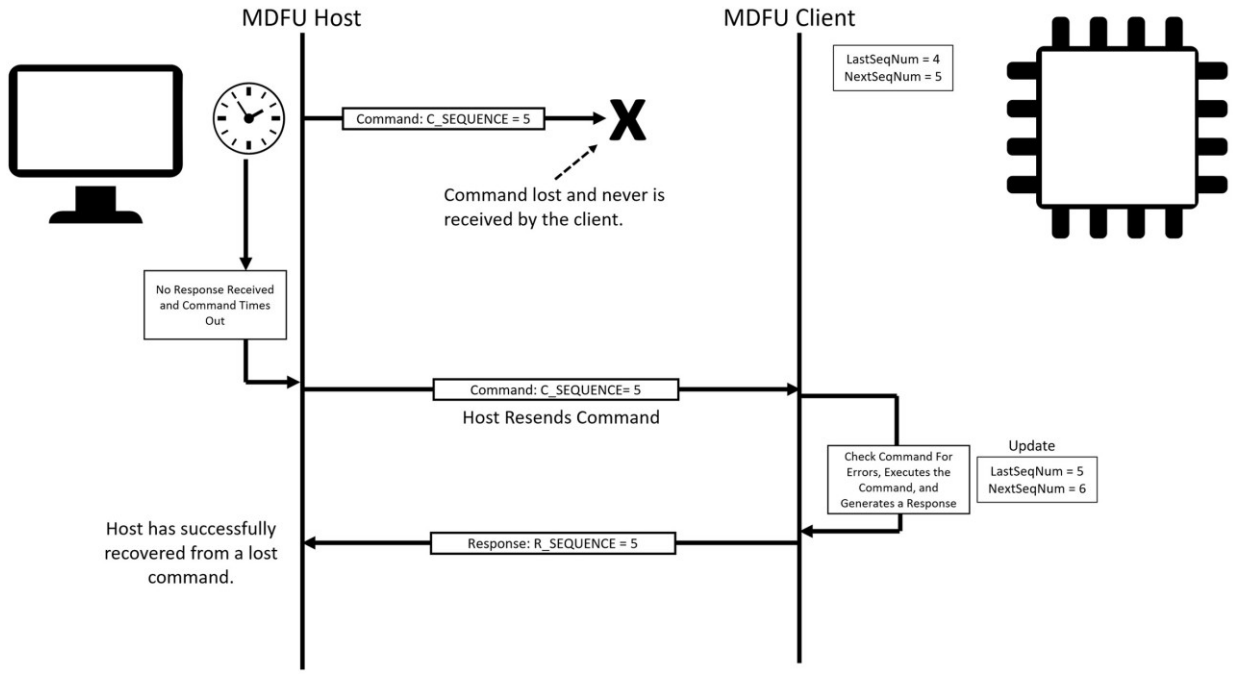
下の図は、レスポンスが破損した後にホストが再送信したコマンドも破損した場合にホストがどのように回復するのかを示しています。クライアントは再送信されたコマンドの破損を検出し、NextSeqNum を持つコマンドの再送信要求をホストへ送信します。ホストは直前のコマンドに対するレスポンスをまだ正常に受信していないため、同じコマンドをもう 1 度再送信します。クライアントは、このコマンドが既に実行済みである事を知り、そのコマンドを再実行せずに対応するするレスポンスを再送信します。以上のプロセスにより、この状況から回復できます。



3.7.2.4.5 コマンドの喪失からの回復

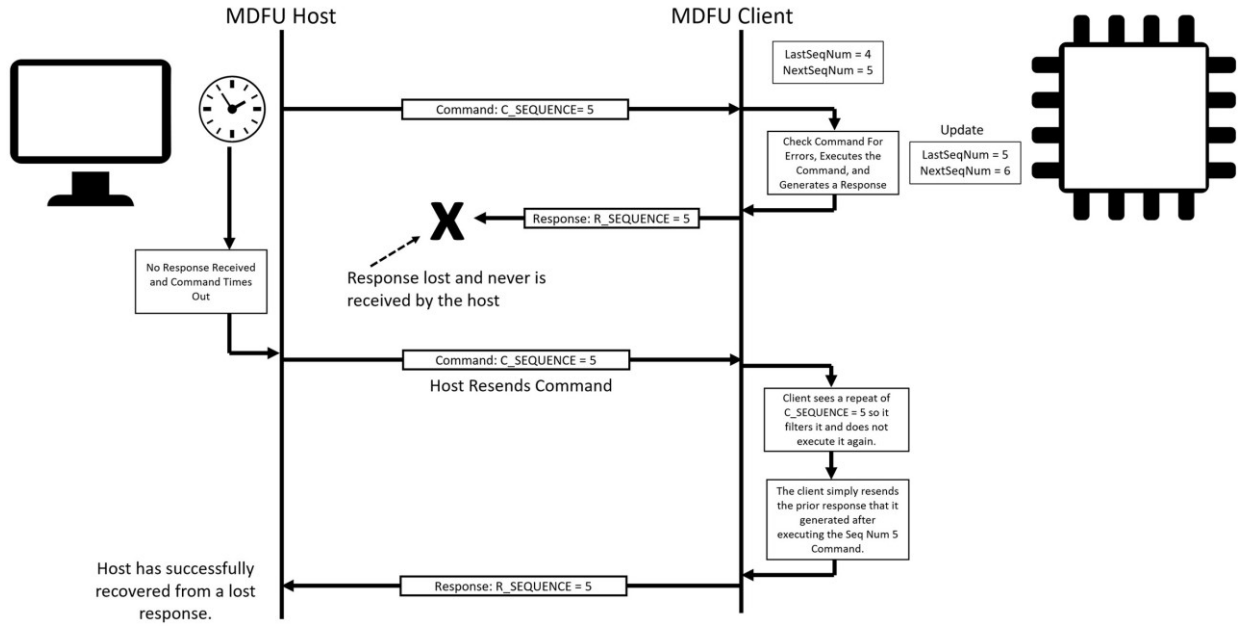
特定の通信インターフェイス上で生じる可能性のある特定形態のコマンド破損により、クライアントがコマンドを完全に受信できない場合があります。これらの状況ではコマンドは喪失し、クライアントは喪失したコマンドに対してレスポンスを生成できません。

ホストは、クライアントからのレスポンスの待機中にタイムアウトが発生した時点で、コマンドが喪失した事を検出します。この時点で、ホストはコマンドをクライアントへ再送信する事により回復できます。



3.7.2.4.6 レスポンスの喪失からの回復

ホストは、クライアントからのレスポンスの待機中にタイムアウトが発生した時点で、レスポンスが喪失した事を検出します。この時点で、ホストはコマンドをクライアントへ再送信する事により回復できます。



3.7.3 回復不可能なエラー

ファームウェアの更新中に、自動的な回復機能では回復できない各種のエラーが発生する可能性があります。MDFU プロトコルは、これらのエラーを回復不可能なエラーとして分類します。

回復不可能なエラーから自動的に回復する事はできないため、クライアントが回復不可能なエラーを検出した時点で即座に更新を停止し、その理由をクライアントから報告する機能が MDFU プロトコルによって提供されます。

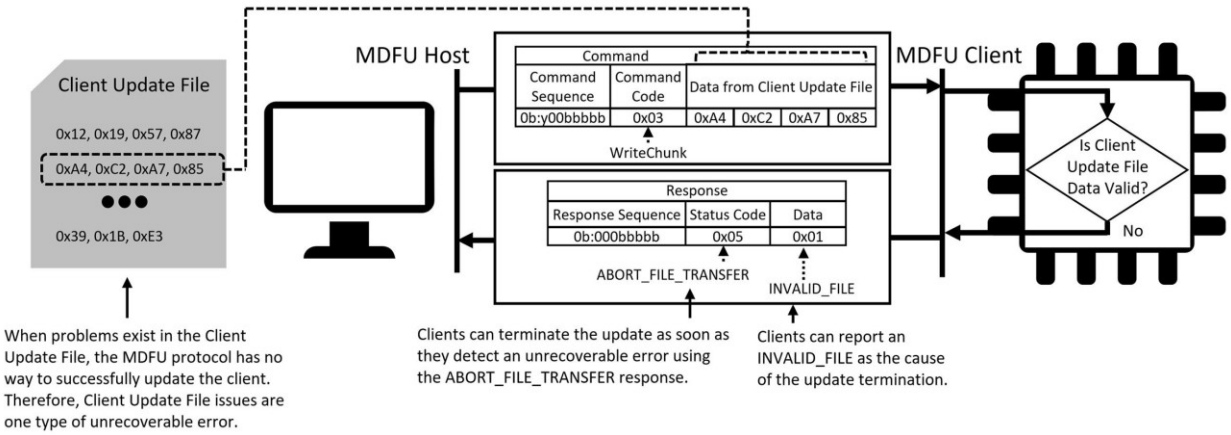
回復不可能なエラーは MDFU クライアントによって検出され、後述のファイル転送中止機能を使ってホストへ報告されます。クライアントが検出する回復不可能エラーには様々なタイプが存在します。回復不可能エラー検出機能は、必要に応じてクライアントに実装できるオプションの MDFU プロトコル機能です。クライアントに実装可能な回復不可能エラー検出機能は各種存在し、クライアントの開発者は必要に応じてそれらの機能を選択して追加できます。検出機能を追加するとクライアント実装のメモリ使用量は増加しますが、以下の利点が得られます。

1. トラブルシュートとデバッグの支援
   - ファイル転送中止プロセスによりクライアントは更新失敗の原因を報告できるため、デバッグを支援できます。
2. 時間の節約
   - 回復不可能エラーが検出された時に直ちに更新を中止するため、時間の浪費を防ぐ事ができます。この機能を実装しない場合、プロトコルはファイル転送の最後まで待機した後にはじめて更新が失敗した事を知ります。これにより、製造ラインの更新環境における時間の浪費を防ぐ事ができます。

3.7.3.1 例 1: 無効な更新ファイル

更新プロセス中に、ホストからクライアントへ更新ファイルが転送されます。更新ファイル内のデータを使ってクライアントを正常に更新するには、クライアントにとって有効なアプリケーションがクライアントにとって有効なフォーマットでファイルに格納されている必要があります。

MDFU プロトコルは、更新ファイル内のデータをクライアントへ確実に送信する事にのみ責任を持ちます。更新ファイルの内容に問題があるために更新が失敗した場合、MDFU プロトコルまたは MDFU ホストはそれらの問題を訂正する事も問題から回復する事もできません。これは回復不可能エラーの一例です。



クライアントには、更新ファイルの各種の問題をチェックする機能を選択して実装できます。クライアントは更新ファイル内のデータの問題を検出した時に、直ちにファイル転送を中止してその理由を報告する事ができます。

更新ファイル内のデータに関連する問題は各種存在します。例えば、更新ファイルが無効なメモリアドレスや無効なアプリケーション番号を含んでいる場合があります。クライアントが報告可能なファイル問題の一覧は「ファイル転送中止の原因」に記載しています。

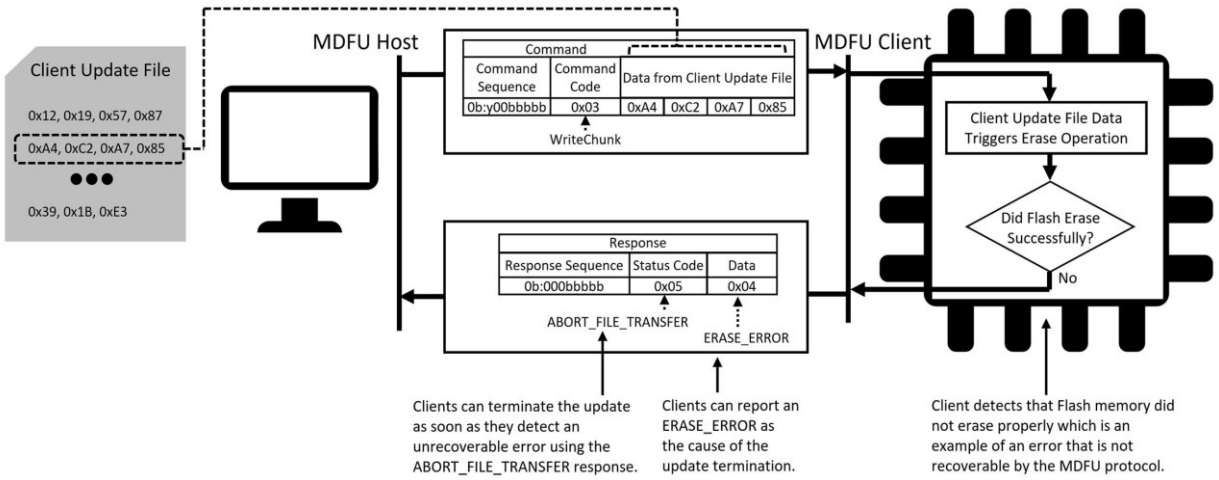
3.7.3.2 例 2: クライアントのフラッシュメモリの問題

ファームウェアの更新中にクライアント プロセッサ上の不揮発性メモリ(NVM)の消去と書き込みが発生します。各種の理由により、クライアントのフラッシュメモリが完全に消去されなかったり、正しい値が完全に書き込まれなかったりする場合があります。クライアントには、消去/書き込み動作の結果が期待される値と一致するかどうかを検証する機能をオプションとして選択できます。

クライアントがフラッシュの消去/書き込みに失敗した事を検出した場合、以下の理由により、更新プロセスを中止する事を推奨します。

1. ユーザは、何も知らされずに問題からの回復が試みられるよりも、クライアント フラッシュの問題を認識すべきです。
2. 消去/書き込み動作の再実行に自動的な再試行ループが適用されると、クライアント フラッシュに対して多数回の消去/書き込みが実行され、ユーザが気付く事なくメモリの寿命が浪費される可能性があります。

これらの理由により、MDFU プロトコルはフラッシュメモリの問題に対する自動的な回復機能を一切備えておらず、それらの問題を回復不可能なエラーと見なします。



3.7.3.3 検出

回復不可能エラーを検出するのはクライアントの責任です。

クライアントは、エンド アプリケーションの要件に応じて必要な数の回復不可能エラー検出機能を選択できます。これにより、クライアントの開発者は、クライアントのコードサイズとトラブルシュート/デバッグの容易さの間で最適なバランスを選択できます。

3.7.3.4 転送の中止

更新プロセス中のどの時点であっても、クライアントは回復不可能エラーを検出した後に、コマンドに対して ABORT_FILE_TRANSFER ステータスを格納したレスポンスを送信し、更新を中止した理由を FileAbortCause として報告できます。

3.8 シーケンス番号

MDFU プロトコルは、コマンド/レスポンス ペアを使ってホストとクライアントの間で信号とデータを転送します。シーケンス番号は、ホストとクライアントが互いに対応するコマンドとレスポンスをコマンド/レスポンス ペアとして明確に識別するための方法を提供します。シーケンス フィールドは追加の制御情報も提供します。

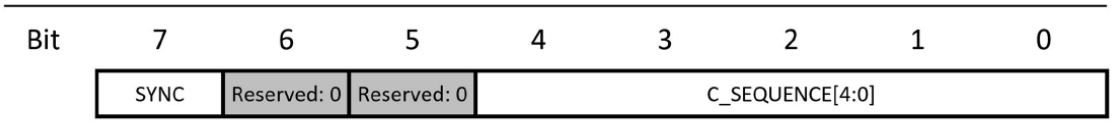
シーケンス番号により、以下が可能となります。

1. ホストは、クライアントからのレスポンスをホストから送信済みの特定コマンドと正しく関連付ける事ができます。
2. クライアントは、全てのコマンドを正しい順序で 1 度だけ確実に実行できます。
3. クライアントは、破損した特定のコマンドを再送信するようホストに要求できます。
4. ホストは、破損したレスポンスをコマンドの実行を繰り返す事なく再送信するようクライアントに要求できます。

後述の「ホストによるコマンド シーケンス番号の選択」と「クライアントによるシーケンス番号の処理」を参照してください。

3.8.1 コマンド シーケンス フィールドの定義

コマンド シーケンス フィールド:



Bit 7-SYNC

ホストはSYNCビットを「1」に設定する事で、クライアントのシーケンス番号をホストのシーケンス番号に同期させます。ホストは、更新プロセスの最初のコマンドでSYNC = 1に設定し、後続の全てのコマンドでSYNC = 0に設定する必要があります。

値	概要
0x0	クライアントはC_SEQUENCE番号に基づいて、このコマンドを実行すべきか、それとも実行せずに直前のレスポンスを再送信すべきか判断します。また、C_SEQUENCEが無効であれば、このコマンドを拒絶します。
0x1	クライアントは、自身のシーケンス番号をC_SEQUENCEビットフィールド内の番号に同期させます。クライアントは、このコマンドの実行を受け付けます。

Bit 6 = 予約済み

将来用に予約済みです。ホストはこのビットを0x0に設定する必要があります。

Bit 5 = 予約済み

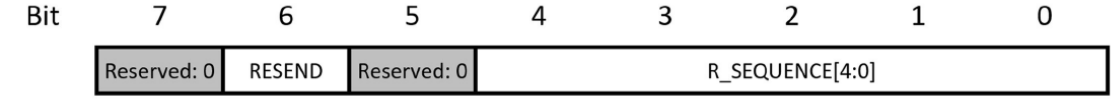
将来用に予約済みです。ホストはこのビットを0x0に設定する必要があります。

Bits 4:0 - C_SEQUENCE

C_SEQUENCE ビットフィールドはコマンドのシーケンス番号を格納します。

3.8.2 レスポンス シーケンス フィールドの定義

レスポンス シーケンス フィールド:



Bit 7 = 予約済み

将来用に予約済みです。クライアントはこのビットを0x0に設定する必要があります。

Bit 6- RESEND

クライアントはこのビットを使って、特定のコマンドを再送信するようホストに要求するか、レスポンス シーケンス番号R_SEQUENCE[4:0]に等しいC_SEQUENCE[4:0]を持つコマンドに対するレスポンスを提供します。

値	概要
0x0	クライアントは、このレスポンスのR_SEQUENCE[4:0]に等しいC_SEQUENCE[4:0]を持つコマンドに応答します。
0x1	クライアントは、このレスポンスのR_SEQUENCE[4:0]に等しいC_SEQUENCE[4:0]を持つコマンドを再送信するようホストに要求します。

Bit 5 = 予約済み

将来用に予約済みです。クライアントはこのビットを0x0に設定する必要があります。

Bits 4:0 - R_SEQUENCE

R_SEQUENCE ビットフィールドはレスポンスのシーケンス番号を格納します。

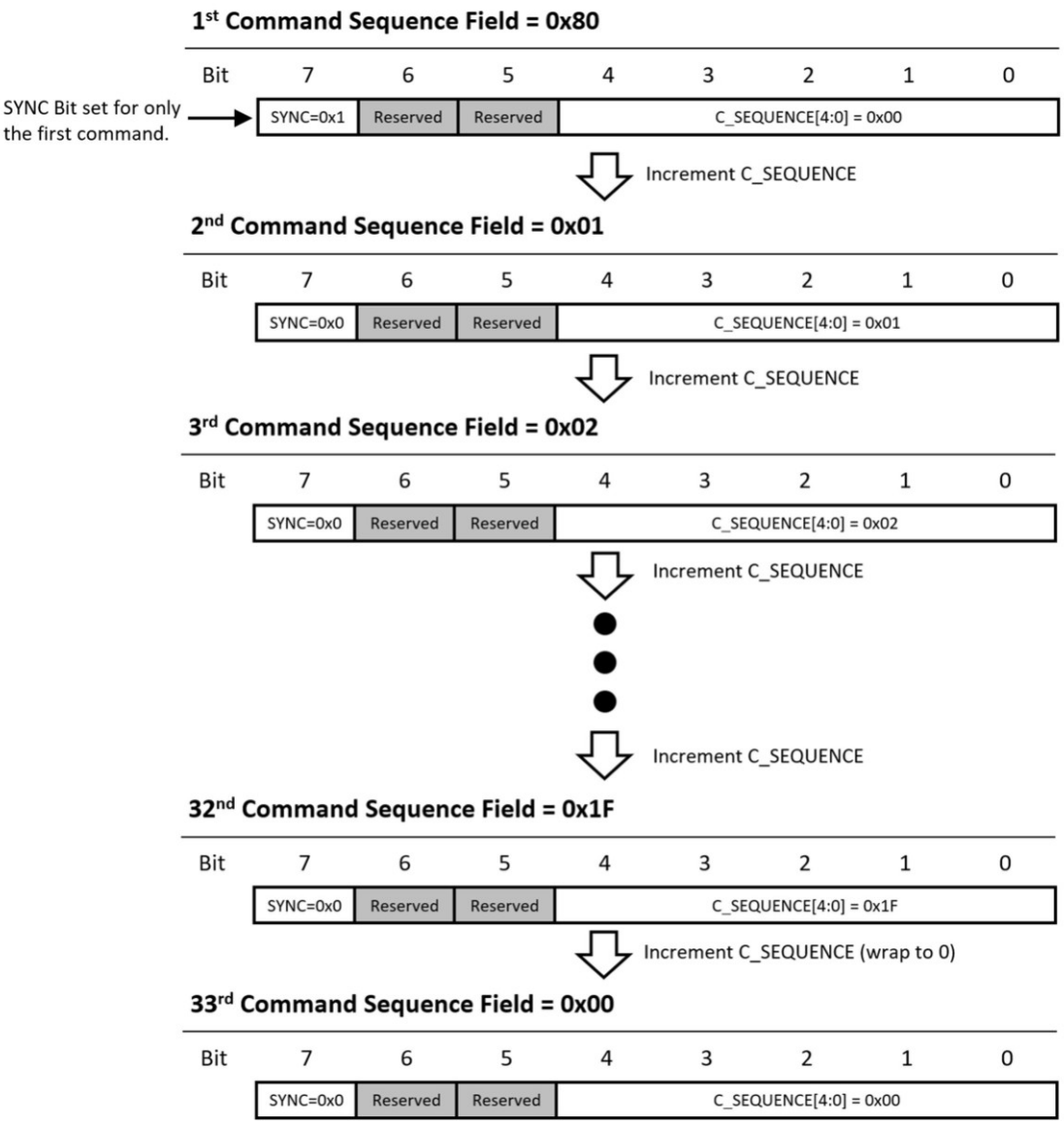
3.8.3 ホストによるコマンド シーケンス番号の選択

ホストは、以下の手順でコマンドのシーケンス番号を選択します。

1. 最初のコマンド:
   - シーケンス番号フィールドは、受信したコマンドを正しい順序で 1 度だけ確実に実行できるようにフィルタ処理するための仕組みをクライアントに提供します。
   - 更新プロセスの開始時に、ホストはクライアントが期待しているシーケンス番号を知りません。この状況を解消するため、本プロトコルは、クライアントのシーケンス番号をホストのシーケンス番号に同期させるための機能を提供します。
   - 更新のためにホストからクライアントへ最初に送信されるコマンドに対してのみホストはシーケンス フィールドの SYNC ビットを「1」に設定し、シーケンス番号を 0 に設定します。
   - クライアントは、「1」に設定された SYNC ビットを検出した時に、クライアント側の現在のシーケンス番号をホストからのコマンド内のシーケンス番号(= 0)に書き換えます。
2. 後続の全てのコマンド:
   - 最初のコマンドによってクライアントのシーケンス番号がホストと同期した後は、後続の全てのコマンドのシーケンス番号は 1 ずつインクリメントし、SYNC ビットは「0」に設定される必要があります。
   - シーケンス番号は 5 ビット値であり、5 ビットがオーバーフローすると値は 0 へロールオーバーします。

3.8.3.1 コマンド シーケンス番号の例

次ページの図に、ホストがコマンド用に選択するシーケンス番号の例を示します。



3.8.4 クライアントによるシーケンス番号の処理

クライアントは、各コマンドが正しい順序で 1 度だけ実行される事を確実にするためのフィルタ処理ロジックを実装します。

クライアントは、最後に実行されたコマンドのシーケンス番号(LastSeqNum)と、次のコマンドに期待するシーケンス番号(NextSeqNum)を追跡します。

クライアントがコマンド シーケンス番号の処理/フィルタリングを実行する前に、そのコマンドはトランスポート層の整合性チェックに合格済みである事が必要です。シーケンス番号の処理/フィルタリングが発生するタイミングについては、「クライアントによるコマンド処理/レスポンス生成」を参照してください。

クライアントは以下のチェック/ロジックを使って各種タスクを実行します。

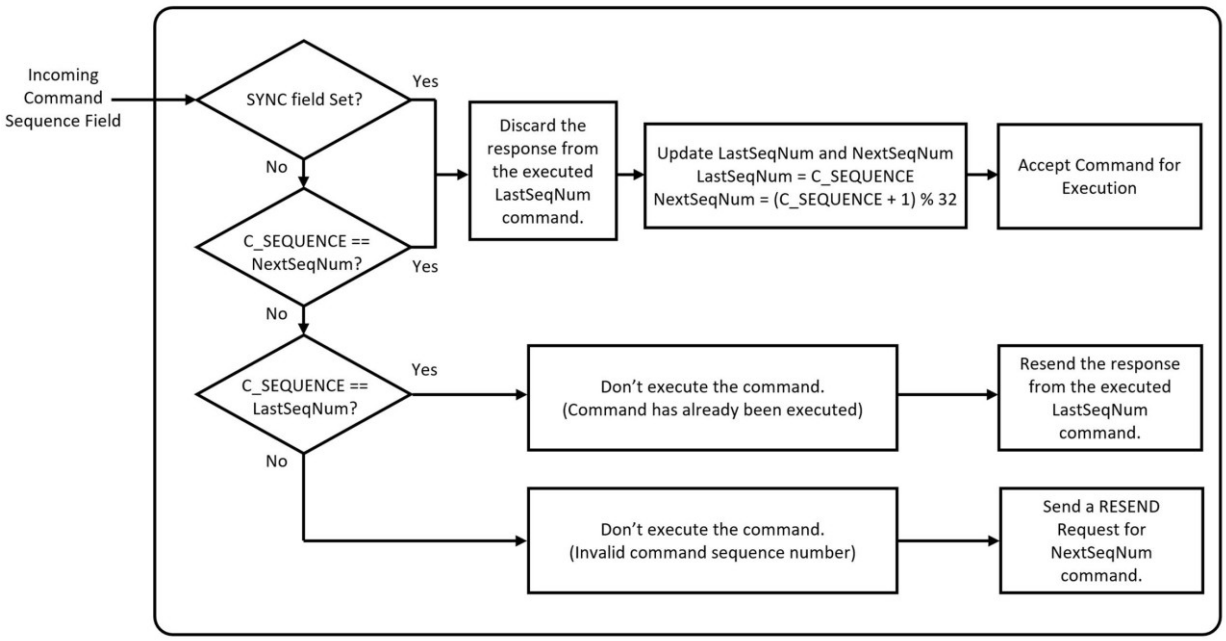
- クライアント シーケンス番号の同期:
  - 更新の開始時に、ホストはクライアントの LastSeqNum と NextSeqNum の値を知りません。クライアントは、C_SEQUENCE が NextSeqNum と一致しないコマンドを拒絶するため、ホストは同期機能を使ってクライアントのシーケンス番号を設定します。

  - 更新プロセスにおいてホストがクライアントへ送信する最初のコマンドのみ SYNC ビットが「1」(クライアントのシーケンス番号をホストのシーケンス番号に同期させるようクライアントに要求する)に設定されます。
  - クライアントは、このコマンドの実行を受け付けます。
- 有効なシーケンス番号を持つコマンドの処理:
  - ホストからのコマンドの C_SEQUENCE が NextSeqNum と等しい場合、クライアントはこのコマンドの実行を受け付けます。
- 破損したレスポンスの回復:
  - レスポンスが破損していた場合、ホストはクライアントへコマンドを再送信します。クライアントがこのコマンドを既に実行済み(C_SEQUENCE = LastSeqNum)であった場合、クライアントはそのコマンドを再度実行せずに、そのコマンドを最初に実行した時のレスポンスを単純に再送信する必要があります。
  - **重要**: クライアントは、コマンドの実行後にホストがそのコマンドに対するレスポンスの再送信をもはや要求する必要はないと結論付けるまで、そのレスポンスを保持する必要があります。クライアントは、C_SEQUENCE が次に期待するシーケンス番号(NextSeqNum)と一致する新しいコマンドを受信した時、またはホストがシーケンス番号を同期させた時にのみ、LastSeqNum コマンドに対するレスポンスを破棄できます。
- 無効なシーケンス番号の通知:
  - SYNC ビット = 「0」かつシーケンス番号が NextSeqNum と LastSeqNum のどちらとも一致しないコマンドをホストが送信した場合、そのシーケンス番号は無効であり、クライアントはそのコマンドを実行してはいけません。
  - クライアントはホストへの応答として、次に期待するシーケンス番号(NextSeqNum)を持つコマンドの再送信を要求する必要があります。このために、クライアントは下記の一時的再送信要求(Ephemeral Resend Request)レスポンスを生成する必要があります。
    - レスポンス シーケンス
      - RESEND = 1
      - R_SEQUENCE = NextSeqNum
    - レスポンス ステータス = COMMAND_NOT_EXECUTED
    - レスポンスデータには何も格納しないか、原因コードとして SEQUENCE_NUMBER_INVALID を格納できます。
  - この一時的再送信要求レスポンスはコマンド実行の結果として生成されるのではなく、クライアントは送信後にこのレスポンスを保持する必要はありません(クライアントがこのレスポンスを再送信する事は決してないため)。

3.8.4.1 処理アルゴリズム図

次ページの図 3-11 に、クライアントによるシーケンス番号処理/フィルタリング アルゴリズムの定義を示します。

図 3-11. クライアントによるシーケンス番号の処理/フィルタリング

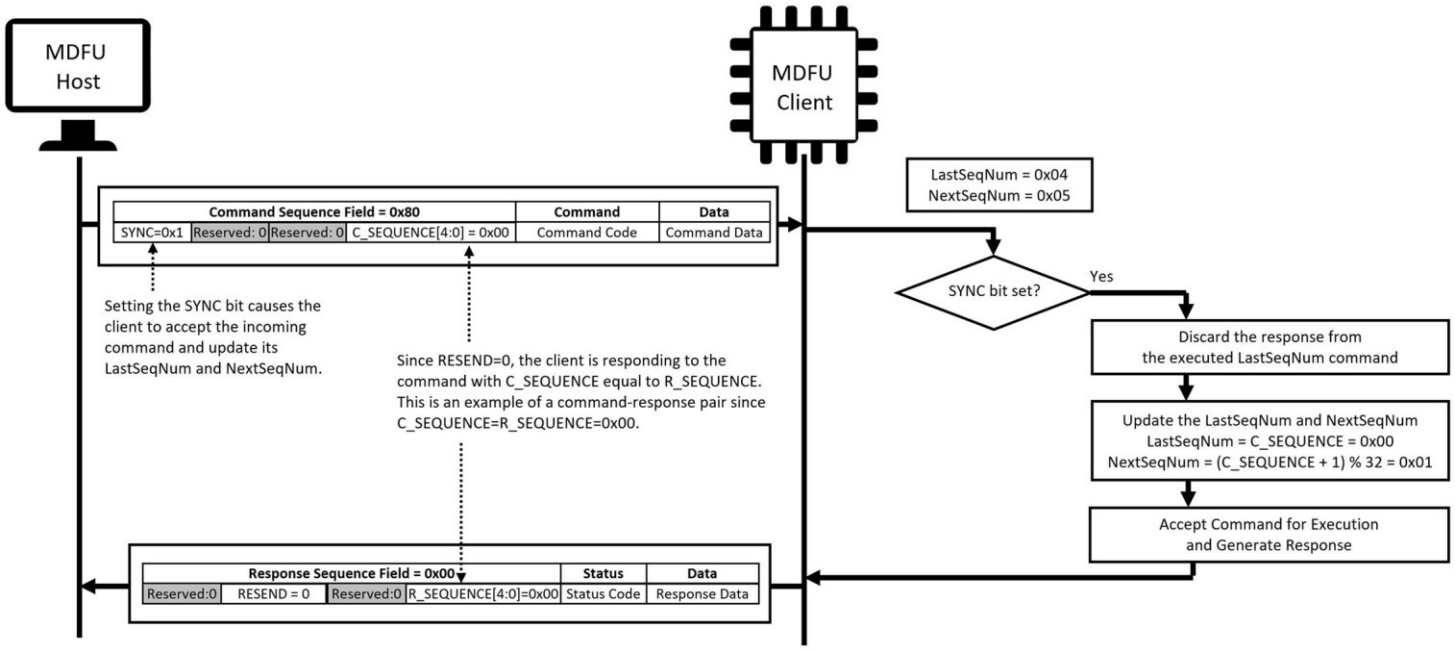


3.8.4.2 クライアントによるシーケンス番号処理の例

以下に記載する例は、前述の 4 つの状況におけるコマンドとレスポンスのシーケンス番号フィールドの詳細を示します。

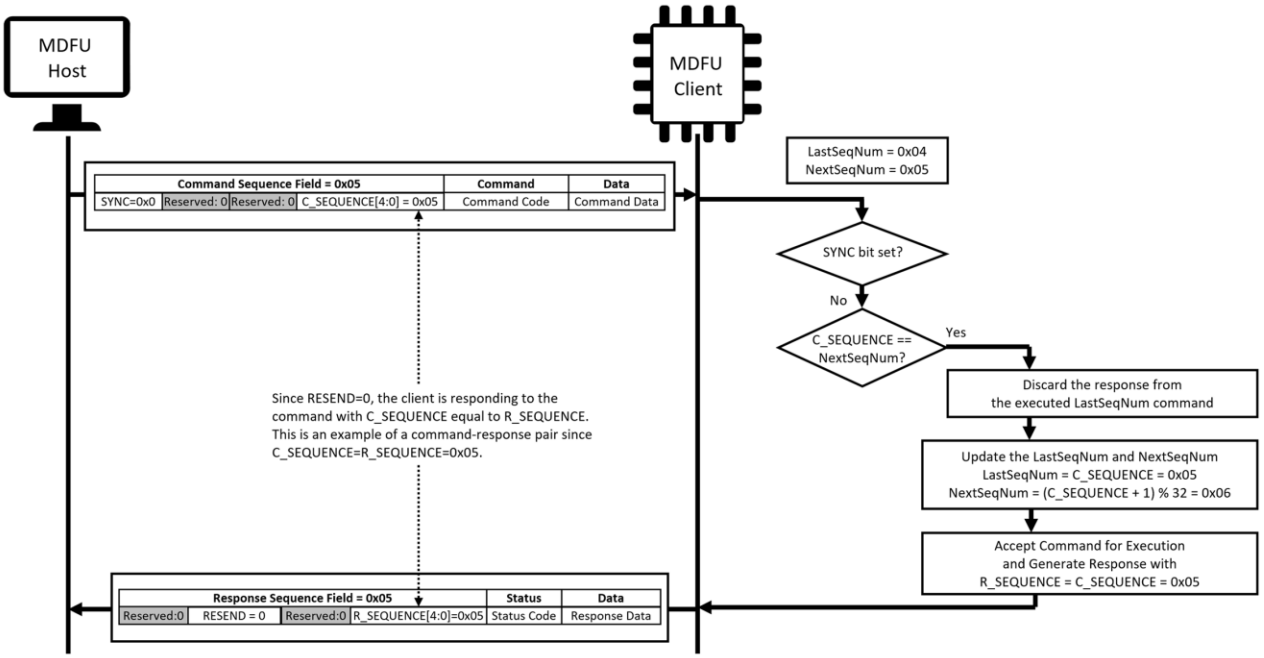
3.8.4.2.1 シーケンス番号同期

図 3-12. クライアントは シーケンス番号をホストに同期させる



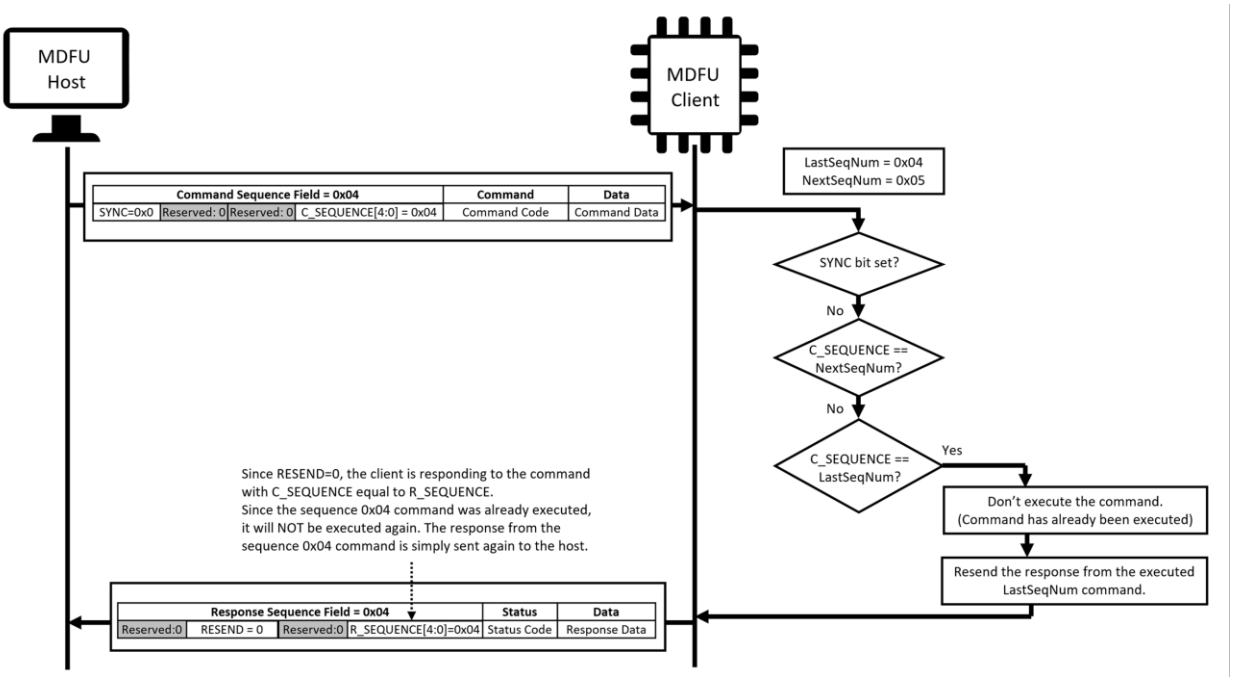
3.8.4.2.2 有効なシーケンス番号を持つコマンドの処理

図 3-13. クライアントは NextSeqNum コマンドを受信する



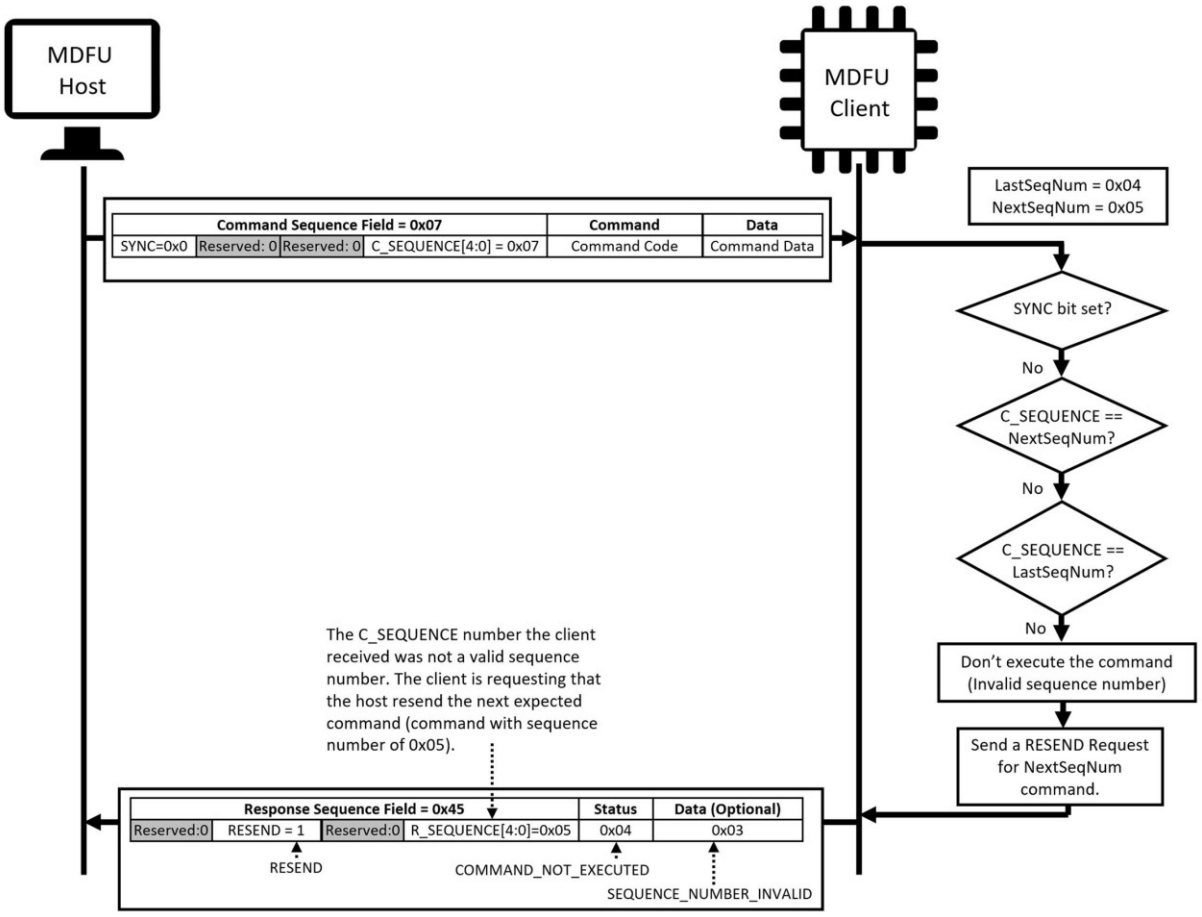
3.8.4.2.3 破損したレスポンスの回復

図 3-14. クライアントは LastSeqNum コマンドに対するレスポンスを再送信する



3.8.4.2.4 無効なシーケンス番号と再送信要求

図 3-15. クライアントは NextSeqNum コマンドを再送信するようホストに要求する



4. トランスポート層

MDFU プロトコルは 2 つの階層に分割されます(「MDFU プロトコルの階層」参照)。ファームウェア更新プロトコル層は、全ての物理的通信インターフェイスで共通するプロトコルの詳細を定義します。トランスポート層は、特定の物理的通信バスに固有のプロトコルの詳細を定義します。

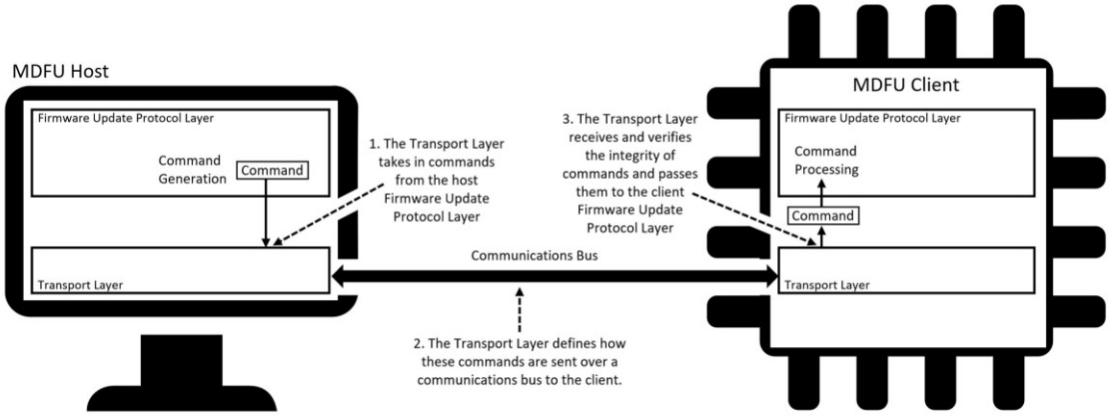
以下に、各種通信バス向けのトランスポート層定義を記載します。これらのトランスポート層定義は、各種通信バスを介してコマンドとレスポンスを送受信する方法を規定します。

4.1 トランスポート層の役目

各通信バス向けのトランスポート層定義の詳細を以下に記載します。

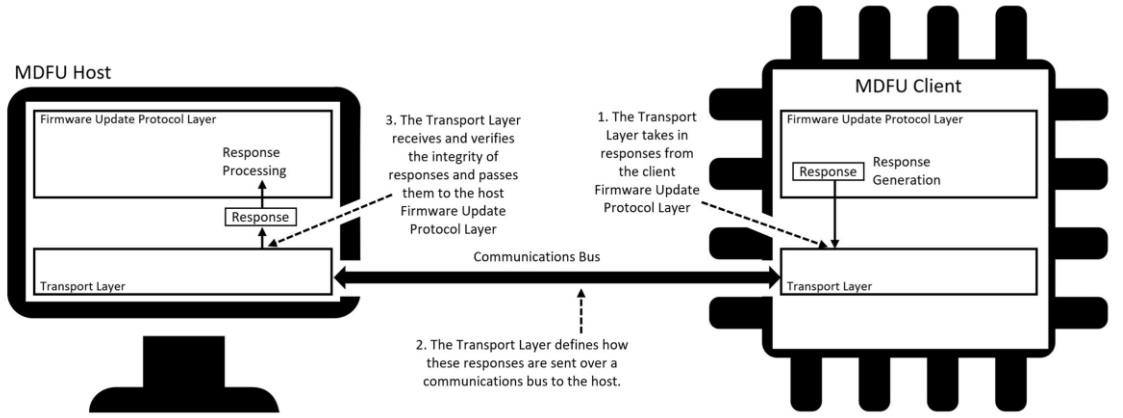
4.1.1 ホストからクライアントへのコマンド転送

1. コマンドはファームウェア更新プロトコル層によって生成され、トランスポート層へ渡されます。
2. コマンドは通信バスを介してホストからクライアントへ送信されます。
3. クライアントが受信したコマンドは、クライアントのトランスポート層によって整合性が検証された後に、ファームウェア更新プロトコル層へ渡されます。



4.1.2 クライアントからのレスポンスの取得

1. クライアントのファームウェア更新プロトコル層はレスポンスを生成し、それらをトランスポート層へ渡します。
2. レスポンスは通信バスを介してクライアントからホストへ送信されます。
3. ホストが受信したレスポンスは、ホストのトランスポート層によって整合性が検証された後に、ホストのファームウェア更新プロトコル層へ渡されます。



4.1.3 コマンド/レスポンスの整合性チェック

- MDFU プロトコルでは、コマンド/レスポンスの破損(例: ノイズによるビット反転等)はトランスポート層によって検出されます。
- 破損が検出された場合、エラー回復機能を使ってこれらのエラーから回復できるよう、ファームウェア更新プロトコル層にエラーが通知されます。

4.1.4 ホスト トランスポート層のレスポンス取得タイムアウト

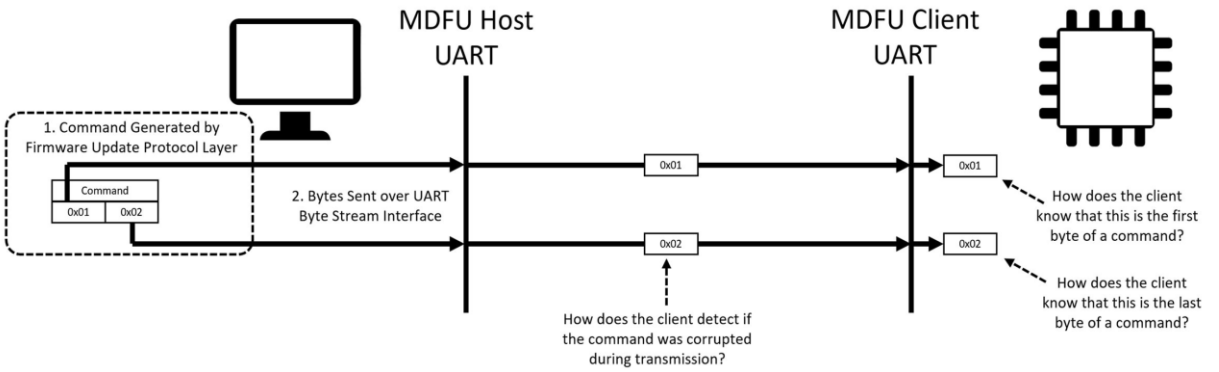
ホストは、タイムアウトを使ってコマンドまたはレスポンスの喪失から回復します。タイムアウトはホストのファームウェア更新層またはトランスポート層のどちらかに実装できます(選択可能)。詳細は「ホストによるコマンド生成/レスポンス処理」を参照してください。

4.2 UART コマンド/レスポンス トランスポート層

MDFU UART トランスポート層は、UART バイトストリーム インターフェイスを介してコマンドとレスポンスを確実に転送する責任を負います。

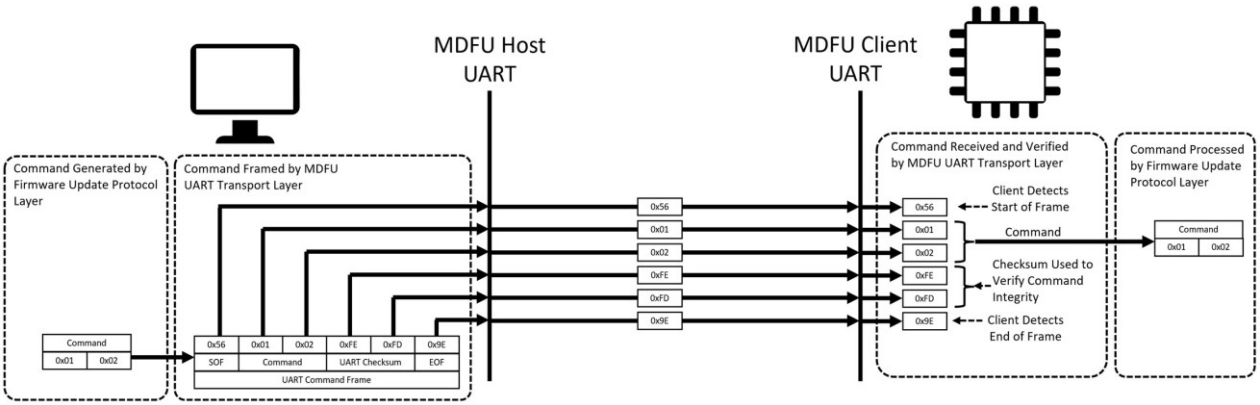
UART バイトストリーム インターフェイスを介してコマンドを送信する場合、クライアントがコマンドを受信する際に以下が要求されます。これらはホストがレスポンスを受信する際にも同様に要求されます。

- バイトストリーム内でコマンドの最初のバイトの位置を検出できる事
- バイトストリーム内でコマンドの最後のバイトの位置を検出できる事
- 転送中にコマンドが破損した事を検出できる事



以下では、MDFU UART トランスポート層においてコマンド/レスポンスの開始/終了の検出とコマンド/レスポンスの破損の検出を可能にする UART コマンド/レスポンスフレーミング機能について説明します。

図 4-1. MDFU UART コマンド フレーミングの例



4.2.1 UART コンフィグレーション

MDFU UART トランスポート層は、以下の通りパリティ フィールドを使わずに 8 ビットデータ転送の汎用バイトストリームを介して動作するように設計されています。

UAR コンフィグレーションの最小要件:

- 双方向
- 全二重
- 非同期
- データレート: 任意
- データ幅: 8 ビット
- パリティ: なし
- ストップビット: 1 以上

4.2.2 ホストによるレスポンスの取得

クライアントは受信したコマンドを実行してレスポンスを生成します。UART コンフィグレーションは非同期/双方向/全二重リンクであるため、クライアントはレスポンスの準備が整うと直ちにレスポンスをホストへ返送できます。

クライアントへコマンドを送信したホストは、クライアントからレスポンスが返送されるのを単純に待機します。

ホストは、タイムアウトを使ってコマンドまたはレスポンスの喪失から回復します。タイムアウトはホストのファームウェア更新層またはトランスポート層のどちらかに実装できます(選択可能)。詳細は「ホストによるコマンド生成/レスポンス処理」を参照してください。

4.2.3 UART コマンド/レスポンスのフレーミング

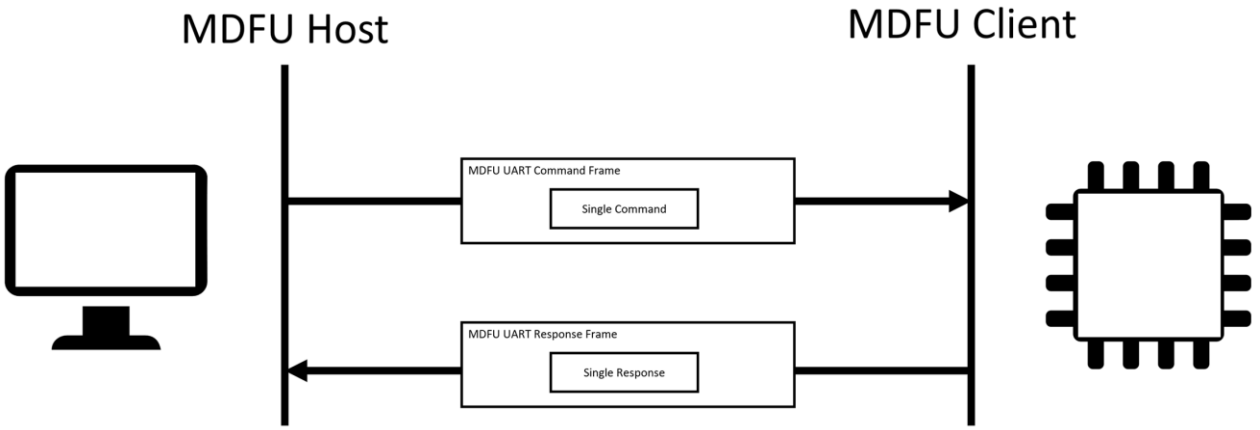
以下では、下記の目標を達成する MDFU UART フレーミング機能を定義します。

1. UART バイトストリーム内でコマンドまたはレスポンスの開始を識別する
2. UART バイトストリーム内でコマンドまたはレスポンスの終了を識別する
3. コマンドとレスポンスの破損を検出するための機能を提供する

4.2.3.1 UART コマンド/レスポンス フレームの内容

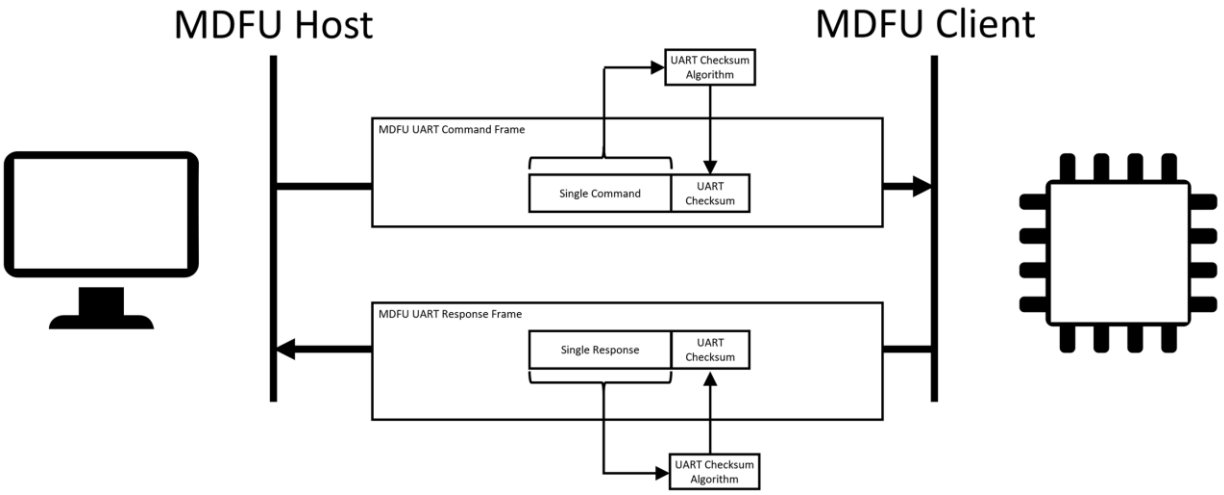
MDFU UART コマンド/レスポンス フレームは、コマンドをホストからクライアントへ転送し、レスポンスをクライアントからホストへ転送するために使われます。MDFU UART コマンド/レスポンス フレームは常に 1 つの完全なコマンドまたは 1 つの完全なレスポンスをフレーム全体に格納します。コマンドまた

はレスポンスが分割されて複数の MDFU UART コマンド/レスポンス フレームに格納される事は決してありません。1 つの MDFU UART コマンド/レスポンス フレーム内に複数のコマンドまたは複数のレスポンスが格納される事も決してありません。



4.2.3.2 コマンド/レスポンス フレームの破損を検出するための UART チェックサム

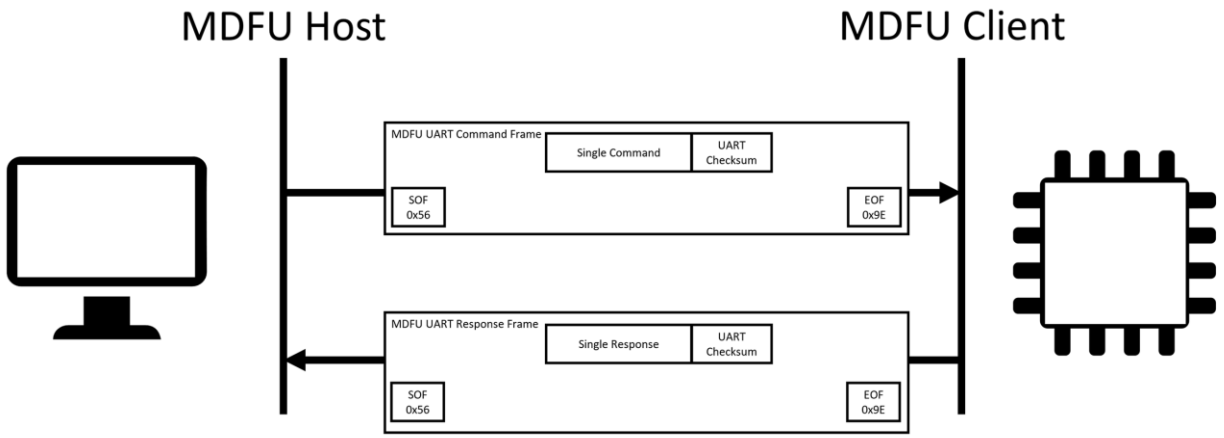
MDFU UART トランスポート層は、UART 物理層内に独自のエラー検出機能等を持たない UART コンフィグレーションを想定して設計されています。このため MDFU UART トランスポート層は、コマンド/レスポンスの破損を検出可能とするために、コマンド/レスポンスの末尾に 16 ビット チェックサムを付加します。



4.2.3.3 フレームの開始とフレームの終了

MDFU UART トランスポート層は、UART コマンド/レスポンス フレームの開始を示すためにコード 0x56 を予約済みです。UART バイトストリーム内で 0x56 が検出された場合、それは常にフレームの開始を意味します。コード 0x56 がバイトストリーム内のフレーム開始以外の位置に存在してはいけません。

MDFU UART トランスポート層は、UART コマンド/レスポンス フレームの終了を示すためにコード 0x9E を予約済みです。UART バイトストリーム内で 0x9E が検出された場合、それは常にフレームの終了を意味します。コード 0x9E がバイトストリーム内のフレーム終了以外の位置に存在してはいけません。



4.2.3.4 予約済みコードの置換

コマンドやレスポンスを構成するフレーム中のデータ内やチェックサム内に 0x56 や 0x9E のコードが出現する場合は、次のように予約済みのバイトコードを使った代替表現に置換する必要があります。

UART トランスポート層のフレーミング機能はコマンド、レスポンス、チェックサム内に現れる予約済みコードを 2 バイトシーケンスに置換して表現する方法を提供します。

- 2 バイトシーケンスの第 1 バイトはエスケープコード(0xCC)を格納します。
- 2 バイトシーケンスの第 2 バイトは表現されるコードの 1 の補数を格納します。

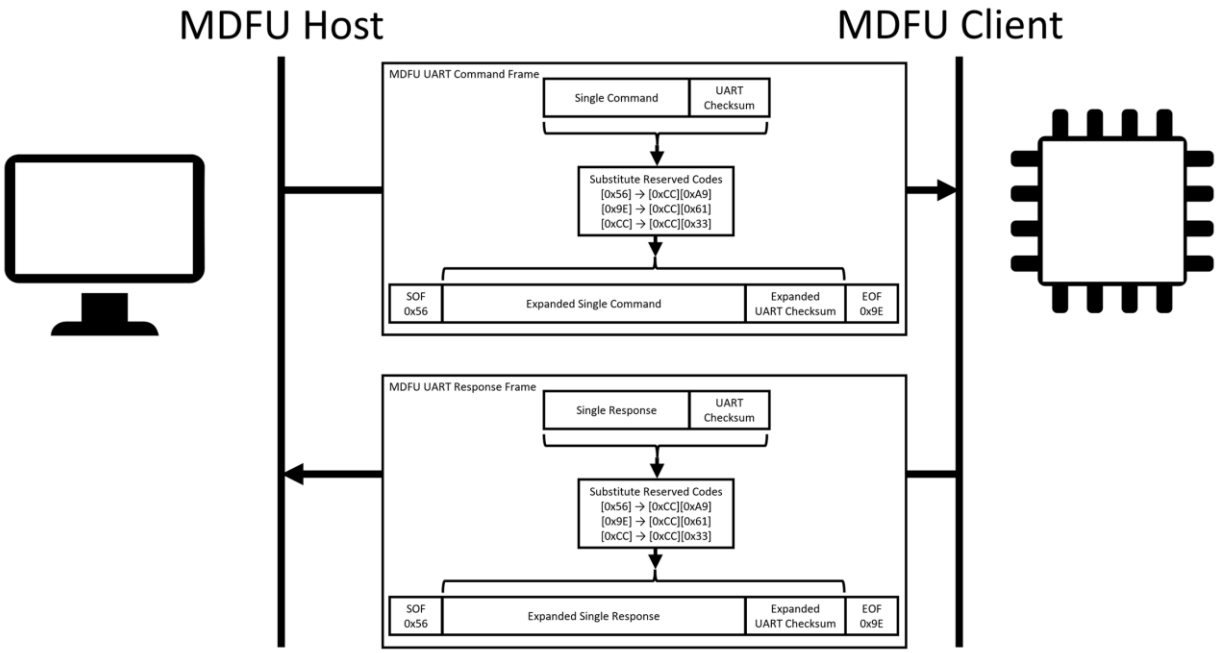
下の表に、予約済みコードとそれらに対応する 2 バイト代替コードを示します。

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

次ページの図にコマンド、レスポンス、チェックサムが予約済みコードの置換処理を経て MDFU UART コマンド/レスポンス フレーム内に格納される様子を示します。

コマンド/レスポンス フレームは最終的にこの形態で UART バイトストリームを介して送信されます。

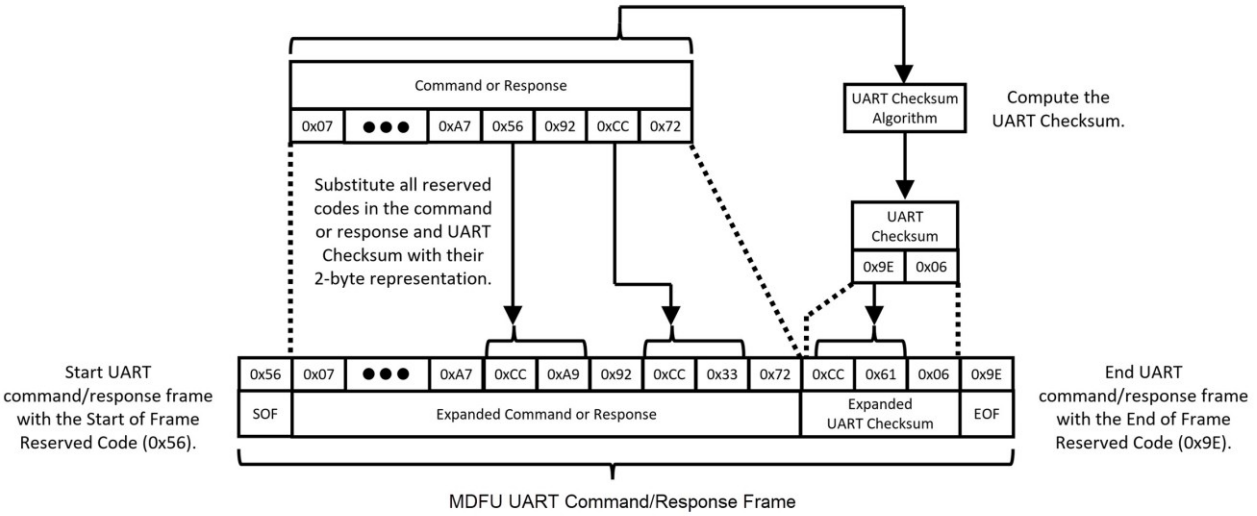


4.2.4 UART コマンド/レスポンス フレームの生成

UART コマンド/レスポンス フレームの生成には以下の手順が必要です。

1. 各コマンド/レスポンス フレームをフレーム開始コード(0x56)で開始する
2. UART チェックサムを計算する
3. コマンド/レスポンスおよびチェックサム内の全ての予約済みコードを 2 バイト代替コードに置換する
4. 各コマンド/レスポンス フレームをフレーム終了コード(0x9E)で終了する

図 4-2. MDFU UART コマンド/レスポンス フレーム



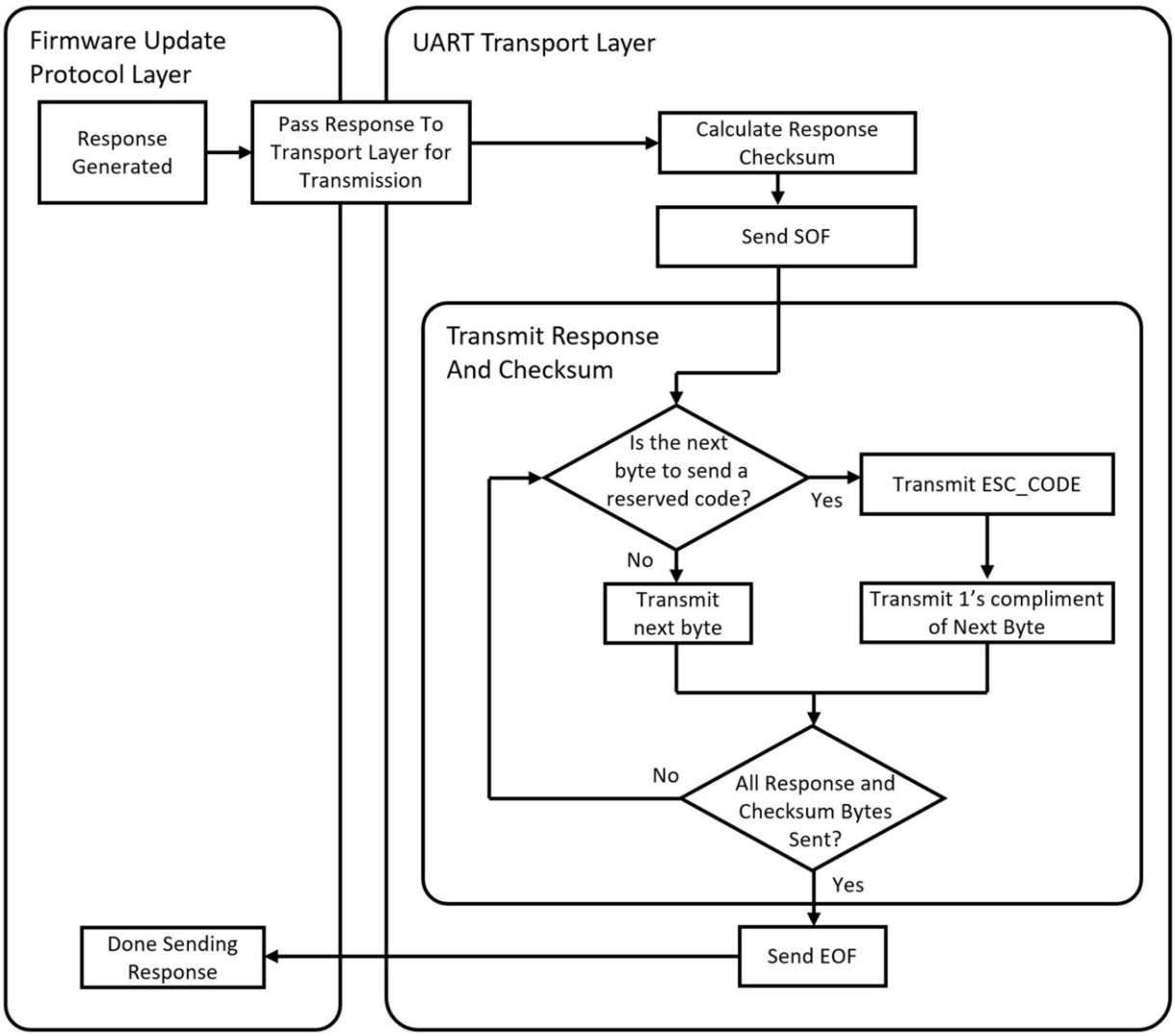
4.2.5 UART コマンド/レスポンス フレームの送信

MDFU ホストまたはクライアントを設計する際に以下のトレードオフが生じます。

一部のアプリケーションでは、UART コマンド/レスポンス フレームの RAM バッファサイズの最小化が優先される場合があります。そのような場合、完全なコマンド/レスポンス フレームを拡張形態(予約済みコ

ードを 2 バイトコードに置換した状態)で保存せずに、オンザフライで MDFU UART コマンド/レスポンス フレームを構成する事が可能です。図 4-3 に、レスポンスフレームを拡張形態で保存する事なくオンザフライで構成する場合のアルゴリズム例を示します。

図 4-3. クライアントがオンザフライで MDFU UART レスポンス フレームを構成する場合の例



他方、コマンドおよびレスポンスを予め UART コマンド/レスポンス フレームバッファ内に格納した後に送信を開始する事が望ましい場合もあります。拡張された完全な UART コマンド/レスポンス フレームを保存する場合のメモリ要件については、「UART コマンド/レスポンス フレームの最大サイズ」を参照してください。

4.2.6 UART コマンド/レスポンス フレームの受信

UART コマンド/レスポンス フレームの受信には以下の手順が必要です。

1. UART コマンド/レスポンス フレームの開始を検出する
2. UART コマンド/レスポンス フレームの終了を検出する
3. コマンド/レスポンス フレームのチェックサムを検証する

4. 特定タイプのコマンド/レスポンス フレーム受信エラー(転送エラー)をファームウェア更新プロトコル層に通知する

加えて、下記を目標として受信バイト処理アルゴリズムが定義されます。

1. 一部の形態のデータ破損が生じた後にコマンド/レスポンス フレームの受信を UART バイトストリームに再同期する事を可能にする
2. 無効なコマンドまたは破損したコマンドに対するエラーレスポンスの数を制限する

4.2.6.1 フレーム受信の詳細

以下では、フレーム開始およびフレーム終了の検出に関する詳細と、これらに関連する受信ウィンドウについて説明します。

4.2.6.1.1 フレーム開始(SOF)

全ての UART コマンド/レスポンス フレームは、フレーム開始(SOF)を示す予約済みコードで始まります。このコードは同期用に使われます。SOF を受信すると以下の対応が取られます。

1. 受信が未完了の UART コマンド/レスポンス フレームを破棄する
2. 後続の受信バイトがコマンド/レスポンス フレームの一部として処理される事を許可するために受信ウィンドウを開く

4.2.6.1.2 フレーム終了(EOF)

全ての UART コマンド/レスポンス フレームはフレーム終了(EOF)を示す予約済みコードで終了します。EOF 信号は、コマンドまたはレスポンスが完全に受信され、この後の処理が可能である事を示します。EOF を受信すると以下の対応が取られます。

1. 受信ウィンドウを閉じる
2. コマンドまたはレスポンスのチェックサムを検証する
   - チェックサムが有効であれば、コマンド/レスポンスは後続の処理に渡されます。
   - チェックサムが無効であれば、コマンド/レスポンス フレームは廃棄され、転送エラーがファームウェア プロトコル層へ報告されます。

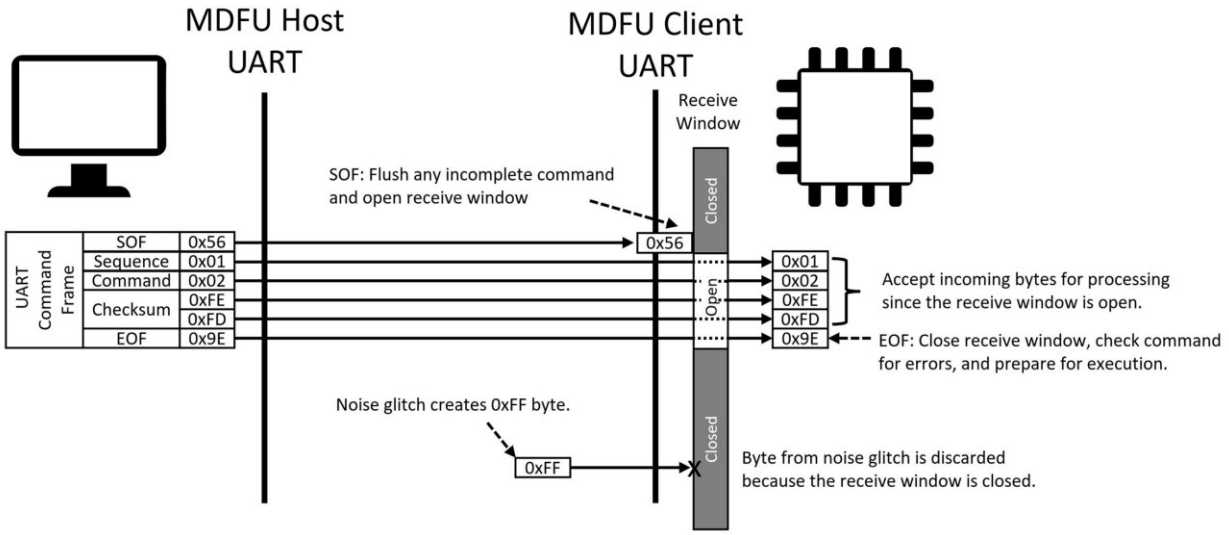
4.2.6.1.3 受信ウィンドウ

バイトストリームが破損してコマンド/レスポンス フレームの同期が失われた場合、無効なデータが受信処理アルゴリズムに渡される可能性があります。

破損したデータストリームが処理される可能性を低減するため、MDFU UART トランスポート層は受信データが処理される事を制限する受信ウィンドウを実装します。

受信ウィンドウが閉じている時に受信したバイトは破棄されて処理されません。このウィンドウが開いている時に受信したバイトは受け入れられます。SOF コードはウィンドウが閉じている時に受け入れられる唯一のコードであり、ウィンドウを開くために使われます。EOF コードまたは特定タイプのエラーによりウィンドウは閉じられます。

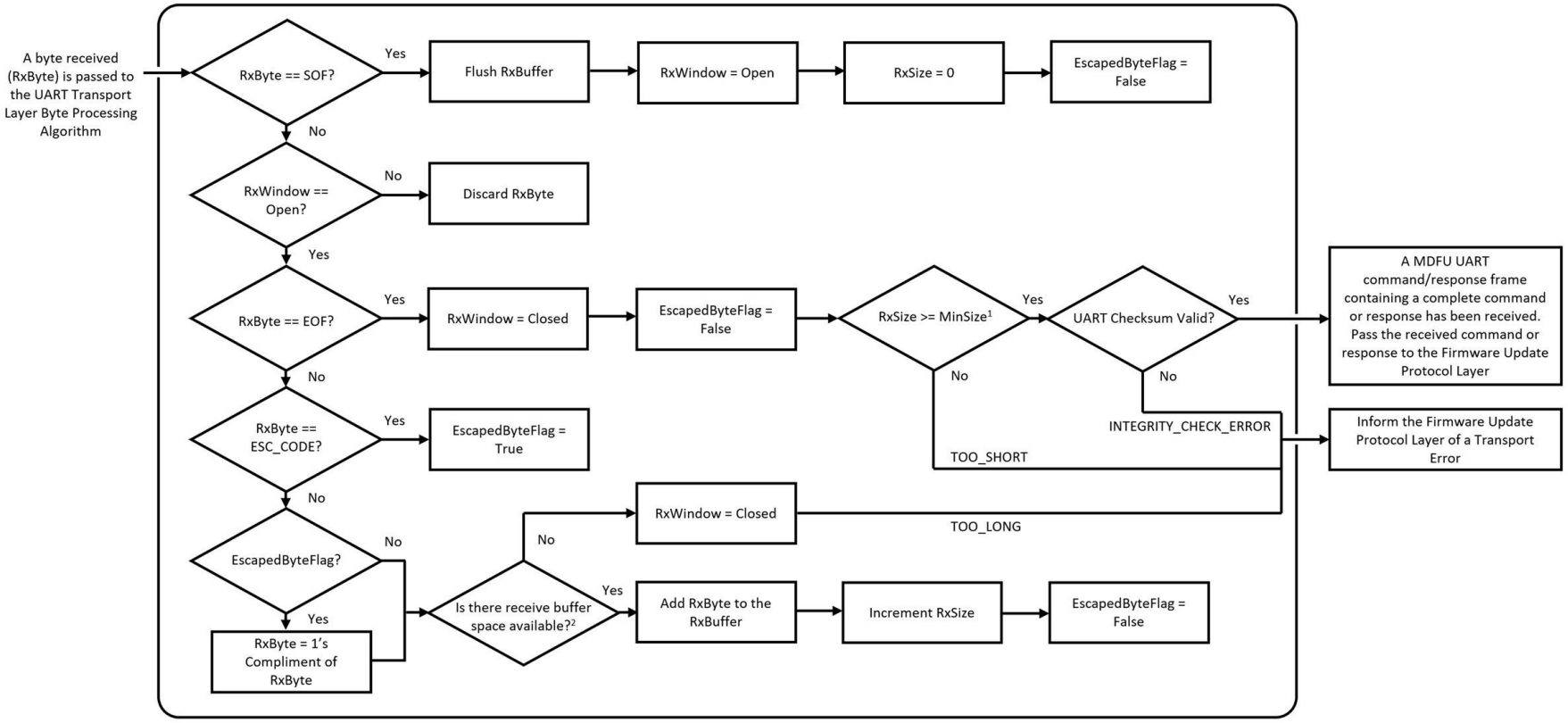
次ページの図に示すコマンドフレームの例では、SOF によって受信ウィンドウが開き、後続の受信バイトは MDFU UART コマンド/レスポンス フレームの一部として受け入れられます。EOF コードによりウィンドウは閉じられ、コマンドを処理するために準備されます。この図は、受信ウィンドウが閉じた後にノイズグリッチによって生じた 0xFF バイトが拒絶される様子も示しています。



4.2.6.2 受信バイトの処理アルゴリズム

次ページの図に示すアルゴリズムに従って受信バイトストリームを処理する事により、コマンド/レスポンスを正しく受信し、それらの整合性をチェックし、各種エラー条件に対応する事が可能となります。

図 4-4. MDFU UART トランスポート層の受信バイト処理アルゴリズム



Note 1: MinSize = 4: 1 バイトのシーケンス + 1 バイトのコマンドコードまたはステータス + 2 バイトのチェックサム = 4 バイト

Note 2: **コマンドの場合:** RxSize < (MaxCommandDataLength + 4)であればバッファ空間として割り当て可能です。
1 バイトのシーケンス + 1 バイトのコマンドコード + MaxCommandDataLength バイトのデータペイロード + 2 バイトのチェックサム = MaxCommandDataLength + 4 バイト

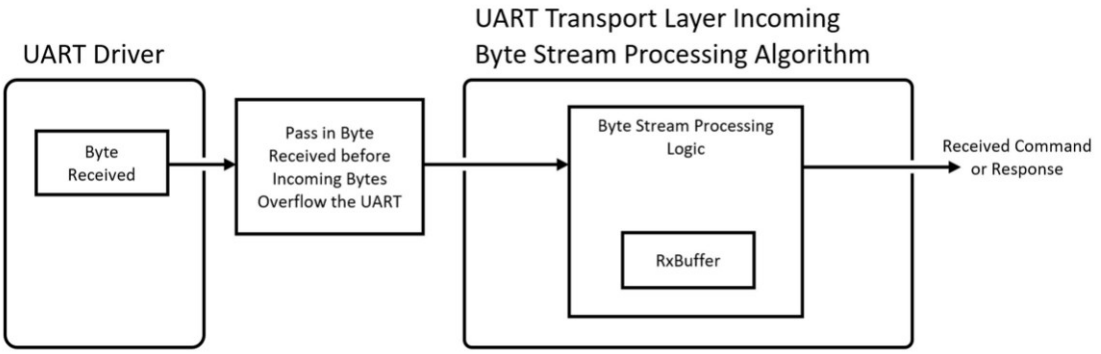
レスポンスの場合: ホストは更新するクライアント向けに十分なサイズのレスポンス バッファを割り当ててレスポンス バッファのオーバーフローを防ぐ必要があります。レスポンスのサイズについては「コマンド/レスポンス ペア」を参照してください。

4.2.6.3 UART ドライバのバッファサイズ

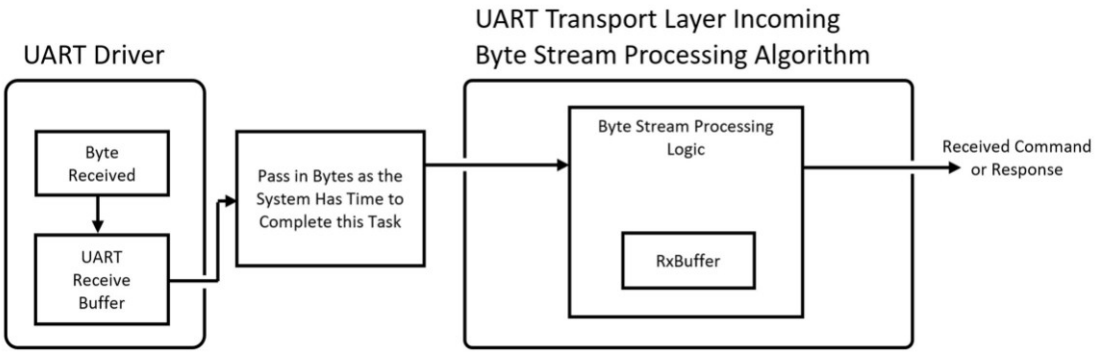
受信バイト処理アルゴリズムは 1 度に 1 バイトずつ演算します。このアルゴリズムは受信中のバイトを極めて高速に処理可能です。

バイトを UART ドライバからトランスポート層の受信バイトストリーム処理アルゴリズムへ渡すためのタイミング要件は、UART ドライバ内のバッファ容量に基づきます。UART ドライバ内のバッファ容量は、アプリケーションで利用可能なシステムリソース(フラッシュ/RAM/処理能力)に基づいてシステム設計者が選択できます。

メモリ容量が制限される場合、受信バイトが届き次第オンザフライで処理するために十分な処理優先度を割り当てる事ができるのであれば、UART ドライバ内でほとんどまたは全くバッファリングを行わない方法(下図参照)を選択できます。



他方、MDFU タスクと同時により高優先度のタスクを実行するシステムでは、UART ドライバと MDFU タスクに十分なタイミング優先度または処理サイクル(もしくはその両方)を与える事ができない場合があります。このようなシステムでは、下図のように UART ドライバの内部にバッファを追加する事で、MDFU タスクに戻って UART バッファからバイトをトランスポート層の受信バイトストリーム処理アルゴリズムに渡す事が必要になる前に、より多くの時間を他のタスクの処理用に使う事ができます。ホストとクライアントの設計者は、この UART 受信バッファのサイズを適切に選定する必要があります。



クライアントの UART 受信バッファのサイズを選定する際は、以下の 2 点を考慮する必要があります。

1. クライアント UART 受信バッファが全てのコマンドのサイズよりも小さい場合、オーバーフローさせずに UART バッファからバイトを取り出すためのタイミング要件は、バッファサイズと UART データレートによって決まります。
2. クライアント UART 受信バッファが最大サイズの MDFU UART コマンドフレームを保持可能なサイズである場合、UART バッファからバイトを取り出すタイミング要件は、バッファサイズおよび UART データレートによって決まりません。この場合のタイミング要件は、コマンド タイムアウト パラメータによって設定されます。このタイムアウト パラメータを長時間に設定すると、MDFU タスクを非常に低優先度のバックグラウンド タスクとして非常に低速なタイミング要件に緩和させて実行させる事ができます。

UART コマンドフレーム全体を受信するために必要な UART バッファのサイズについては、「UART コマンド/レスポンス フレームの最大サイズ」を参照してください。

同様に、ホストの UART 受信バッファのサイズを選定する際は、以下の 2 点を考慮する必要があります。

1. ホスト UART 受信バッファが全てのレスポンスのサイズよりも小さい場合、オーバーフローさせずに UART バッファからバイトを取り出すためのタイミング要件は、バッファサイズと UART データレートによって決まります。
2. ホスト UART 受信バッファがクライアントから送信される最大サイズの UART レスポンス フレームを保持可能なサイズである場合、バッファのオーバーフローを防ぐためにレスポンスが届き次第 UART 受信バッファからバイトを取り出す必要はありません。これにより MDFU タスクの処理が必要になる前にホストが他のタスク向けに使える時間が増えるため、他の高優先度タスクを実行する必要があるホストには有利です。

4.2.7 UART コマンド/レスポンスの整合性チェック

MDFU UART トランスポート層は、コマンドおよびレスポンス内の破損を検出するために 1 の補数表現とした 16 ビット チェックサムを使います。チェックサム計算の詳細を以下に記載します。

4.2.7.1 チェックサム計算の詳細

以下の手順により、コマンドまたはレスポンスの全体に対して 1 の補数チェックサムが計算されます。

1. コマンドまたはレスポンスを一連の uint16_t 値に変換する
   - コマンドまたはレスポンスが奇数個のバイトを格納している場合、最後の uint16_t 値の上位バイトは 0x00 に設定されます。
   - 追加された 0x00 は送信されるコマンドまたはレスポンスに含まれません。これはチェックサム計算にのみ使われます。
2. 全ての uint16_t 値を加算する
3. 加算結果を uint16_t 値に切り詰める
4. 切り詰めた値の 1 の補数を計算する

4.2.7.2 例

図 4-5 に、偶数個のバイトを含むコマンドのチェックサム計算例を示します。

図 4-5. 偶数バイトのチェックサム

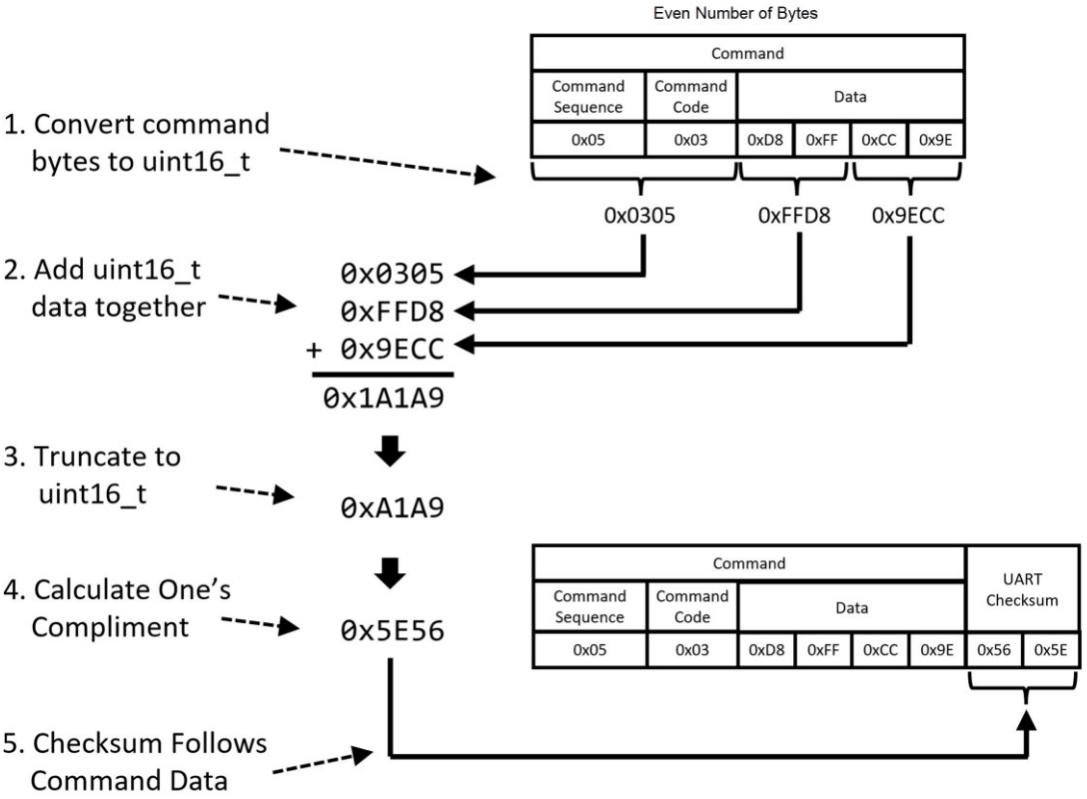
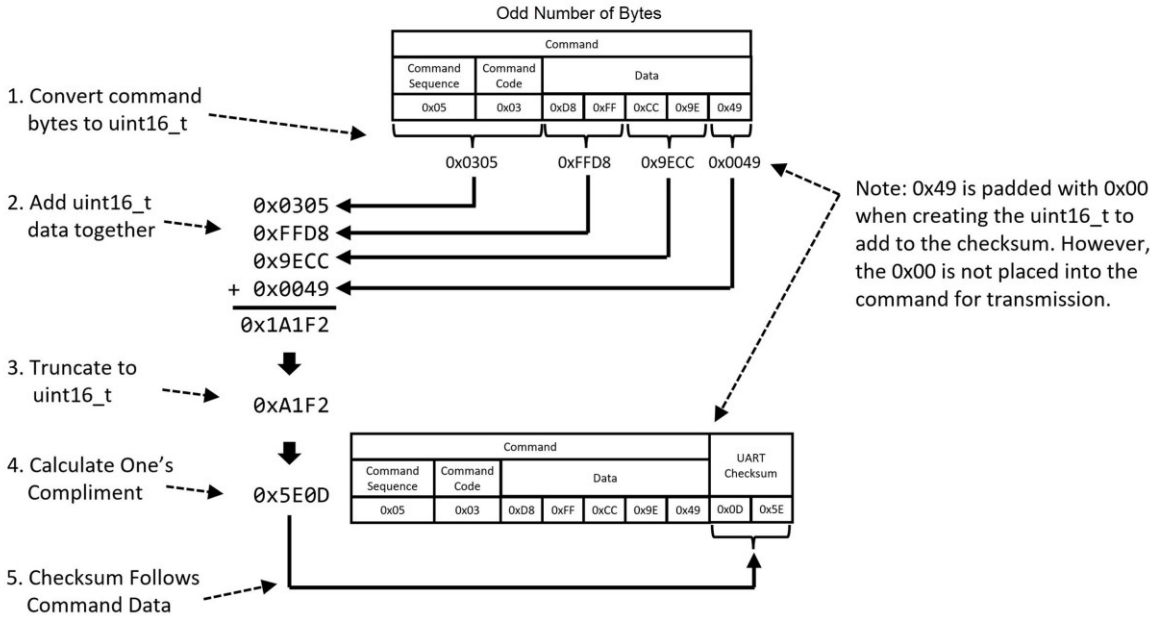


図 4-6 に、奇数個のバイトを含むコマンドのチェックサム計算例を示します。

図 4-6. 奇数バイトのチェックサム



4.2.8 UART コマンド/レスポンス フレームの最大サイズ

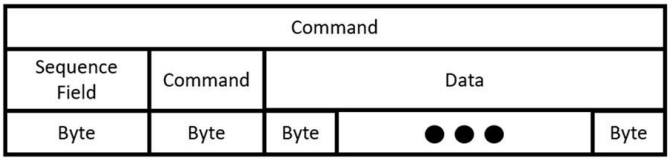
以下では、最悪ケースでの UART コマンド/レスポンス フレームサイズの計算方法について説明します。

4.2.8.1 コマンド/レスポンスの最大サイズ

コマンドとレスポンスの最大データペイロード サイズはクライアントに依存します。コマンド データペイロードに格納される最大バイト数は MaxCommandDataLength パラメータによって定義されます。コマンドとレスポンスのサイズに関する詳細は「コマンド/レスポンス ペア」を参照してください。

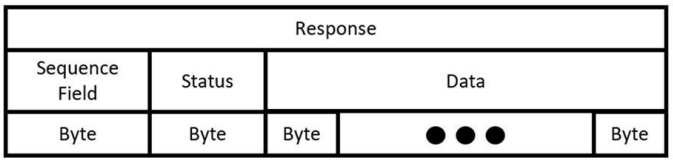
コマンドとレスポンスの最大サイズの計算式を下図に示します。

Maximum Size of Command



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

Maximum Size of Response

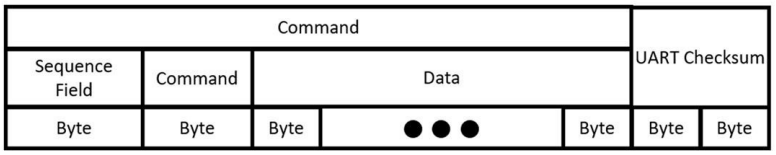


1 Byte + 1 Byte + N_R = 2 + N_R Bytes

4.2.8.2 チェックサム付きコマンド/レスポンスの最大サイズ

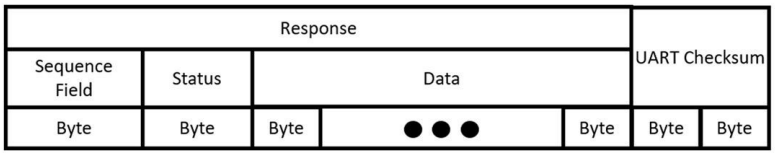
下図に示す通り、UART トランスポート層は 2 バイトのチェックサムをコマンドとレスポンスに追加します。

Maximum Size of Command With Checksum



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

Maximum Size of Response With Checksum



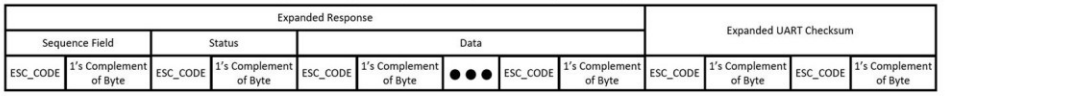
1 Byte + 1 Byte + N_R Bytes + 2 Bytes = 4 + N_R Bytes

4.2.8.3 予約済みコードの置換によって拡張された最悪ケースのコマンド/レスポンスとチェックサムのサイズ

最悪の状況はコマンドまたはレスポンス内の全てのバイトが予約済みコードであり、それらを 2 バイトの代替コードに置換する必要がある場合です。そのような状況で拡張されたコマンド/レスポンスおよびチェックサムを下図に示します。

Worst Case Size of Expanded Command With Expanded Checksum

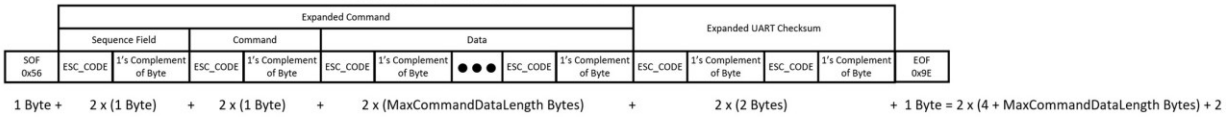
Worst Case Size of Expanded Response With Expanded Checksum



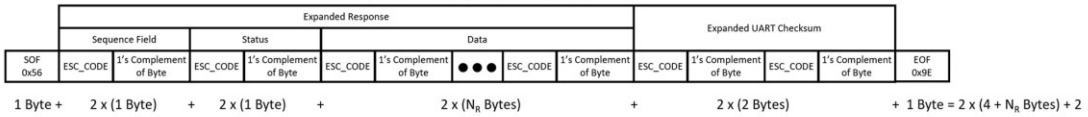
4.2.8.4 最悪ケースの MDFU UART コマンド/レスポンス フレームのサイズ

MDFU UART コマンド/レスポンスには、さらに SOF および EOF コードが追加されます。これにより、MDFU UART コマンドフレームおよびレスポンス フレームの最悪ケースでの最大サイズが最終的に求まります。

Worst Case Command Frame Size



Worst Case Response Frame Size



5. 改訂履歴

本書の各リビジョンでの改訂内容を以下に記載します。

Note: 文字「I 」と「O」はリビジョンの表記に使われません(一部のフォントでは、これらの文字が数字「1」および「0」と見分けにくいため)。

PDF: リビジョン A、WebHelp Ver 1、プロトコル 1.0.0 (2024 年 9 月)

本書は初版です。

Microchip 社の情報

Microchip 社ウェブサイト

Microchip 社はウェブサイト(www.microchip.com)を通してオンライン サポートを提供しています。当ウェブサイトでは、お客様に役立つ情報やファイルを簡単に見つけ出せます。以下を含む各種の情報をご覧になれます。

- **製品サポート** - データシートとエラッタ、アプリケーション ノートとサンプル プログラム、設計リソース、ユーザガイドとハードウェア サポート文書、最新のソフトウェアと過去のソフトウェア
- **技術サポート** – FAQ(よく寄せられる質問)、技術サポートのご依頼、オンライン ディスカッション グループ、Microchip 社のデザイン パートナー プログラムおよびメンバーリスト
- **ご注文とお問い合わせ** - 製品セレクタと注文ガイド、最新プレスリリース、セミナー/イベントの一覧、お問い合わせ先(営業所/正規代理店)の一覧

製品変更通知サービス

Microchip 社の製品変更通知サービスは、お客様に Microchip 社製品の最新情報をお届けする配信サービスです。ご興味のある製品ファミリまたは開発ツールに関する変更、更新、リビジョン、エラッタ情報をいち早くメールにてお知らせします。

http://www.microchip.com/pcn にアクセスし、登録手続きをしてください。

カスタマサポート

Microchip 社製品をお使いのお客様は、以下のチャンネルからサポートをご利用になれます。

- 正規代理店
- 技術サポート

サポートは正規代理店にお問い合わせください。各地の営業所もご利用になれます。本書の最後のページに各国の営業所の一覧を記載しています。

技術サポートは以下のウェブページからもご利用になれます。www.microchip.com/support

製品識別システム

ご注文や製品の価格、納期につきましては弊社または正規代理店にお問い合わせください。

PART NO.	[X](1)	-	X	/XX	XXX
デバイス	テープ&リール オプション		温度レンジ	パッケージ	パターン

デバイス:	PIC16F18313、PIC16LF18313、PIC16F18323、PIC16LF18323	
テープ&リール オプション:	空白	= 標準梱包(チューブまたはトレイ)
	T	= テープ&リール(1)
温度レンジ:	I	= -40～+85 ℃ (産業用温度レンジ)
	E	= -40～+125 ℃ (拡張温度レンジ)
パッケージ:(2)	JQ	= UQFN
	P	= PDIP
	ST	= TSSOP
	SL	= SOIC-14
	SN	= SOIC-8
	RF	= UDFN
パターン:	QTP、SQTP、その他のコード等 (または空白)	

例:

- PIC16LF18313- I/P: 産業用温度レンジ、PDIP パッケージ
- PIC16F18313- E/SS: 拡張温度レンジ、SSOP パッケージ

Note:

1. テープ&リールの識別情報は、カタログの製品番号説明に記載しています。これは製品の注文時に使う識別情報であり、デバイスのパッケージには印刷していません。テープ&リールが選択できるパッケージの在庫/供給状況は正規代理店にお問い合わせください。
2. 小型パッケージ オプションがご利用になれる場合があります。小型パッケージについては www.microchip.com/ packaging をご覧になるか、弊社正規代理店までお問い合わせください。

Microchip 社のデバイスコード保護機能

Microchip 社製品のコード保護機能について以下の点にご注意ください。

- Microchip 社製品は、該当する Microchip 社データシートに記載の仕様を満たしています。
- Microchip 社では、通常の条件ならびに仕様に従って使った場合、Microchip 社製品のセキュリティ レベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- Microchip 社はその知的財産権を重視し、積極的に保護しています。Microchip 社製品のコード保護機能の侵害は固く禁じられており、デジタル ミレニアム著作権法に違反します。
- Microchip 社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、Microchip 社が製品を「解読不能」として保証するものではありません。コード保護機能は常に進歩しています。Microchip 社では、常に製品のコード保護機能の改善に取り組んでいます。

法律上の注意点

本書および本書に記載されている情報は、Microchip 社製品を設計、テスト、お客様のアプリケーションと統合する目的を含め、Microchip 社製品に対してのみ使用する事ができます。それ以外の方法でこの情報を使用する事はこれらの条項に違反します。デバイス アプリケーションの情報は、ユーザの便宜のためにのみ提供されるものであり、更新によって変更となる事があります。お客様のアプリケーションが仕様を満たす事を保証する責任は、お客様にあります。

その他のサポートについては、弊社または代理店にお問い合わせになるか、www.microchip.com/en-us/support/design-help/ client-support-services をご覧ください。

Microchip 社は本書の情報を現状のままで提供しています。Microchip 社は明示的、暗黙的、書面、口頭、法定のいずれであるかを問わず、本書に記載されている情報に関して、非侵害性、商品性、特定目的への適合性の暗黙的保証、または状態、品質、性能に関する保証をはじめとするいかなる類の表明も保証も行いません。

いかなる場合も Microchip 社は、本情報またはその使用に関連する間接的、特殊的、懲罰的、偶発、的または必然的損失、損害、費用、経費のいかんにかかわらず、また Microchip 社がそのような損害が生じる可能性について報告を受けていた場合あるいは損害が予測可能であった場合でも、一切の責任を負いません。法律で認められる最大限の範囲を適用しようとも、本情報またはその使用に関連する一切の申し立てに対する Microchip 社の責任限度額は、使用者が当該情報に関連して Microchip 社に直接支払った額を超えません。

Microchip 社の明示的な書面による承認なしに、生命維持装置あるいは生命安全用途に Microchip 社の製品を使用する事は全て購入者のリスクとし、また購入者はこれによって発生したあらゆる損害、クレーム、訴訟、費用に関して、Microchip 社は擁護され、免責され、損害をうけない事に同意するものとします。特に明記しない場合、暗黙的あるいは明示的を問わず、Microchip 社が知的財産権を保有しているライセンスは一切譲渡されません。

商標

Microchip 社の名称とロゴ、Microchip ロゴ、Adaptec、AVR、AVR ロゴ、AVR Freaks、BesTime、BitCloud、CryptoMemory、CryptoRF、dsPIC、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi ロゴ、MOST、MOST ロゴ、MPLAB、OptoLyzer、PIC、picoPower、PICSTART、PIC32 ロゴ、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST ロゴ、SuperFlash、Symmetricom、SyncServer、Tachyon、TimeSource、tinyAVR、UNI/O、Vectron、XMEGA は米国およびその他の国における Microchip Technology Incorporated の登録商標です。

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 ロゴ、Quiet-Wire、SmartFusion、SyncWorld、TimeCesium、TimeHub、TimePictra、TimeProvider、ZL は米国における Microchip Technology Incorporated の登録商標です。

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 ロゴ、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、ZENA は米国およびその他の国における Microchip Technology Incorporated の商標です。

SQTP は米国における Microchip Technology Incorporated のサービス マークです。

Adaptec ロゴ、Frequency on Demand、Silicon Storage Technology、Symmcom はその他の国における Microchip Technology Incorporated の登録商標です。

GestIC は、その他の国における Microchip Technology Germany II GmbH & Co. KG (Microchip Technology Inc.の子会社)の登録商標です。

その他の商標は各社に帰属します。

ISBN: 979-8-3371-0084-5

品質管理システム

Microchip 社の品質管理システムについては www.microchip.com/quality をご覧ください。

各国の営業所とサービス

北米	アジア/太平洋	アジア/太平洋	ヨーロッパ
本社 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel:480-792-7200 Fax:480-792-7277 技術サポート： http://www.microchip.com/support URL: www.microchip.com	**オーストラリア - シドニー** Tel:61-2-9868-6733	**インド - バンガロール** Tel:91-80-3090-4444	**オーストリア - ヴェルス** Tel:43-7242-2244-39 Fax:43-7242-2244-393
アトランタ Duluth, GA Tel:678-957-9614 Fax:678-957-1455	**中国 - 北京** Tel:86-10-8569-7000	**インド - ニューデリー** Tel:91-11-4160-8631	**デンマーク - コペンハーゲン** Tel:45-4485-5910 Fax:45-4485-2829
オースティン、TX Tel:512-257-3370	**中国 - 成都** Tel:86-28-8665-5511	**インド - プネ** Tel:91-20-4121-0141	**フィンランド - エスポー** Tel:358-9-4520-820
ボストン Westborough, MA Tel:774-760-0087 Fax:774-760-0088	**中国 - 重慶** Tel:86-23-8980-9588	**日本 - 大阪** Tel:81-6-6152-7160	**フランス - パリ** Tel:33-1-69-53-63-20 Fax:33-1-69-30-90-79
シカゴ Itasca, IL Tel:630-285-0071 Fax:630-285-0075	**中国 - 東莞** Tel:86-769-8702-9880	**日本 - 東京** Tel:81-3-6880- 3770	**ドイツ - ガーヒング** Tel:49-8931-9700
ダラス Addison, TX Tel:972-818-7423 Fax:972-818-2924	**中国 - 広州** Tel:86-20-8755-8029	**韓国 - 大邱** Tel:82-53-744-4301	**ドイツ - ハーン** Tel:49-2129-3766400
デトロイト Novi, MI Tel:248-848-4000	**中国 - 杭州** Tel:86-571-8792-8115	**韓国 - ソウル** Tel:82-2-554-7200	**ドイツ - ハイルブロン** Tel:49-7131-72400
ヒューストン、TX Tel:281-894-5983	**中国 - 香港 SAR** Tel:852-2943-5100	**マレーシア - クアラルンプール** Tel:60-3-7651-7906	**ドイツ - カールスルーエ** Tel:49-721-625370
インディアナポリス Noblesville, IN Tel:317-773-8323 Fax:317-773-5453 Tel:317-536-2380	**中国 - 南京** Tel:86-25-8473-2460	**マレーシア - ペナン** Tel:60-4-227-8870	**ドイツ - ミュンヘン** Tel:49-89-627-144-0 Fax:49(-89/-627)-144/-44
ロサンゼルス Mission Viejo, CA Tel:949-462-9523 Fax:949-462-9608 Tel:951-273-7800	**中国 - 青島** Tel:86-532-8502-7355	**フィリピン - マニラ** Tel:63-2-634-9065	**ドイツ - ローゼンハイム** Tel:49-8031-354-560
ローリー、NC Tel:919-844-7510	**中国 - 上海** Tel:86-21-3326-8000	**シンガポール** Tel:65-6334-8870	**イスラエル - ラーナナ** Tel:972-9-744-7705
ニューヨーク、NY Tel:631-435-6000	**中国 - 瀋陽** Tel:86-24-2334-2829	**台湾 - 新竹** Tel:886-3-577-8366	**イタリア - ミラノ** Tel:39-0331-742611 Fax:39-0331-466781
サンノゼ、CA Tel:408-735-9110 Tel:408-436-4270	**中国 - 深圳** Tel:86-755-8864-2200	**台湾 - 高雄** Tel:886-7-213-7830	**イタリア - パドヴァ** Tel:39-049-7625286
カナダ - トロント Tel:905-695-1980 Fax:905-695-2078	**中国 - 蘇州** Tel:86-186-6233-1526	**台湾 - 台北** Tel:886-2-2508-8600	**オランダ - ドリューネン** Tel:31-416-690399 Fax:31-416-690340
	中国 - 武漢 Tel:86-27-5980-5300	**タイ - バンコク** Tel:66-2-694-1351	**ノルウェー - トロンハイム** Tel:47-7288-4388
	中国 - 西安 Tel:86-29-8833-7252	**ベトナム - ホーチミン** Tel:84-28-5448-2100	**ポーランド - ワルシャワ** Tel:48-22-3325737
	中国 - 厦門 Tel:86-592-2388138		**ルーマニア - ブカレスト** Tel:40-21-407-87-50
	中国 - 珠海 Tel:86-756-3210040		**スペイン - マドリッド** Tel:34-91-708-08-90 Fax:34-91-708-08-91
			スウェーデン - ヨーテボリ Tel:46-31-704-60-40
			スウェーデン - ストックホルム Tel:46-8-5090-4654
			イギリス - ウォーキンガム Tel:44-118-921-5800 Fax:44-118-921-5820