

## RX ファミリ

### フラッシュメモリ データ管理モジュール Firmware Integration Technology

---

#### 要旨

本アプリケーションノートでは、ルネサス エレクトロニクス製 RX MCU 内蔵のフラッシュメモリを使用したデータ管理方法とその使用方法を説明します。

フラッシュメモリデータ管理モジュール（以降、DATFRX と称す）は、内蔵フラッシュメモリをデータ管理目的で利用するための上位層に位置するソフトウェアです。

MCU 個別のフラッシュメモリを制御するため別途、下位層に位置するソフトウェア、フラッシュ FIT モジュールの最新版をルネサス エレクトロニクスホームページから入手してください。

- フラッシュ FIT モジュール（内蔵フラッシュ書き換え）**Rev.3.40、もしくはそれ以降の Rev.**

RX ファミリ フラッシュモジュール Firmware Integration Technology (R01AN2184)

#### 対象デバイス

- ・ RX ファミリ

#### 関連ドキュメント

- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
- e<sup>2</sup> studio に組み込む方法 Firmware Integration Technology (R01AN1723)
- CS+に組み込む方法 Firmware Integration Technology (R01AN1826)
- Renesas e<sup>2</sup> studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)
- RX ファミリ フラッシュモジュール Firmware Integration Technology (R01AN2184)

## 目次

1. 概要 .....	4
1.1 DATFRX とは .....	4
1.2 DATFRX の概要 .....	4
1.2.1 用語の定義 .....	4
1.2.1.1 フラッシュタイプ .....	4
1.2.1.2 データフラッシュメモリ .....	4
1.2.1.3 ブロック .....	4
1.2.1.4 BGO .....	4
1.2.2 機能概要 .....	5
1.2.3 DATFRX レイヤ概要 .....	6
1.3 API の概要 .....	7
1.3.1 BGO 動作の設定 .....	7
1.4 処理例 .....	8
1.4.1 フラッシュタイプ 1 .....	8
1.4.1.1 全体像(main 関数の処理例) .....	8
1.4.1.2 初期化のフロー .....	9
1.4.2 フラッシュタイプ 2,3,4 .....	10
1.4.2.1 全体像(main 関数の処理例) .....	10
1.4.2.2 初期化のフロー .....	11
1.4.3 コールバック関数 .....	12
1.5 状態遷移図 .....	13
1.6 制限事項 .....	14
2. API 情報 .....	15
2.1 ハードウェアの要求 .....	15
2.2 ソフトウェアの要求 .....	15
2.3 サポートされているツールチェーン .....	15
2.4 使用する割り込みベクタ .....	15
2.5 ヘッダファイル .....	15
2.6 整数型 .....	15
2.7 コンパイル時の設定 .....	16
2.7.1 データ番号の追加方法 .....	17
2.7.1.1 r_datfrx_rx_config.h の修正例 .....	17
2.7.1.2 r_dm_1.c/r_dm_2.c/r_dm_3.c/r_dm_4.c の修正例 .....	17
2.8 コードサイズ .....	18
2.8.1 フラッシュタイプ 1 .....	18
2.8.2 フラッシュタイプ 2 .....	19
2.8.3 フラッシュタイプ 3 .....	20
2.8.4 フラッシュタイプ 4 .....	21
2.9 引数 .....	22
2.10 戻り値 .....	22
2.11 コールバック関数 .....	23
2.12 FIT モジュールの追加方法 .....	24
2.13 for 文、while 文、do while 文について .....	24

3. API 関数 .....	25
4. 端子設定 .....	45
5. デモプロジェクト .....	46
5.1 ワークスペースにデモを追加する .....	46
5.2 デモのダウンロード方法 .....	46
6. 付録 .....	47
6.1 動作確認環境 .....	47
6.2 トラブルシューティング .....	48
6.3 データ管理 .....	49
6.3.1 DATFRX 領域 .....	49
6.3.1.1 1ブロック 1024 バイト (フラッシュタイプ 1) .....	49
6.3.1.2 1ブロック 128 バイト (フラッシュタイプ 2) .....	50
6.3.1.3 1ブロック 64 バイト (フラッシュタイプ 3、4) .....	50
6.3.1.4 1ブロック 32 バイト (フラッシュタイプ 2) .....	50
6.3.2 ブロック領域【フラッシュタイプ 1】 .....	51
6.3.2.1 ブロックヘッダ (フラッシュタイプ 1) .....	51
6.3.2.2 データヘッダ (フラッシュタイプ 1) .....	52
6.3.2.3 データ (フラッシュタイプ 1) .....	54
6.3.3 ブロック管理【フラッシュタイプ 2、3、4】 .....	55
6.3.3.1 ブロックヘッダ (フラッシュタイプ 2、3、4) .....	55
6.3.3.2 データヘッダ (フラッシュタイプ 2、3、4) .....	55
6.3.3.3 データ (フラッシュタイプ 2、3、4) .....	57
6.3.4 ブロックの状態と判定 .....	58
6.3.4.1 フラッシュタイプ 1 .....	58
6.3.4.2 フラッシュタイプ 2、3、4 .....	62
7. 参考ドキュメント .....	65

## 1. 概要

### 1.1 DATFRX とは

ルネサス エレクトロニクス製 RX MCU 内蔵フラッシュメモリをデータ管理目的で利用するための上位層に位置するソフトウェアです。

### 1.2 DATFRX の概要

#### 1.2.1 用語の定義

##### 1.2.1.1 フラッシュタイプ

下位層に位置するフラッシュ FIT モジュールは、使用する技術およびシーケンサにより、フラッシュタイプをフラッシュタイプ 1、フラッシュタイプ 2、フラッシュタイプ 3、フラッシュタイプ 4 に分類します。

フラッシュタイプの詳細はフラッシュ FIT モジュールの最新版をルネサス エレクトロニクスホームページから入手し、確認ください。

RX ファミリ フラッシュモジュール Firmware Integration Technology (R01AN2184)

##### 1.2.1.2 データフラッシュメモリ

データ格納用フラッシュメモリです。

フラッシュタイプにより、データフラッシュメモリの名称が異なります。フラッシュタイプ別の、データフラッシュメモリの名称を記載します。本書ではデータフラッシュメモリの呼称を用います。

表 1-1 データフラッシュメモリ

フラッシュタイプ	名称
フラッシュタイプ 1	E2 データフラッシュ
フラッシュタイプ 2	E2 データフラッシュ
フラッシュタイプ 3	データフラッシュメモリ
フラッシュタイプ 4	データフラッシュメモリ

##### 1.2.1.3 ブロック

データフラッシュメモリは特定の領域を持つブロックが複数集まり、構成されます。

ブロックサイズや個数は使用 MCU により異なります。ブロックの詳細についてはユーザズマニュアルハードウェア編「フラッシュメモリ」の項を参照ください。

##### 1.2.1.4 BGO

バックグラウンドオペレーションの略です。

データ管理対象のデータフラッシュメモリ領域のデータを更新中に、RAM または外部メモリに配置されたユーザプログラムの動作が可能です。

## 1.2.2 機能概要

以下に、フラッシュタイプと機能の概略を示します。

表 1-2 機能概要

機能			フラッシュ タイプ1	フラッシュ タイプ2、 3、4
データ管理	API コールでユーザの指定するデータ番号のデータの更新と読み出しができます。		○	○
	ユーザデータを DATFRX の論理データで管理します。データ数、データサイズはユーザが設定できます。		○	○
	データ更新処理では、空いたブロックにデータを更新します。データ更新を行うブロックは DATFRX が選択します。DATFRX では、特定のブロックにデータ更新が集中しないように、ブロックの更新順番を設定しています。データ更新処理では旧データの消去を行いません。		—	○
	不要な旧データを保管しているブロックをブロック消去処理で消去できます。消去するブロックは DATFRX が選択します。		○	○
	データ更新処理中の電源遮断／リセット	再起動後の初期化関数実行にて、電源遮断／リセットを検出します。	○	○
		データが有効でない時、更新前のデータに復旧します。	○	○
	ブロック消去処理中の電源遮断／リセット	再起動後の初期化関数実行にて、管理するブロックの消去中電源遮断、リセットを検出します。	○	○
		更新データの状態により、そのデータが有効かどうか判断。有効でない場合、更新前のデータに復旧します。 ※有効ブロックと誤認した場合、間違ったデータを有効データとして扱う場合があります。	○	○
データフラッシュメモリ	データフラッシュメモリ領域のデータと、BGO 動作に対応。		○	○
	ビッグエンディアン／リトルエンディアンでの動作が可能。		○	○

## 1.2.3 DATFRX レイヤ概要

DATFRX とフラッシュ FIT モジュールの関係を以下に示します。

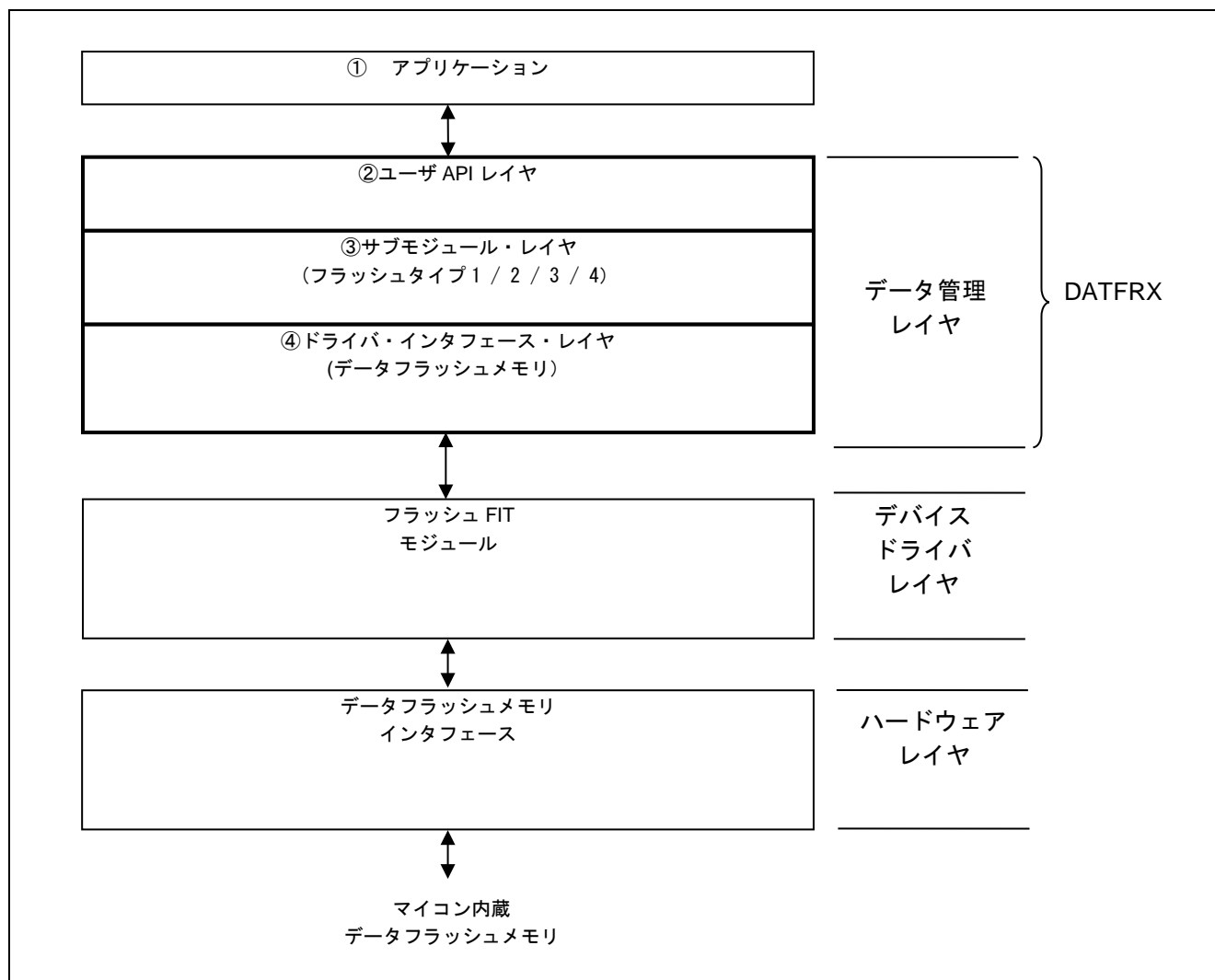


図 1-1 DATFRX とフラッシュ FIT モジュールの関係

## ① アプリケーション

本 FIT モジュールは、データフラッシュメモリの制御例を同梱しています。FITDemos サブディレクトリを参照してください。

## ② ユーザ API レイヤ

データフラッシュメモリのデータ管理用 API で、下位層のデバイスドライバに依存しない部分です。

## ③ サブモジュール・レイヤ

データフラッシュメモリのデータ管理サブモジュールで、下位層のデバイスドライバに依存しない部分です。

## ④ ドライバ・インタフェース・レイヤ

下位層のデバイスドライバとの接続部分です。

### 1.3 API の概要

「表 1-3 API 関数」に DATFRX に含まれる API 関数を示します。

DATFRX はデータフラッシュメモリの BGO 動作に対応します。必ず BGO 動作の設定とデータフラッシュメモリサイズの設定を行ってください。

フラッシュタイプにより、使用可能な関数が異なります。

表 1-3 API 関数

関数名	説明	フラッシュタイプ 1	フラッシュタイプ 2、3、4
R_FLASH_DM_Open()	DATFRX のオープン処理	○	○
R_FLASH_DM_Close()	DATFRX のクローズ処理	○	○
R_FLASH_DM_Init()	初期化処理（分割）	○	○
R_FLASH_DM_InitAdvance()	初期化処理継続（分割）	—	○
R_FLASH_DM_Format()	フォーマット処理	○	○
R_FLASH_DM_Read()	データ読み出し処理	○	○
R_FLASH_DM_Write()	データ更新処理	○	○
R_FLASH_DM_Erase()	ブロック消去処理	○	○
R_FLASH_DM_Reclaim()	リクレーム処理	○	—
R_FLASH_DM_Control()	各種状態チェック処理	○	○
R_FLASH_DM_GetVersion()	バージョン取得	○	○

#### 1.3.1 BGO 動作の設定

フラッシュ FIT モジュールの BGO 機能を利用した制御に対応しています。r\_flash\_rx\_config.h を下記設定にしてください。

表 1-4 フラッシュ FIT モジュール設定

フラッシュ FIT モジュール r_flash_rx_config.h #define 定義	BGO 動作あり	BGO 動作なし
	データフラッシュメモリ	データフラッシュメモリ
FLASH_CFG_CODE_FLASH_ENABLE	0	未サポート
FLASH_CFG_CODE_FLASH_BGO	0	
FLASH_CFG_DATA_FLASH_BGO	1	

## 1.4 処理例

## 1.4.1 フラッシュタイプ 1

## 1.4.1.1 全体像(main 関数の処理例)

DATFRX の R\_FLASH\_DM\_Open()から R\_FLASH\_DM\_Close()までの例を示します。

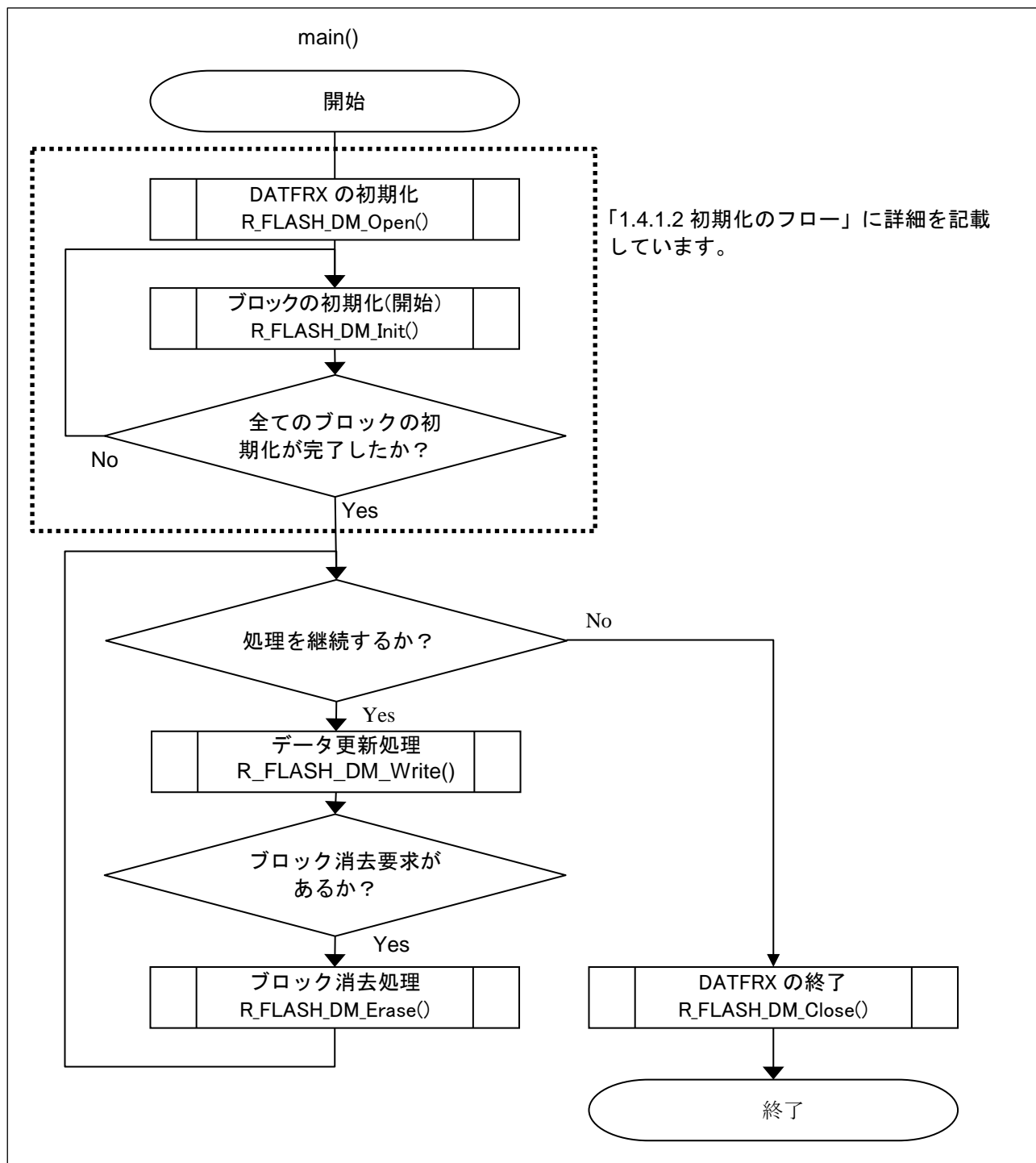


図 1-2 main 関数の処理例(フラッシュタイプ 1)



## 1.4.1.2 初期化のフロー

R\_FLASH\_DM\_Init()後、ユーザデータが扱えるようになるまでの詳細な例をフローに示します。

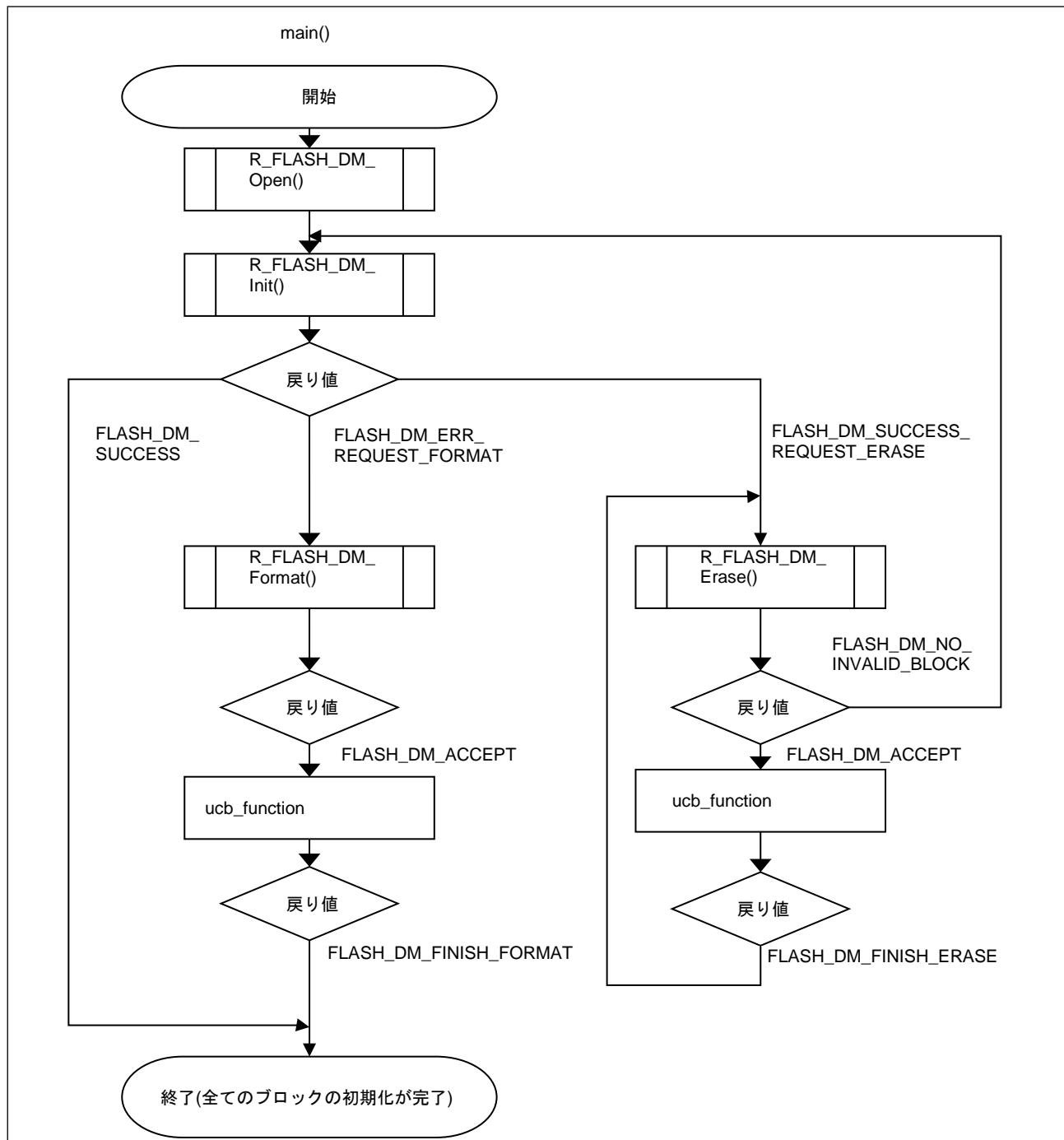


図 1-3 R\_FLASH\_DM\_Init()後の処理例(フラッシュタイプ 1)

## 1.4.2 フラッシュタイプ 2,3,4

## 1.4.2.1 全体像(main 関数の処理例)

DATFRX の R\_FLASH\_DM\_Open()から R\_FLASH\_DM\_Close()までの例を示します。

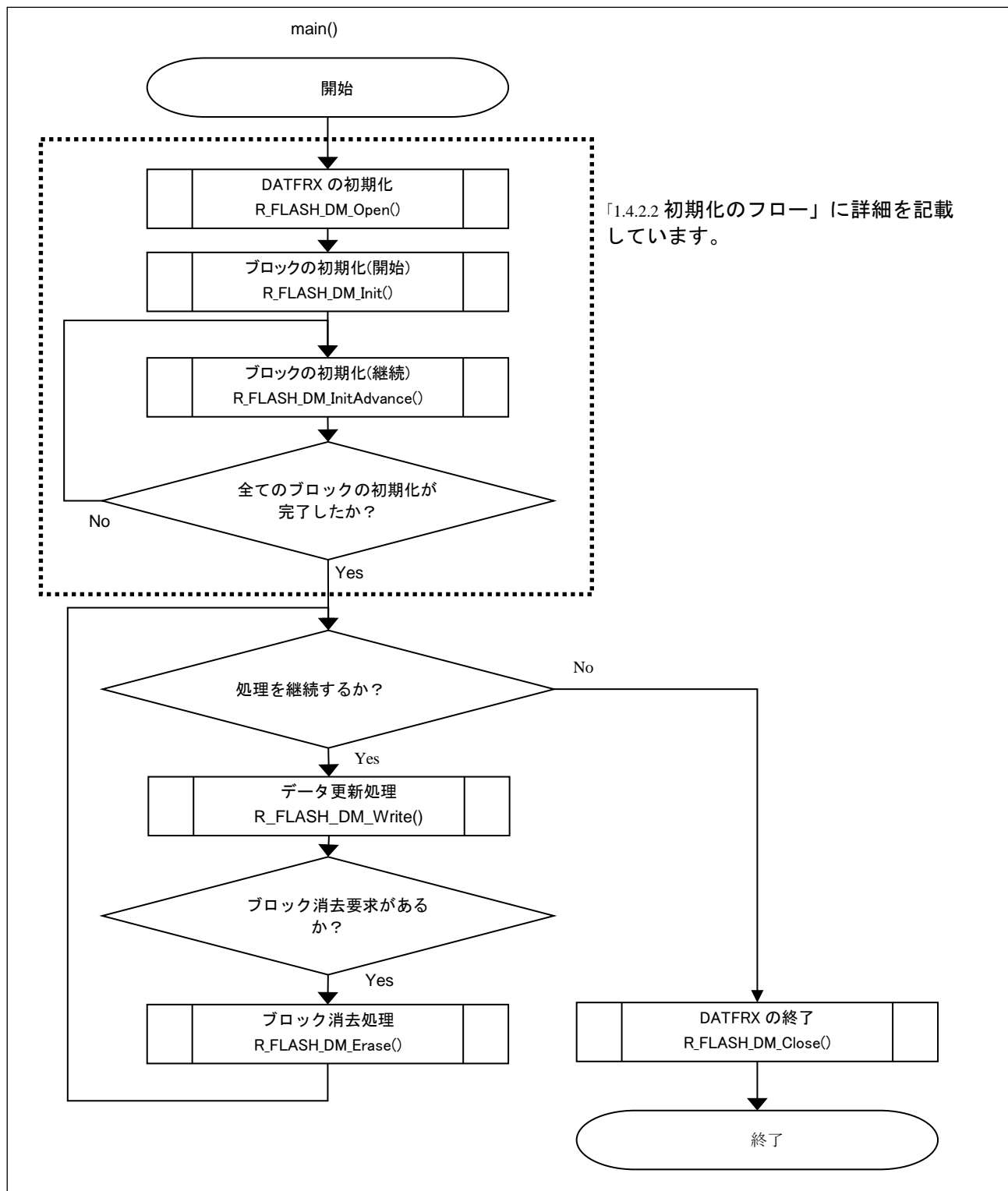


図 1-4 main 関数の処理例(フラッシュタイプ 2,3,4)

## 1.4.2.2 初期化のフロー

R\_FLASH\_DM\_Init()後、ユーザデータが扱えるようになるまでの詳細な例をフローに示します。

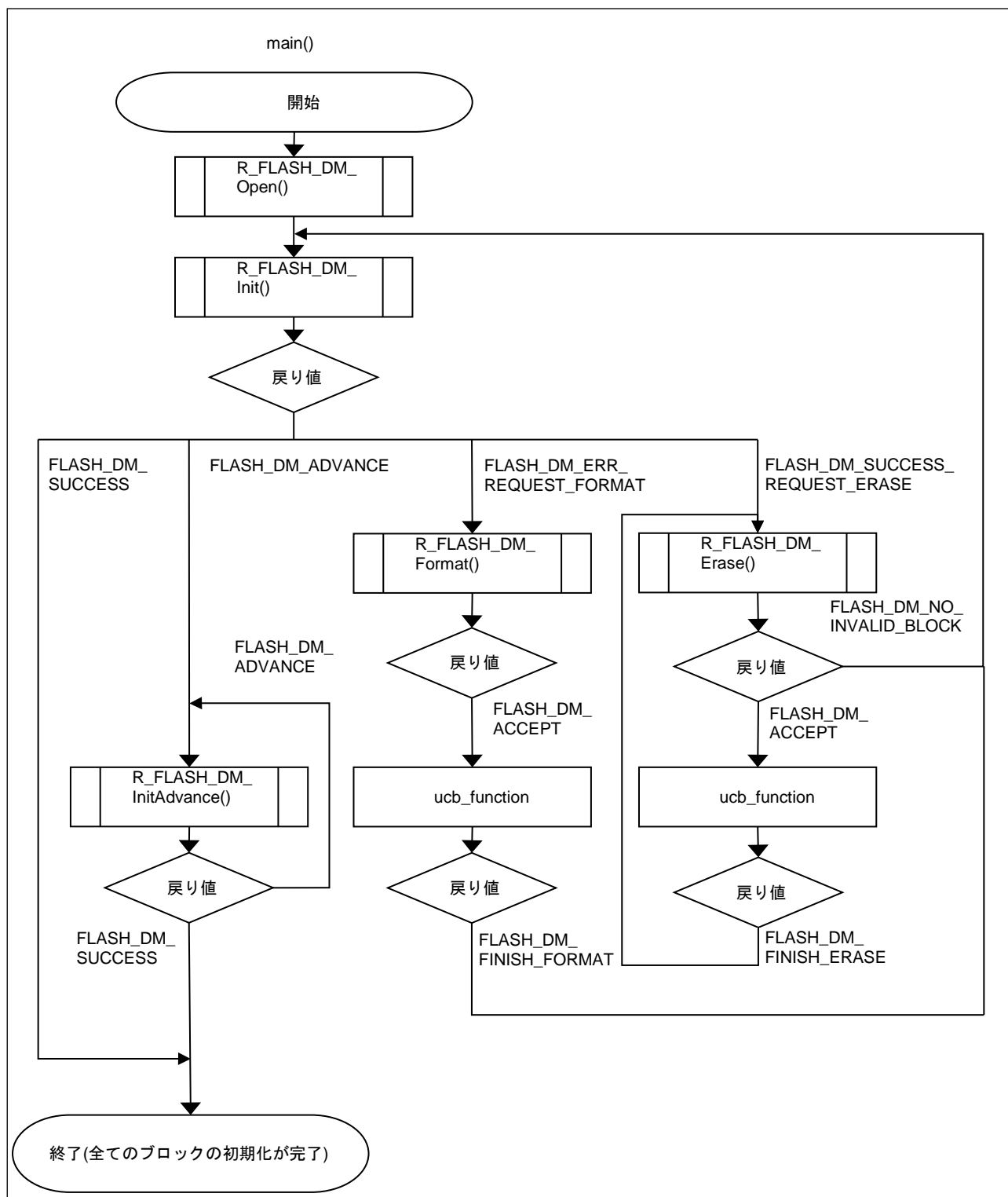


図 1-5 R\_FLASH\_DM\_Init()後の処理例(フラッシュタイプ 2,3,4)

1.4.3 コールバック関数

コールバック関数の中身はユーザが作成します。

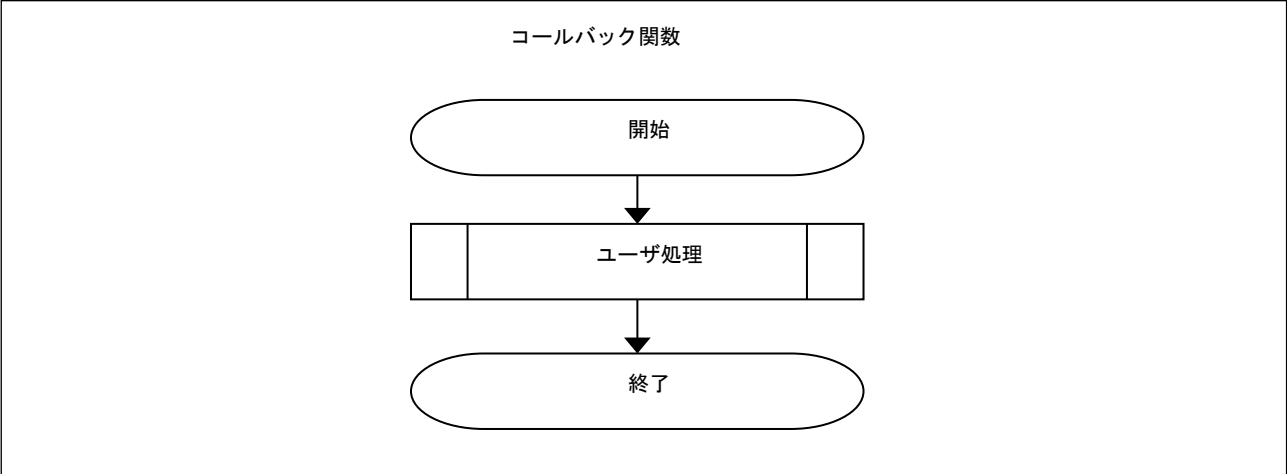


図 1-6 コールバック関数

## 1.5 状態遷移図

「図 1-7 状態遷移図」に本モジュールの状態遷移図を示します。

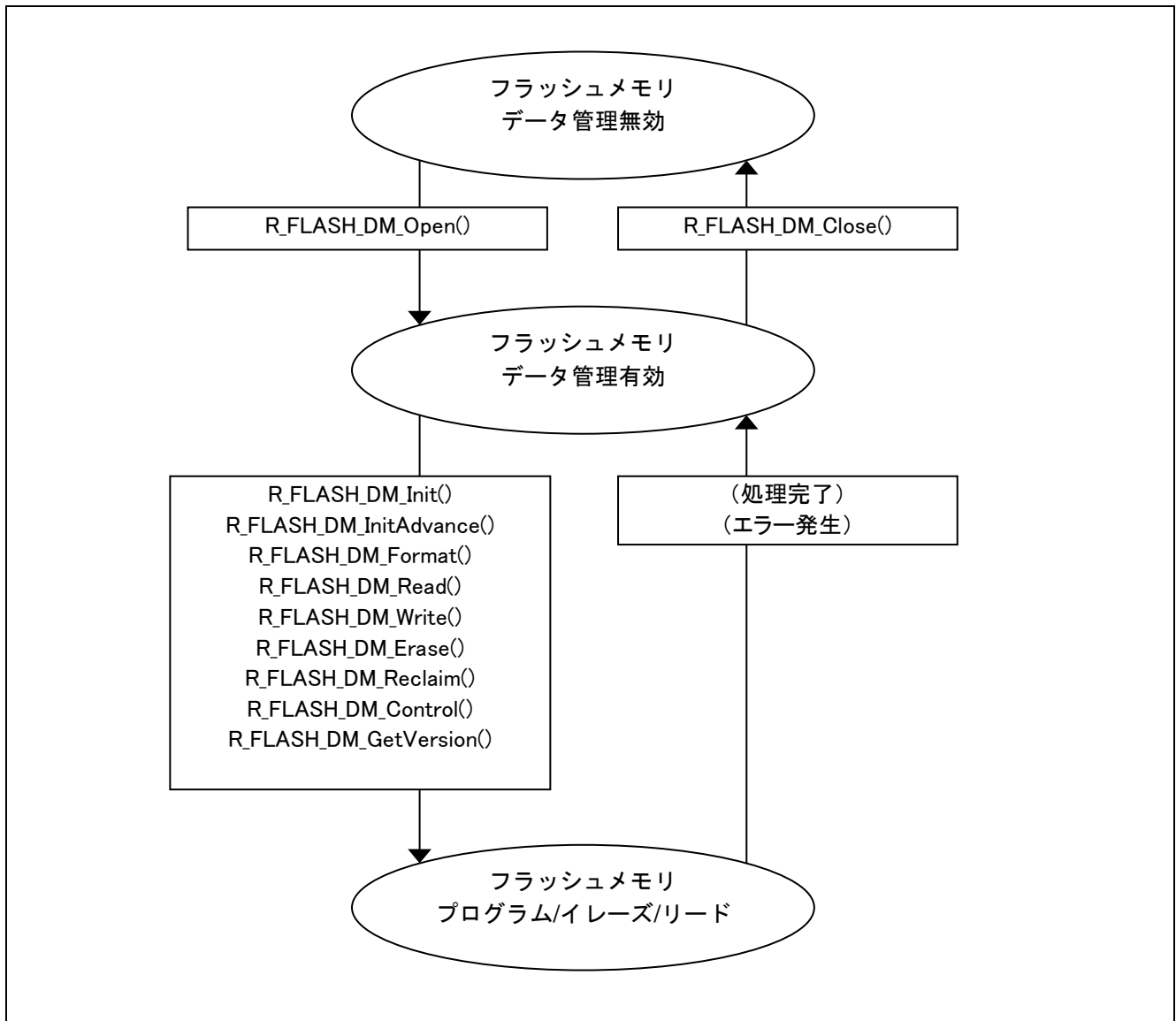


図 1-7 状態遷移図

## 1.6 制限事項

制限事項を記載します。

表 1-5 制限事項

名称	内容
電源電圧について	DATFRX は、フラッシュ FIT モジュールを用いて、データフラッシュメモリに対してプログラムやブロックイレースを行います。プログラムコマンドやブロックイレースコマンドを実行する API を使用する際は、ユーザーズマニュアル・ハードウェア編が定める MCU の電源電圧条件を満たすようにしてください。
データフラッシュメモリ仕様について	フラッシュメモリ制御レジスタや電気的特性等のデータフラッシュメモリ仕様については、使用する MCU のユーザーズマニュアル・ハードウェア編を参照してください。
他のユーザプログラムとの排他について	DATFRX の処理が完了するまでは、フラッシュメモリレジスタへのアクセスは禁止です。アクセスした場合、正常に動作しません。 DATFRX は、フラッシュ FIT モジュールを用いて必要に応じてフラッシュメモリのレジスタへアクセスします。DATFRX の処理を実行している間、同様にフラッシュメモリへアクセスするユーザプログラムとの並行動作は考慮されていません。
プログラム中／イレース中／ブランクチェック中のリセット	確実にプログラム／ブロックイレースを完了させるため、ユーザシステムにて下記事項を守るようにしてください。 電源電圧が低下した場合であっても、プログラムコマンド／ブロックイレースコマンドの処理が完了するまでの間、スーパー・キャパシタ等により MCU の動作電圧を保持してください。動作電圧と保持時間に関して、使用する MCU のユーザーズマニュアル・ハードウェア編の電気的特性を参照してください。 プログラムコマンド／ブロックイレースコマンド実行中に電源電圧低下を検出した場合、次のコマンドが発行されないように、API 関数をコールしないでください。また、電源電圧低下により動作電圧を下回った場合、MCU とフラッシュメモリが不定状態になる可能性があります。MCU をリセットし、DATFRX の初期化処理を実行してください。
ドライバのフォーマット、初期化について	フォーマット中に電源遮断またはリセットが発生しないようにしてください。 電源遮断またはリセットが発生すると、フォーマットが未完了状態のため、その後の初期化処理でブロックを誤認識する場合があります。
初期化処理について	初期化処理中に電源遮断またはリセットが発生しないようにしてください。 電源遮断またはリセットが発生すると、その後の初期化処理の結果、データを消失する可能性があります。
API コールの制限	DATFRX の API は、C 言語で記述したアプリケーションプログラムから発行してください。割り込みハンドラおよび周期ハンドラ等から発行した場合、正常動作を保証できません。
引数の設定規則、レジスタの保証規則	DATFRX の引数の設定規則やレジスタの保証規則は、C コンパイラの設定規則および保証規則に準じています。関連マニュアルをご参照ください。
セクションについて	初期値無し領域のセクションは、0 に初期化してください。

## 2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

### 2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- データフラッシュメモリ

### 2.2 ソフトウェアの要求

DATFRX を FIT 対応させて使用する場合、以下のパッケージに依存しています。

- r\_bsp
- r\_flash\_rx

### 2.3 サポートされているツールチェイン

本 FIT モジュールは「6.1 動作確認環境」に示すツールチェインで動作確認を行っています。

### 2.4 使用する割り込みベクタ

データフラッシュメモリへの書き込み、および、消去の完了を検出するため、FRDYI/FRDYIE 割り込みを使用します。DATFRX のオープン処理 R\_FLASH\_DM\_Open() をコールするまでにシステムを割り込み許可状態にしてください。FRDYI/FRDYIE 割り込みの詳細は、フラッシュ FIT モジュールのアプリケーションノートを参照してください。

### 2.5 ヘッダファイル

すべての API 呼び出しと使用されるインタフェース定義は r\_flash\_dm\_rx\_if.h に記載しています。

ビルド毎の構成オプションは、r\_datfrx\_rx\_config.h で選択します。

### 2.6 整数型

このプロジェクトは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

## 2.7 コンパイル時の設定

本制御ソフトウェアのコンフィギュレーションオプションの設定は、r\_datfrx\_rx\_config.h で行います。  
オプション名および設定値に関する説明を下表に示します。

なお、フラッシュ FIT モジュール r\_flash\_rx\_config.h の#define 定義を以下のとおりに設定してください。

```
#define FLASH_CFG_CODE_FLASH_ENABLE (0)
```

```
#define FLASH_CFG_CODE_FLASH_BGO (0)
```

```
#define FLASH_CFG_DATA_FLASH_BGO (1)
```

表 2-1 Configuraiton options

Configuration options in r_datfrx_rx_config.h	
定義	内容
FLASH_DM_CFG_FRDYI_INT_PRIORITY デフォルト値は "1"	データフラッシュメモリ BGO 動作で使用する場合、FRDYI/FRDYIE 割り込みプライオリティレベルを定義してください。 設定可能な値は 1~15 です。
FLASH_DM_CFG_DF_BLOCK_NUM デフォルト値は "4"	データフラッシュメモリで管理するブロック数を定義してください。 設定可能な値は、 フラッシュタイプ 1 の場合：2~8 フラッシュタイプ 2,3,4 の場合：2~1024 です。 データ管理対象のブロック情報は、「6.3 データ管理」を参照してください。 ブロック数の最大値は使用する MCU のユーザーズマニュアル・ハードウェア編を参照してください。
FLASH_DM_CFG_DF_DATA_NUM デフォルト値は "10"	データフラッシュメモリで管理するデータ数を定義してください。 設定可能な値は フラッシュタイプ 1 の場合：1~255（管理データ番号は No.0~No.254） フラッシュタイプ 2,3,4 の場合：1~1024（管理データ番号は No.0~No.1023） です。
FLASH_DM_CFG_DF_SIZE_NOx データ 0 のデフォルト値は "1" "x" はデータ番号	データフラッシュメモリで管理するデータ番号ごとにデータサイズを定義してください。 設定可能な値は、 フラッシュタイプ 1 の場合：1~256 フラッシュタイプ 2,3,4 の場合：1~1024 です。 使用しないデータ番号の設定値は無視されます。 ただし、No.40 以降は定義していないため、別途、定義を追加する必要があります。また、一部ソースコードの変更が必要です。
FLASH_DM_CFG_CRC_HARDWARE デフォルト値は "0"	この定義はフラッシュタイプ 2,3,4 に適用されます。 RX の CRC 演算器を使用するかどうか設定してください。 "0" の場合、ソフトウェアで CRC 演算を行います。 "1" の場合、RX MCU に搭載されている CRC 演算器を使用して高速な CRC 演算を行います。  ※将来展開用です。本バージョンでは"0"のみサポートします。



## 2.7.1 データ番号の追加方法

データフラッシュメモリで管理するデータ番号として、No.40 以降を使用したい場合、別途定義を追加する必要があります。また、一部ソースコードの変更が必要です。

以下にデータフラッシュメモリ制御にてデータ番号 No.40-No.47 を追加する例を示します。

### 2.7.1.1 r\_datfrx\_rx\_config.h の修正例

以下のコードを追加してください。

なお、()内のデータサイズは任意の値を定義してください。

[DATA FLASH : SET THE DATA LENGTH FOR THE DATA NUMBER]

```
#define FLASH_DM_CFG_DF_SIZE_NO40    (4)
#define FLASH_DM_CFG_DF_SIZE_NO41    (4)
#define FLASH_DM_CFG_DF_SIZE_NO42    (4)
#define FLASH_DM_CFG_DF_SIZE_NO43    (4)
#define FLASH_DM_CFG_DF_SIZE_NO44    (4)
#define FLASH_DM_CFG_DF_SIZE_NO45    (4)
#define FLASH_DM_CFG_DF_SIZE_NO46    (4)
#define FLASH_DM_CFG_DF_SIZE_NO47    (4)
```

### 2.7.1.2 r\_dm\_1.c/r\_dm\_2.c/r\_dm\_3.c/r\_dm\_4.c の修正例

const 変数 gc\_dm\_data\_size[]の No.40-No.47 のコメントを外してください。

<r\_dm\_1.c 314 行目付近、またはr\_dm\_2.c/r\_dm\_3.c/r\_dm\_4.c 60 行目付近>

```
const uint16 gc_dm_data_size[] =
{
```

```
    FLASH_DM_CFG_DF_SIZE_NO0, FLASH_DM_CFG_DF_SIZE_NO1,
    FLASH_DM_CFG_DF_SIZE_NO2, FLASH_DM_CFG_DF_SIZE_NO3,
    FLASH_DM_CFG_DF_SIZE_NO4, FLASH_DM_CFG_DF_SIZE_NO5,
    FLASH_DM_CFG_DF_SIZE_NO6, FLASH_DM_CFG_DF_SIZE_NO7,
```

(省略)

```
/* FLASH_DM_CFG_DF_SIZE_NO40, FLASH_DM_CFG_DF_SIZE_NO41,
FLASH_DM_CFG_DF_SIZE_NO42, FLASH_DM_CFG_DF_SIZE_NO43, */ ←行の先頭と最後のコメント
を外してください。
/* FLASH_DM_CFG_DF_SIZE_NO44, FLASH_DM_CFG_DF_SIZE_NO45,
FLASH_DM_CFG_DF_SIZE_NO46, FLASH_DM_CFG_DF_SIZE_NO47, */ ←行の先頭と最後のコメント
を外してください。
```

## 2.8 コードサイズ

### 2.8.1 フラッシュタイプ 1

表 2-2 にデータフラッシュメモリ BGO 動作の必要コードサイズを示します。

表 2-2 コードサイズ（データフラッシュメモリ BGO 動作）

使用 MCU	使用メモリ	サイズ (注 1、注 2、注 3、注 4、注 5、注 6、注 7、注 8)
RX111	ROM	5045 バイト + (4 バイト × n ブロック数) + (2 バイト × m 管理データ数)
	RAM	31 バイト + (4 バイト × n ブロック数) + (2 バイト × m 管理データ数)
	最大使用ユーザスタック	120 バイト
	最大使用割り込みスタック	56 バイト
RX231	ROM	5039 バイト + (4 バイト × n ブロック数) + (2 バイト × m 管理データ数)
	RAM	33 バイト + (4 バイト × n ブロック数) + (2 バイト × m 管理データ数)
	最大使用ユーザスタック	120 バイト
	最大使用割り込みスタック	56 バイト

(注 1) 「コンパイル時の設定」のデフォルト設定を選択した場合の値です。選択する定義により、コードサイズは異なります。

(注 2) 動作条件は以下のとおりです。

```
r_flash_dm_rx_if.c
r_dispatch_1.c
r_dm_1.c
```

(注 3) 必要メモリサイズは、C コンパイラのバージョンやコンパイル・オプションにより異なります。

(注 4) リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。

(注 5)  $n = 2 - 8$

(注 6)  $m = 1 - 255$

(注 7) フラッシュ FIT モジュール（内蔵フラッシュ書き換え）のサイズを含みません。

(注 8) 最大使用割り込みスタックは Excep\_FCU\_FRDYI() コール時の値です。

## 2.8.2 フラッシュタイプ 2

表 2-3 にデータフラッシュメモリ BGO 動作の必要コードサイズを示します。

表 2-3 コードサイズ (データフラッシュメモリ BGO 動作)

使用 MCU	使用メモリ	サイズ (注 1、注 2、注 3、注 4、注 5、注 6、注 7)
RX210	ROM	5099 バイト
	RAM	18 バイト + (3 バイト × n ブロック数)
	最大使用ユーザスタック	192 バイト
	最大使用割り込みスタック	248 バイト
RX63N	ROM	5026 バイト
	RAM	18 バイト + (3 バイト × n ブロック数)
	最大使用ユーザスタック	192 バイト
	最大使用割り込みスタック	248 バイト

(注 1) 「コンパイル時の設定」のデフォルト設定を選択した場合の値です。選択する定義により、コードサイズは異なります。

(注 2) 動作条件は以下のとおりです。

```
r_flash_dm_rx_if.c
r_dispatch_2.c
r_datf_crc.c
r_dm_2.c
```

(注 3) 必要メモリサイズは、C コンパイラのバージョンやコンパイル・オプションにより異なります。

(注 4) リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。

(注 5) (RX210)  $n = 4 - 64$  , (RX63N)  $n = 4 - 1024$

(注 6) フラッシュ FIT モジュール (内蔵フラッシュ書き換え) のサイズを含みません。

(注 7) 最大使用割り込みスタックは Excep\_FCU\_FRDYI() コール時の値です。

## 2.8.3 フラッシュタイプ 3

表 2-4 にデータフラッシュメモリ BGO 動作の必要コードサイズを示します。

表 2-4 コードサイズ (データフラッシュメモリ BGO 動作)

使用 MCU	使用メモリ	サイズ (注 1、注 2、注 3、注 4、注 5、注 6、注 7)
RX64M	ROM	4999 バイト
	RAM	18 バイト + (3 バイト × n ブロック数)
	最大使用ユーザスタック	160 バイト
	最大使用割り込みスタック	72 バイト
RX71M	ROM	4999 バイト
	RAM	18 バイト + (3 バイト × n ブロック数)
	最大使用ユーザスタック	160 バイト
	最大使用割り込みスタック	72 バイト
RX66T	ROM	4999 バイト
	RAM	18 バイト + (3 バイト × n ブロック数)
	最大使用ユーザスタック	160 バイト
	最大使用割り込みスタック	72 バイト

(注 1) 「コンパイル時の設定」のデフォルト設定を選択した場合の値です。選択する定義により、コードサイズは異なります。

(注 2) 動作条件は以下のとおりです。

```
r_flash_dm_rx_if.c
r_dispatch_2.c
r_datf_crc.c
r_dm_3.c
```

(注 3) 必要メモリサイズは、C コンパイラのバージョンやコンパイル・オプションにより異なります。

(注 4) リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。

(注 5) (RX64M)  $n = 4 - 1024$ , (RX71M)  $n = 4 - 1024$ , (RX66T)  $n = 4 - 512$

(注 6) フラッシュ FIT モジュール (内蔵フラッシュ書き換え) のサイズを含みません。

(注 7) 最大使用割り込みスタックは `Excep_FCU_FRDYI()` コール時の値です。

## 2.8.4 フラッシュタイプ 4

表 2-5 にデータフラッシュメモリ BGO 動作の必要コードサイズを示します。

表 2-5 コードサイズ（データフラッシュメモリ BGO 動作）

使用 MCU	使用メモリ	サイズ (注 1、注 2、注 3、注 4、注 5、注 6、注 7)
RX65N-2MB	ROM	4999 バイト
	RAM	18 バイト + (3 バイト × n ブロック数)
	最大使用ユーザスタック	156 バイト
	最大使用割り込みスタック	68 バイト

(注 1) 「コンパイル時の設定」のデフォルト設定を選択した場合の値です。選択する定義により、コードサイズは異なります。

(注 2) 動作条件は以下のとおりです。

```
r_flash_dm_rx_if.c
r_dispatch_2.c
r_datf_crc.c
r_dm_4.c
```

(注 3) 必要メモリサイズは、C コンパイラのバージョンやコンパイル・オプションにより異なります。

(注 4) リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。

(注 5) (RX65N-2MB) n = 4 – 512

(注 6) フラッシュ FIT モジュール（内蔵フラッシュ書き換え）のサイズを含みません。

(注 7) 最大使用割り込みスタックは Excep\_FCU\_FRDYI() コール時の値です。

## 2.9 引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともに `r_flash_dm_rx_if.h` に記載されています。

```
typedef struct _flash_dm_info
{
    uint8_t    data_no;
    uint8_t    rsv[3];
    uint8_t    * p_data;
} st_flash_dm_info_t;
```

## 2.10 戻り値

API 関数の戻り値およびエラーコードを示します。この列挙型は API 関数のプロトタイプ宣言とともに `r_flash_dm_rx_if.h` で記載されています。

表 2-6 戻り値

戻り値	説明
FLASH_DM_SUCCESS	処理成功
FLASH_DM_ACCEPT	受付処理成功
FLASH_DM_SUCCESS_REQUEST_ERASE	処理成功、消去要求
FLASH_DM_ADVANCE	アドバンス要求
FLASH_DM_FINISH_FORMAT	フォーマット成功
FLASH_DM_FINISH_WRITE	データ更新成功
FLASH_DM_FINISH_ERASE	ブロック消去成功
FLASH_DM_FINISH_RECLAIM	リクレーム成功
FLASH_DM_FINISH_INITIALIZE	初期化成功
FLASH_DM_NO_INVALID_BLOCK	無効ブロックなし
FLASH_DM_ERR_INIT	初期化処理失敗
FLASH_DM_ERR_BUSY	ビジー状態
FLASH_DM_ERR_ARGUMENT	パラメーターエラー
FLASH_DM_ERR_REQUEST_INIT	初期化要求
FLASH_DM_ERR_REQUEST_FORMAT	フォーマット要求
FLASH_DM_ERR_REQUEST_ERASE	ブロック消去要求
FLASH_DM_ERR_DATA_NOT_PRESENT	データ番号不整合
FLASH_DM_ERR_CANT_RECLAIM	リクレーム処理実行不可
FLASH_DM_ERR_REQUEST_RECLAIM	リクレーム要求
FLASH_DM_ERR_FORMAT	フォーマット失敗
FLASH_DM_ERR_WRITE	データ更新失敗
FLASH_DM_ERR_ERASE	ブロック消去失敗
FLASH_DM_ERR_RECLAIM	リクレーム失敗
FLASH_DM_ERR_OPEN	オープン失敗
FLASH_DM_ERR_CLOSE	クローズ失敗
FLASH_DM_ERR	エラー

## 2.11 コールバック関数

フォーマット処理／データ更新処理／ブロック消去処理／リクレーン処理の正常終了時またはエラー終了時、ユーザ設定のコールバック関数が呼び出されます。

コールバック関数の登録方法は「3 API 関数」を参照してください。

### Format

```
void user_cb_function(
    void* event
)
```

### Parameters

\*event

コマンドの結果が格納されます。

### Return Values

なし

### Properties

ユーザプログラムでプロトタイプ宣言します。

### Description

フォーマット処理／データ更新処理／ブロック消去処理／リクレーン処理の完了が通知されます。

通知内容は引数 void\* event に格納されます。通知内容の取得方法は「3 R\_FLASH\_DM\_Open()」を参照してください。

表 2-7 コールバック関数への通知

引数に格納される値	意味	フラッシュタイプ
FLASH_DM_FINISH_FORMAT	フォーマット完了	1,2,3,4
FLASH_DM_FINISH_WRITE	データ更新完了	1,2,3,4
FLASH_DM_FINISH_ERASE	ブロック消去完了	1,2,3,4
FLASH_DM_FINISH_RECLAIM	リクレーン完了	1
FLASH_DM_ERR_FORMAT	フォーマット失敗	1,2,3,4
FLASH_DM_ERR_WRITE	データ更新失敗	1,2,3,4
FLASH_DM_ERR_ERASE	ブロック消去失敗	1,2,3,4
FLASH_DM_ERR_RECLAIM	リクレーン失敗	1
FLASH_DM_ERR	処理失敗	1,2,3,4

### 2.12 FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

- (1) e<sup>2</sup> studio 上で Smart Configurator を使用して FIT モジュールを追加する場合  
e<sup>2</sup> studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e<sup>2</sup> studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。
- (2) e<sup>2</sup> studio 上で FIT Configurator を使用して FIT モジュールを追加する場合  
e<sup>2</sup> studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e<sup>2</sup> studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。
- (3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合  
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e<sup>2</sup> studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。
- (4) CS+上で FIT モジュールを追加する場合  
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

### 2.13 for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT\_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT\_LOOP」で該当の処理を検索できます。



### 3. API 関数

#### R\_FLASH\_DM\_Open()

データ管理処理を開始する際、最初に使用する関数です。DATFRX が使用するワーク領域を確保し、コールバック関数を登録します。

##### Format

```
e_flash_dm_status_t R_FLASH_DM_Open(  
    uint32_t* p_flash_dm_work,  
    p_flash_dm_callback func  
)
```

##### Parameters

*\*p\_flash\_dm\_work*

ワーク領域のポインタ

領域サイズは以下の通り。

(フラッシュタイプ 1)140 バイト

(フラッシュタイプ 2,3,4)261+2 バイト×FLASH\_DM\_CFG\_DF\_DATA\_NUM

上記に示したサイズを満たすワーク領域を用意してください

*func*

コールバック関数のポインタ

フォーマット処理／データ更新処理／ブロック消去処理／リクレーム処理の正常終了時またはエラー終了時にコールされます。

##### Return Values

FLASH\_DM\_SUCCESS

正常終了

FLASH\_DM\_ERR\_ARGUMENT

パラメーターエラー

→引数の確認後、再度オープン関数をコールしてください。

FLASH\_DM\_ERR\_OPEN

フラッシュ FIT モジュールのオープン関数エラー

→再度オープン関数をコールしてください。

##### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

##### Description

フラッシュ FIT モジュールのオープン関数 R\_FLASH\_Open()をコールします。

##### Reentrant

リエントラントは不可能です。

**Example**

```
static uint32_t g_flash_dm_work[314/sizeof(uint32_t)];

void user_cb_function(void * event) /* callback function */
{
    e_flash_dm_status_t callback_event = (e_flash_dm_status_t)event;

    /* Perform callback functionality here */
    switch(callback_event)
    {
        case FLASH_DM_FINISH_FORMAT:
        {
            nop();
        }
        break;
        case FLASH_DM_FINISH_WRITE:
        {
            nop();
        }
        break;
        /* : */
        /* : */
        default:
        {
            nop();
        }
        break;
    }
}

void main(void)
{
    if (FLASH_DM_SUCCESS != R_FLASH_DM_Open(&g_flash_dm_work,
    &user_cb_function))
    {
        /* Error */
    }
}
```

**Special Notes:**

なし

## R\_FLASH\_DM\_Close()

---

データ管理を終了し、DATFRX が使用するワーク領域を解放します。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Close(  
    void  
)
```

### Parameters

なし

### Return Values

<i>FLASH_DM_SUCCESS</i>	正常終了
<i>FLASH_DM_ERR_CLOSE</i>	クローズ失敗

→もう一度 *R\_FLASH\_DM\_Close()* をコールしてください。

### Properties

*r\_flash\_dm\_rx\_if.h* にプロトタイプ宣言されています。

### Description

データ管理を終了し、DATFRX が使用するワーク領域を解放します。  
フラッシュ FIT モジュールのクローズ関数 *R\_FLASH\_Close()* をコールします。

### Reentrant

リエントラントは不可能です。

### Example

```
if (FLASH_DM_SUCCESS != R_FLASH_DM_Close())  
{  
    /* Error */  
}
```

### Special Notes:

なし

## R\_FLASH\_DM\_Init()

ドライバを初期化します。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Init(  
    void  
)
```

### Parameters

なし

### Return Values

FLASH_DM_SUCCESS	正常終了(フラッシュタイプ1) →フラッシュタイプ1の初期化が完了しました。
FLASH_DM_ADVANCE	初期化処理中(フラッシュタイプ2,3,4) →フラッシュタイプ2,3,4はFLASH_DM_SUCCESSが戻るまでFLASH_DM_InitAdvance()をコールし、初期化を完了させてください。
FLASH_DM_SUCCESS_REQUEST_ERASE	正常終了、かつ消去要求(フラッシュタイプ1) →ブロック消去関数をコールしてください。
FLASH_DM_ERR_BUSY	API実行中またはフラッシュメモリビジー状態 →実行中の処理の終了後、再度初期化関数をコールしてください。
FLASH_DM_ERR_REQUEST_FORMAT	未フォーマットまたはブロックヘッダ異常(フラッシュタイプ1) →フォーマット関数をコールしてください。
FLASH_DM_ERR_REQUEST_INIT	未初期化状態 →初期化関数をコールしてください。
FLASH_DM_ERR_INIT	初期化エラー →R_FLASH_DM_Open()が行われているか確認し、再度R_FLASH_DM_Init()をコールしてください。

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

DATFRX を初期化します。

フォーマット関数 R\_FLASH\_DM\_Format()以外の API を実行する前に本関数を実行してください。

フラッシュタイプ 2,3,4 の場合、初期化処理時間が長い場合、初期化処理を分割実行します。

戻り値の FLASH\_DM\_ADVANCE が返った後、R\_FLASH\_DM\_InitAdvance()をコールし、初期化処理を完了してください。

### Reentrant

リエントラントは不可能です。

**Example**

```
e_flash_dm_status_t ret;

do
{
    ret = R_FLASH_DM_Init();
}while (FLASH_DM_ERR_BUSY == ret);
if(ret == FLASH_DM_ERR_REQUEST_FORMAT)
{
    ret = R_FLASH_DM_Format();
}
else if(ret == FLASH_DM_SUCCESS_REQUEST_ERASE)
{
    ret = R_FLASH_DM_Erase();
}
else if(ret != FLASH_DM_ADVANCE)
{
    ret = R_FLASH_DM_InitAdvance();
}
else
{
}
```

**Special Notes:**

初期化処理はノンブロッキングではありません。**API** 内で処理終了を待ちます。  
コールバック関数の中で本関数をコールしないでください。

---

## R\_FLASH\_DM\_InitAdvance()

---

初期化処理の実行を継続します。

本関数はフラッシュタイプ 2, 3, 4 専用です。

### Format

```
e_flash_dm_status_t R_FLASH_DM_InitAdvance(  
    void  
)
```

### Parameters

なし

### Return Values

FLASH_DM_SUCCESS	正常終了
FLASH_DM_ADVANCE	初期化中（次処理を開始） →再度 R_FLASH_DM_InitAdvance 関数をコールしてください。
FLASH_DM_ERR_BUSY	API 実行中またはフラッシュメモリビジー状態 →実行中の処理の終了後、再度初期化関数をコールしてください。
FLASH_DM_ERR_REQUEST_FORMAT	未フォーマットまたはブロックヘッダ異常 →フォーマット関数をコールしてください。
FLASH_DM_ERR_REQUEST_INIT	初期化エラー →再度初期化関数をコールしてください。
FLASH_DM_ERR_INIT	初期化エラー →R_FLASH_DM_Open()が行われているか確認し、再度 R_FLASH_DM_Init()をコールしてください。

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

DATFRX の初期化処理を継続して行います。

R\_FLASH\_DM\_Init()で初期化処理を開始した後、R\_FLASH\_DM\_InitAdvance()をコールし、初期化処理を完了してください。

フォーマット関数 R\_FLASH\_DM\_Format()以外の API を実行する前に初期化処理を完了してください。

### Reentrant

リエントラントは不可能です。

### Example

```
e_flash_dm_status_t ret;  
  
do  
{  
    ret = R_FLASH_DM_InitAdvance();  
}  
while (FLASH_DM_ERR_BUSY == ret);
```

### Special Notes:

初期化処理はノンブロッキングではありません。API 内で処理終了を待ちます。  
コールバック関数の中で本関数をコールしないでください。

## R\_FLASH\_DM\_Format()

データフラッシュメモリ内のデータを消去します。

初期化処理可能な状態にします。

(フラッシュタイプ 1 の場合、フォーマット処理の正常終了後は、初期化処理の実行は不要です。)

### Format

```
e_flash_dm_status_t R_FLASH_DM_Format(  
    void  
)
```

### Parameters

なし

### Return Values

FLASH\_DM\_ACCEPT

処理受付

FLASH\_DM\_ERR\_BUSY

API 実行中またはフラッシュメモリビジー状態

→実行中の処理の終了後、再度フォーマット関数をコールしてください。

FLASH\_DM\_ERR\_FORMAT

フォーマット失敗（ワーク領域が不正）

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

フォーマット処理を開始します。

エラーが発生した時点、またはフォーマット処理が完了した時点でコールバック関数が呼び出されます。

### Reentrant

リエントラントは不可能です。

### Example

```
e_flash_dm_status_t ret;  
  
ret = FLASH_DM_Format();  
if (FLASH_DM_ACCEPT != ret)  
{  
    /* Error */  
}  
else  
{  
    /* 初期化処理 */  
}
```

### Special Notes:

フラッシュタイプ 2,3,4 の場合、フォーマット処理ではドライバを初期化しません。フォーマット処理後、速やかにドライバの初期化処理を行ってください。



---

## R\_FLASH\_DM\_Read()

---

指定されたデータ番号のデータを読み出します。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Read(  
    st_flash_dm_info_t * p_flash_dm_info  
)
```

### Parameters

*\*p\_flash\_dm\_info*

DATFRX 情報構造体

*data\_no*

読み出し対象のデータ番号

使用可能なデータ番号は 0～ (FLASH\_DM\_CFG\_DF\_DATA\_NUM-1) です。

*\*p\_data*

読み出したデータの格納先バッファ

領域サイズは FLASH\_DM\_CFG\_DF\_SIZE\_NOx に指定した値です。

### Return Values

FLASH\_DM\_SUCCESS

正常終了

FLASH\_DM\_ERR\_ARGUMENT

パラメーターエラー

→オープン関数をコールしていない場合、オープン関数をコールしてから本関数をコールしてください。オープン関数をコールしている場合、引数の確認後、再度データ読み出し関数をコールしてください。

FLASH\_DM\_ERR\_BUSY

API 実行中またはフラッシュメモリビジー状態

→実行中の処理の終了後、再度データ読み出し関数をコールしてください。

FLASH\_DM\_ERR\_DATA\_NOT\_PRESENT

指定したデータ番号のデータがフラッシュメモリ上にない。

→データ番号の確認後、再度データ読み出し関数をコールしてください。

FLASH\_DM\_ERR\_REQUEST\_INIT

未初期化状態

→初期化関数をコールしてください。

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

データフラッシュメモリから指定したデータを読み出し、指定したバッファに格納します。

データ更新中のデータ番号を指定し、データ読み出し関数を実行した場合、データ更新中の新しいデータはまだ未成立のため、既にデータフラッシュメモリに書き込み済の古いデータを読み出します。

### Reentrant

リエントラントは不可能です。

### Example

```
st_flash_dm_info_t flash_dm_info;
static uint8_t g_test_r_buff[FLASH_DM_CFG_DF_SIZE_NO0];

flash_dm_info.data_no = 0;
flash_dm_info.p_data = &g_test_r_buff[0];
if (FLASH_DM_SUCCESS != R_FLASH_DM_Read(&flash_dm_info))
{
    /* Error */
}
```

### Special Notes:

なし

## R\_FLASH\_DM\_Write()

指定されたデータ番号のデータを更新します。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Write(  
    st_flash_dm_info_t * p_flash_dm_info  
)
```

### Parameters

*\*p\_flash\_dm\_info*

DATFRX 情報構造体

*data\_no*

データ更新対象のデータ番号

使用可能なデータ番号は 0～ (FLASH\_DM\_CFG\_DF\_DATA\_NUM-1) です。

*\*p\_data*

更新データの格納元バッファ

領域サイズは FLASH\_DM\_CFG\_DF\_SIZE\_NOx に指定した値です。

### Return Values

FLASH\_DM\_ACCEPT

処理受付

FLASH\_DM\_ERR\_REQUEST\_INIT

未初期化状態

→初期化関数をコールしてください。

FLASH\_DM\_ERR\_ARGUMENT

パラメーターエラー

→オープン関数をコールしていない場合、オープン関数をコールしてから本関数をコールしてください。オープン関数をコールしている場合、引数の確認後、再度データ更新関数をコールしてください。

FLASH\_DM\_ERR\_REQUEST\_ERASE

消去ブロックなしのため、データ更新処理実行不可能(フラッシュタイプ2,3,4)

→ブロック消去関数をコールしてください。

FLASH\_DM\_ERR\_REQUEST\_RECLAIM

アクティブブロック内に指定されたデータ番号のデータを更新するための書き込み可能領域なし(フラッシュタイプ1)

→リクレーム関数をコールしてください。

FLASH\_DM\_ERR\_BUSY

API 実行中またはフラッシュメモリビジー状態

→実行中の処理の終了後、再度データ更新関数をコールしてください。

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

指定したデータ番号のデータ更新を開始します。指定したバッファのデータをデータフラッシュメモリに書き込みます。

エラーが発生した時点、またはデータ更新が完了した時点でコールバック関数が呼び出されます。

**Reentrant**

リエントラントは不可能です。

**Example**

```
e_flash_dm_status_t ret;
uint32_t status;
st_flash_dm_info_t flash_dm_info;
static uint8_t g_test_w_buff[FLASH_DM_CFG_DF_SIZE_NO0];

flash_dm_info.data_no = 0;
flash_dm_info.p_data = &g_test_w_buff[0];
if (FLASH_DM_ACCEPT != R_FLASH_DM_Write(&flash_dm_info))
{
    /* Reclaim or error */
}
do
{
    ret = R_FLASH_DM_Control(FLASH_DM_GET_STATUS, &status);
    if (FLASH_DM_SUCCESS == ret)
    {
        if(status == FLASH_DM_ACT_IDLE)
        {
            break;
        }
    }
}while(1);
```

**Special Notes:**

プログラム時にエラーが発生した場合、データ更新エラー終了を返します。再度データ更新関数をコールしてください。書き込み先アドレスを更新し、データ更新処理を実行します。

データ更新処理が終了するまで、**p\_data** の値を変更しないでください。変更された場合、更新データは正しくない可能性があります。

---

## R\_FLASH\_DM\_Erase()

---

ブロック消去します。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Erase(  
    void  
)
```

### Parameters

なし

### Return Values

FLASH_DM_ACCEPT	処理受付
FLASH_DM_ERR_REQUEST_INIT	未初期化状態 →初期化関数をコールしてください。
FLASH_DM_NO_INVALID_BLOCK	無効ブロックなし
FLASH_DM_ERR_BUSY	API 実行中またはフラッシュメモリビジー状態 →実行中の処理の終了後、再度ブロック消去関数をコールしてください。
FLASH_DM_ERR_ERASE	ブロック消去失敗（ワーク領域が不正）

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

無効ブロックを消去し、消去ブロックを作成します。

無効ブロックがない状態で本関数をコールした場合、ブロック消去処理を実行しません。

エラーが発生した時点、またはブロック消去が完了した時点でコールバック関数が呼び出されます。

### Reentrant

リエントラントは不可能です。

## 1. Example

```
e_flash_dm_driver_status_t ret = FLASH_DM_SUCCESS;

if (FLASH_DM_ACCEPT != R_FLASH_DM_Erase())
{
    /* Error */
}
do
{
    ret = R_FLASH_DM_Control(FLASH_DM_GET_STATUS, &status);
    if (FLASH_DM_SUCCESS == ret)
    {
        if(status == FLASH_DM_ACT_IDLE)
        {
            break;
        }
    }
}
while(1);
```

## 2. Special Notes:

プログラム時またはブロックイレーズ時にエラーが発生した場合、ブロック消去エラー終了を返します。再度ブロック消去関数をコールしてください。エラーが発生した物理ブロックに対してブロック消去処理を実行します。なお、ブロック消去エラー終了が続く場合は、データフラッシュメモリの劣化の可能性があります。

## R\_FLASH\_DM\_Reclaim()

リクレーム処理を開始します。

データを更新するための容量を確保します。

本関数はフラッシュタイプ 1 専用です。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Reclaim(  
    void  
)
```

### Parameters

なし

### Return Values

FLASH_DM_ACCEPT	処理受付
FLASH_DM_ERR_REQUEST_INIT	未初期化状態 →初期化関数をコールしてください。
FLASH_DM_ERR_REQUEST_ERASE	消去ブロックなしのため、リクレーム処理実行不可能 →ブロック消去関数をコールしてください。
FLASH_DM_ERR_CANT_RECLAIM	消去ブロックおよび消去対象ブロックなしのため、リクレーム処理実行不可能 →フォーマット関数をコールしてください。
FLASH_DM_ERR_BUSY	API 実行中またはフラッシュメモリビジー状態 →実行中の処理の終了後、再度リクレーム関数をコールしてください。

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

アクティブブロックを切り替え、最も古いリクレームブロックの全有効データを新しいアクティブブロックにコピーします。コピー元のリクレームブロックをガーベジブロックに設定します。

リクレーム処理は、ブロック内の全有効データのコピー処理を含むため、完了までに時間がかかります。

### Reentrant

リエントラントは不可能です。

**Example**

```
e_flash_dm_driver_status_t ret = FLASH_DM_SUCCESS;

if (FLASH_DM_ACCEPT != R_FLASH_DM_Reclaim())
{
    /* Error */
}
do
{
    ret = R_FLASH_DM_Control(FLASH_DM_GET_STATUS, &status);
    if (FLASH_DM_SUCCESS == ret)
    {
        if(status == FLASH_DM_ACT_IDLE)
        {
            break;
        }
    }
}
while(1);
```

**Special Notes:**

リクレーン処理関数 **R\_FLASH\_DM\_Reclaim()** をコールすると、アクティブブロックの書き込み可能領域の有無にかかわらず、リクレーン処理を開始します。データ更新回数を増やすために、アクティブブロックの書き込み可能領域の無い状態<sup>1</sup>で、リクレーン関数 **R\_FLASH\_DM\_Reclaim()** をコールしてください。

リクレーン処理の正常終了時に、コピー元のリクレーンブロックをガーベジブロック→消去対象ブロックに設定します。次のリクレーン処理実行<sup>2</sup>までに、ブロック消去処理が必要です。

プログラム時にエラーが発生した場合、リクレーンエラー終了を返します。<sup>3</sup>

---

<sup>1</sup> データ更新関数 **R\_FLASH\_DM\_Write()** コール時の戻り値で判断できます。

<sup>2</sup> リクレーン関数 **R\_FLASH\_DM\_Reclaim()** コール時の戻り値で、ブロック消去が必要なタイミングを知ることができます。ユーザシステムに時間的余裕があるときに、ブロック消去処理を行うことを推奨します。

<sup>3</sup> リクレーンエラー終了時、データを再構築するために初期化処理を実行し、戻り値に従って処理を実行してください。その後、リクレーン関数 **R\_FLASH\_DM\_Reclaim()** またはデータ更新関数 **R\_FLASH\_DM\_Write()** をコールしてください。なお、データ更新関数 **R\_FLASH\_DM\_Write()** をコールすることによって、リクレーン関数 **R\_FLASH\_DM\_Reclaim()** のコール要求する戻り値を返します。



---

## R\_FLASH\_DM\_Control()

---

コントロール関数で各種機能を組み込みます。

### Format

```
e_flash_dm_status_t R_FLASH_DM_Control(  
    e_flash_dm_cmd_t cmd,  
    uint32_t* pcfg  
)
```

### Parameters

cmd

実行するコマンド

\*pcfg

コマンドに要求される設定用引数。コマンドの要求が無い場合は NULL で構いません。

### Return Values

FLASH_DM_SUCCESS	プロテクト処理正常終了
FLASH_DM_ERR_REQUEST_INIT	未初期化状態 →初期化関数をコールしてください。
FLASH_DM_ERR_ARGUMENT	パラメーターエラー →オープン関数をコールしていない場合、オープン関数をコールしてから本関数をコールしてください。オープン関数をコールしている場合、引数の確認後、再度コントロール関数をコールしてください。
FLASH_DM_ERR_BUSY	API 実行中またはフラッシュメモリビジー状態 →実行中の処理の終了後、再度コントロール関数をコールしてください。
FLASH_DM_ERR	モジュール実行エラー →再度コントロール関数をコールしてください。

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

**Description**

リード、ライト、ブロック消去以外のシーケンサの機能を組み込むための拡張機能です。コマンドのタイプによって、引数の型も異なります。

表 3-1 コントロール関数オプション

コマンド	引数	動作
FLASH_DM_GET_WRITABLE_SIZE	uint32_t*	(フラッシュタイプ 1)アクティブブロック内の書き込み可能サイズ <sup>4</sup> (単位: バイト) を取得し、引数に格納します。この値以下のユーザデータのデータ更新が可能です <sup>5</sup> 。 (フラッシュタイプ 2,3,4)消去ブロックを基に書き込み可能サイズ (単位: バイト) を取得し、引数に格納します。
FLASH_DM_GET_STATUS	uint32_t*	ユーザが実行した API が実行中かどうかの状態を取得し、引数に格納します。 0x00:FLASH_DM_ACT_IDLE(アイドル) 0x01:FLASH_DM_ACT_WRITING(データ更新処理中) 0x02:FLASH_DM_ACT_RECLAIMING(リクレーン処理中) 0x04:FLASH_DM_ACT_ERASING(ブロック消去処理中) 0x08:FLASH_DM_ACT_FORMATTING(フォーマット処理中) 0x10:FLASH_DM_ACT_INITIALIZING(初期化処理中)
FLASH_DM_GET_DATA_SIZE	uint32_t*	引数で指定したデータ番号のデータサイズを取得し、引数に格納します。
FLASH_DM_GET_DATA_NUM	uint32_t*	ユーザ設定のデータ数を取得し、引数に格納します。

**Reentrant**

本関数は再入不可です。

<sup>4</sup> データヘッダとユーザデータの境界 (セパレータ) としてプログラム単位サイズが必要です。

書き込み可能サイズは、空き領域から、このプログラム単位サイズとデータヘッダサイズを引いた値です。

<sup>5</sup> 未初期化時、フォーマット処理中、データ更新処理中またはリクレーン処理中は書き込み可能サイズの取得ができません。また、他の API 実行中には書き込み可能サイズを取得できません。

データフラッシュメモリの場合、データ更新処理に必要なデータヘッダ 7 バイト、データヘッダとユーザデータの境界用に 1 バイト、合計 8 バイト差し引いた値を返します。

**Example1:API の状態を取得する。(フラッシュタイプ 1,2,3,4)**

```
e_flash_dm_status_t ret;
uint32_t* pcfg;

ret = R_FLASH_DM_Control(FLASH_DM_GET_STATUS, pcfg);
if (FLASH_DM_SUCCESS != ret)
{
    /* Error */
}
```

**Example2:データ数を取得する。(フラッシュタイプ 1,2,3,4)**

```
e_flash_dm_status_t ret;
uint32_t* pcfg;

ret = R_FLASH_DM_Control(FLASH_DM_GET_DATA_NUM, pcfg);
if (FLASH_DM_SUCCESS != ret)
{
    /* Error */
}
```

**Special Notes:**

なし

## R\_FLASH\_DM\_GetVersion()

---

DATFRX のバージョン情報を取得します。

### Format

```
uint32_t R_FLASH_DM_GetVersion(  
    void  
)
```

### Parameters

なし

### Return Values

上位 2 バイト:	メジャーバージョン (10 進表示)
下位 2 バイト:	マイナーバージョン (10 進表示)

### Properties

r\_flash\_dm\_rx\_if.h にプロトタイプ宣言されています。

### Description

バージョン情報を返します。

### Reentrant

リエントラントは可能です。

### Example

```
uint32_t version;  
version = R_FLASH_DM_GetVersion();
```

### Special Notes:

なし

#### 4. 端子設定

DATFRX は端子を使用しないため、端子設定は不要です。

## 5. デモプロジェクト

FITDemos フォルダに、以下のプロジェクトを格納しています。また、サンプルプログラム r\_datfrx\_rx\_main.c を格納しています。

表 5-1 プロジェクトリスト

ユーザ条件					プロジェクト
使用 MCU	フラッシュ タイプ	フラッシュ メモリ	ブロック サイズ	IDE	
RX66T	フラッシュ タイプ 3	データフラッシュ	64 バイト	e2 studio	type3_rx66t_rsk_sample
RX65N-2MB	フラッシュ タイプ 4	データフラッシュ	64 バイト	e2 studio	type4_rx65n_2mb_rsk_sample

サンプルプログラムは、オープン、初期化処理後、データ更新、データ読み出し、ベリファイチェックを 50 回繰り返して、クローズします。

データ更新、データ読み出し、ベリファイチェックが 1 セット終了すると、データ番号を +1 します。  
(デモでは、データ番号 0～データ番号 13 を昇順に使用します)

初期化処理時、フォーマットが必要な場合はフォーマットを行います。

データ更新時、ブロック消去が必要な場合はブロック消去を行います。

### 5.1 ワークスペースにデモを追加する

デモプロジェクトは、本アプリケーションノートで提供されるファイルの FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」→「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し、「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「完了」をクリックします。

### 5.2 デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

## 6. 付録

### 6.1 動作確認環境

表 6-1 に動作確認環境を示します。

表 6-1 動作確認環境

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e <sup>2</sup> studio V7.2.0 ルネサス エレクトロニクス製 CS+ V8.0.0
C コンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.3.00.00
	コンパイル・オプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのバージョン	Ver.2.01
使用ボード	Renesas Starter Kit for RX231(型名：R0K505231xxxxxx) Renesas Starter Kit for RX210(型名：R0K505210xxxxxx) Renesas Starter Kit for RX66T(型名：RTK50566T0Cxxxxxxx) Renesas Starter Kit+ for RX65N-2MB (型名：RTK50565N2S1xxxxxx)

### 6.2 トラブルシューティング

- (1) Q : 本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A : FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合

アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」

- e<sup>2</sup> studio を使用している場合

アプリケーションノート RX ファミリ e<sup>2</sup> studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)」を参照してください。

- (2) Q : 本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current r\_datfrx\_rx module.」エラーが発生します。

A : 追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。



## 6.3 データ管理

DATFRX のデータ管理領域を記載します。

### 6.3.1 DATFRX 領域

DATFRX が管理するブロック領域を以下に示します。

なお、以下枠内の読み出し用アドレスはユーザーズマニュアル・ハードウェア編の表記に合わせています。ブロック A～ブロック H は DATFRX での管理番号です。

#### 6.3.1.1 1 ブロック 1024 バイト（フラッシュタイプ 1）

表 6-2 1 ブロック 1024(フラッシュタイプ 1)

1 ブロック=1,024 バイト 読み出し用アドレス	フラッシュ FIT モジュール 管理ブロック	DATFRX 管理ブロック
0x0010 0000 0x0010 03FF	DB0000	ブロック A（先頭ブロック）
0x0010 0400 0x0010 07FF	DB0001	ブロック B
0x0010 0800 0x0010 0BFF	DB0002	ブロック C
0x0010 0C00 0x0010 0FFF	DB0003	ブロック D
0x0010 1000 0x0010 13FF	DB0004	ブロック E
0x0010 1400 0x0010 17FF	DB0005	ブロック F
0x0010 1800 0x0010 1BFF	DB0006	ブロック G
0x0010 1C00 0x0010 1FFF	DB0007	ブロック H（最終ブロック）

## 6.3.1.2 1 ブロック 128 バイト (フラッシュタイプ 2)

表 6-3 1 ブロック 128 バイト(フラッシュタイプ 2)

1 ブロック=128 バイト 読み出し用アドレス	フラッシュ FIT モジュール 管理ブロック	DATFRX 管理ブロック
0x0010 0000 0x0010 007F	ブロック 0(128 バイト)	ブロック A
0x0010 0080 0x0010 00FF	ブロック 1(128 バイト)	ブロック B
・ ・	・ ・	・ ・
0x0010 7F80 0x0010 1FFF	ブロック 63(128 バイト)	ブロック Z

## 6.3.1.3 1 ブロック 64 バイト (フラッシュタイプ 3、4)

表 6-4 1 ブロック 64 バイト(フラッシュタイプ 3,4)

1 ブロック=64 バイト 読み出し用アドレス	フラッシュ FIT モジュール 管理ブロック	フラッシュ FIT モジュール 管理ブロック	DATFRX 管理ブロック
0x0010 0000 0x0010 003F	ブロック 0 (64 バイト)	ブロック 0 (64 バイト)	ブロック A
0x0010 0040 0x0010 007F	ブロック 1 (64 バイト)	ブロック 1 (64 バイト)	ブロック B
・ ・	・ ・	・ ・	・ ・
0x0010 7FC0 0x0010 7FFF	・ ・	ブロック 511 (64 バイト)	ブロック Y
0x0010 FFC0 0x0010 FFFF	ブロック 1023 (64 バイト)	-	ブロック Z

## 6.3.1.4 1 ブロック 32 バイト (フラッシュタイプ 2)

表 6-5 1 ブロック 32 バイト(フラッシュタイプ 2)

1 ブロック=32 バイト 読み出し用アドレス	フラッシュ FIT モジュール 管理ブロック	DATFRX 管理ブロック
0x0010 0000 0x0010 001F	ブロック 0(32 バイト)	ブロック A
0x0010 0020 0x0010 003F	ブロック 1(32 バイト)	ブロック B
・ ・	・ ・	・ ・
0x0010 7FD0 0x0010 7FE0	ブロック 1023(32 バイト)	ブロック Z

### 6.3.2 ブロック領域【フラッシュタイプ 1】

ブロックフォーマットを以下に示します。ブロック内は、ブロックヘッダ領域、データヘッダ領域とユーザデータ領域に分けられ、データヘッダ領域とユーザデータ領域の間は空き領域となります。

#### 6.3.2.1 ブロックヘッダ（フラッシュタイプ 1）

ブロックヘッダ領域はブロックを管理します。

消去開始フラグ、消去完了フラグ、初期化済みフラグ等があります。

ブロックをブロックヘッダ領域で管理します。

ブロックヘッダ領域に下記フラグを記録し、初期化処理時にブロックのフラグ状態を確認して、ブロックの種類を判定します。

表 6-6 データ(フラッシュタイプ 1)

フラグ名称	処理	コマンド 1	コマンド 2
消去開始フラグ	ブロック消去処理	イレースコマンド実行前 0x00	イレースコマンド成功 0xFF
消去完了フラグ	ブロック消去処理	ブロックイレース完了 0xAA	-
初期化済みフラグ	ブロック消去処理	初期化済みブロック作成完了 0x00	-
アクティブフラグ	リクレーム処理	アクティブフラグの切り替え完了 0x00	-
フルフラグ	リクレーム処理	リクレーム処理開始 0x00	-
リクレームフラグ	リクレーム処理	リクレーム処理完了	-

## 6.3.2.2 データヘッダ（フラッシュタイプ1）

データヘッダ領域はユーザデータを管理します。

データ更新要求を受けるごとにデータヘッダを1個作ります。

同じデータ番号のデータ更新要求を受けた時、同じデータ番号を持つ新たなデータヘッダをアクティブブロック内に作成します、アドレスが大きい方のデータヘッダを最新（有効）と判定します。

初期化処理時、リクレームブロックとアクティブブロックのデータヘッダをチェックし、ブロックヘッダの直後からアドレスが増える方向に、データヘッダをプログラムします。

以下にデータヘッダフォーマットを示します。

表 6-7 データフラッシュメモリ(フラッシュタイプ1)

(1 ブロック : 1024 バイト)

: プログラム単位 1 バイト

オフセット	内容	サイズ
0x000	データヘッダ更新開始フラグ	1 バイト
0x001	データ番号	1 バイト
0x002	データアドレス（下位） <sup>6</sup>	1 バイト
0x003	データアドレス（上位）6	1 バイト
0x004	データヘッダ更新完了フラグ	1 バイト
0x005	データ更新完了フラグ	1 バイト
0x006	有効フラグ	1 バイト

<sup>6</sup> リトルエンディアン時のデータ配置です。ビッグエンディアン時は、データアドレス（下位）とデータアドレス（上位）のデータ配置が逆になります。

表 6-8 フラグの種類

フラグ名称	処理
データヘッダ更新開始フラグ	データヘッダ書き込み開始時に 0x7F をプログラムします。データヘッダの存在を示します。 初期化処理時にこのフラグをチェックし、0x7F,0xFF 以外であれば無効なデータヘッダと判定します。
データ番号	r_datfrx_config.h で、ユーザが設定したデータ番号です。データ番号ごとのデータサイズを設定できます。設定方法は「2.7 コンパイル時の設定」を参照してください。
データアドレス	ユーザデータが格納されている実アドレスです。
データヘッダ更新完了フラグ	データ番号とデータアドレスをプログラムした後に、0xBF をプログラムします。データ番号とデータアドレスのプログラム完了を示します。 初期化処理時にこのフラグをチェックし、0xBF 以外の場合、データヘッダのプログラムが未完了と判断し、無効なデータヘッダとして扱います。
データ更新完了フラグ	ユーザデータをプログラムした後に、0xDF をプログラムします。ユーザデータのプログラム完了を示します。 初期化処理時にこのフラグをチェックし、0xDF 以外の場合、ユーザデータのプログラムが未完了と判断し、無効なデータヘッダとして扱います。
有効フラグ	データ更新完了フラグをプログラムした後に、0x0F をプログラムします。データヘッダの有効性を示します。 初期化処理時にこのフラグをチェックし、0x0F 以外の場合、データ更新処理が終了していないと判断し、無効なデータヘッダとして扱います。

## 6.3.2.3 データ（フラッシュタイプ 1）

表 6-9 データフラッシュメモリ(フラッシュタイプ 1)

(1 ブロック : 1,024 バイト) :  
プログラム単位 1 バイト

オフセット	内容	サイズ	領域
0x000	消去開始フラグ	2 バイト	ブロックヘッダ 領域
0x002	消去完了フラグ	8 バイト	
0x00A	初期化済みフラグ	2 バイト	
0x00C	アクティブフラグ	2 バイト	
0x00E	フルフラグ	2 バイト	
0x010	リクレームフラグ	2 バイト	データヘッダ 領域
0x012	データヘッダ(a)	7 バイト	
0x019	データヘッダ(b)	7 バイト	
0x020	データヘッダ(c)	7 バイト	
:	:		
:	データヘッダ(n)	7 バイト	空き領域
	↓		
	↑		
0x400 - Size(a..n)	ユーザデータ (n)	Size(n)	ユーザデータ 領域
:	:		
0x400 - Size(a..c)	ユーザデータ (c)	Size(c)	
0x400 - Size(a..b)	ユーザデータ (b)	Size(b)	
0x400 - Size(a)	ユーザデータ (a)	Size(a)	

Size(a) : ユーザデータ a のデータサイズ

Size(a..n) : ユーザデータ a~ユーザデータ n までのデータサイズの合計

## 6.3.3 ブロック管理【フラッシュタイプ 2、3、4】

## 6.3.3.1 ブロックヘッダ（フラッシュタイプ 2、3、4）

フラッシュタイプ 2、3、4 にブロックヘッダはありません。

## 6.3.3.2 データヘッダ（フラッシュタイプ 2、3、4）

初期化処理時にブロックそのものの状態を確認し、ブロックの種類を判定します。

ブロック内の管理情報を以下に纏めます。

表 6-10 データフラッシュメモリ(フラッシュタイプ 2、3、4)

フラグ名称	処理	コマンド
data_type	データの type を判別する。	data_type の値 <sup>7</sup> 1: Long format の最初のブロック 2: Long format の中間のブロック 4: Long format の最終ブロック 8: Short format のブロック  Format1 では、data_type=1、又は data_type=8 である。 Format2 では、data_type=2、又は data_type=4 である。
chain	1 ブロック内に収まりきらないユーザデータに関し、次のユーザデータを保管しているブロック番号を抽出する。	Long format の場合、ユーザデータを保管している次のブロック番号情報を格納する。 data_type=4 および data_type=8 は、ユーザデータを保管する次のブロック番号が無い場合、0xFFFF を格納する。
data_No	保管しているユーザデータのデータ番号を抽出する。	データ番号<データ数で設定可能。
ser_No	ユーザデータの新旧を判別する。	データ更新ごとに+1 更新するシリアル番号 Pass/Fail に関係なく更新する。 更新回数：0xFFFFFFFF 回 MAX 0xFFFFFFFF 回はソフトウェア上の更新可能回数であり、データフラッシュメモリのデータ更新可能回数ではありません。
crc_ccitt	格納している管理情報が正しいか判断する。	以下の管理情報に関し、CRC コードを生成する。 (1) data_No (2) ser_No
write_end	データ更新が終了したことを確認する	データ更新処理で、最後に本フラグをプログラムする。 0x0000 のデータをプログラムする。 データ更新処理が正常に終了した場合、本フラグは未消去状態となる。
erase_start	ブロック消去処理が異常終了したか、判別する。	イレーズする前に本フラグをプログラムする。 0x0000 のデータをプログラムする。 本フラグが未消去状態の場合、ブロック消去処理が異常終了したと判断する。

<sup>7</sup> Short format は 1 つのデータが 1 ブロックの中に納まるユーザデータに適用され、Format1 で構成されます。Long format は 1 つのデータを複数のブロックに跨いで保管する形態です（データが 1 つのブロックに収まりきれない場合に使用します）。Format1 と Format2 で構成されます。

		ブロック消去処理が正常に終了した場合、本フラグは消去状態となる。
erase_end	ブロック消去処理が終了したことを確認する	ブロック消去処理で、最後に本フラグをプログラムする。 0x0000 のデータをプログラムする。 ブロック消去処理が正常に終了した場合、本フラグは未消去状態となる。



## 6.3.3.3 データ（フラッシュタイプ 2、3、4）

Short format は 1 つのデータが 1 ブロックの中に納まるユーザデータに適用され、Format1 で構成されます。Long format は 1 つのデータを複数のブロックに跨いで保管する形態です（データが 1 つのブロックに収まりきれない場合に使用します）。Format1 と Format2 で構成されます。

## ● Format1

Format1 は 1 つのデータを保管するブロックの先頭ブロックに適用します。

表 6-11 Format1

フラッシュタイプ 2				フラッシュタイプ 3		フラッシュタイプ 4		データ	記号
128 バイト		32 バイト		64 バイト		64 バイト			
アドレス	サイズ	アドレス	サイズ	アドレス	サイズ	アドレス	サイズ		
0x00	1	0x00	1	0x00	1	0x00	1	データタイプ	data_type
0x01	2	0x01	2	0x01	2	0x01	2	チェーン情報	chain
0x03	2	0x03	2	0x03	2	0x03	2	データ番号	data_No
0x05	4	0x05	4	0x05	4	0x05	4	シリアル番号	ser_No
0x09	2	0x09	2	0x09	2	0x09	2	CRC 符号	crc_ccitt
0x0B	111	0x0B	15	0x0B	41	0x0B	41	ユーザデータ	data
0x7A	2	0x1A	2	0x34	4	0x34	4	書き込み終了フラグ	write_end
0x7C	2	0x1C	2	0x38	4	0x38	4	消去開始フラグ	erase_start
0x7E	2	0x1E	2	0x3C	4	0x3C	4	消去終了フラグ	erase_end

## ● Format2

Format2 は 1 つのデータを複数ブロックに跨って保管する場合（Long format）、2 番目以降のブロックに適用します。

表 6-12 Format2

フラッシュタイプ 2				フラッシュタイプ 3		フラッシュタイプ 4		データ	記号
128 バイト		32 バイト		64 バイト		64 バイト			
アドレス	サイズ	アドレス	サイズ	アドレス	サイズ	アドレス	サイズ		
0x00	1	0x00	1	0x00	1	0x00	1	データタイプ	data_type
0x01	2	0x01	2	0x01	2	0x01	2	チェーン情報	chain
0x03	121	0x03	25	0x03	53	0x03	53	ユーザデータ	data
0x78	2	0x18	2	0x38	4	0x38	4	消去開始フラグ	erase_start
0x7C	2	0x1C	2	0x3C	4	0x3C	4	消去終了フラグ	erase_end

## 6.3.4 ブロックの状態と判定

## 6.3.4.1 フラッシュタイプ 1

## (1) ブロックの状態（フラッシュタイプ 1）

DATFRX 管理下（フラッシュタイプ 1）の物理ブロックは、以下のいずれかの状態に分類されます。

表 6-13 ブロックの状態と判定

ブロック状態	内容	判定
消去対象	ブロック消去対象に設定されたブロックで、消去が必要なブロックです。 ブロック消去処理は、消去対象ブロックを初期化済ブロックに切り替えます。	消去開始フラグが 0xFF でない場合、消去対象ブロックと判定します。 消去完了フラグの各バイトデータが 1 つでも 0xAA でない場合、消去対象ブロックと判定します。
初期化済み	ユーザデータが全く書かれていないブロックで、初期化済ブロックが 1 個必要で、使用可能なブロックです。 フォーマット処理は、ブロック B を初期化済みブロックに設定します。また、ブロック消去処理は、初期化済みブロックを作成します。	消去済みブロックでない、かつアクティブフラグが 0xFF の場合、初期化済みブロックと判定します。
アクティブ	ユーザデータを書き込むブロックで、アクティブブロックが 1 個必要で、データ更新対象のブロックです。 フォーマット処理は、ブロック A をアクティブブロックに設定します。また、リクレーン（アクティブブロックの切り替えと有効データのコピー）処理は、アクティブブロックをフルブロックに切り替えます。その後、初期化済みブロックをアクティブブロックに切り替えます。 したがって、フォーマット後アクティブブロックは、ブロック A→ブロック B→ブロック C→ブロック D→ブロック E→ブロック F→ブロック G→ブロック H→ブロック A の順に切り替わります。	初期化済みブロックでない、かつフルフラグが 0xFF の場合、アクティブブロックと判定します。 アクティブブロックが複数個あると判定された場合、異常状態と判断します。異常解消方法として、フォーマット処理が必要です。
フル	データ更新可能領域がなくなったアクティブブロックで、リクレーン処理期間中にのみ設定されます。リクレーン処理終了時には、フルブロックをリクレーンブロックに切り替えます。	アクティブブロックでない、かつリクレーンフラグが 0xFF の場合、フルブロックと判定します。 フルブロックが複数個あると判定された場合、異常状態と判断します。異常解消方法として、フォーマット処理が必要です。
リクレーン	データ更新可能領域がなくなった読み出し専用のブロックです。リクレーン処理は、最も古いリクレーンブロックをコピー元ブロックとし、有効データを切り替わった新アクティブブロックにコピーします。 リクレーン処理終了時には、コピー元ブロックをガーベージブロックに設定します。	どのブロックの判定に当てはまらない場合、リクレーンブロックと判定します。
ガーベージ	リクレーン処理により有効なデータがなくなったコピー元ブロック、または初期化処理でブロックヘッダ異常と判定されたブロックです。	アクティブブロックの次のブロックが初期化済みブロックでない場合、そのブ

	ブロック消去処理は、ガーベージブロックを消去対象ブロックとして扱います。	ロックをガーベージブロックに設定します。 フルブロックが1個とアクティブブロックが1個ある場合、そのアクティブブロックをガーベージブロックに設定します。
消去済み	ブロックイレースコマンドが完了し、消去完了フラグのみがプログラムされたブロックです。ブロック消去処理の途中で処理が中断された場合、初期化処理によりブロックヘッダ異常と判定されたブロックとして扱われます。	消去対象ブロックでない、かつ初期化済みフラグが 0xFF の場合、消去済みブロックと判定します。

## (2) ブロックの状態遷移 (フラッシュタイプ 1)

DATFRX 管理下 (フラッシュタイプ 1) の物理ブロックは、以下の状態を遷移します。

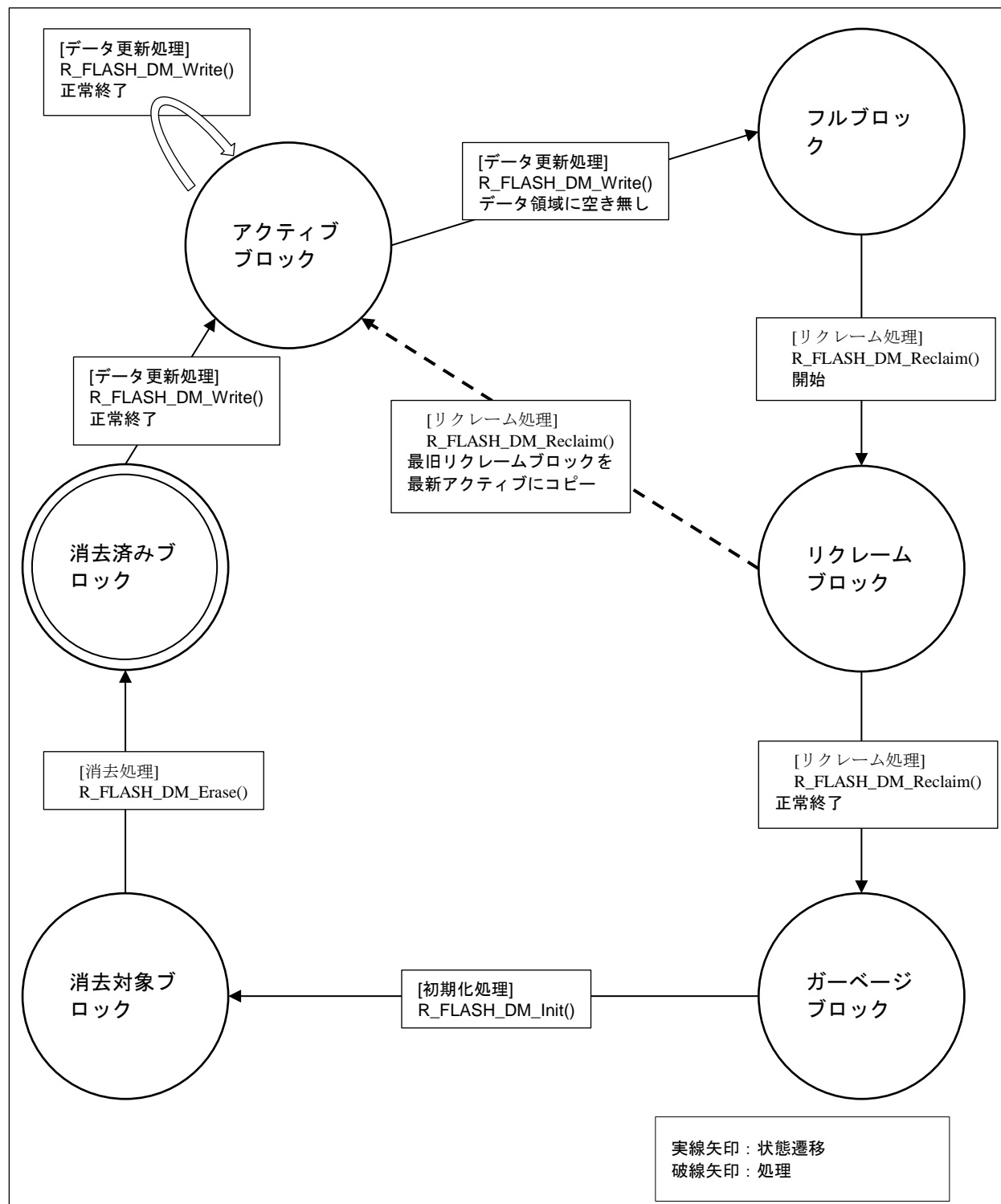


図 6-1 ブロックの状態遷移(フラッシュタイプ 1)

## (3) ブロック判定フロー（フラッシュタイプ 1）

DATFRX 管理下（フラッシュタイプ 1）の物理ブロックは、以下のような判定により状態が決まります。

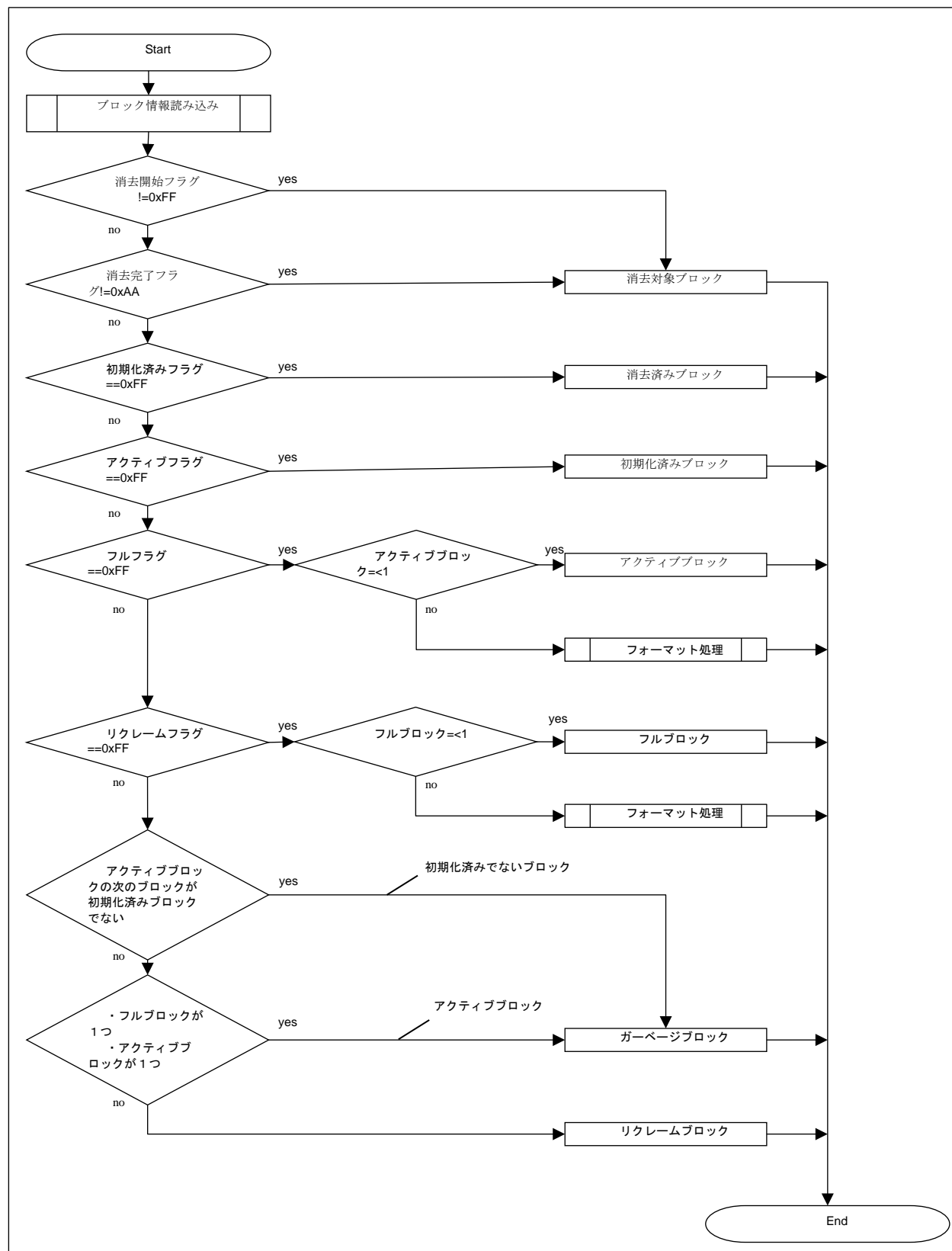


図 6-2 ブロック判定フロー(フラッシュタイプ 1)

## 6.3.4.2 フラッシュタイプ 2、3、4

## (1) ブロックの状態（フラッシュタイプ 2、3、4）

DATFRX 管理下（フラッシュタイプ 2、3、4）の物理ブロックは、以下のいずれかのブロック状態に分類されます。

フラグが判定を満たすとき、ブロック状態が決定します。

表 6-14 ブロックの状態(フラッシュタイプ 2、3、4)

ブロック状態	意味	フラグ	ブロック状態を判定するための、フラグの状態
有効ブロック	有効なユーザデータを保管しているブロックです。 <sup>8</sup>	data_type	未消去状態であること、仕様通りの値であること。
		chain	未消去状態であること、ブロック番号がブロック数未満の値であること。
		data_No	未消去状態であること、データ番号が登録されたデータ数未満の値であること。
		ser_No	未消去状態であること、同一データ番号のブロックの中で、最大の番号であること。
		crc_ccitt	未消去状態であること、data_No および ser_No から生成された符号と一致すること。
		data	ブロックに格納されているデータの範囲で未消去状態であること、その他の部分は消去状態であること。
		write_end	未消去状態であること。
		erase_start	消去状態であること。
		erase_end	未消去状態であること。
消去ブロック	データが正常に消去されているブロックです。データ更新処理は消去ブロックに対し行います。	data_type	消去状態であること。
		chain	消去状態であること。
		data_No	消去状態であること。
		ser_No	消去状態であること。
		crc_ccitt	消去状態であること。
		data	消去状態であること。
		write_end	消去状態であること。
		erase_start	消去状態であること。
		erase_end	未消去状態であること。
無効ブロック	その他のブロックです。ブロック消去処理は、無効ブロックに対し行います。	有効ブロック又は消去ブロックに判定されなかったブロックを、無効ブロックと判定します。	

<sup>8</sup> Format2 では先頭ブロックが有効ブロックの場合、その中間ブロック、最終ブロックを、有効ブロックと判定する。

(2) ブロックの状態遷移（フラッシュタイプ 2、3、4）

DATFRX 管理下（フラッシュタイプ 2、3、4）の物理ブロックは、以下の状態を遷移します。

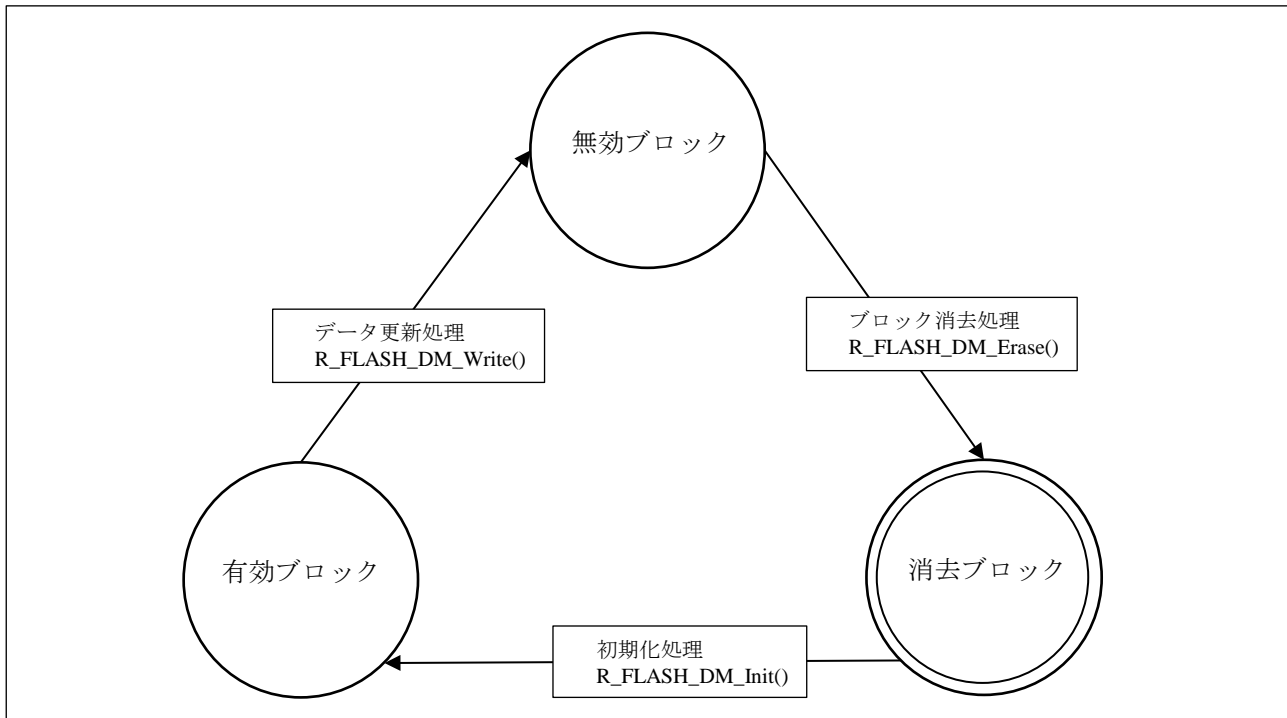


図 6-3 ブロックの状態遷移(フラッシュタイプ 2、3、4)

## (3) ブロック判定フロー（フラッシュタイプ 2、3、4）

DATFRX 管理下（フラッシュタイプ 2、3、4）の物理ブロックは、以下のような判定により状態が決まります。

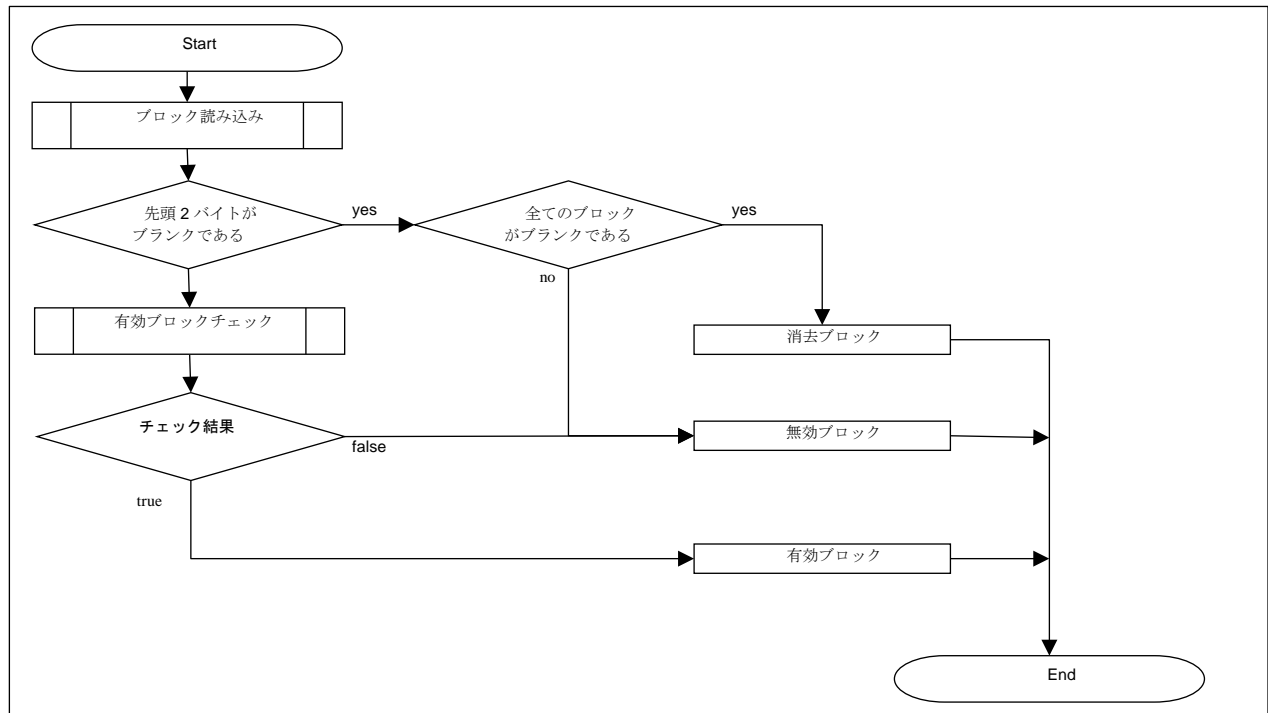


図 6-4 ブロック判定フロー(フラッシュタイプ 2、3、4)



## 7. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

(最新版をルネサス エレクトロニクスホームページから入手してください。)

•

テクニカルアップデート／テクニカルニュース

(最新の情報をルネサス エレクトロニクスホームページから入手してください。)

•

ユーザーズマニュアル：開発環境

RX ファミリ CC-RX コンパイラ ユーザーズマニュアル (R20UT3248)

(最新版をルネサス エレクトロニクスホームページから入手してください。)

## テクニカルアップデートの対応について

本モジュールは以下のテクニカルアップデートの内容を反映しています。

なし

## ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

<http://japan.renesas.com>

お問合せ先

<http://japan.renesas.com/contact/>

すべての商標および登録商標は、それぞれの所有者に帰属します。

## 改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2018.09.28	—	初版発行
2.01	2019.02.01	—	英語版 PDF を追加 1.4 処理例を追加

## 製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

### 1. 静電気対策

CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

### 2. 電源投入時の処置

電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

### 3. 電源オフ時における入力信号

当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

### 4. 未使用端子の処理

未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

### 5. クロックについて

リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

### 6. 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、 $V_{IL}$  (Max.) から  $V_{IH}$  (Min.) までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、 $V_{IL}$  (Max.) から  $V_{IH}$  (Min.) までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

### 7. リザーブアドレス（予約領域）のアクセス禁止

リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

### 8. 製品間の相違について

型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違くと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

## ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

高品質水準： 輸送機器（自動車、電車、船舶等）、交通管制（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。

6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものいたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

## 本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

[www.renesas.com](http://www.renesas.com)

## お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

[www.renesas.com/contact/](http://www.renesas.com/contact/)

## 商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。