User's Manual

32

RH850 ファミリ

データフラッシュライブラリ Type01

対象インストーラ名：RENESAS_FDL_RH850_T01_Vx.xx※

※ x.xx ： 2.01以上

32 ビット・シングルチップ・マイクロコントローラ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.03　2019.05

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）、交通制御（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て、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といたします。
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

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/

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。

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うと、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、ノイズ幅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。

このマニュアルの使い方

対 象 者　このユーザーズマニュアルは、RH850ファミリのデータフラッシュライブラリ Type01の機能を理解し、それを用いたアプリケーションシステムを設計するユーザを対象としています。

目　　的　このユーザーズマニュアルは、RH850ファミリのデータフラッシュの書き換えを行うために使用するRH850 データフラッシュライブラリ Type01の使用方法を理解していただくことを目的としています。

構　　成　このマニュアルは、大きく分けて次の内容で構成しています。

・はじめに
・アーキテクチャ
・機能仕様
・ユーザインタフェース（API）
・FDLの設定方法と使用方法
・注意事項

読 み 方　このマニュアルを読むにあたっては、電気、論理回路、マイクロコントローラの一般知識を必要とします。

□一通りの機能を理解しようとするとき
→目次に従って読んでください。
□関数の機能の詳細を知りたいとき
→このユーザーズ·マニュアルの**第4章　ユーザインタフェース（API）**を参照してください。

なお、初版以降において本文欄外の★印は、本版で改訂された主な箇所を示しています。

凡　　例

データ表記の重み	：左が上位桁、右が下位桁
アクティブ・ロウの表記	：$\overline{\times\times\times}$（端子、信号名称に上線）
注	：本文中につけた注の説明
注意	：気をつけて読んでいただきたい内容
備考	：本文の補足説明
数の表記	：2進数…××××または××××B
	10進数…××××
	16進数…××××Hまたは0x××××

目次

第1章　はじめに

本ユーザマニュアルでは、RH850 ファミリ用データフラッシュライブラリ Type 01（以降 FDL と呼称する）の機能、アプリケーションプログラミングインタフェース（API）について説明します。

備考：

- 本 FDL の対象デバイス以外には本 FDL を使用しないでください。あらゆる動作不良につながる恐れがあります。本 FDL はソースコードで供給されます。意図しない動作不良やプログラミング不良を生じる恐れがあるので、ご採用の際には細心の注意を払い、十分に動作確認し、ご使用ください。
- 本 FDL は、GHS 社製コンパイラ用 FDL とルネサスエレクトロニクス社製コンパイラ用 FDL が含まれます。コンパイラとアセンブラの機能は多様で、特にアセンブラファイルは環境によって異なります。したがって、本 FDL とサンプルアプリケーションは、適切な環境を選択できるようにインストーラツールを使用して提供されます。
- 本 FDL は、デバイス依存のサンプルアプリケーションと一緒に提供されます。
- 本 FDL は無償製品であり、サンプルプログラムとしてソースコード形態で提供しています。
- 本 FDL、本ユーザマニュアルは、常に最新バージョンのご使用を推奨します。なお、対応デバイスにつきましては本 FDL パッケージに含まれている support.txt にてご確認ください。
- 本ユーザマニュアルのすべての章をよくお読みください。本 FDL の誤使用に起因する誤動作を回避するために、本ユーザーズマニュアルが定める手順と推奨事項に従ってください。また、本ユーザーズマニュアルは、必ず、本 FDL パッケージに添付されているリリースノート、および対象デバイスのユーザーズマニュアルと合わせてご使用ください。

フラッシュメモリの基本構造

RH850 ファミリの多くには、コードフラッシュに加えてデータフラッシュという独立したフラッシュメモリ領域が搭載されています。このフラッシュメモリ領域はデータ格納用に設計されています。命令実行（コードフェッチ）に用いることはできません。

- デバイスによってはデータフラッシュに関する仕様（データフラッシュのサイズなど）が異なる場合や注意事項がある場合があります。

例：RH850/F1L データフラッシュ32Kバイト搭載 1ブロックサイズ64バイト		FDL操作アドレス
ブロック番号	アドレス	
511	0xFF20_7FC0 ～ 0xFF20_7FFF	0x0000_7FC0 ～ 0x0000_7FFF
510	0xFF20_7F80 ～ 0xFF20_7FBF	0x0000_7F80 ～ 0x0000_7FBF
509	0xFF20_7F40 ～ 0xFF20_7F7F	0x0000_7F40 ～ 0x0000_7F7F
・ ・ ・	・ ・ ・	・ ・ ・
5	0xFF20_0140 ～ 0xFF20_017F	0x0000_0140 ～ 0x0000_017F
4	0xFF20_0100 ～ 0xFF20_013F	0x0000_0100 ～ 0x0000_013F
3	0xFF20_00C0 ～ 0xFF20_00FF	0x0000_00C0 ～ 0x0000_00FF
2	0xFF20_0080 ～ 0xFF20_00BF	0x0000_0080 ～ 0x0000_00BF
1	0xFF20_0040 ～ 0xFF20_007F	0x0000_0040 ～ 0x0000_007F
0	0xFF20_0000 ～ 0xFF20_003F	0x0000_0000 ～ 0x0000_003F

データフラッシュの動作単位

RH850 ファミリのデータフラッシュは 64 バイトのブロック構造になっており、このブロックが FDL における消去操作単位です。必ず、消去はブロック単位、書き込みは 1 ワード（4 バイト）単位で実行されます。また、消去されたデータフラッシュ領域を読み出しますと、不定値が読み出されます。

なお、対象デバイスのデータフラッシュのハードウェア（データフラッシュのサイズなど）は、必ず対象デバイスのユーザーズマニュアルで仕様、注意事項をご確認ください。

第2章　アーキテクチャ

本章では、ユーザが FDL を使用してデータフラッシュの書き換えを行う上で、必要な FDL のアーキテクチャについて説明します。

2. 1　階層アーキテクチャ

本章では、データフラッシュ書き換えシステムに含まれるすべてのブロックの機能について説明します。このマニュアルは FDL について説明するものですが、関係するすべての機能ブロックとそれらの関係を簡潔に説明します。

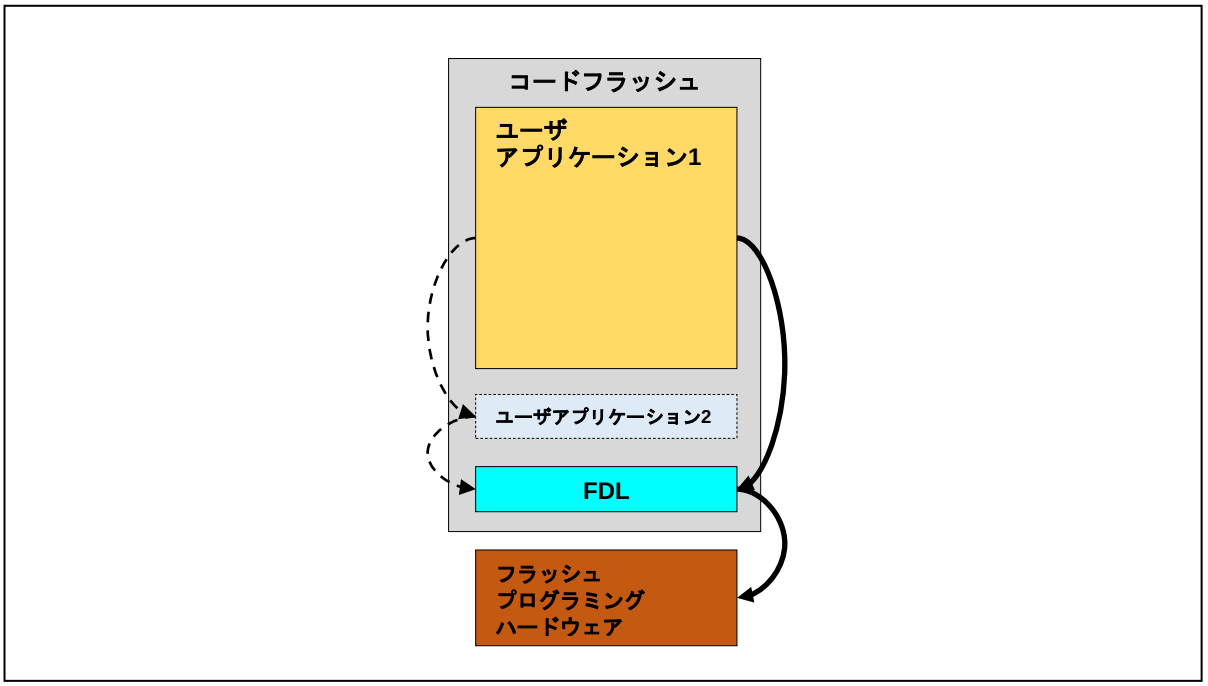


図 1　抽象化したデータフラッシュ書き換えシステムの機能ブロック間の関係

上図に示すように、データフラッシュ書き換えシステムのソフトウェアアーキテクチャは以下の数個の機能ブロックで構成されています。

- **ユーザアプリケーション 1 :**

  この機能ブロックは、直接 FDL が提供する関数を使用します。

  また、拡張機能的な使用方法として、直接 FDL を制御せず、ユーザアプリケーション 2※を経由して FDL を操作する方法もあります。

  ※ユーザアプリケーション 2 はユーザにて作成します。

- **データフラッシュライブラリ（FDL）：**

  データフラッシュ領域へのアクセスインタフェース、いわゆる「FDL プール」（「2. 2　FDL プールの定義」参照）を提供します。FDL は、書き込み、消去、ブランクチェックコマンドなどの機能を提供します。

- **フラッシュプログラミングハードウェア：**

  この機能ブロックは、FDL により制御されるフラッシュプログラミングハードウェア(フラッシュシーケンサ)を表します。

- **ユーザアプリケーション 2：**

  この機能ブロックは、単純な書き換え機能として提供している FDL にユーザが機能を追加するユーザアプリケーションです。

  機能拡張例）書き込むデータのチェックサムや長さの管理など

  このユーザアプリケーション 2 は任意でユーザによって開発される機能ブロックで、使用しない場合は不要です。

2. 2　FDLプールの定義

FDLプールとはFDLが管理するデータフラッシュ領域です。データフラッシュへのアクセスを必ずFDLを使用するというシステムを構築する場合、FDLプールのサイズは対象デバイスに搭載されているデータフラッシュのサイズ（ブロック数）を設定してください。データフラッシュにFDLプールとして設定されていない領域がある場合、そのFDLプール未設定領域にはFDLからアクセスすることはできません。また、標準的な使用方法としてFDLプールのすべての領域はユーザアプリケーション1※から直接アクセスできるユーザプールとして扱います。ただし、拡張機能としてFDLプールをユーザアプリケーション1※からアクセスできないEELプールに分けることも可能です。設定方法につきましては「4. 2　ランタイム設定」をご参照ください。

以下が各プールの説明です。
・FDL プール：FDL が管理するデータフラッシュ領域（データフラッシュ全領域）
・ユーザプール：ユーザアプリケーション 1 のみが FDL を介してアクセス可能な領域

≪拡張機能用≫
・EEL プール：ユーザアプリケーション 2※のみが FDL を介してアクセス可能な領域

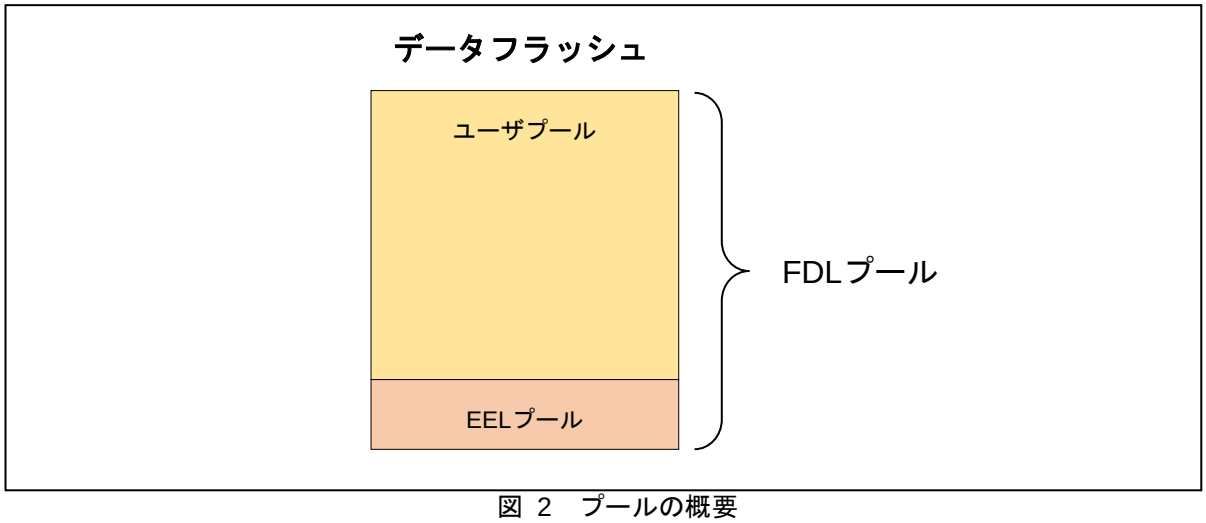


図 2　プールの概要

※　「2. 1　階層アーキテクチャ」参照

2. 3　アーキテクチャ関連の注意事項

- FDL を使用する場合、データフラッシュ関連の動作はすべて FDL によって実行されます。したがって、ユーザアプリケーションはデータフラッシュへ直接アクセス（消去、書き込みなど）できません。例外はデータフラッシュの読み出しです。データフラッシュが CPU のアドレス空間にマップされている為、CPU から直接読み出すことができます。
- FDL はデータフラッシュにのみアクセス可能です。
- FDL と FCL の同時実行は禁止です。

第3章　機能仕様

この章では、ユーザがデータフラッシュの書き換えを行う上で必要なFDLの機能について説明します。

3. 1　FDL関数とコマンド一覧

下表に、FDLが提供しているFDL関数を示します。

表 1　FDL関数

FDL関数名	機能概要
R_FDL_Init	FDLの初期化
R_FDL_Execute	各コマンドの実行開始
R_FDL_Handler	実行中のFDLを制御
R_FDL_SuspendRequest	FDLへのサスペンド要求
R_FDL_ResumeRequest	FDLのサスペンド状態からの復帰要求
R_FDL_CancelRequest	FDLへのキャンセル要求
R_FDL_StandBy	FDLのスタンバイ
R_FDL_WakeUp	FDLのスタンバイ状態からの復帰
R_FDL_GetVersionString	FDLのバージョン情報の取得

コマンドは R_FDL_Execute を使って起動し､その後は R_FDL_Handler を実行することにより進捗を制御します。下表のコマンドを使用して、下表のデータフラッシュ操作を実行することができます。

表 2　FDL のコマンドと動作

コマンド名	機能概要
R_FDL_CMD_PREPARE_ENV	FDLの実行環境を準備します。 - フラッシュプログラミングハードウェアの初期化 - ランタイム設定値の確認（ CPU動作周波数など）
R_FDL_CMD_WRITE	指定した書き込み開始アドレスから指定したデータ数、指定したRAM領域のデータを読み出して書き込みます。書き込む領域はあらかじめ消去（R_FDL_CMD_ERASEコマンド）を実行しておく必要があります。
R_FDL_CMD_ERASE	指定したブロックのデータフラッシュの内容を消去します。
R_FDL_CMD_BLANKCHECK	本FDLのブランクチェック機能は、対象領域のデータの読み出し前に、データの書き込み済み領域か未書き込み領域かを判別する機能です。本FDLの対象デバイスはデータの未書き込み領域に対し読み出しを行った場合、不定値が読み出されます。誤って"データが書かれている領域"のデータとして扱わないように、読み出しの前にブランクチェック処理を実行することで、データが書かれている領域か、データの書かれていない未書き込みの領域かを判別することができます。必要に応じてご使用ください。
R_FDL_CMD_READ	指定した読み出し開始アドレスから指定したデータ数、指定したRAM領域へデータを読み出します。読み出す領域はあらかじめデータが書かれている領域か把握しておく必要があります。

データフラッシュの 2 つのブロックを消去する例を使って、データフラッシュ操作の基本的なフローを下図に示します。フラッシュプログラミングハードウェアは消去または書き込みを 1 単位ずつ（1 ブロックの消去、1 ワードの書き込み）しかできませんが、ブランクチェックは複数ワード単位を一括で実行できます。ただし、ブランクチェックはアドレス 0x1000 バイト境界を跨いでの一括処理はできませんので、アドレス 0x1000 バイト境界を跨ぐ場合、処理は分割されます。

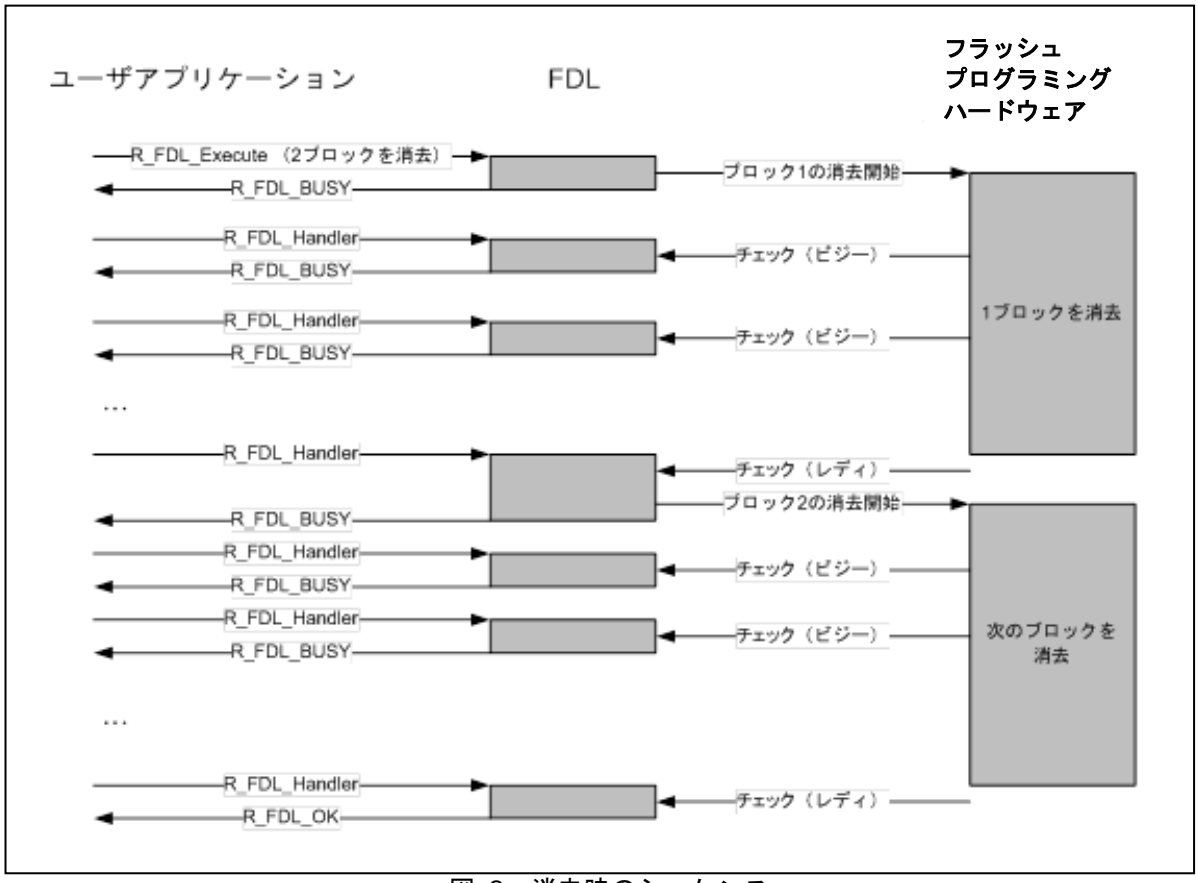


図 3　消去時のシーケンス

3. 2　リクエスト/レスポンス型アーキテクチャ

FDL はコマンドの起動にリクエスト/レスポンス型アーキテクチャを利用しています。各コマンドは関数で実行し、リクエスト構造体経由でコマンドやデータサイズなどを FDL へ受け渡します。FDL はリクエスト構造体の内容を解釈し、そのリクエスト構造体のメンバの整合性をチェックして実行を開始します。また、逆に FDL の状態、エラー情報などをリクエスト構造体経由で取得します。

リクエスト構造体メンバの status_enu は、状態が R_FDL_BUSY の場合、直ちに R_FDL_Execute 関数から処理が戻ります。"R_FDL_BUSY"である限り R_FDL_Handler を呼び出して、コマンドを完了させます。ユーザアプリケーションは、フラッシュのリソースを使用しない限り、残りの時間他の動作を自由に実行できます。

次の図に示すように、リクエスト構造体は FDL 動作の中心に位置しています。

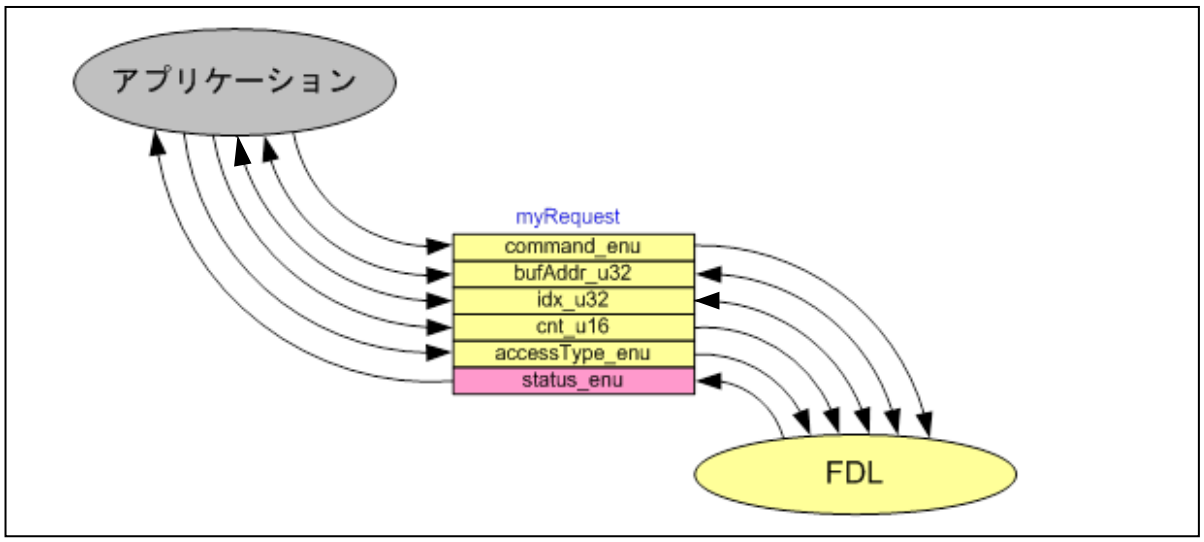


図 4　リクエスト構造体の使用法

リクエスト構造体変数とリクエスト構造体メンバの詳細については、「4. 3. 3　r_fdl_request_t リクエスト構造体」で説明します。また、すべてのリクエスト構造体メンバがどのコマンドにも必要とは限らないことに注意してください。「4. 5　コマンド」で個々のコマンドについて詳述します。

3. 3　BGO（Back Ground Operation）機能

FDL 関数はコマンド動作を開始した後､ユーザコードに戻りますので､セルフプログラミング中にユーザアプリケーションを実行することができます。コマンドは次の 2 つのステップに分けて実行されます。

1.　R_FDL_Execute 関数を使ったコマンドの起動

2.　要求されたコマンドの状態に対する R_FDL_Handler 関数を使った処理

R_FDL_Execute 関数はコマンドの実行を開始するだけで、実行後の最初の内部状態として、リクエスト状態である「ビジー」（または、リクエストを受け付け場合はエラー）が直ちに返されることに注意してください。

デバイスのフラッシュプログラミングハードウェアは、動作をバックグラウンドで実行します。このハードウェアの動作は複数の動作に分割され、それぞれがブロックやデータ項目の数に応じて個別に実行されます。最初の動作は R_FDL_Execute 関数の呼び出しによって実行されます。2 番目以降の動作は R_FDL_Handler 関数を呼び出して開始させます。このため、R_FDL_Handler 関数を複数回呼び出す必要があります。それぞれの動作が終了してから次の動作が開始するまでの間、処理が停止しています｡したがって､R_FDL_Handler 関数を呼び出す間隔が長くなると､処理時間全体が延びます。

ただし、R_FDL_CMD_READ コマンドはユーザコードに戻らず、R_FDL_Execute 関数内で全ての処理が終了します(R_FDL_Handler 関数の実行が不要です)。

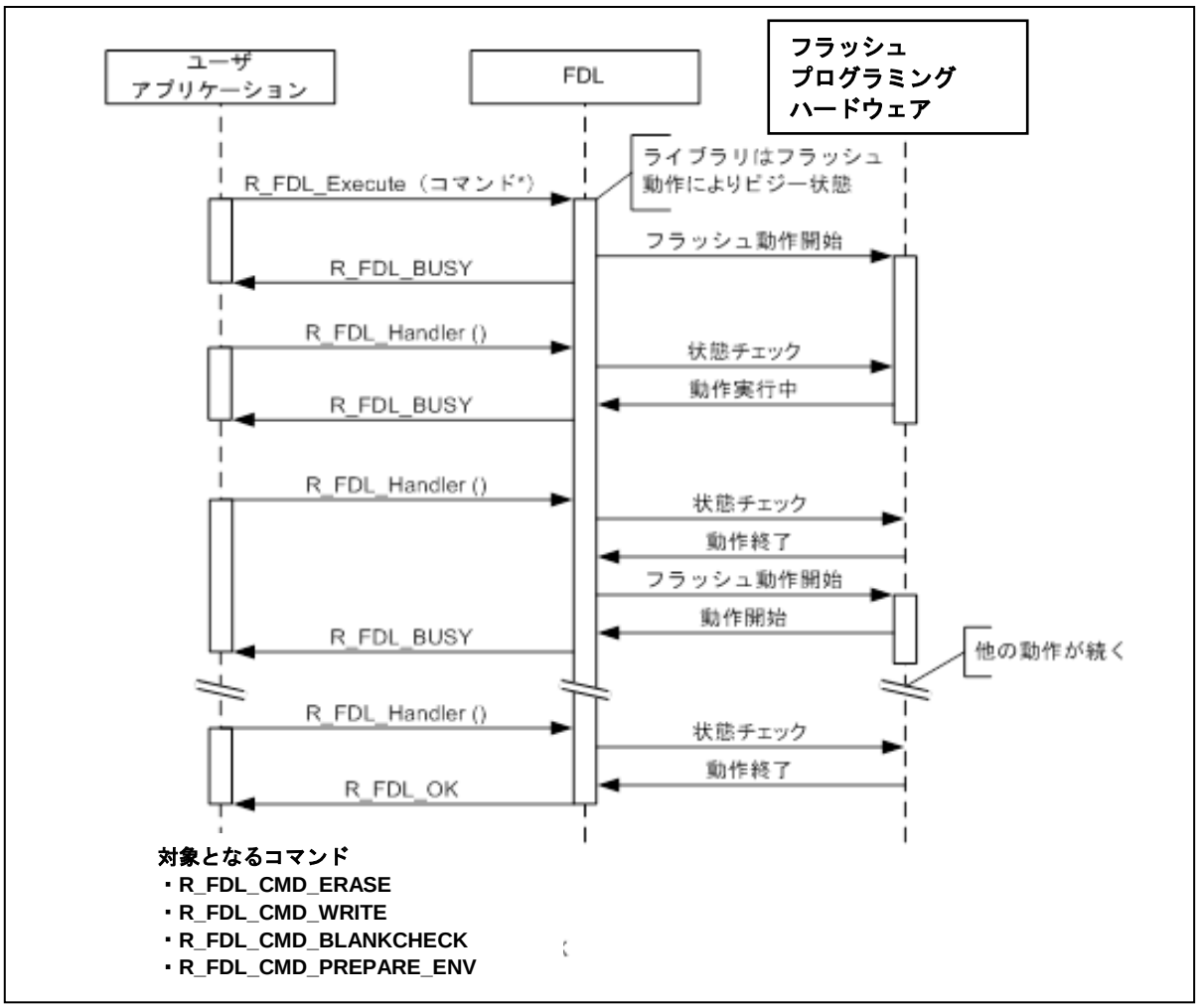


図 5　BGO（Back Ground Operation）機能

3.4　フラッシュアクセス保護

FDL のフラッシュアクセス保護は、想定外のデータフラッシュのアドレスへのアクセスを防ぎます。

この保護機能はユーザプールと EEL プールのブロックを区別します（詳細については「2.2　FDLプールの定義」参照）。ユーザプールへはユーザアプリケーション 1※から直接 FDL を介してのみアクセス可能で、EEL プールへはユーザアプリケーション 2※が FDL を介してのみアクセス可能です。

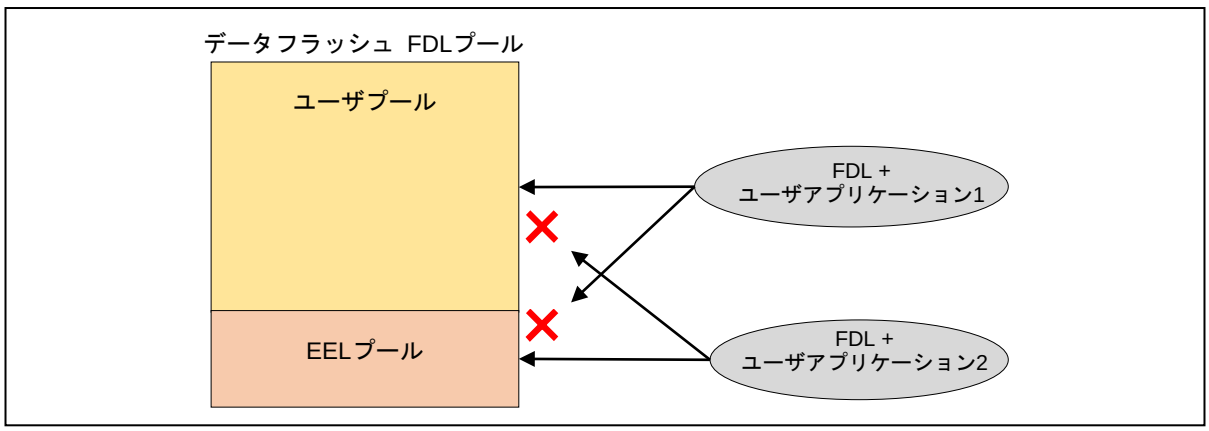


図 6　フラッシュへのアクセス権

※　「2.1　階層アーキテクチャ」参照

3.5　サスペンド/レジュームメカニズム

消去または書き込み動作は長く続くこともあり、終了を待たず、途中で中断したい場合があります。FDL には、その動作をサスペンドして、後からレジュームするオプション機能があります。 コマンドのサスペンド後は、別のコマンドを起動したり、ユーザアプリケーションで FDL と無関係な別の動作を実行したりできます。サスペンド状態中は、書き込み、ブランクチェック、読み出し実行後、サスペンド状態で終了します。

≪サスペンド許可シーケンス≫

書き込み　　--> サスペンド --> ブランクチェック※、読み出し※

消去　　--> サスペンド --> ブランクチェック※、読み出し※、書き込み※

ブランクチェック --> サスペンド --> ブランクチェック、読み出し、書き込み、消去

上記以外のシーケンス、サスペンドのネストは禁止です。

※ただし、書き込み中、消去中のブロック以外に対してのみ許可されます。

消去中にサスペンドし、レジュームした場合、消去が再度実行されますが、これは中断された消去の再開の為、消去回数は追加されません。

消去コマンド/書き込みコマンドをサスペンドする例を下図に示します。

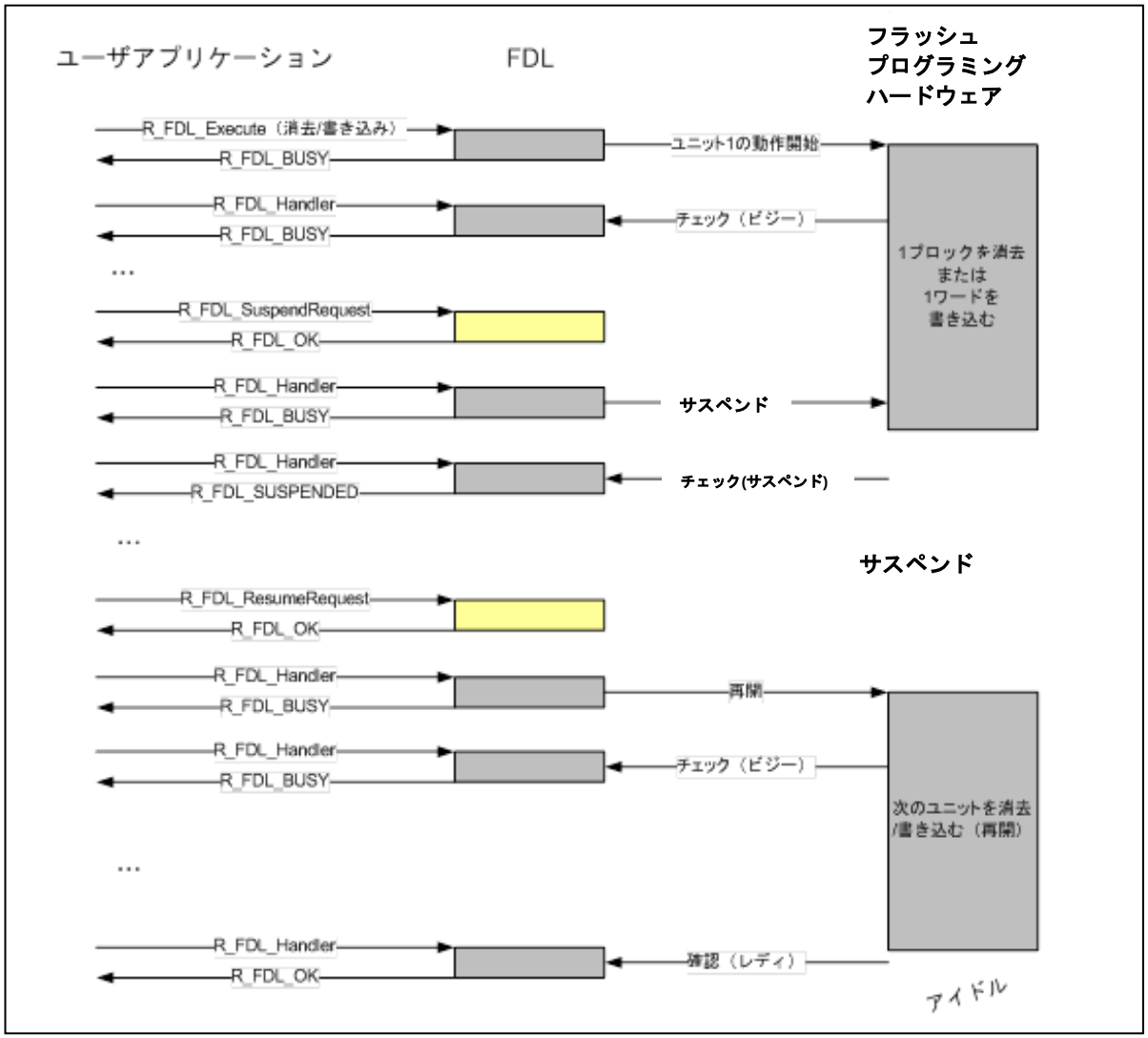


図 7　消去/書き込みのサスペンド/レジュームフロー例

ブランクチェック動作は、その動作がフラッシュマクロの境界（0x1000 バイト単位）に達するか動作が終了した場合を除き、サスペンド要求によって中断されることはありません。

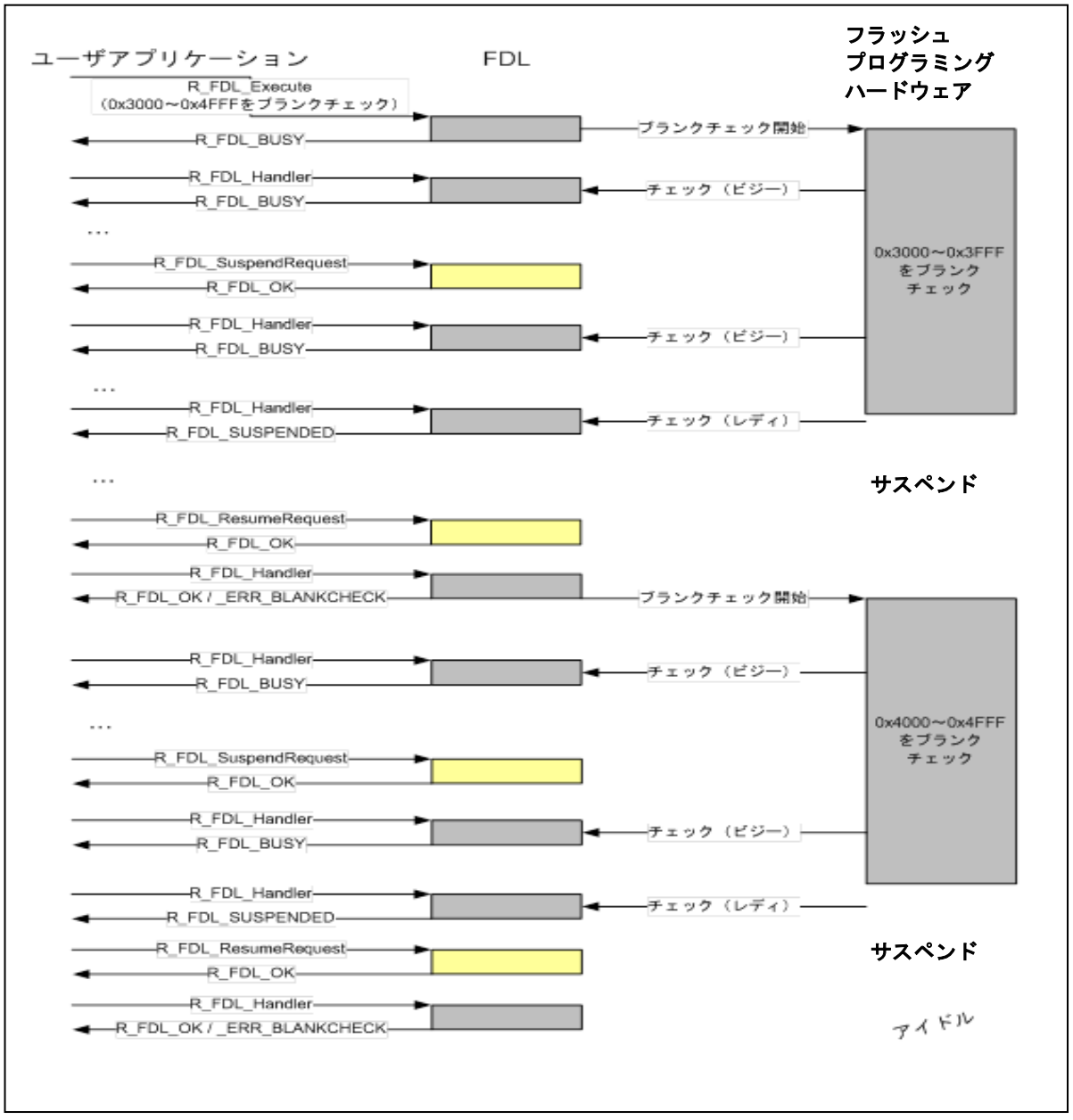


図 8　ブランクチェック動作のサスペンド/レジューム

3. 6　スタンバイ/ウェイクアップメカニズム

データフラッシュの操作中、デバイスの低消費電力で動作するモードへの移行は禁止です。このため、特にデータフラッシュの消去動作を行っていた場合は、デバイスの低消費電力で動作するモードに入るのがかなり遅れる可能性があります。このモード移行を素早く実行できるように、R_FDL_StandBy 関数と R_FDL_WakeUp 関数を用意しています。これらの関数の主な機能は、実行中のデータフラッシュの消去/書き込み動作があればそれをスタンバイし（R_FDL_StandBy 関数）、HALT モードからウェイクアップした後（R_FDL_WakeUp 関数）再開することです。なお、スタンバイ状態中に FDL 操作はできません。

スタンバイ処理を開始したら、エラーの発生以外、必ずスタンバイ状態になるまで処理を続けてください。スタンバイがデバイス内部のハードウェアによって遅れる場合や、スタンバイ機能がサポートされていない場合がある（消去と書き込みのみスタンバイ可能）ので、R_FDL_StandBy 関数の呼び出しで、必ずしもあらゆるデータフラッシュ操作が直ちにスタンバイするわけではありません。このような場合、状態がスタンバイになるまで R_FDL_StandBy 関数を繰り返し呼び出してください。

★ ウェイクアップ処理は、スタンバイが実行された状態により、直ちにスタンバイから復帰しない(R_FDL_BUSY が返る)場合があります。このような場合、エラーの発生以外、状態がスタンバイから復帰する(R_FDL_OK が返る)まで R_FDL_WakeUp 関数を繰り返し呼び出してください。

また、FDL のバージョンにより、スタンバイ/ウェイクアップで、一部動作が異なる場合がありますので、ご注意ください。

<FDL V2.12 以前>

FDL がアイドル状態（実行中のデータフラッシュ操作がない）の場合、R_FDL_StandBy 関数を呼び出すと FDL は直ちにスタンバイ状態になります。R_FDL_WakeUp 関数を呼び出すと、FDL は前の状態（この場合はアイドル状態）に戻ります。

<FDL V2.13 以降>

FDL がアイドル状態（実行中のデータフラッシュ操作がない）の場合、R_FDL_StandBy 関数を呼び出すと R_FDL_BUSY が返ります。以降、R_FDL_StandBy 関数を繰り返し呼び出すことでスタンバイ状態になります。その後、R_FDL_WakeUp 関数を呼び出すと、FDL は前の状態（この場合はアイドル状態）に戻ります。

FDL の動作中にスタンバイ要求が出された場合の FDL の動作例を図 9 と図 10 で説明します。

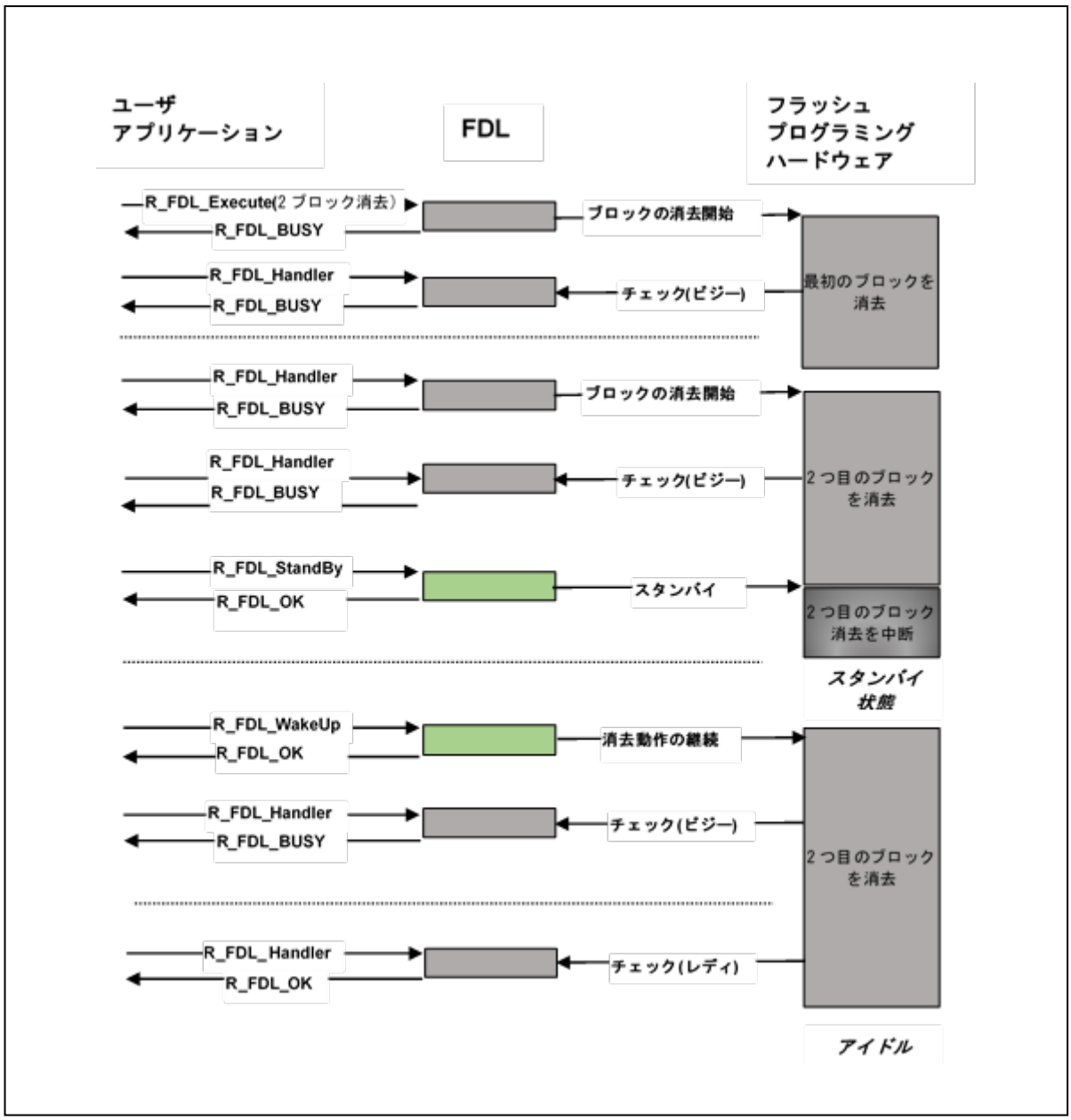


図 9　データフラッシュの消去動作におけるスタンバイ処理

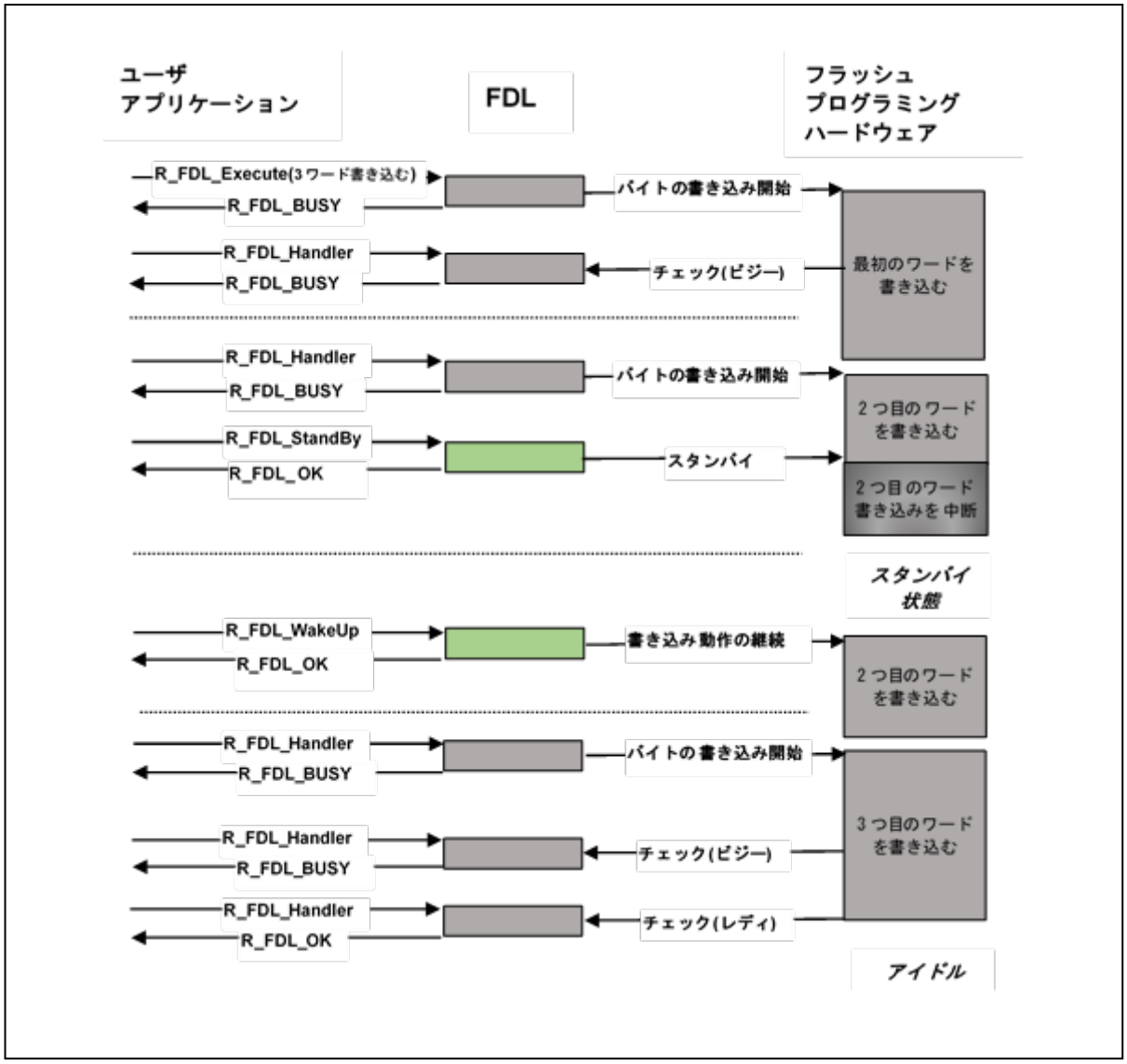


図 10　データフラッシュの書き込み動作におけるスタンバイ処理

3. 7　キャンセルメカニズム

消去または書き込み動作によっては長く続くため、終了するのを待てない場合があります。FDL には、その動作をキャンセルする機能があります。消去コマンドをキャンセルする方法を下図に示します。

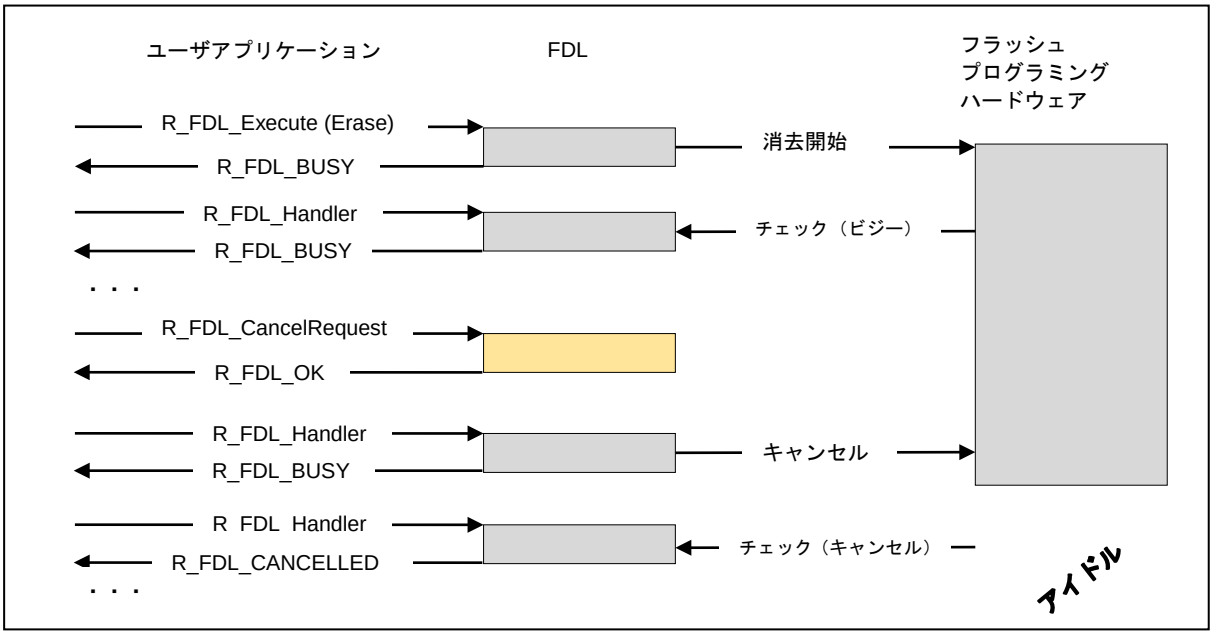


図 11　キャンセルフロー例

サスペンド状態に対して R_FDL_CancelRequest 関数を実行した場合、サスペンド状態の前に実行していた R_FDL_CMD_ERASE コマンド、R_FDL_CMD_WRITE コマンド、R_FDL_CMD_BLANKCHECK コマンドもキャンセルされます。

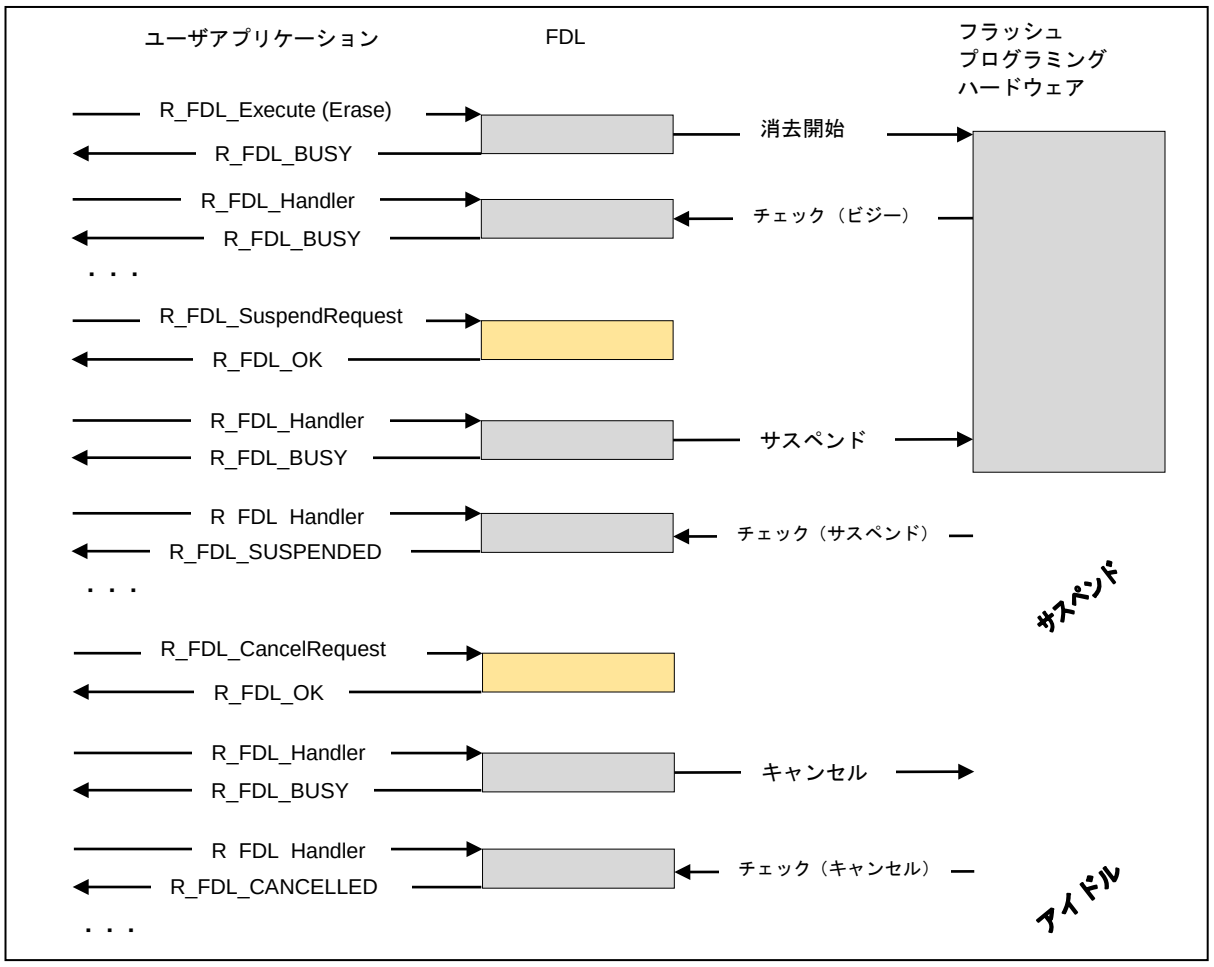


図 12　キャンセルフロー例

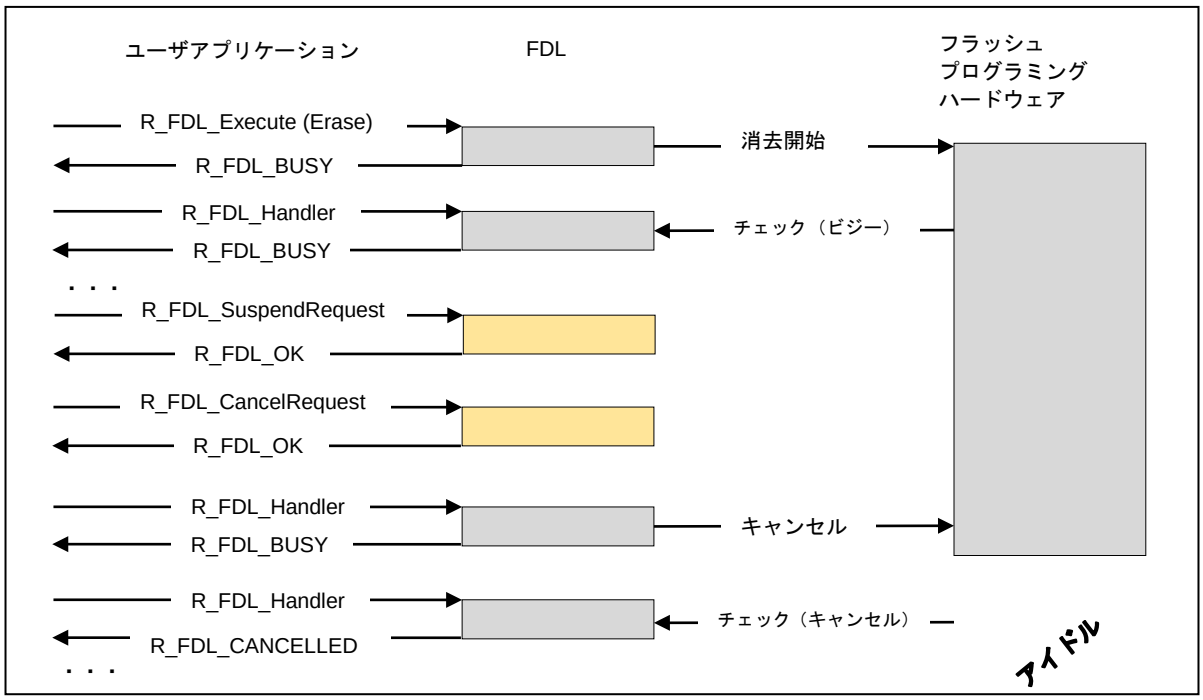


図 13　キャンセルフロー例

3.8　タイムアウト処理

デバイスのレジスタを監視しているループ処理にタイムアウト機能が実装されています。

（対応バージョン：V2.11以降）

第4章　ユーザインタフェース（API）

本章では、本 FDL のアプリケーションプログラミングインタフェース（API）について説明します。

4. 1　プリコンパイル設定

プリコンパイル設定は、コンパイラによって生成されるオブジェクトファイルに影響を与えます。よって、ユーザは、fdl_cfg.h ヘッダファイル内の定数を確認し、必要に応じて設定してください。

1.　クリティカルセクション定義

本定義の説明に関しましては「注意A　【R_FDL_CMD_PREPARE_ENVコマンド実行中のコードフラッシュへのアクセス禁止】」をご参照ください。

マクロ定義：

```
#define FDL_CRITICAL_SECTION_BEGIN FDL_User_CriticalSetionBegin();
#define FDL_CRITICAL_SECTION_END     FDL_User_CriticalSetionEnd();
```

2.　バージョン 1 互換定義

これはバージョンV1.xxとの互換用のオプションです。V2.00ではFDLの初期化処理がR_FDL_Init関数とR_FDL_CMD_PREPARE_ENV コマンドの二つに分割されていますが、このオプションを有効にすることで、R_FDL_Init関数内でR_FDL_CMD_PREPARE_ENV コマンドを実行するようなります。

定義：

```
#ifdef R_FDL_LIB_V1_COMPATIBILITY
・・・省略・・・
#else
・・・省略・・・
#endif
```

3.　RH850/F1K 専用ビルド定義（対応バージョン：V2.11 以降）

下記の仕様を無効に設定し、RH850/F1K, F1KM, F1KH専用にビルドします。
・ユーザ領域とFCUファームウェア格納領域の切り替え。
・FCUファームウェア転送。
本オプションを有効にした場合、RH850/F1K, F1KM, F1KH以外では使用しないでください。なお、本オプションを有効にした場合、R_FDL_MIRROR_FCU_COPYオプションとR_FDL_NO_FCU_COPYオプションは有効にしないでください。

#define R_FDL_NO_BFA_SWITCH

4.　FDL スタック実行処理定義（対応バージョン：V2.11 以降）

V2.11は従来スタックに一部配置していたFDLの処理を専用のRAMセクション"R_FDL_CodeRam"へ配置することで、FDLが使用するスタックの使用量を軽減させています。下記のオプションを有効にすることで、V2.11以前のスタックへの配置に戻すことが可能です。

#define R_FDL_EXE_INIT_CODE_ON_STACK

5.　FCU ファームウェア転送無効定義（対応バージョン：V2.12 以降）

FCU ファームウェア転送を無効に設定し、RH850/D1M1A, D1M1-V2, D1S1 専用にビルドします。

本オプションを有効にした場合、RH850/D1M1A, D1M1-V2, D1S1 以外では使用しないでください。
なお、本オプションを有効にした場合、R_FDL_MIRROR_FCU_COPY オプションと
R_FDL_NO_BFA_SWITCH オプションは有効にしないでください。

#define R_FDL_NO_FCU_COPY

6.　ユーザ領域と FCU ファームウェア格納領域の切り替え無効定義（対応バージョン：V2.12 以降）

ユーザ領域と FCU ファームウェア格納領域の切り替えを無効にします。

#define R_FDL_MIRROR_FCU_COPY

★　**注意事項**：RH850 FDL Type01 V2.12(FDL V2.12)以降のバージョンで、R_FDL_NO_FCU_COPY、R_FDL_MIRROR_FCU_COPY、および R_FDL_NO_BFA_SWITCH のプリコンパイル定義は各デバイスグループごとに有効/無効を設定する必要があります。

FDL V2.12 以降のバージョンのプリコンパイル設定：

FDL V2.12 以降 プリコンパイル定義	F1L/F1M/ F1H	D1L/ D1M1/D1M1H D1M2/D1M2H	D1M1A/ D1M1-V2 D1S1	F1K/F1KM/ F1KH	Futute Product
R_FDL_NO_BFA_SWITCH	**無効**	**無効**	**無効**	**有効**	**無効**
R_FDL_MIRROR_FCU_COPY	**無効**	**無効**	**無効**	**無効**	**有効**
R_FDL_NO_FCU_COPY	**無効**	**無効**	**有効**	**無効**	**無効**

注意：バージョン毎の対象製品は、各ライブラリに付属のsupport.txtをご確認ください。

4. 2　ランタイム設定

以下の設定はユーザが行ってください。

この設定には重要な本 FDL 関連の情報（CPU 周波数、FDL プールサイズなど）が含まれています。ランタイム設定はディスクリプタ構造体（r_fdl_descriptor_t 参照）に格納されており、r_fdl_types.h で宣言されますが、ユーザアプリケーションで定義され、関数 R_FDL_Init によって FDL に渡されます。ファイル fdl_descriptor.c はディスクリプタ構造体の定義および設定の例を示し、fdl_descriptor.h はディスクリプタ構造体への設定に必要な定義の例を示します。

以下の定義の値は必ずご確認の上、ユーザによって設定してください。

1. **CPU_FREQUENCY_MHZ**：

   CPU 動作周波数を設定してください。この周波数から FDL 内部でタイミング計算に使用します。CPU の動作周波数については、対象デバイスのユーザーズマニュアルをご参照ください。

   - 小数点が存在する場合、小数点以下を切り上げた値を設定してください。

   例 小数点がない場合）

   32MHzの場合、32を設定してください。

   例 小数点がある場合）

   25. 3MHzの場合、26を設定してください。

2. **FDL_POOL_SIZE**：

   FDL がアクセスするブロック数を定義します。デバイスに搭載されているデータフラッシュのブロックの総数に設定します。たとえば、デバイスが 32K バイトのデータフラッシュを備えており、ブロックサイズが 64 バイトの場合、FDL_POOL_SIZE には 512 を設定します。

3. **EEL_POOL_START**：

   EEL プールの先頭ブロックを定義します。ユーザアプリケーション 2※を使用しない場合は、この値を 0 に設定してください。

4. **EEL_POOL_SIZE**：

   EEL プールに使用するブロックの数を定義します。ユーザアプリケーション 2※を使用しない場合は、この値を 0 に設定してください。

例　ディスクリプタ設定値：

```
#define CPU_FREQUENCY_MHZ          (80)
/* FDL pool will use 512 blocks * 64 bytes = 32KB, no EEL pool */
#define FDL_POOL_SIZE              (512)
#define EEL_POOL_START             (0)
#define EEL_POOL_SIZE              (0)
```

例　ユーザアプリケーション 2※を使用する場合のディスクリプタ設定値：

```
#define CPU_FREQUENCY_MHZ          (80)
/* FDL pool will use 32KB, EEL pool occupies fist 16 KB */
#define FDL_POOL_SIZE              (512)
#define EEL_POOL_START             (0)
#define EEL_POOL_SIZE              (256)
```

※　「2. 1　階層アーキテクチャ」参照

4. 3　データ型

本節では、FDL が使用および提供するすべてのデータ定義について説明します。ユーザアプリケーションにおいて型の不一致が起こる可能性を減らすために、提供されている型を正確に使用してください。

★ **RH850 FDL Type01 V2.13(FDL V2.13)以降のバージョンでコンパイルをする場合：**

FDL V2.13 以降のバージョンでは、データ型の定義を ISO で定められた標準的な C 言語の規格(C99 規格以降)でコンパイルすることを前提としています。

GHS 社製コンパイラとルネサスエレクトロニクス社製コンパイラで、C99 以降の規格が指定されている場合、各コンパイラが提供するヘッダファイル”stdint.h”が使用されます。

C99 規格以降の規格が指定されていない場合は、"r_typedefs.h”内の定義（stdint.h と同等）が使用されます。

※C99 規格：ISO/IEC 9899:1999

RH850 FDL Type01 V2.12(FDL V2.12)以前のバージョンでコンパイルをする場合：

FDL V2.12 以前のバージョンで C99 規格のコンパイルをする場合、以下の 2 ファイルで定義されている「#include "r_typedefs.h"」を「#include "stdint.h”」に変更してください。

・r_fdl_global.h

・r_fdl_user_if_init.c

≪変更例≫

```
#include "stdint.h"
/* #include "r_typedefs.h" */
```

また、添付サンプルを使用する場合も同様に、定義されている「#include "r_typedefs.h"」を

#include "stdint.h"に変更してください。

RH850 FDL Type01 の対象サンプルファイルは、以下の通りです。

・fdlapp_control.c

・fdlapp_main.c

・fdl_descriptor.c

・fdl_user.c

4. 3. 1　単純型定義

★　**コンパイラでC99以降の規格が指定されている場合の型定義：**

各コンパイラが提供するヘッダファイル”stdint.h”をご確認ください。

コンパイラでC99以降の規格が指定されていない場合の型定義("r_typedefs.h”内記述の抜粋）：

```
typedef signed char             int8_t;
typedef unsigned char           uint8_t;
typedef signed short            int16_t;
typedef unsigned short          uint16_t;
typedef signed long             int32_t;
typedef unsigned long           uint32_t;
```

説明：

これらの単純型はFDL API全体を通じて使用されます。

FDL固有のすべての単純型定義については、FDLのインストールパッケージに入っているファイルr_typedefs.hを参照してください。(C99規格でコンパイルする場合は、stdint.hを参照してください。)

4. 3. 2　r_fdl_descriptor_t ディスクリプタ構造体

型定義：

```
typedef struct R_FDL_DESCRIPTOR_T
{
    uint16_t    cpuFrequencyMHz_u16;
    uint16_t    fdlPoolSize_u16;
    uint16_t    eelPoolStart_u16;
    uint16_t    eelPoolSize_u16;
} r_fdl_descriptor_t;
```

説明：

ランタイム設定（「4. 2　ランタイム設定」参照）は独立したデータ型で定義されます。データ型の変数は初期化フェーズで読み出され、設定に応じて内部変数がセットされます。

メンバ：

メンバ	説明
cpuFrequencyMHz_u16	CPU 動作周波数（MHz）
fdlPoolSize_u16	FDL プールのサイズ（ブロック数）
eelPoolStart_u16	EEL プールの先頭ブロック
eelPoolSize_u16	EEL プールに使用するブロックの数

4. 3. 3　r_fdl_request_t リクエスト構造体

型定義：

```
typedef volatile struct R_FDL_REQUEST_T
{
    r_fdl_command_t     command_enu;
    uint32_t            bufAddr_u32;
    uint32_t            idx_u32;
    uint16_t            cnt_u16;
    r_fdl_accessType_t  accessType_enu;
    r_fdl_status_t      status_enu;
} r_fdl_request_t;
```

説明：

ユーザ動作は、主にR_FDL_Execute関数により開始されます。実行に必要なすべての情報はリクエスト構造体によってFDLに渡されます。また、そのエラーは同じリクエスト構造体によって返されます。このリクエスト構造体の詳細については「3. 2　リクエスト/レスポンス型アーキテクチャ」を参照してください。目的のコマンド毎にこのリクエスト構造体に値を設定する方法と結果を確認する方法については、「4. 5　コマンド」に詳説します。

リクエスト構造体メンバのidx_u32にアドレスを設定する場合、簡素化のため、データフラッシュの先頭アドレスを0x0000_0000で始まる仮想アドレスで扱います。ハードウェアのユーザマニュアルのデータフラッシュの記述にあるアドレスではありません。

メンバ：

メンバ	説明
command_enu	R_FDL_Execute 関数へ設定するコマンド
bufferAdd_u32	データバッファの先頭アドレス 【R_FDL_CMD_WRITEコマンドの場合】 ユーザの書き込むデータを格納しているデータバッファ 【R_FDL_CMD_READコマンドの場合】 データフラッシュから読み出したデータを格納するデータバッファ 【その他の場合】 設定無効
idx_u32	インデックス 【ユーザが設定する場合】 ≪R_FDL_CMD_ERASEコマンドの場合≫ 消去開始ブロック番号 ≪R_FDL_CMD_WRITE コマンド、R_FDL_CMD_READ コマンド、 R_FDL_CMD_BLANKCHECK コマンドの場合≫ 各コマンドを実行する開始アドレス 【ユーザがデータを取得する場合】 ≪R_FDL_CMD_READ コマンド、 R_FDL_CMD_BLANKCHECK コマンドの場合≫ 処理が失敗したアドレス
cnt_u16	カウント 【書き込みコマンド（R_FDL_CMD_WRITE）の場合】 書き込みデータ数 【消去コマンド（R_FDL_CMD_ERASE）の場合】 消去を実行するブロック数 【その他の場合】 設定無効
accessType_enu	アクセスタイプ 【FDL プールへのアクセス元がユーザアプリケーション 1※の場合】 R_FDL_ACCESS_USER 【FDL プールへのアクセス元がユーザアプリケーション 2※の場合】 R_FDL_ACCESS_EEL
status_enu	FDLの状態、エラー情報。 なお、R_FDL_Execute 関数、R_FDL_Handler 関数以外に R_FDL_SuspendRequest 関数、R_FDL_ResumeRequest 関数、 R_FDL_CancelRequest 関数実行時も status_enu を参照します。

※　「2.1　階層アーキテクチャ」参照

★

4. 3. 4　r_fdl_command_t

型定義：

```
typedef enum R_FDL_COMMAND_T
{
      R_FDL_CMD_ERASE                = 0,
      R_FDL_CMD_WRITE                = 1,
      R_FDL_CMD_BLANKCHECK           = 2,
      R_FDL_CMD_READ                 = 3,
      R_FDL_CMD_PREPARE_ENV          = 5
} r_fdl_command_t;
```

説明：

コマンドは1つの関数、R_FDL_Execute関数により開始され、その後、R_FDL_Handler関数により制御されます。使用可能なコマンドの詳細については「4. 5　コマンド」を参照してください。

メンバ：

メンバ	説明
R_FDL_CMD_ERASE	指定したブロックのデータフラッシュの内容を消去します。
R_FDL_CMD_WRITE	指定した書き込み開始アドレスから指定したデータ数、指定したRAM 領域のデータを読み出して書き込みます。書き込む領域はあらかじめ消去（R_FDL_CMD_ERASE コマンド）を実行しておく必要があります。
R_FDL_CMD_BLANKCHECK	本 FDL のブランクチェック機能は、対象領域のデータの読み出し前に、データの書き込み済み領域か未書き込み領域かを判別する機能です。本 FDL の対象デバイスはデータの未書き込み領域に対し読み出しを行った場合、不定値が読み出されます。誤って"データが書かれている領域"のデータとして扱わないように、読み出しの前にブランクチェック処理を実行することで、データが書かれている領域か、データの書かれていない未書き込みの領域かを判別することができます。必要に応じてご使用ください。
R_FDL_CMD_READ	指定した読み出し開始アドレスから指定したデータ数、指定したRAM 領域へデータを読み出します。読み出す領域はあらかじめデータが書かれている領域か把握しておく必要があります。
★ R_FDL_CMD_WRITE16B (FDL V2.13 以降　非サポート)	【FDL V2.12 以前のバージョンで、一部デバイスのみサポート】 R_FDL_CMD_WRITE16B コマンドは一部の製品のみサポートしています。サポート対象外の製品では R_FDL_CMD_WRITE16B コマンドを使用しないでください。以下にサポート対象外製品の例を示します。 - サポート対象外製品例：RH850/F1L, F1M, F1H RH850/F1K, F1KM, F1KH, RH850/D1x なお、R_FDL_CMD_WRITE16B コマンドのサポートデバイス情報に関しましては、対象デバイスのユーザーズマニュアルをご確認ください。
R_FDL_CMD_PREPARE_ENV	FDLの実行環境を準備します - フラッシュプログラミングハードウェアの初期化 - ランタイム設定値の確認（ CPU動作周波数など）

4. 3. 5　r_fdl_accessType_t

型定義：

```
typedef enum R_FDL_ACCESS_TYPE_T
{
      R_FDL_ACCESS_NONE,
      R_FDL_ACCESS_USER,
      R_FDL_ACCESS_EEL
} r_fdl_accessType_t;
```

説明：

FDLプールへのアクセス権の設定です。データフラッシュ操作を開始するために、設定されたアクセス対象のプールに応じてデータフラッシュへのアクセスタイプを設定する必要があります。プールの範囲はディスクリプタで定義され、FDLに渡されます（「図 6　フラッシュへのアクセス権」を確認してください）。

誤ったアクセスを防ぐために、動作が終わるたびにアクセス権はR_FDL_ACCESS_NONEにリセットされます。

メンバ：

メンバ	説明
R_FDL_ACCESS_NONE	FDL プールへのアクセス不可
R_FDL_ACCESS_USER	ユーザプールへのアクセス要求
R_FDL_ACCESS_EEL	EEL プールへのアクセス要求

4. 3. 6　r_fdl_status_t

型定義：

```
typedef enum R_FDL_STATUS_T
{
      R_FDL_OK,
      R_FDL_BUSY,
      R_FDL_SUSPENDED,
      R_FDL_ERR_CONFIGURATION,
      R_FDL_ERR_PARAMETER,
      R_FDL_ERR_PROTECTION,
      R_FDL_ERR_REJECTED,
      R_FDL_ERR_WRITE,
      R_FDL_ERR_ERASE,
      R_FDL_ERR_BLANKCHECK,
      R_FDL_ERR_COMMAND,
      R_FDL_ERR_ECC_SED,
      R_FDL_ERR_ECC_DED,
      R_FDL_ERR_INTERNAL,
      R_FDL_CANCELLED
} r_fdl_status_t;
```

説明：

列挙型r_fdl_status_tはFDLの状態を定義しています。FDLコマンドの現在の状態を示すために、上記の状態/エラーコードが本FDLによって返されます。初期化とサスペンド/レジューム、キャンセルに関する他のAPI関数も、この状態コードを返します。すべての状態コードとエラーコードの根本的な原因と解釈は、実行される動作または呼び出される関数に依存します。動作を示すコードの意味を次の表で説明します。

メンバ：

メンバ	説明
R_FDL_OK	要求された動作が正常終了しました。
R_FDL_BUSY	要求された動作が正常に起動し、進行中です。
R_FDL_SUSPENDED	現在の動作がサスペンドしています。
R_FDL_ERR_CONFIGURATION	ディスクリプタのデータが不正なため、要求されたコマンドが実行されませんでした。
R_FDL_ERR_PARAMETER	入力データが不正なため、要求されたコマンドが実行されませんでした。
R_FDL_ERR_PROTECTION	保護（FHVE15 レジスタ/ FHVE3 レジスタ）している領域/設定に対して、禁止されているコマンドを実行しました。 FDL 内部のパラメータ、またはデータフラッシュを制御するハードウェアに FDL 経由で設定したパラメータがコマンド実行中に不正に改変されました。
R_FDL_ERR_REJECTED	別のコマンドが操作中のため、要求されたコマンドが実行されませんでした。
R_FDL_ERR_WRITE	未消去領域への書き込みまたはハードウェア故障のため、要求された書き込みコマンドにエラーが生じました。
R_FDL_ERR_ERASE	ハードウェアの消去エラーにより、要求された消去コマンドにエラーが生じました。
R_FDL_ERR_BLANKCHECK	指定した領域が消去されていません。
R_FDL_ERR_COMMAND	要求されたコマンドが存在しません。
R_FDL_ERR_ECC_SED	FDL 操作は実行され、訂正されたデータが読み出されましたが ECC エラー（1 ビットエラー)※を検出しました。
R_FDL_ERR_ECC_DED	FDL 操作は実行され、訂正できなかったデータが読み出されました。ECC エラー（2 ビットエラー) ※を検出しました。
R_FDL_CANCELLED	現在のコマンド操作がキャンセルされました。
R_FDL_ERR_INTERNAL	予期せぬエラーです。

※　ECCの詳細は対象デバイスのユーザーズマニュアルをご参照ください。

4. 4　関数

API 関数は、インタフェースにおける役割によって次のように分類されます。

初期化：

- R_FDL_Init

データフラッシュ操作：

- R_FDL_Execute
- R_FDL_Handler

動作制御：

- R_FDL_SuspendRequest
- R_FDL_ResumeRequest
- R_FDL_StandBy
- R_FDL_WakeUp
- R_FDL_CancelRequest

管理：

- R_FDL_GetVersionString

4. 4. 1　初期化

4.4.1.1. R_FDL_Init

概要：FDLの初期化

インタフェース：Cインタフェース

```
r_fdl_status_t R_FDL_Init (const r_fdl_descriptor_t * descriptor_pstr);
```

引数：パラメータ

引数	型	アクセス	説明
descriptor_pstr	r_fdl_descriptor_t *	R	FDL のコンフィグレーションディスクリプタ（「4. 3. 2　r_fdl_descriptor_t ディスクリプタ構造体」参照）

戻り値：

型	説明		
r_fdl_status_t	R_FDL_OK	説明	正常終了しました
		原因	正常動作です
		処置	無し
	R_FDL_ERR_CONFIGURATION	説明	今回のR_FDL_Init関数の実行を拒否しました
		原因	ランタイムの設定値、引数のポインタが不正です
		処置	正しいランタイムの設定値、引数のポインタを設定してください
	R_FDL_ERR_INTERNAL※	意味	今回のコマンドの実行は正常に終了しませんでした
		理由	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1） ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2） FDLが破壊されている可能性もあります
		処置	さらなるFDL操作を中止し、原因を調査してください

※　バージョン 1 互換定義（「2　バージョン1 互換定義」参照）を有効にした場合のみ。

事前条件：【バージョン1 互換定義※　無効の場合】
なし

【バージョン1 互換定義※　有効の場合】
本関数内でR_FDL_CMD_PREPARE_ENVコマンドが実行されます。必ず、「4.4.2.1 R_FDL_Execute」を参照しR_FDL_CMD_PREPARE_ENVコマンドの仕様をご確認ください。

※　「2　バージョン1 互換定義」参照。

事後条件：なし

説明：FDLの内部変数の初期化を行います。
R_FDL_Init関数はr_fdl_user_if_init.cファイル内で定義されています。必ずr_fdl_user_if_init.cファイルをご使用ください。

例：

```
const r_fdl_descriptor_t    sampleApp_fdlConfig_enu =
{
      CPU_FREQUENCY_MHZ,
      FDL_POOL_SIZE,
      EEL_POOL_START,
      EEL_POOL_SIZE
};

r_fdl_status_t ret;

ret = R_FDL_Init (&sampleApp_fdlConfig_enu);

if (ret != R_FDL_OK)
{
      /* Error handler */
}
```

4. 4. 2　データフラッシュ操作

4.4.2.1. R_FDL_Execute

概要：データフラッシュ操作を開始します

インタフェース：Cインタフェース

```
void R_FDL_Execute (r_fdl_request_t * request_pstr);
```

引数：パラメータ

引数	型	アクセス	説明
request_pstr	r_fdl_request_t *	RW	この引数は、コマンドを定義するリクエスト構造体、コマンドパラメータ、さらに実行結果も指します。 リクエスト構造体の詳細については「4. 3. 3 r_fdl_request_t リクエスト構造体」参照。

戻り値：なし

事前条件：・R_FDL_Init関数を実行し、R_FDL_Init関数の戻り値がR_FDL_OKとなっていること。
・R_FDL_CMD_PREPARE_ENVコマンド以外は、すでにR_FDL_CMD_PREPARE_ENVコマンドを実行し、R_FDL_Handler関数の戻り値がR_FDL_OKとなっていること。

事後条件：・R_FDL_Execute関数実行後、正常終了の場合、status_enuはR_FDL_BUSYになります。
・status_enuがR_FDL_BUSY中は、繰り返しR_FDL_Handler関数※を実行してください。
・コマンドの動作中、ユーザアプリケーションはリクエスト構造体のメンバを変更しないでください。
※　R_FDL_CMD_READコマンドにはR_FDL_Handler関数の実行は必要ありません。

説明：FDLの実行環境の準備とデータフラッシュに対し、消去、書き込み、ブランクチェック、読み出しを実行する関数です。コマンドなどをパラメータとして扱い、リクエスト構造体に設定し実行します。
また、R_FDL_CMD_PREPARE_ENVコマンドの場合、R_FDL_Execute関数実行後のR_FDL_Handler関数実行中、コードフラッシュへのアクセス禁止の区間があります。詳細は「注意A　【R_FDL_CMD_PREPARE_ENVコマンド実行中のコードフラッシュへのアクセス禁止】」をご参照ください。
なお、V2.11以降、R_FDL_CMD_PREPARE_ENVコマンド実行中にFLERR割り込み注が発生しないように、FLERR割り込み[注]をマスクし、復帰時にマスクを解除しています。
（対応バージョン：V2.11以降）
注 FLERR割り込みの詳細は対象デバイスのユーザーズマニュアルを参照してください。

例：ブロック0～3を消去する。

```
r_fdl_request_t myRequest;

myRequest.command_enu          = R_FDL_CMD_ERASE;
myRequest.idx_u32              = 0;
myRequest.cnt_u16              = 4;
myRequest.accessType_enu       = R_FDL_ACCESS_USER;

R_FDL_Execute (&myRequest);
while (myRequest.status_enu == R_FDL_BUSY)
{
      R_FDL_Handler ();
}

if (myRequest.status_enu != R_FDL_OK)
{
      /* Error handler */
}
```

例：アドレス0x10から始まる8バイトを書き込む。

```
r_fdl_request_t myRequest;

uint32_t data[]                = { 0x11223344, 0x55667788 };

myRequest.command_enu          = R_FDL_CMD_WRITE;
myRequest.idx_u32              = 0x10;
myRequest.cnt_u16              = 2;
myRequest.bufAddr_u32          = (uint32_t)&data[0];
myRequest.accessType_enu       = R_FDL_ACCESS_USER;

R_FDL_Execute (&myRequest);
while (myRequest.status_enu == R_FDL_BUSY)
{
      R_FDL_Handler ();
}

if (myRequest.status_enu != R_FDL_OK)
{
      /* Error handler */
}
```

例：アドレス 0x10～0x17 をブランクチェックする。

```
r_fdl_request_t myRequest;

myRequest.command_enu          = R_FDL_CMD_BLANKCHECK;
myRequest.idx_u32              = 0x10;
myRequest.cnt_u16              = 2;
myRequest.accessType_enu       = R_FDL_ACCESS_USER;

R_FDL_Execute(&myRequest);

while (myRequest.status_enu == R_FDL_BUSY)
{
      R_FDL_Handler();
}

if (myRequest.status_enu != R_FDL_OK)
{
      /* Error handler */
}
```

例：アドレス 0x10 から始まる 2 ワードを読み出す。

```
r_fdl_request_t myRequest;

uint32_t data[2];

myRequest.command_enu          = R_FDL_CMD_READ;
myRequest.idx_u32              = 0x10;
myRequest.cnt_u16              = 2;
myRequest.bufAddr_u32          = (uint32_t)&data[0];
myRequest.accessType_enu       = R_FDL_ACCESS_USER;

R_FDL_Execute(&myRequest);

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
}
```

例：R_FDL_CMD_PREPARE_ENV コマンドの実行

```
r_fdl_request_t myRequest;
myRequest.command_enu = R_FDL_CMD_PREPARE_ENV;
R_FDL_Execute(&myRequest);

while (myRequest.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler();
}

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
}
```

注意A【R_FDL_CMD_PREPARE_ENVコマンド実行中のコードフラッシュへのアクセス禁止】※1

R_FDL_CMD_PREPARE_ENVコマンド実行中はコードフラッシュへのアクセス禁止の区間があります。必ずPSWレジスタのNP、IDを使用し、割り込み禁止に設定もしくは割り込みハンドラをRAMに移動※2してください。また、R_FDL_CMD_PREPARE_ENVコマンド処理内のコードフラッシュへのアクセス禁止の区間の前後でコールされるコールバック関数のインタフェースも提供しています（fdl_user.c内）。必ず下記a）、b）のどちらかの方法により割り込みを禁止に設定してください。

a) R_FDL_CMD_PREPARE_ENVコマンド処理実行前に割り込みを禁止 or 割り込みハンドラをRAMに設定する。かつ、R_FDL_CMD_PREPARE_ENVコマンド処理実行後に割り込み設定 or 割り込みハンドラ設定を復帰する。
b) FDL_User_CriticalSetionBegin関数内で割り込みを禁止 or 割り込みハンドラをRAMに設定する。かつ、FDL_User_CriticalSetionEnd関数内で割り込み設定 or 割り込みハンドラ設定を復帰する。

※1　RH850/F1K, F1KM, F1KHは対象外です。RH850/F1K, F1KM, F1KHの場合、上記のコードフラッシュへのアクセス禁止はありません。また、次のページに記載されているコールバック関数（FDL_User_CriticalSetionBegin関数、FDL_User_CriticalSetionEnd 関数）もコールされることはありません。（対応バージョン：V2.11以降）

※2　割り込みハンドラの移動方法の詳細については、ハードウェアのユーザーズマニュアルを参照してください。

なお、提供しているコールバック関数内の処理はサンプルアプリケーションですので、ユーザによる設計が必要です。必要に応じてユーザが変更し、必ず十分な評価を行ってください。また、ユーザシステム上において割り込み、例外ハンドラ以外でも、コードフラッシュへのアクセス禁止区間中に、コードフラッシュへアクセスが発生することがないか十分に検証してください（例えば、転送元をコードフラッシュとしたDMA転送など）。

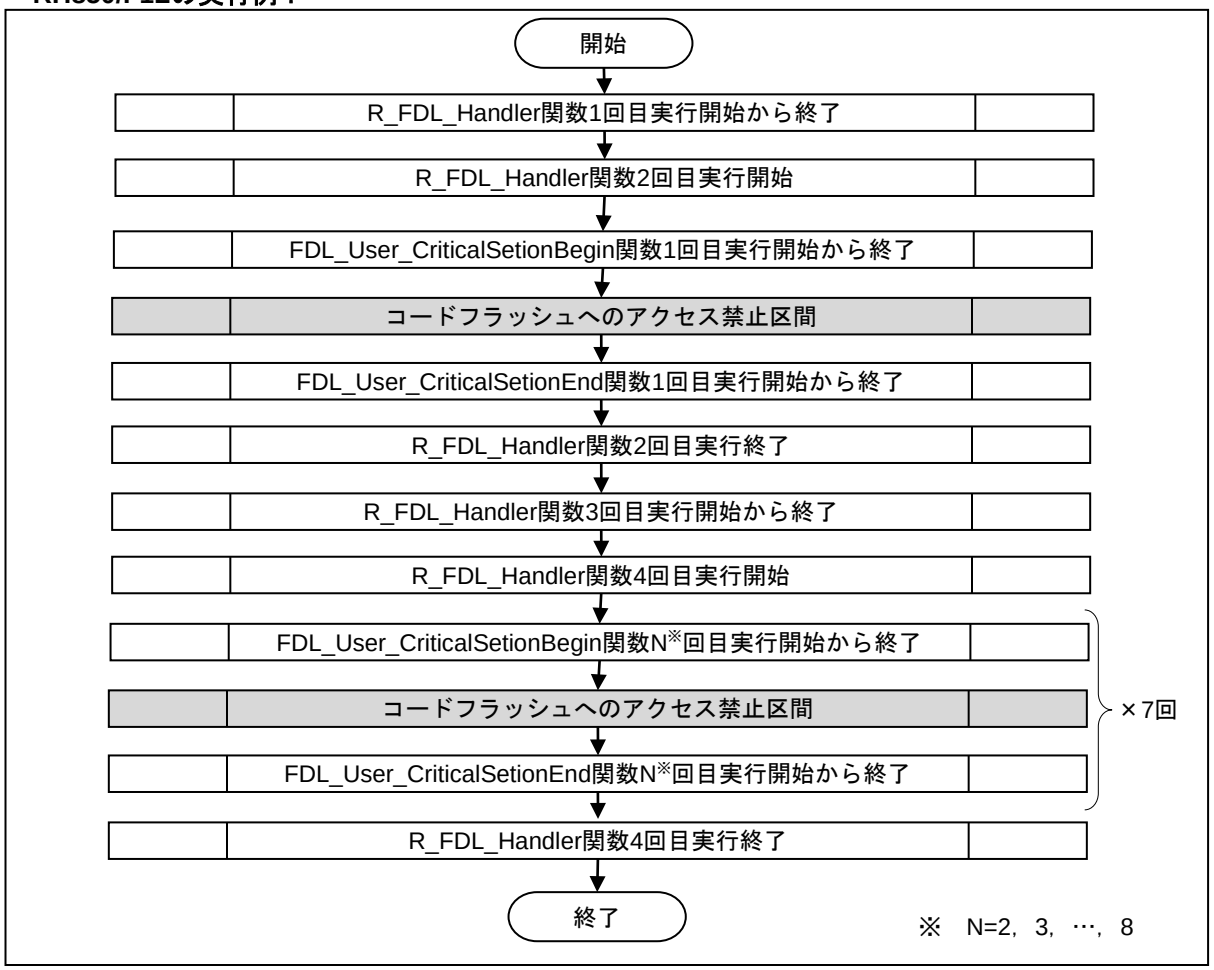
コールバック関数を使用しない場合、それぞれのコールバック関数内のサンプルコードは削除してください。

- ファイル名：fdl_user.c
- コールバック関数名：FDL_User_CriticalSetionBegin関数、FDL_User_CriticalSetionEnd関数

- コールバック関数実行タイミング

FDL_User_CriticalSetionBegin関数はR_FDL_CMD_PREPARE_ENVコマンド実行時のコードフラッシュへのアクセス禁止の区間の直前で実行され、FDL_User_CriticalSetionEnd関数はコードフラッシュへのアクセス禁止の区間終了直後に実行されます。

RH850/F1Lの実行例：



4.4.2.2. R_FDL_Handler

概要：FDLによるコマンド操作を進行させ、終了を確認する関数です。

インタフェース：Cインタフェース

```
void R_FDL_Handler (void);
```

引数：なし

戻り値：なし

事前条件：・R_FDL_Init関数を実行し、R_FDL_Init関数の戻り値がR_FDL_OKとなっていること。
・R_FDL_Execute関数を実行し※、status_enuがR_FDL_BUSYとなっていること。
※　R_FDL_CMD_READコマンドにはR_FDL_Handler関数の実行は必要ありません。

事後条件：・status_enuがR_FDL_BUSY中は、繰り返しR_FDL_Handler関数※を実行してください。
・コマンドの動作中、ユーザアプリケーションはリクエスト構造体のメンバを変更しないでください。

説明：完了していないコマンドを動作させその進捗を監視するために、この関数を一定間隔で呼び出す必要があります。コマンド実行の状態/エラーコードによりstatus_enuが更新されます。

注意　コマンドの処理を行わなければ、R_FDL_HandlerはCPUサイクルをほとんど消費しません。

例：

```
while (true)
{
      R_FDL_Handler();
      User_Task_A();
      User_Task_B();
      User_Task_C();
      User_Task_D();
}
```

4. 4. 3　動作制御

4.4.3.1. R_FDL_SuspendRequest

概要：FDL実行中にFDLのサスペンドを要求する関数です。

インタフェース：Cインタフェース

```
r_fdl_status_t R_FDL_SuspendRequest (void);
```

引数：なし

戻り値：

型	説明		
r_fdl_status_t	R_FDL_OK	説明	正常にサスペンド要求を受け付けました
		原因	正常動作です
		処置	無し
	R_FDL_ERR_REJECTED	説明	今回のR_FDL_SuspendRequest関数の実行を拒否しました
		原因	FDLは、初期化を実行していない状態、または不正な状態です
		処置	さらなるFDL操作を中止し、原因を調査してください

事前条件：・R_FDL_Init関数を実行し、R_FDL_Init関数の戻り値がR_FDL_OKとなっていること。
・すでにstatus_enuがR_FDL_CMD_ERASEコマンド、R_FDL_CMD_WRITEコマンド、R_FDL_CMD_BLANKCHECKコマンドによってR_FDL_BUSYとなっていること。
・サスペンド中でないこと。

事後条件：・status_enuがR_FDL_BUSY 中は、繰り返しR_FDL_Handler関数を繰り返し実行してください。

説明：FDL実行中にFDLのサスペンドを要求する関数です。実行中の消去、書き込み、ブランクチェックを中断します。

例：

```
r_fdl_status_t    srRes_enu;
r_fdl_request_t myReq_str_str;
uint32_t              i;

/* Start Erase operation */
myReq_str_str.command_enu        = R_FDL_CMD_ERASE;
myReq_str_str.idx_u32               = 0;
myReq_str_str.cnt_u16               = 4;
myReq_str_str.accessType_enu    = R_FDL_ACCESS_USER;

R_FDL_Execute (&myReq_str_str);

/* Now call the handler some times */
i = 0;
while ( (myReq_str_str.status_enu == R_FDL_BUSY) && (i < 10) )
{
      R_FDL_Handler ();
      i++;
}

/* Suspend request and wait until suspended */
srRes_enu = R_FDL_SuspendRequest ();

if (R_FDL_OK != srRes_enu)
{
      /* error handler */
      while (1)
            ;
}

while (R_FDL_SUSPENDED != myReq_str_str.status_enu)
{
      R_FDL_Handler ();
}

/* Now the FDL is suspended and we can handle other operations or read the Data Flash ... */

/* Erase resume */
srRes_enu = R_FDL_ResumeRequest();

if (R_FDL_OK != srRes_enu)
{
      /* Error handler */
}

/* Finish the erase */
while (myReq_str_str.status_enu == R_FDL_SUSPENDED)
{
      R_FDL_Handler();
}
while (myReq_str_str.status_enu == R_FDL_BUSY)
{
      R_FDL_Handler();
}

if (R_FDL_OK != myReq_str_str.status_enu)
{
      /* Error handler */
}
```

4.4.3.2. R_FDL_ResumeRequest

概要：サスペンド状態からの復帰を要求する関数です。

インタフェース：Cインタフェース

```
r_fdl_status_t R_FDL_ResumeRequest (void);
```

引数：なし

戻り値：

型	説明		
r_fdl_status_t	R_FDL_OK	説明	正常にサスペンドからの復帰の要求を受け付けました
		原因	正常動作です
		処置	無し
	R_FDL_ERR_REJECTED	説明	今回のR_FDL_ResumeRequest関数の実行を拒否しました
		原因	FDLは、初期化を実行していない状態、または不正な状態です
		処置	さらなるFDL操作を中止し、原因を調査してください

事前条件：・R_FDL_Init 関数を実行し、R_FDL_Init 関数の戻り値が R_FDL_OK となっていること。
・status_enu が R_FDL_SUSPENDED となっていること。

事後条件：status_enu が R_FDL_SUSPENDED 中は、R_FDL_Handler 関数を繰り返し実行してください。

説明：サスペンド状態からの復帰を要求する関数です。再開を要求できるのはこの関数だけです。
再開処理は R_FDL_Handler 関数により実行されます。

例：R_FDL_SuspendRequest を参照してください。

4.4.3.3. R_FDL_StandBy

概要：FDL実行中にスタンバイ状態にする関数です。

インタフェース：Cインタフェース

```
r_fdl_status_t R_FDL_StandBy (void);
```

引数：なし

戻り値：

型	説明		
r_fdl_status_t	R_FDL_OK	説明	スタンバイを開始しました
		原因	正常動作です
		処置	無し
	R_FDL_BUSY	説明	正常にスタンバイ要求を受け付けました
		原因	正常動作です
		処置	操作を続けてください
	R_FDL_ERR_REJECTED	説明	今回のR_FDL_StandBy関数の実行を拒否しました
		原因	FDLは、初期化を実行していない状態、または不正な状態です
		処置	さらなるFDL操作を中止し、原因を調査してください

事前条件：・R_FDL_Init関数を実行し、R_FDL_Init関数の戻り値がR_FDL_OKとなっていること。
・すでにstatus_enuがR_FDL_BUSYとなっていること。
・スタンバイ状態でないこと。

事後条件：・R_FDL_StandBy関数の戻り値がR_FDL_BUSY中はR_FDL_StandBy関数を繰り返し実行してください。
・スタンバイ後、次回実行可能なFDL関数はR_FDL_WakeUp関数と
R_FDL_GetVersionString関数です。

説明：FDL実行中にスタンバイ状態にする関数です。実行中のコマンド操作を中断し、スタンバイ状態中はR_FDL_WakeUp関数とR_FDL_GetVersionString関数以外のFDL関数、コマンド実行はできません。

注意　フラッシュプログラミングハードウェアのリセットや、RAMのデータを保持できない低消費で動作するモードに入ると、以前の動作を再開できなくなるので、このようなモードに入らないように注意してください。FDLはその故障を検出できません。

例：

```
r_fdl_status_t    fdlRet_enu;
r_fdl_request_t myReq_str_str;

/* Start Erase operation */
myReq_str_str.command_enu        = R_FDL_CMD_ERASE;
myReq_str_str.idx_u32            = 0;
myReq_str_str.cnt_u16            = 4;
myReq_str_str.accessType_enu    = R_FDL_ACCESS_USER;

R_FDL_Execute (&myReq_str_str);

...
do
{
      fdlRet = R_FDL_StandBy ();
}
while (R_FDL_BUSY == fdlRet);
if (R_FDL_OK != fdlRet)
{
      /* error handler */
}

...
/* device enters power save mode */
...

...
/* device recovers from power save mode */
...

★ do
{
      fdlRet = R_FDL_WakeUp ();
}
while (R_FDL_BUSY == fdlRet);
if (R_FDL_OK != fdlRet)
{
      /* error handler */
}

/* Finish erase command */

while (myReq_str_str.status_enu == R_FDL_BUSY)
{
      R_FDL_Handler ();
}

if (R_FDL_OK != myReq_str_str.status_enu)
{
      /* Error handler */
      while (1)
            ;
}
```

★ 4.4.3.4. R_FDL_WakeUp

概要：FDLのスタンバイ状態から復帰する関数です。

インタフェース：Cインタフェース

```
r_fdl_status_t R_FDL_WakeUp (void);
```

引数：なし

戻り値：

型	説明		
r_fdl_status_t	R_FDL_OK	説明	正常にスタンバイから復帰しました
		原因	正常動作です
		処置	無し
	R_FDL_BUSY	説明	正常にスタンバイからの復帰の要求を受け付けました
		原因	正常動作です
		処置	操作を続けてください
	R_FDL_ERR_REJECTED	説明	今回のR_FDL_WakeUp関数の実行を拒否しました
		原因	FDLは、初期化を実行していない状態、または不正な状態です
		処置	さらなるFDL操作を中止し、原因を調査してください

事前条件：・R_FDL_Init関数を実行し、R_FDL_Init関数の戻り値がR_FDL_OKとなっていること。
・R_FDL_StandBy関数の戻り値がR_FDL_BUSY から変化しR_FDL_OKとなり、スタンバイ状態であること。
・status_enuがR_FDL_BUSYとなっていること。
・ハードウェアの条件（CPUの周波数、電圧など）を変更している場合、スタンバイ要求を出す前の状態に復元しておくこと。

事後条件：・R_FDL_WakeUp関数の戻り値がR_FDL_BUSY中はR_FDL_WakeUp関数を繰り返し実行してください。

説明：FDLのスタンバイ状態から復帰する関数です。

例：R_FDL_StandByを参照してください。

4.4.3.5. R_FDL_CancelRequest

概要：実行中の消去、書き込み、ブランクチェック、サスペンド動作のキャンセルを要求する。

インタフェース：Cインタフェース

```
r_fdl_status_t R_FDL_CancelRequest (void)
```

引数：なし

戻り値：

型	説明		
r_fdl_status_t	R_FDL_OK	説明	正常にキャンセル要求を受け付けました
		原因	正常動作です
		処置	無し
	R_FDL_ERR_REJECTED	説明	今回のFDL関数の実行を拒否しました
		原因	以下のいずれかの状態です ・R_FDL_CancelRequest関数が実行不可能なコマンドが実行中 ・FDLの初期化を実行していない状態 ・R_FDL_BUSYでない状態 ・サスペンドではない状態
		処置	さらなるFDL操作を中止し、根本的な原因を調査してください
	R_FDL_ERR_PROTECTION	説明	今回のコマンドの実行を拒否しました
		原因	コマンド実行中にFDL内部のパラメータ、またはデータフラッシュを制御するハードウェアにFDL経由で設定したパラメータが不正に改変されました
		処置	さらなるFDL操作を中止し、原因を調査してください
	R_FDL_ERR_INTERNAL	説明	今回のコマンドの実行は正常に終了しませんでした
		原因	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1)ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2)FDLが破壊されている可能性もあります
		処置	さらなるFDL操作を中止し、原因を調査してください

事前条件：すでにstatus_enuがR_FDL_CMD_ERASEコマンド、R_FDL_CMD_WRITEコマンド、R_FDL_CMD_BLANKCHECKコマンドによってR_FDL_BUSYとなっていること、またはサスペンド中であること。

事後条件：status_enuがR_FDL_BUSY中は、繰り返しR_FDL_Handler関数を実行してください。

説明：FDLの操作をキャンセルします。キャンセル可能な状態はR_FDL_CMD_ERASEコマンド実行中、R_FDL_CMD_WRITEコマンド実行中、R_FDL_CMD_BLANKCHECKコマンド実行中またはサスペンド状態です。サスペンド状態に対してR_FDL_CancelRequest関数を実行した場合、サスペンド状態の前に実行していたR_FDL_CMD_ERASEコマンド、R_FDL_CMD_WRITEコマンド、R_FDL_CMD_BLANKCHECKコマンドもキャンセルされます。

例：

```
/* Erase block 0,1,2 and 3 */
r_fdl_request_t myRequest ;
r_fdl_status_t srRes_enu ;
uint32_t i ;

myRequest.command_enu = R_FDL_CMD_ERASE
myRequest.idx_u32 = 0
myRequest.cnt_u16 = 4
myRequest.accessType_enu = R_FDL_ACCESS_USER;

R_FDL_Execute(&myRequest);
/* call the handler some time */

i= 0;
while ((myRequest.status_enu == R_FDL_BUSY) && (i<10))
{
      R_FDL_Handler ();
      i++;
}

/* Cancel request and wait until cancelled */

srRes_enu = R_FDL_CancelRequest () ;

if (R_FDL_OK != srRes_enu)
{
/* Error treatment */
      ...
)

while (R_FDL_CANCELLED != myRequest.status_enu)
{
      R_FDL_Handler ();
}
```

4. 4. 4　管理

4.4.4.1. R_FDL_GetVersionString

概要：FDLバージョン情報のポインタを取得します。

インタフェース：Cインタフェース

```
(const uint8_t*) R_FDL_GetVersionString (void);
```

引数：なし

戻り値：

型	説明
const uint8_t *	アドレス値

事前条件：なし

事後条件：なし

説明：

FDL のバージョンは次の形式の文字列値：“DH850T01xxxxxYZabcD”

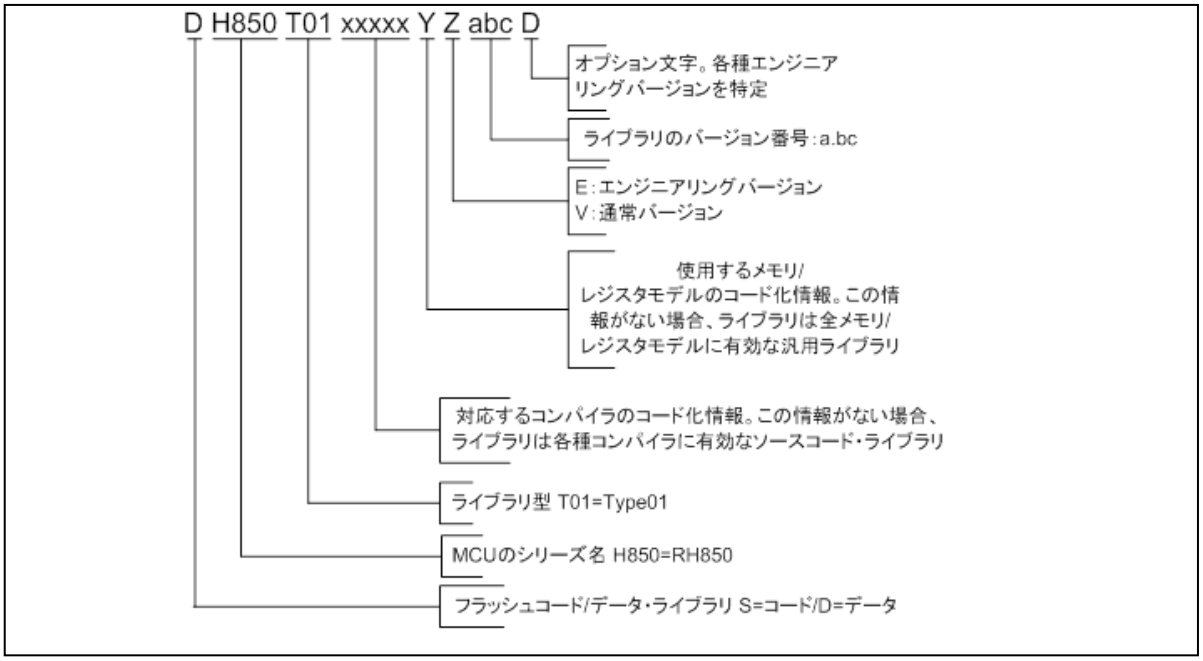


図 14　バージョン文字列

例：

```
uint8_t * vstr = (uint8_t *)R_FDL_GetVersionString ();
```

4.5　コマンド

下記に本 FDL の基本フローチャートを示します。

1. リクエスト構造体を設定します。
2. R_FDL_Execute 関数を使ってコマンドの実行を開始します。
3. status_enu が R_FDL_BUSY の間、R_FDL_Handler を呼び出して FDL コマンドの実行を進めます。
4. status_enu が R_FDL_BUSY から変化した後、正常終了ならば終了し、エラーであればエラーを解析してください。

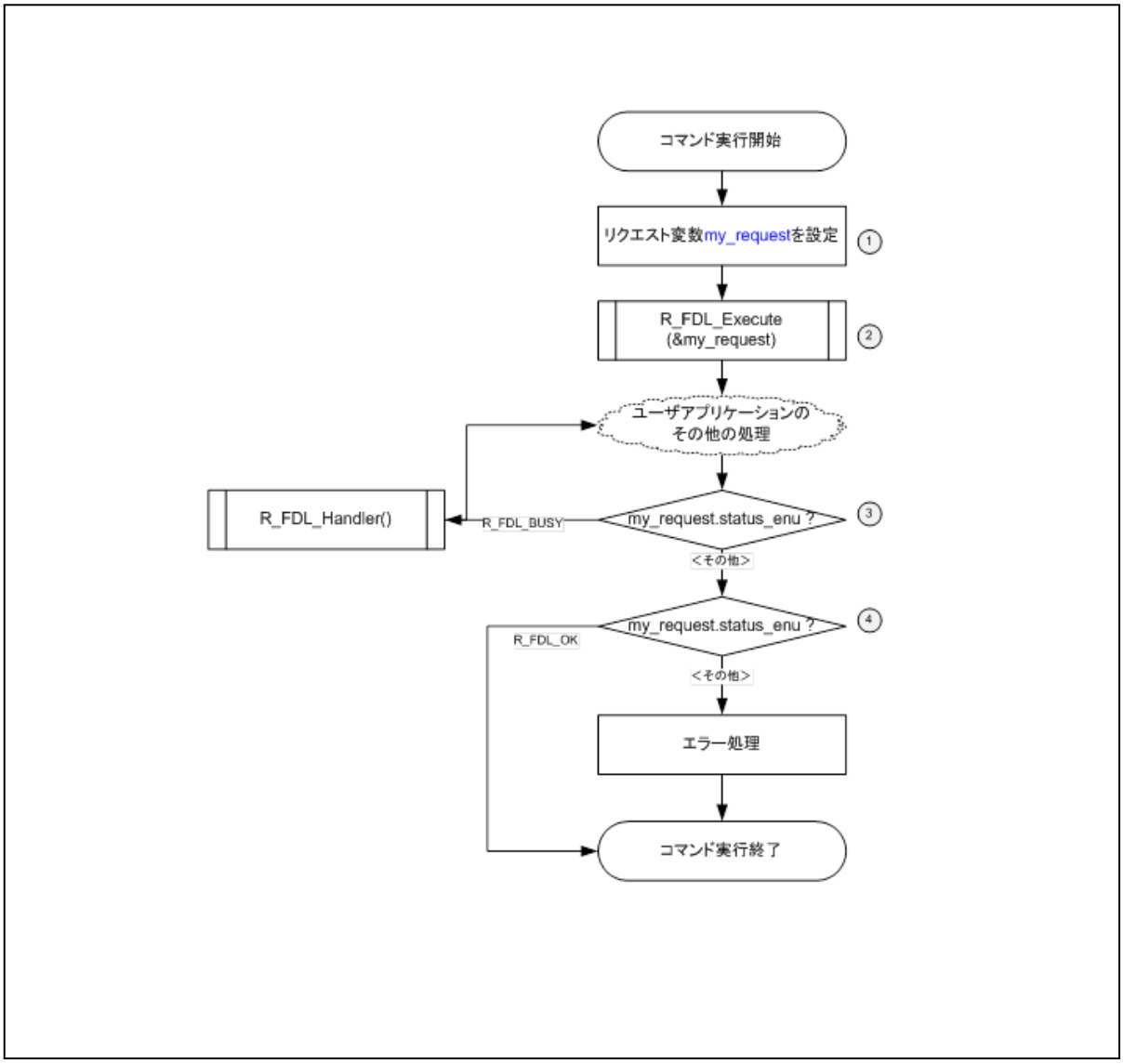


図 15　一般的なコマンドの実行フロー

4. 5. 1　R_FDL_CMD_ERASE

先頭ブロックとブロック数で指定された複数のフラッシュブロックを消去するために使用するコマンドです。R_FDL_Execute に提供されるリクエスト構造体のメンバを下表に記載します。

表 3　消去コマンドに使用されるリクエスト構造体

構造体のメンバ	値	説明
command_enu	R_FDL_CMD_ERASE	ブロックの消去動作を要求します
bufAddr_u32	-	使用しません
idx_u32	例1）ブロック0の場合 設定値：0 例2）　ブロック1の場合 設定値：1	消去する先頭ブロックの番号。データフラッシュのブロックは、64 バイトの消去単位で次のように指定されます。 例） ブロック 0：0x00 ... 0x3F ブロック 1：0x40 ... 0x7F
cnt_u16	例）　3ブロックを消去する場合 設定値：3	消去するブロック数
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	コマンドが実行される FDL プールを選択します
status_enu	-	コマンドの実行中および実行後の動作の状態が入っています。可能性のある値を「表 4　消去動作からの状態戻り値」で説明します

戻る可能性のあるすべての状態を「表 4　消去動作からの状態戻り値」に示します。

表 4　消去動作からの状態戻り値

状態	背景と対応	
R_FDL_BUSY	説明	コマンド操作は正常に開始しました または実行中です
	原因	正常動作です
	処置	操作を続けてください
R_FDL_OK※1	説明	正常終了しました
	原因	正常動作です
	処置	無し
R_FDL_SUSPENDED※1	説明	サスペンド中です
	原因	正常動作です
	処置	無し
R_FDL_CANCELLED※1	説明	実行中だった FDL 操作をキャンセルしました
	原因	正常動作です
	処置	無し
R_FDL_ERR_PARAMETER	説明	今回のコマンドの実行を拒否しました
	原因	設定しているパラメータが不正です
	処置	さらなる FDL 操作を中止し、原因を調査してください
R_FDL_ERR_PROTECTION	説明	今回のコマンドの実行を拒否しました
	原因	1) 保護（FHVE15 レジスタ/ FHVE3 レジスタ）している領域/設定に対して、禁止されているコマンドを実行しました 2) FDL 内部のパラメータ、またはデータフラッシュを制御するハードウェアに FDL 経由で設定したパラメータがコマンド実行中に不正に改変されました
	処置	1) FHVE15 レジスタ/ FHVE3 レジスタの設定をご確認ください 2) さらなる FDL 操作を中止し、原因を調査してください
R_FDL_ERR_REJECTED※2	説明	今回のコマンドの実行を拒否しました
	原因	すでに他のコマンド、FDL 関数が実行中です
	処置	実行中のコマンドの終了後に再度今回実行しようとしたコマンドを実行してください
R_FDL_ERR_ERASE※1	説明	消去が失敗しました
	原因	データフラッシュが故障している可能性があります
	処置	さらなる FDL 操作を中止し、原因を調査してください
R_FDL_ERR_INTERNAL	説明	今回のコマンドの実行は正常に終了しませんでした
	原因	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1)ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2)FDLが破壊されている可能性もあります
	処置	さらなる FDL 操作を中止し、原因を調査してください

※1 R_FDL_Execute ではこの状態コードは設定されません。

※2 R_FDL_Handler ではこの状態コードは設定されません。

4. 5. 2　R_FDL_CMD_WRITE

内蔵 RAM 上の書き込みデータを、仮想アドレスで指定された場所に書き込むためのコマンドです。

注意：データが書かれている領域に消去をせずデータを上書きすることは禁止です。書き込み先の領域を必ず消去し、データを書き込んでください。

R_FDL_Execute に提供されるリクエスト構造体のメンバを下表に示します。

表 5　書き込みコマンドに使用されるリクエスト構造体

構造体のメンバ	値	説明
command_enu	R_FDL_CMD_WRITE	書き込み動作を要求します
bufAddr_u32	例）データバッファ名 “data”の配列の場合 bufAddr_u32=&data[0];	データバッファ※ の先頭アドレス
idx_u32	例1） データフラッシュのブロック0の先頭から書き込みを実行する場合 設定値 ： 0x0000_0000 例2） データフラッシュのブロック1の先頭+4バイトから書き込みを実行する場合 設定値 ： 0x0000_0044 例3） データフラッシュのブロック0の先頭+3バイトの下記アドレスを設定し実行した場合、4バイトアラインではない為、status_enuはR_FDL_ERR_PARAMETERとなります 設定値 ： 0x0000_0003	書き込みの開始アドレスはデータフラッシュのブロック0の先頭アドレスを0x0000_0000とした仮想アドレスで指定します なお、設定可能なアドレスはデータフラッシュのブロック0の先頭アドレスを0x0000_0000とした4バイトアラインのアドレス（アドレスの最下位が0H, 4H, 8H, CH）です
cnt_u16	例）8バイトの場合 設定値：2	書き込みを実行するデータ数を設定してください [4 バイト単位]
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	コマンドが実行される FDL プールを選択します
status_enu	-	コマンドの実行中および実行後の動作の状態が入っています。可能性のある値を「表 6　書き込み動作からの状態戻り値」で説明します

※　ユーザの書き込むデータが配置されているバッファ

戻る可能性のあるすべての状態を「表 6　書き込み動作からの状態戻り値」に示します。

表 6　書き込み動作からの状態戻り値

状態	背景と対応	
R_FDL_BUSY	説明	コマンド操作は正常に開始しました または実行中です
	原因	正常動作です
	処置	操作を続けてください
R_FDL_OK※1	説明	正常終了しました
	原因	正常動作です
	処置	無し
R_FDL_SUSPENDED※1	説明	サスペンド中です
	原因	正常動作です
	処置	無し
R_FDL_CANCELLED※1	説明	実行中だったFDL操作をキャンセルしました
	原因	正常動作です
	処置	無し
R_FDL_ERR_PARAMETER	説明	今回のコマンドの実行を拒否しました
	原因	設定しているパラメータが不正です
	処置	さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_PROTECTION	説明	今回のコマンドの実行を拒否しました
	原因	1) 保護（FHVE15レジスタ/ FHVE3レジスタ）している領域/設定に対して、禁止されているコマンドを実行しました 2) FDL内部のパラメータ、またはデータフラッシュを制御するハードウェアにFDL経由で設定したパラメータがコマンド実行中に不正に改変されました
	処置	1) FHVE15レジスタ/ FHVE3レジスタの設定をご確認ください 2)さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_REJECTED※2	説明	今回のコマンドの実行を拒否しました
	原因	すでに他のコマンド、FDL関数が実行中です
	処置	実行中のコマンドの終了後に再度今回実行しようとしたコマンドを実行してください
R_FDL_ERR_WRITE※1	説明	書き込みが失敗しました
	原因	1)データフラッシュが故障している可能性があります 2)書き込み先が消去した状態でない可能性があります
	処置	1)書き込み先が消去されていなかった場合 消去を実行後、再度書き込みを実行してください 2)書き込み先が消去されていた場合 さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_INTERNAL	説明	今回のコマンドの実行は正常に終了しませんでした
	原因	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1)ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2)FDLが破壊されている可能性もあります
	処置	さらなるFDL操作を中止し、原因を調査してください

※1 R_FDL_Execute ではこの状態コードは設定されません。

※2 R_FDL_Handler ではこの状態コードは設定されません。

4. 5. 3　R_FDL_CMD_BLANKCHECK

指定アドレスから指定サイズ分のメモリ領域に対してブランクチェックをするために使用するコマンドです。メモリ内に未消去の場所が初めて現れたとき、コマンド動作を停止して R_FDL_ERR_BLANKCHECK を返します。

注意：

1．　本コマンドで R_FDL_ERR_BLANKCHECK が発生した場合、対象の領域にはデータが書かれています。ただし、これはデータフラッシュへの書き込みが正常に行われた場合のほか、消去や書き込み動作の中断による場合も考えられます。

2．　本コマンドが正常終了の場合、対象の領域にはデータが書かれていません。これは対象領域の消去が正常に行われていた場合のほか、消去や書き込み動作の中断による場合も考えられます。

3．　使用事例として、誤って"データが書かれている領域"のデータとして扱わないように、読み出しの前にブランクチェック処理を実行することで、データが書かれている領域か、データの書かれていない未書き込みの領域かを判別することができます。必要に応じてご使用ください

4．　FDL 内部では、ブランクチェック動作が 0x1000 バイトの境界を越えるたびに、動作を分割します。これは、1 回のブランクチェック処理にかかる時間を短くするためです。

R_FDL_Execute に提供されるリクエスト構造体のメンバを下表に示します。

表 7　ブランクチェックコマンドに使用されるリクエスト構造体

構造体のメンバ	値	説明
command_enu	R_FDL_CMD_BLANKCHECK	ブランクチェック動作を要求します
bufAddr_u32	-	使用しません
idx_u32	例1） データフラッシュのブロック0の先頭からブランクチェックを実行する場合 設定値 ： 0x0000_0000 例2） データフラッシュのブロック1の先頭+4バイトからブランクチェックを実行する場合 設定値 ： 0x0000_0044 例3） データフラッシュのブロック0の先頭+3バイトの下記アドレスを設定し実行した場合､4バイトアラインではない為、status_enuはR_FDL_ERR_PARAMETER となります 設定値 ： 0x0000_0003	ブランクチェックの開始アドレスはデータフラッシュのブロック0の先頭アドレスを0x0000_0000とした仮想アドレスで指定します なお、設定可能なアドレスはデータフラッシュのブロック0の先頭アドレスを0x0000_0000とした4バイトアラインのアドレス（アドレスの最下位が0H, 4H, 8H, CH）です また、R_FDL_ERR_BLANKCHECK発生時、ブランクチェックに失敗したアドレスを取得します
cnt_u16	例） 8バイトの場合 設定値：2	ブランクチェックを実行するデータ数を設定してください [4バイト単位]
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	コマンドが実行される FDL プールを選択します
status_enu	-	コマンドの実行中および実行後の動作の状態が入っています。可能性のある値を「表 8　ブランクチェック動作からの状態戻り値」で説明します

戻る可能性のあるすべての状態を「表 8　ブランクチェック動作からの状態戻り値」に示します。

表 8　ブランクチェック動作からの状態戻り値

状態	背景と対応	
R_FDL_BUSY	説明	コマンド操作は正常に開始しました または実行中です
	原因	正常動作です
	処置	操作を続けてください
R_FDL_OK※1	説明	正常終了しました
	原因	正常動作です
	処置	無し
R_FDL_SUSPENDED※1	説明	サスペンド中です
	原因	正常動作です
	処置	無し
R_FDL_CANCELLED※1	説明	実行中だったFDL操作をキャンセルしました
	原因	正常動作です
	処置	無し
R_FDL_ERR_PARAMETER	説明	今回のコマンドの実行を拒否しました
	原因	設定しているパラメータが不正です
	処置	さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_PROTECTION	説明	今回のコマンドの実行を拒否しました
	原因	1) 保護（FHVE15レジスタ/ FHVE3レジスタ）している領域/設定に対して、禁止されているコマンドを実行しました 2) FDL内部のパラメータ、またはデータフラッシュを制御するハードウェアにFDL経由で設定したパラメータがコマンド実行中に不正に改変されました
	処置	1) FHVE15レジスタ/ FHVE3レジスタの設定をご確認ください 2)さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_REJECTED※2	説明	今回のコマンドの実行を拒否しました
	原因	すでに他のコマンド、FDL関数が実行中です
	処置	実行中のコマンドの終了後に再度今回実行しようとしたコマンドを実行してください
R_FDL_ERR_BLANKCHECK※1	説明	指定した領域が消去されていません
	原因	データが書き込まれています
	処置	1)データが書かれていない状態を期待していた場合 データが書き込まれている状態ため、消去を実行してください 2)データが書かれている状態を期待していた場合 操作を続けてください
R_FDL_ERR_INTERNAL	説明	今回のコマンドの実行は正常に終了しませんでした
	原因	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1)ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2)FDLが破壊されている可能性もあります
	処置	さらなるFDL操作を中止し、原因を調査してください

※1 R_FDL_Execute ではこの状態コードは設定されません。

※2 R_FDL_Handler ではこの状態コードは設定されません。

4. 5. 4　R_FDL_CMD_READ

読み出し動作はデータフラッシュ内の指定された領域を読み出して、そのデータを指定されたデータバッファにコピーします。

FDL の読み出し動作は ECC エラーを発生させる割り込みの生成を一時的に無効にします。読み出し動作が終了すると、ECC 割り込みの生成状態は復元されます。読み出し動作中に検出されたエラーは、status_enu 変数と idx_u32 変数によってユーザアプリケーションに伝えられます。

1 ビットエラーの場合、データの読み出しが続けられ ECC エラーの最初の発生が返されます。2 ビットエラーの場合は、読み出し動作が停止して、エラーのあるアドレスが返されます。先に 1 ビットエラーが検出されていた場合、そのエラーが生じたアドレスは上書きされます。

読み出しコマンドは、R_FDL_Execute 関数の実行に同期して実行されます。したがって、このコマンドは R_FDL_Handler 関数を実行する必要がありませんので、スタンバイ及びサスペンドはできません。

R_FDL_Execute に提供されるリクエスト構造体のメンバを「表 9　読み出しコマンドに使用されるリクエスト構造体」に示します。

表 9　読み出しコマンドに使用されるリクエスト構造体

構造体のメンバ	値	説明
command_enu	R_FDL_CMD_READ	読み出し動作を要求します
bufAddr_u32	例） データバッファ名 “data”の配列の場合 bufAddr_u32=&data[0];	データバッファ の先頭アドレス バッファは 32 ビットにアラインする必要があります
idx_u32	例1） データフラッシュのブロック0の先頭から読み出しを実行する場合 設定値 ： 0x0000_0000 例2） データフラッシュのブロック1の先頭+4バイトから読み出しを実行する場合 設定値 ： 0x0000_0044 例3） データフラッシュのブロック0の先頭+3バイトの下記アドレスを設定し実行した場合、4バイトアラインではない為、status_enu は R_FDL_ERR_PARAMETER となります 設定値 ： 0x0000_0003	読み出しの開始アドレスはデータフラッシュのブロック0の先頭アドレスを0x0000_0000とした仮想アドレスで指定します なお、設定可能なアドレスはデータフラッシュのブロック0の先頭アドレスを0x0000_0000とした4バイトアラインのアドレス（アドレスの最下位が0H, 4H, 8H, CH）です また、R_FDL_ERR_ECC_SED、R_FDL_ERR_ECC_DED 発生時、エラーを検出したアドレスを取得します
cnt_u16	例） 8バイトの場合 設定値：2	読み出しを実行するデータ数を設定してください[4バイト単位]
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	コマンドが実行される FDL プールを選択します
status_enu	-	コマンドの実行中および実行後の動作の状態が入っています。可能性のある値を「表 10 読み出し動作からの状態戻り値」で説明します

※　データフラッシュから読み出したデータを配置するバッファ

戻る可能性のあるすべての状態を「表 10　読み出し動作からの状態戻り値」に示します。

表 10　読み出し動作からの状態戻り値

状態	背景と対応	
R_FDL_OK	説明	正常終了しました
	原因	正常動作です
	処置	無し
R_FDL_ERR_PARAMETER	説明	今回のコマンドの実行を拒否しました
	原因	設定しているパラメータが不正です
	処置	さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_PROTECTION	説明	今回のコマンドの実行を拒否しました
	原因	コマンド実行中にFDL内部のパラメータ、またはデータフラッシュを制御するハードウェアにFDL経由で設定したパラメータが不正に改変されました
	処置	さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_REJECTED	説明	今回のコマンドの実行を拒否しました
	原因	すでに他のコマンド、FDL関数が実行中です
	処置	実行中のコマンドの終了後に再度今回実行しようとしたコマンドを実行してください
R_FDL_ERR_ECC_SED	説明	FDL操作は実行され、訂正されたデータが読み出されました
	原因	ECC エラー（1ビットエラー)※を検出しました
	処置	さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_ECC_DED	説明	FDL操作は実行され、訂正できなかったデータが読み出されました
	原因	ECC エラー（2ビットエラー)※を検出しました
	処置	さらなるFDL操作を中止し、原因を調査してください
R_FDL_ERR_INTERNAL	説明	今回のコマンドの実行は正常に終了しませんでした
	原因	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1)ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2)FDLが破壊されている可能性もあります
	処置	さらなるFDL操作を中止し、原因を調査してください

※　読み出しコマンドの実行中に ECC エラーが発生したレジスタの処理について「図 16　読み出しコマンド実行時のECCエラーレジスタの処理」に示します。

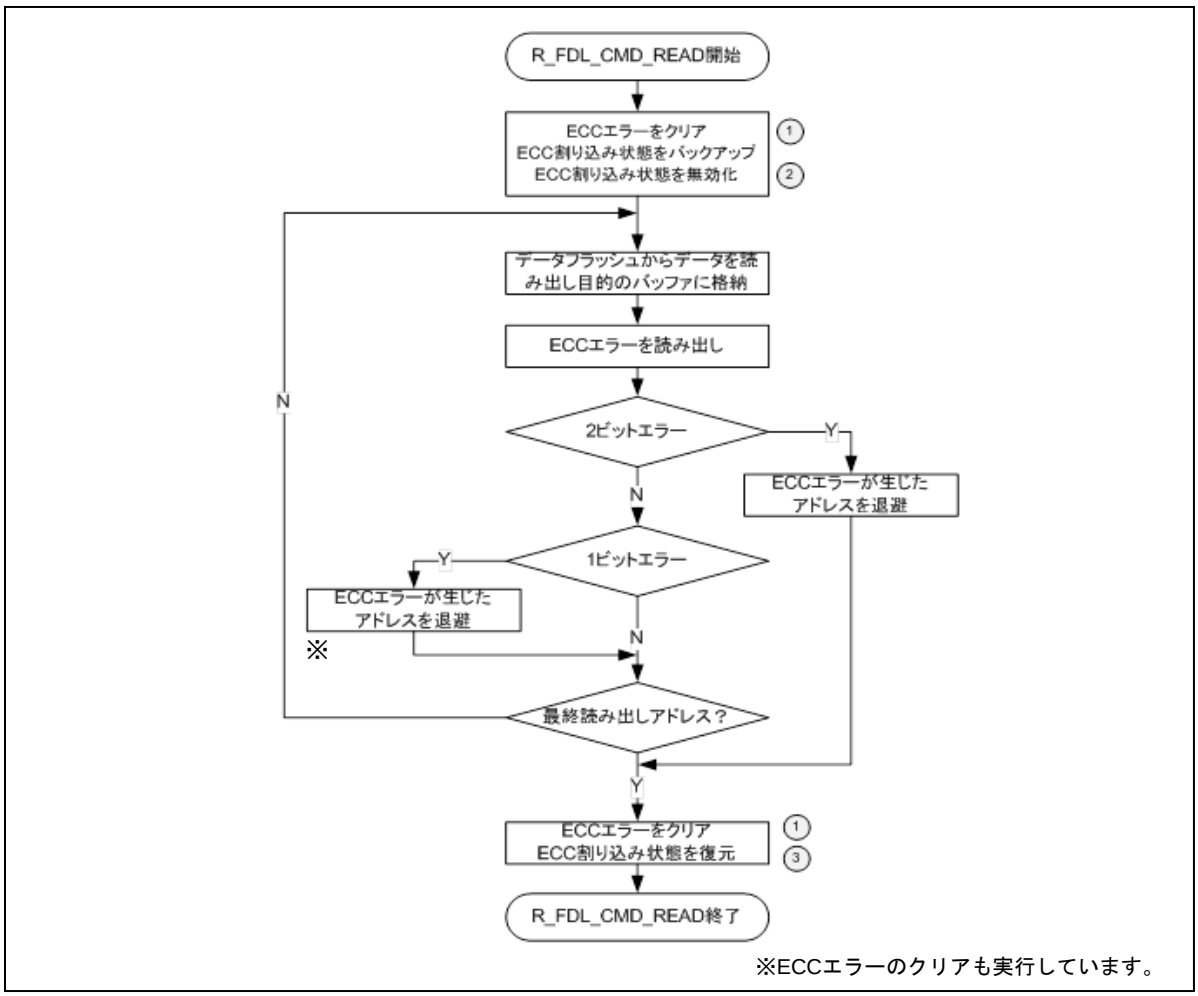


図 16　読み出しコマンド実行時の ECC エラーレジスタの処理

以下のレジスタが書き換えられることを考慮してください。

1. DFFSTERSTR 内のエラーをクリアするために、DFERSTC レジスタへの書き込みが行われます。
2. DFERRINT レジスタのバックアップと無効化が行われます。
3. DFERRINT レジスタにバックアップしたデータが復帰されます。

4. 5. 5　R_FDL_CMD_PREPARE_ENV

FDLの実行環境を準備します。

- フラッシュプログラミングハードウェアの初期化
- ランタイム設定値の確認（ CPU動作周波数など）
- FDLが使用するプログラムをRAMにコピーします。ただし、RH850/F1K, F1KM, F1KH, D1M1A, D1M1-V2, D1S1は対象外です。

以下にリクエスト構造体の設定方法を示します。

表 11　リクエスト構造体の設定

リクエスト構造体	値	説明
command_enu	R_FDL_CMD_PREPARE_ENV	実行環境の準備動作を要求します
bufferAdd_u32	設定不要	
idx_u32	設定不要	
cnt_u16	設定不要	
accessType_enu	設定不要	
status_enu	-	コマンドの実行中および実行後の動作の状態が入っています。可能性のある値を「表 12　本コマンドが取り得るstatus_enuの値」で説明します

戻る可能性のあるすべての状態を「表 12　本コマンドが取り得るstatus_enuの値」に示します。

表 12　本コマンドが取り得る status_enu の値

状態	背景と対応	
R_FDL_OK※1	説明	正常終了しました
	原因	正常動作です
	処置	無し
R_FDL_BUSY	説明	コマンド操作は正常に開始しました。または実行中です
	原因	正常動作です
	処置	操作を続けてください
R_FDL_ERR_REJECTED※2	説明	今回のコマンドの実行を拒否しました
	原因	他の FDL 関数、コマンドが実行中です
	処置	さらなる FDL 操作を中止し、根本的な原因を調査してください
R_FDL_ERR_CONFIGURATION※1	説明	今回のコマンドの実行を拒否しました
	原因	設定しているパラメータ（CPU 動作周波数）が不正です
	処置	さらなる FDL 操作を中止し、根本的な原因を調査してください
R_FDL_ERR_INTERNAL※1	説明	通常発生しない予期せぬエラーです
	原因	FDLでは原因を判別できない通常発生しない予期せぬエラーです 1)ユーザプログラムによって、FDLが使用するRAMを破壊している可能性があります 2)FDLが破壊されている可能性もあります
	処置	さらなる FDL 操作を中止し、根本的な原因を調査してください
R_FDL_ERR_PROTECTION	説明	今回のコマンドの実行を拒否しました
	原因	コマンド実行中に FDL 内部のパラメータ、またはデータフラッシュを制御するハードウェアに FDL 経由で設定したパラメータが不正に改変されました
	処置	さらなる FDL 操作を中止し、原因を調査してください

※1 R_FDL_Execute ではこの状態コードは設定されません。

※2 R_FDL_Handler ではこの状態コードは設定されません。

第5章　FDLの設定方法と使用方法

本章には、FDL を動作させる方法と、ユーザアプリケーションに組み込む方法に関する重要な情報を記載しています。FDL の問題や誤動作を回避するために、本章および「第6章　注意事項」をよくお読みください。FDL をユーザのプロジェクトに組み込む前に、FDL 使用方法など必ず読んで理解してください（「第2章　アーキテクチャ」と「第3章　機能仕様」参照）

5. 1　FDLの入手

本 FDL、本ユーザマニュアルは、常に最新バージョンのご使用を推奨します。

5. 2　ファイル構成

対象デバイスでの FDL の実装方法と使用方法を紹介するサンプルアプリケーションと FDL を含むコンパイル可能なサンプルプロジェクトとして、本 FDL は供給されます。

5. 2. 1　概要

FDL とサンプルアプリケーションに関するファイルを下図に示します。

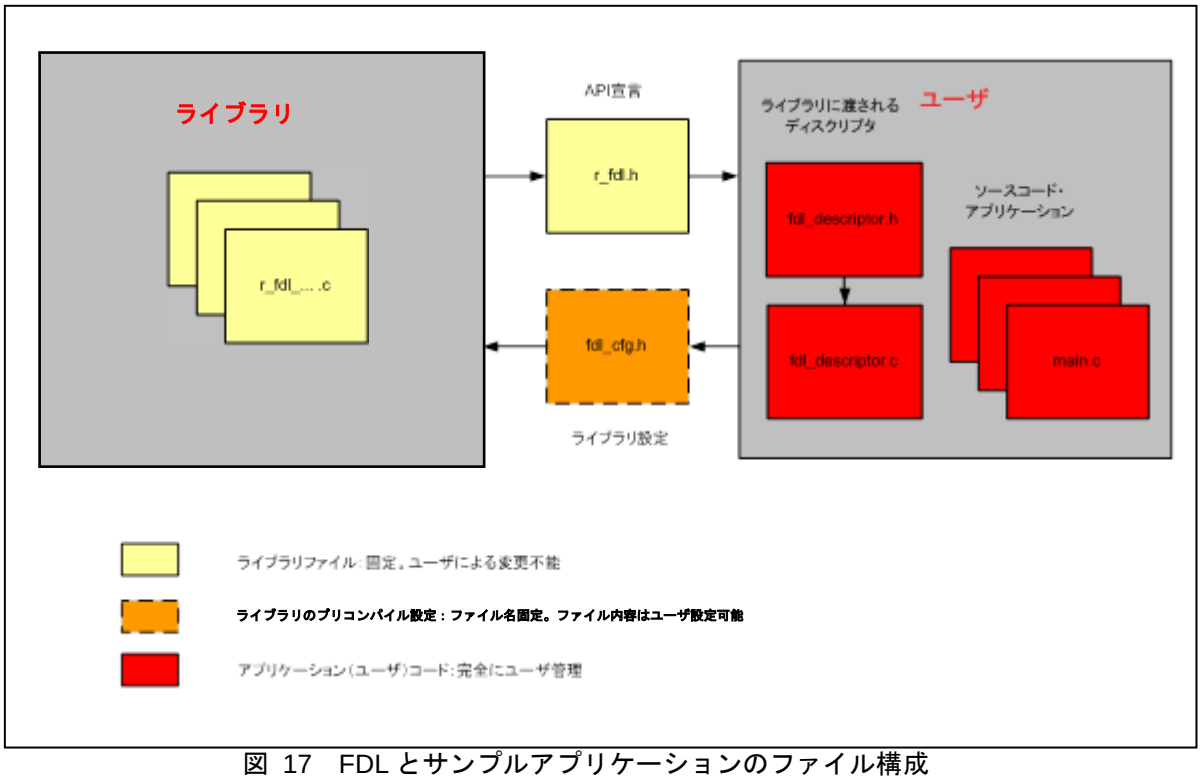


図 17　FDL とサンプルアプリケーションのファイル構成

FDL コードは r_fdl_...で始まる各種ソースファイルで構成されています。これらのファイルをユーザが自由に変更することはできません。

FDL はソースコードで提供されているため、FDL をコンパイルのための設定をする必要があります。ファイル fdl_cfg.h などにはそのための定義が含まれています。これらのファイルは FDL のソースファイルなので、ファイル内容はユーザにより変更可能ですが、ファイル名は変更できません。

5. 2. 2　パッケージのファイルシステム構成

FDL のインストーラを使ってインストールされる全ファイルを「表 13　FDLパッケージのファイル構成」に示します。

- 赤字のファイルはビルド環境に属しており、コンパイル、リンク、ターゲットのビルド処理を制御します
- 青字のファイルはサンプルアプリケーションに属しています
- 緑字のファイルは説明のみです
- 黒字のファイルは FDL に属しています

表 13　FDL パッケージのファイル構成

ファイル名		内容
<installation_folder>/FDL		
Release.txt		リリースノート
support.txt		サポートデバイス一覧
<installation_folder>/FDL /<compiler>/<device_name>		
Build.bat		FDL のサンプルアプリケーションをビルドするためのバッチファイル
Clean.bat		FDL のサンプルアプリケーションを削除するためのバッチファイル
Makefile		ビルド処理と削除処理を制御する Make ファイル
<installation_folder>/FDL /<compiler>/<device_name>/Sample(1)		
dr7f70xxxx_startup.850(1)	<GHS 用>	デバイスとコンパイラに固有のスタートアップコード
cstart.asm	<CC-RH 用>	
dr7f70xxxx.ld(1)	<GHS 用>	コンパイラ固有のリンクディレクティブ
dr7f70xxxx.dir(1)	<CC-RH 用>	
dr7f70xxxx.dvf.h (1) dr7f70xxxx_irq.h(1)	<GHS 用>	デバイス固有のヘッダファイル <GHS 用>は"dr7f70xxxx.dvf.h(1) "、または"dr7f70xxxx_0.h(1)"と"io_macros_v2.h"を使用します。 <CC-RH 用>は"boot.asm"、または"vecttbl.asm".を使用します。
iodefine.h boot.asm	<CC-RH 用>	
app.h		サンプルアプリケーションのコード
fdlapp_control.c		
fdlapp_main.c		
target.h		ターゲットマイクロコントローラの初期化コード
fdl_cfg.h		FDL のプリコンパイル定義
fdl_descriptor.c		サンプルアプリケーションで使用される FDL のディスクリプタ
fdl_descriptor.h		
fdl_user.c		割り込みと FDL の事前の初期化を処理するためのユーザ定義コード
fdl_user.h		
<installation_folder>/FDL /<compiler>/FDL		
r_fdl.h		FDL の API の定義
r_fdl_types.h		ユーザインタフェースによる型定義、エラーコードと状態コード
<installation_folder>/FDL /<compiler>/FDL/lib		
r_typedefs.h		FDL で使用される C 型
r_fdl_mem_map.h		セクションのマッピング定義

ファイル名	内容
r_fdl_env.h	FDL の内部定義
r_fdl_global.h	グローバル変数と設定
r_fdl_hw_access.c	FDL の主要ソースコード
r_fdl_user_if.c	
r_fdl_user_if_init.c	

(1) ファイル名は、選択したデバイス型名に依存します。ここで示すファイル名は、RH850/F1L の例です。

例) デバイスが R7F701007 の場合、dr7f70xxxx_startup.850 は、dr7f701007_startup.850 です。

(2) RH850 データフラッシュライブラリ(FDL) Type01 に同梱のバッチファイルに記載の make.exe ファイルは外部ツールで、make.exe を提供しているサイトからダウンロードする必要があります。添付 release.txt に記述の通り、サンプルアプリケーションは、GNU Make を使用して動作確認をしています。同等の環境でご使用いただく場合は、GNU の Web サイトから make.exe をダウンロード、インストールし、添付のバッチファイルを実行してください。

5. 3　リンカセクション

以下に示すセクションは FDL に関連しており、リンカファイルで定義される必要があります（例については、サンプルのリンカディレクティブファイルを参照してください）。

FDL のデータセクション	
R_FDL_Data	FDL の内部変数です。内蔵 RAM に配置できます。

FDL のコンストセクションとテキストセクション	
R_FDL_Const	FDL の内部定数データです。コードフラッシュ内に配置できます。
R_FDL_Text	FDL のコードです。コードフラッシュ内に配置できます。

5. 3. 1　FDL のリソース

FDL V2.13 が使用するリソースの参考値（下記プリコンパイル設定時※1※2 における一例）を下記に示します。なお、プリコンパイル設定に関しましては「4. 1　プリコンパイル設定」をご参照ください。

また、ビルド条件によってリソースサイズは増減しますのでご注意ください。

【対象バージョン V2.13※1】

★

項目名	サイズ　[バイト]	
	GHS（22レジスタモード）	CC-RH（22レジスタモード）
R_FDL_Textセクション	5212	5650
R_FDL_Constセクション	19	19
R_FDL_Dataセクション	84	104
R_FDL_CodeRamセクション	256	256
FDLスタック最大使用容量※2,3	132	164

※1　上表のプリコンパイル設定条件：R_FDL_LIB_V1_COMPATIBILITY＝有効、
R_FDL_NO_BFA_SWITCH＝無効、R_FDL_EXE_INIT_CODE_ON_STACK＝無効

※2　「R_FDL_EXE_INIT_CODE_ON_STACK＝有効」に設定した場合、R_FDL_CodeRamセクションは0バイトとなり、FDLスタック最大使用容量が（約R_FDL_CodeRamセクションのサイズ分）増加しますのでご注意ください。

※3　レジスタモードは、最大と考えられる32レジスタモードで測定しています。

注意　スタック領域は、ユーザが使用するサイズに加え、FDL が使用するサイズを含めて確保してください。また、上記の FDL が使用するリソースを破壊しないでください。

5. 4　MISRA C 対応

本 FDL コードは MISRA C に対応しております。

使用したツールは、MISRA C 2004 標準規則の検査用の QA C ソースコードアナライザです。

注意：

"MISRA", "MISRA C"は、MISRA Consortium を代表して HORIBA MIRA 社が保有する登録商標です。

"QA C"は Programming Research 社の登録商標です。

5. 5　サンプルアプリケーション

FDL をユーザアプリケーションに正しく実装するには、データフラッシュと FDL を理解することがとても重要です。したがって、このユーザマニュアルを FDL ご使用前に読むことが大切です。最良の方法は、ユーザマニュアルをお読みになられた後、FDL のサンプルアプリケーションを試してみることです。

5. 6　FDLの設定

FDL を使用する前に、必ずを「4. 2　ランタイム設定」の章を読み、必要な設定を行ってください。

5. 7　基本的な再プログラミングフロー

データフラッシュの書き換えを実行する基本的なフローを下図に示します。

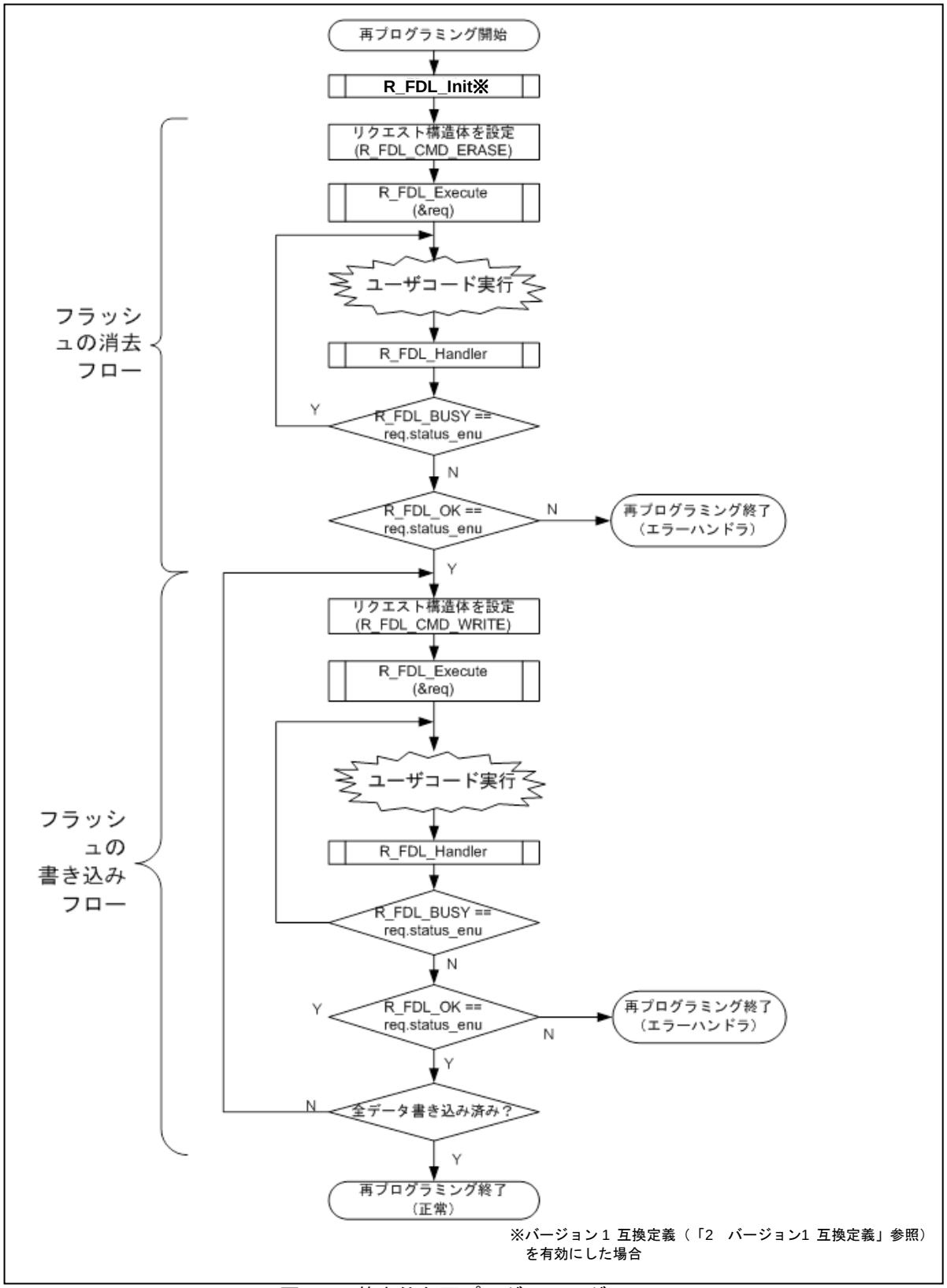


図 18　基本的な再プログラミングフロー

簡略化するため、このフローチャートでは FDL 関数自体のエラー処理について詳述していません。

データフラッシュへのアクセスを有効または無効にする場合の詳細については、ハードウェアのユーザーズマニュアルを参照してください。一例をサンプルアプリケーションのファイル fdl_user.c、関数 FDL_Open および FDL_Close で示しています。

5. 8　R_FDL_Handlerの呼び出し

一度開始した FDL 動作は、ハンドラを連続的に呼び出して進行させる必要があります。
R_FDL_Handler を呼び出すさまざまな方法について、その長所と短所を以下で比較します。

- 動作の実行開始後に R_FDL_Handler を繰り返し呼び出す：

  R_FDL_Handler を、動作状態がビジーでなくなるのを待つためのループ内で実行することで、最良の FDL 動作性能が得られます。ただし、CPU には負荷がかかります。この方法は、本マニュアルに記載されたほとんどのコード例で利用しています。

- 定期タスク内で R_FDL_Handler を呼び出す：

  R_FDL_Handler を一定間隔で呼び出すことにより、他のタスクを CPU が処理している間、FDL コマンドを進行させることができます。状態確認の呼び出しの間隔は FDL 動作の性能に大きく影響する可能性があります。呼び出し間隔が短いほど FDL の動作性能は高まりますが、FDL による CPU への負荷も大きくなります。このトレードオフのため、一般的に推奨できる呼び出し間隔はありません。ユーザアプリケーション毎に解析して個別に調整してください。

- アイドルタスク内で R_FDL_Handler を呼び出す：

  アイドルタスク内で何度も呼び出すようにすると、ハンドラを連続的に呼び出すことができるので、FDL の動作性能を高める可能性があります。ただし、アイドルタスク自体が小さい場合は、FDL の動作性能を高めることは期待できないかもしれません。

注意：

どの FDL 関数もリエントラントに対応しておりません。FDL 関数の実行中に、割り込み処理から別の FDL 関数を呼び出すことはできません。

第6章　注意事項

本 FDL を使用する前に必ず以下の注意事項をご参照ください。

1. 動作周波数の設定：

・FDL を使用する場合、CPU を PLL クロックで動作させてください。FDL では指定した CPU 動作周波数が RH850 製品毎の PLL クロックの下限、上限内に収まっているかをチェックしています。なお、デバイスによってはデータフラッシュに対する操作に対し、CPU 動作周波数に制限がある場合があります。また、FDL 実行中に CPU の周波数を変更しないでください。周波数の変更が必要な場合は、CPU 周波数を変更後、再度初期化から実行してください。

★ ・RH850/F1KM-S4, RH850/F1KH-D8 使用時、FDL で設定しているフラッシュシーケンサ動作周波数は、初期値として CPU 動作周波数(Max 240MHz で使用する CPU クロック)の 1/8 の周波数を設定しています。

fPCLK = 1/8 fCPUCLK_H（CKDIVMD=1 : CPU 動作周波数 : Max 240MHz）

RH850/F1KM-S4, RH850/F1KH-D8 のオプションバイトを CKDIVMD=0（Max 120MHz）に設定して使用する場合は、フラッシュシーケンサ動作周波数を CPU 動作周波数の 1/4 の周波数に切り替えて使用する必要があります。

fPCLK = 1/4 fCPUCLK_H（CKDIVMD=0 : CPU 動作周波数 : Max 120MHz）

FDL V2.13 以降、フラッシュシーケンサ動作周波数を切り替えて、Max 120MHz に対応することが可能です。

切り替え方法は、FDL の”r_fdl_hw_access.c”ファイルの”R_FDL_FCUFct_SetFrequency” 関数内で設定している周波数比の分母を 1/2 にする行を有効にします。

具体的には、”Changing CKDIVMD” のキーワードの後のコメント行に続く、”#if 0”の行を”#if 1”に変更します。

<キーワード>

```
/***********************************************************************
* SAMPLE: Changing CKDIVMD
***********************************************************************/
```

```
#if 0
fDivider = fDivider / 2u;
#endif
```

⇒

```
#if 1
fDivider = fDivider / 2u;
#endif
```

RH850/F1KM-S4, F1KH-D8をCKDIVMD=0 (Max 120MHz)で使用する場合に限り、**”#if 1”**に変更する。

※FDL V2.12 以前は CKDIVMD=0（Max 120MHz）の設定には対応していません。

2. CPU：

・FDL はスーパバイザモード（CPU 動作モード）でのみ使用できます。ユーザモードでは使用できません。

・RH850 デュアル CPU の製品では、メイン CPU（CPU1）でのみ本製品の使用が可能です。パフォーマンス CPU(CPU2)での本製品の使用は禁止です。また、メイン CPU（CPU1）で FDL のコマンドを使用する時、パフォーマンス CPU(CPU2)を停止させてください。CPU1 で FDL のコマンドを実行中、CPU1 からデータフラッシュにアクセスできないだけでなく、CPU2 からもデータフラッシュにアクセスできません。

・RH850 デュアル CPU の製品では、メイン CPU（CPU1）で R_FDL_CMD_PREPARE_ENV コマンドを使用する時、パフォーマンス CPU(CPU2)を停止させてください。CPU1 で R_FDL_CMD_PREPARE_ENV コマンドを実行中、CPU1 からコードフラッシュにアクセスできないだけでなく、CPU2 からもコードフラッシュにアクセスできません。

3. FDL の多重実行：

FDLは多重実行に対応していません。

- FDL関数を割り込み処理内で実行しないでください。
- OS上でFDL関数を実行する場合は、複数のタスクからFDL関数を実行しないでください。

4. 関数間のタスク切り替え、コンテキスト変更、同期：

どの時点でも実行される FDL 関数は 1 つだけである必要があります。したがって、ある FDL 関数を起動した後、それがまだ終了しないうちに別のタスクコンテキストに切り替えて別の FDL 関数を実行することは禁じられています。

5. 低消費で動作するモードへの移行：

コマンド実行中に、低消費電力で動作するモードへの移行することも禁止です。R_FDL_StandBy 関数を実行し、スタンバイモードへ遷移してから、低消費電力で動作するモードへの移行してください。低消費電力で動作するモードについては、対象デバイスのユーザーズマニュアルでご確認ください。

6. HALT 以外の低消費電力で動作するモード：

HALT以外の低消費電力で動作するモードには対応しておりません。HALT以外の低消費電力で動作するモードに設定し、通常のモードに復帰した場合、R_FDL_Init関数からやり直してください。

7. 初期化：

R_FDL_Initの呼び出しによるFDLの初期化は、ほとんどのFDL関数を呼び出す前に実行する必要があります。例外はR_FDL_GetVersionString関数で、このFDL関数はいつでも呼び出すことができます。

8. クリティカルセクション※1の処理：

R_FDL_CMD_PREPARE_ENV コマンド※2は、コードフラッシュを一時的に無効にします。無効期間中はコードフラッシュを使用できないので、FDL はコードを内蔵 RAM から実行します。以下のことに注意してください。

- コードは別の場所（内蔵 RAM など）から実行します。
- コードフラッシュにアクセスできない為、コードフラッシュ上の割り込み/例外へのジャンプ、CPU からのコードフラッシュの直接読み出しまたは実行、コードフラッシュへの DMA アクセスなど、いずれもできません。コールバック関数を使用して割り込みと例外を無効化および復元する方法が、サンプルアプリケーションで例示されています。

※1 本注意事項対象のクリティカルセクションのマクロ定義は、「1.クリティカルセクション定義」を参照。

※2 バージョン 1 互換定義（「2.バージョン1 互換定義」）を有効にした場合は、R_FDL_Init 関数の実行時。

9. データフラッシュ操作中の中断：

書き込み処理中、または消去処理中にリセット/電源瞬断が発生、もしくは R_FDL_CancelRequest 関数を実行した場合、処理は中断され、書き込み処理、または消去処理を実行していたブロックの内容は不定となります。上記の処理中断により不定となった領域を再度ご使用になる場合は、必ず再度消去処理を実行し、ご使用ください。

10. 書き込み動作：

データを書き込むブロックはあらかじめ必ず消去してください。

11. データフラッシュの読み出し：

本 FDL の対象デバイスに搭載されているデータフラッシュの消去状態は不定値です。従って消去状態の領域を読み出しますと、不定値が読み出されます。誤って読み出しを行わないようにどの領域にデータの書き込みを行ったかなど、データの書き込み状況を管理してください。なお、ブランクチェックによってブランク状態を判別することは可能ですので必要に応じブランクチェックを実行してください。

また、データの読み出しは FDL 関数を介さずに直接ユーザプログラムから読み出すことが可能です。なお、R_FDL_Execute 関数によるコマンド実行中は、データフラッシュの読み出しは行えません。読み出しを行う場合は、必ず、R_FDL_Handler 関数によってコマンド操作の終了を確認後、実行してください。

12. デュアルオペレーション：

FDL と FCL の同時実行はできません。

13. コマンド実行中のリクエスト構造体：

コマンド実行中にリクエスト構造体の内容を変更した場合、FDL は誤動作します。

14. ウォッチドッグタイマ：

ウォッチドッグタイマは、FDL の実行中は停止しません。 なお、FDL においてはタイムアウト処理を行っていません。FDL 関数のタイムアウト処理が必要な場合は、ユーザプログラムでウォッチドッグタイマなど使用して処理を行ってください。

15. サスペンドとスタンバイの多重実行とネスティング：

サスペンドは多重実行できません。また、スタンバイも多重実行できません。

サスペンド機能は以下のシーケンスを禁止しています。

サスペンド許可シーケンス：

書き込み　　--> サスペンド --> ブランクチェック※、読み出し※

消去　　--> サスペンド --> ブランクチェック※、読み出し※、書き込み※

ブランクチェック --> サスペンド --> ブランクチェック、読み出し、書き込み、消去

上記以外のシーケンス、サスペンドのネストは禁止です。

※ただし、書き込み中、消去中のブロック以外に対してのみ許可されます。

16. スタンバイ：

FDL がスタンバイモードのときは、R_FDL_GetVersionString 関数、R_FDL_WakeUp 関数以外の FDL 関数を実行しないでください。

17. データアラインメント：

データフラッシュのブロックは 64 バイトにアラインされ、データの書き込みは 4 バイトにアラインされます。

18. プリコンパイルオプション：

本マニュアルに記載されていないプリコンパイル設定オプションは使用しないでください。

19. 対応デバイス：

対応デバイスにつきましては本 FDL パッケージに含まれている support.txt にてご確認ください。

20. プリコンパイルオプション設定時のアクセス領域：

本FDLはプリコンパイルオプション設定によって、初期化時に以下の領域にアクセスします。
R_FDL_ExecuteのPrepareEnvコマンド実行時は、以下の領域のリードアクセスを許可してください。

FDL V2.12 以降 プリコンパイル定義	アクセス領域
R_FDL_NO_BFA_SWITCH	0x01030000～0x0103029F
R_FDL_MIRROR_FCU_COPY	0x01030000～0x0103029F, 0x01037000～0x01037FFF
R_FDL_NO_FCU_COPY	0x00010000～0x0001029F
上記プリコンパイル定義なし	0x00010000～0x0001029F, 0x00017000～0x00017FFF

付録A　改訂記録

A. 1　本版で改訂された主な箇所

Rev.	発行日	改訂内容	
		ページ	ポイント
1.03	2019.05.31	全体	R_FDL_CMD_PREPARE_ENV コマンドの注意に該当する新サポートデバイスを追加(F1KH, D1M1-V2, D1S1)。
		18	「3. 6　スタンバイ/ウェイクアップメカニズム」でウェイクアップ処理動作の説明とバージョンによるスタンバイ/ウェイクアップ動作の違いを追加。
		24	「4. 1　プリコンパイル設定」の注意事項で、プリコンパイル定義毎に対象となる新サポートデバイスを追加。
		26	「4. 3　データ型」に V2.13 以降のバージョンを C99 でコンパイルする場合の注意を追加。
		27	「4. 3. 1　単純型定義」に C99 以降の規格が指定されている場合の説明を追加。
		30	「4. 3. 4　r_fdl_command_t」で、V2.13 以降、”R_FDL_CMD_WRITE16B”が非サポートとなることを追加。
		45	「4.4.3.3. R_FDL_StandBy」の「例：」で、”R_FDL_WakeUp()” 関数の呼び出し例を V2.13 以降の仕様に合わせ変更。
		46	「4.4.3.4. R_FDL_WakeUp」の「戻り値」、「事後条件」を”R_FDL_WakeUp()” 関数の呼び出し例を V2.13 以降の仕様に合わせ変更。
		67	「5. 3　リンカセクション」の「5. 3. 1　FDL のリソース」のリソースサイズを対象バージョン FDL V2.13 用に変更。
		71	「第 6 章　注意事項」の「1.動作周波数の設定」に RH850/F1KM-S4, または、RH850/F1KH-D8 のオプションバイトで CKDIVMD=0（Max 120MHz）に設定して使用する場合の注意事項を追加。
		73	「第 6 章　注意事項」の「8.クリティカルセクションの処理」にマクロ定義参照の補足を追加。

A. 2　前版までの改版履歴

これまでの改版履歴を次に示します。

Rev.	発行日	改訂内容 ポイント
1.00	2015.03.31	初版発行
1.01	2017.02.01	「3. 8　タイムアウト処理」追加
		「4. 1　プリコンパイル設定」に「3.RH850/F1K 専用ビルド定義」と「4.FDL スタック実行処理定義」を追加
		「4.4.2.1.R_FDL_Execute」の「説明」に新機能を追加
		「4.4.2.1.R_FDL_Execute」の「注意 A」に RH850/F1K 仕様差分を追加
		「5. 3. 2　FDL のリソース」に補足文章追加、V2.00 スタック最大使用量(GHS)の誤記訂正、V2.11 のリソース追加。
1.02	2017.12.25	「第 1 章　はじめに」の「備考」に文言を追加
		「4. 1　プリコンパイル設定」の「3.RH850/F1K 専用ビルド定義」に補足と F1KM を追加。
		「4. 1　プリコンパイル設定」に「5.FCU ファームウェア転送無効定義」と「6.ユーザ領域と FCU ファームウェア格納領域の切り替え無効定義」を追加。
		「4. 3　データ型」に C99 でコンパイルする場合の注意を追加。
		「4.4.2.1. R_FDL_Execute」の注意 A 【R_FDL_CMD_PREPARE_ENV コマンド実行中のコードフラッシュへのアクセス禁止】の「※1」に F1KM を追加
		「4.4.3.5. R_FDL_CancelRequest」で status_enu の取り得る値から R_FDL_ERR_PROTECTION, R_FDL_ERR_INTERNAL を追加。
		「4. 5. 5　R_FDL_CMD_PREPARE_ENV」に説明追加。
		「表 13　FDL パッケージのファイル構成」で FDL パッケージのファイル構成のファイル名、内容、注釈を変更。 (2)に make.exe に関する注意書きを追加。
		「5. 3　ファイル構成」の「5. 3. 2　FDL のリソース」の ROM/RAM 容量、処理時間に関する記述を変更。
		「第 6 章　注意事項」の「1.動作周波数の設定」に新デバイス RH850/F1KM-S4 の注意事項追加。
		「第 6 章　注意事項」に「プリコンパイルオプション設定時のアクセス領域」を追加。

RH850 ファミリ　ユーザーズマニュアル
データフラッシュライブラリ Type01

発行年月日　2015年 3月31日　Rev.1.00
2019年 5月31日　Rev.1.03

発行　ルネサス エレクトロニクス株式会社
〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

RH850 ファミリ

ルネサス エレクトロニクス株式会社

R01US0151JJ0103