User's Manual

16

RL78ファミリ

EEPROMエミュレーション・ライブラリ Pack01

日本リリース版

ZIPファイル名：JP_R_EEL_RL78_P01_Vx.xx_x_J

16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　2016.01

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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製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）、交通用信号機器、
   防災・防犯装置、各種安全装置等

   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、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。お客様が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. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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。

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その総株主の議決権の過半数を直接または間接に保有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。

(2012.4)

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部 ROM、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。

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、RL78ファミリのEEPROM エミュレーション･ライブラリPack01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。

★　対応MCU：以下のリストを参照してください。

日本語版：

RL78ファミリ セルフプログラミングライブラリ セルフRAMリスト（R20UT2943）

目　　的　このユーザーズ・マニュアルは、RL78ファミリのデータ・フラッシュ・メモリをEEPROMエミュレーションでデータ格納する方法（アプリケーションによる定数データ書き込み）をユーザに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。

・EEPROMエミュレーション概要

・EEPROMエミュレーションの使用方法

・EEPROMエミュレーション機能

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3章 EEPROMエミュレーション機能**を参照してください。

なお、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

目次

第1章　EEPROMエミュレーション概要

1. 1　EEPROMエミュレーションの基本仕様

EEPROMエミュレーションとは、搭載されているフラッシュ・メモリへEEPROMのようにデータを格納させるための機能です。EEPROMエミュレーションは、データ・フラッシュ・ライブラリとEEPROMエミュレーション・ライブラリを使用して、データ・フラッシュ・メモリへの書き込みや読み出しを実行します。

データ・フラッシュ・ライブラリは、データ・フラッシュ・メモリへの操作を行うためのソフトウェア・ライブラリです。EEPROMエミュレーション・ライブラリは、ユーザ・プログラムからEEPROMエミュレーション機能を実行させるためのソフトウェア・ライブラリです。データ・フラッシュ・ライブラリおよびEEPROMエミュレーション・ライブラリは、コード・フラッシュ・メモリに配置して使用します。

ユーザ・プログラムからは、EEPROMエミュレーション・ライブラリが提供するユーザ・アクセス関数処理（関数）を呼び出す事により、データ・フラッシュ・メモリに関する操作を意識することなく使用することができます。

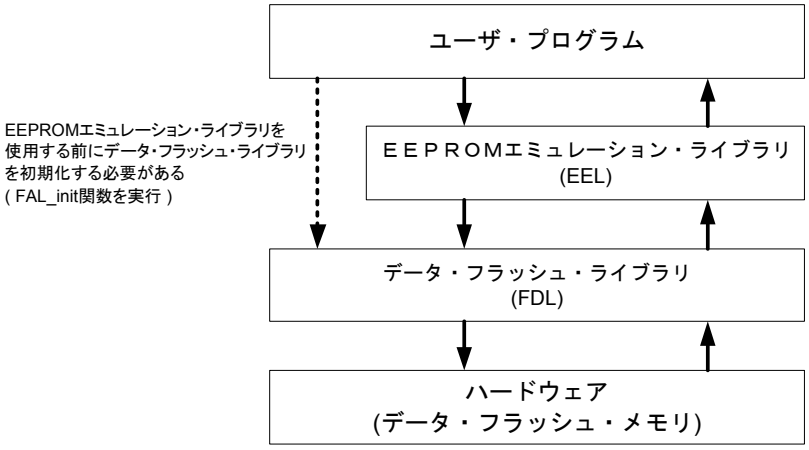
EEPROMエミュレーション・ライブラリ Pack01では、1バイトの識別子（データID：1～255）をデータごとにユーザが割り振り、割り振った識別子ごとに読み出し／書き込みを1～255バイトの任意の単位で操作することができます（識別子ごとに割り振れるデータは最大255個まで扱うことができます）。

また、データを格納するためのデータ・フラッシュ・メモリは連続した4ブロック以上の領域を使用します。このブロックをEEPROMエミュレーション・ブロックと言います。

EEPROMエミュレーションによって書き込まれるデータは、参照用の参照データと、ユーザが指定したユーザ・データに分けられ、参照データはブロックの下位アドレスから、ユーザ・データはブロックの上位アドレスから対象ブロックに書き込みが行われます。

図1－1にEEPROMエミュレーション・ライブラリとデータ・フラッシュ・ライブラリの関係図を、図1－2、図1－3にメモリ・マップ、データ構造例、図1－4から図1－6にブロックの使用方法と遷移例を示します。また、表1－1にEEPROMエミュレーションにおいて設定する各項目と、その項目の範囲について示します。

★　**図 1−1　EEPROMエミュレーション・ライブラリとデータ・フラッシュ・ライブラリの関係図**



★　**図 1−2　メモリ・マップ例**

・R5F100にて、データ・フラッシュ・メモリをEEPROMエミュレーション・ブロックとして使用するためにユーザ・プログラムとデータ・フラッシュ・ライブラリ、EEPROMエミュレーション・ライブラリをコード・フラッシュ・メモリに配置し、EEPROMエミュレーション・ブロックをデータ・フラッシュ・メモリに設定、定義されているユーザ・データ（ユーザ・データA、ユーザ・データB、ユーザ・データC）をそれぞれ書き込んでいった場合のマップ例。

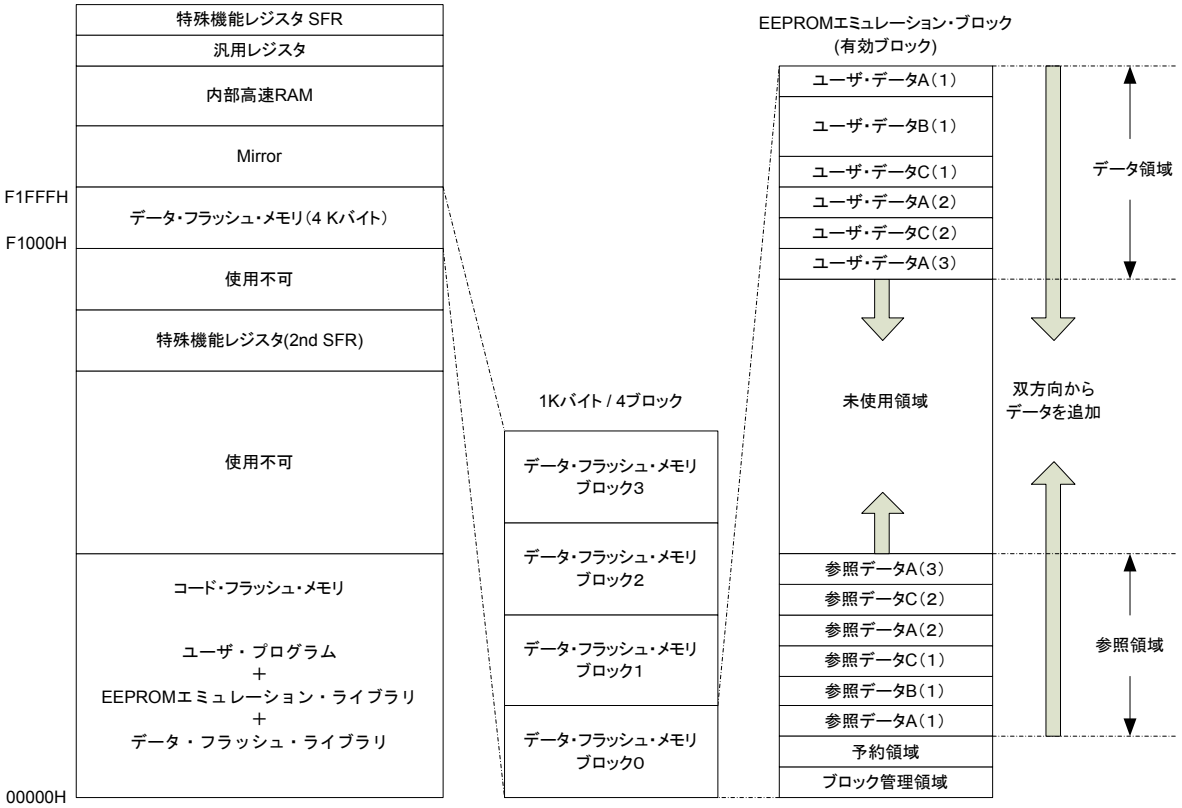
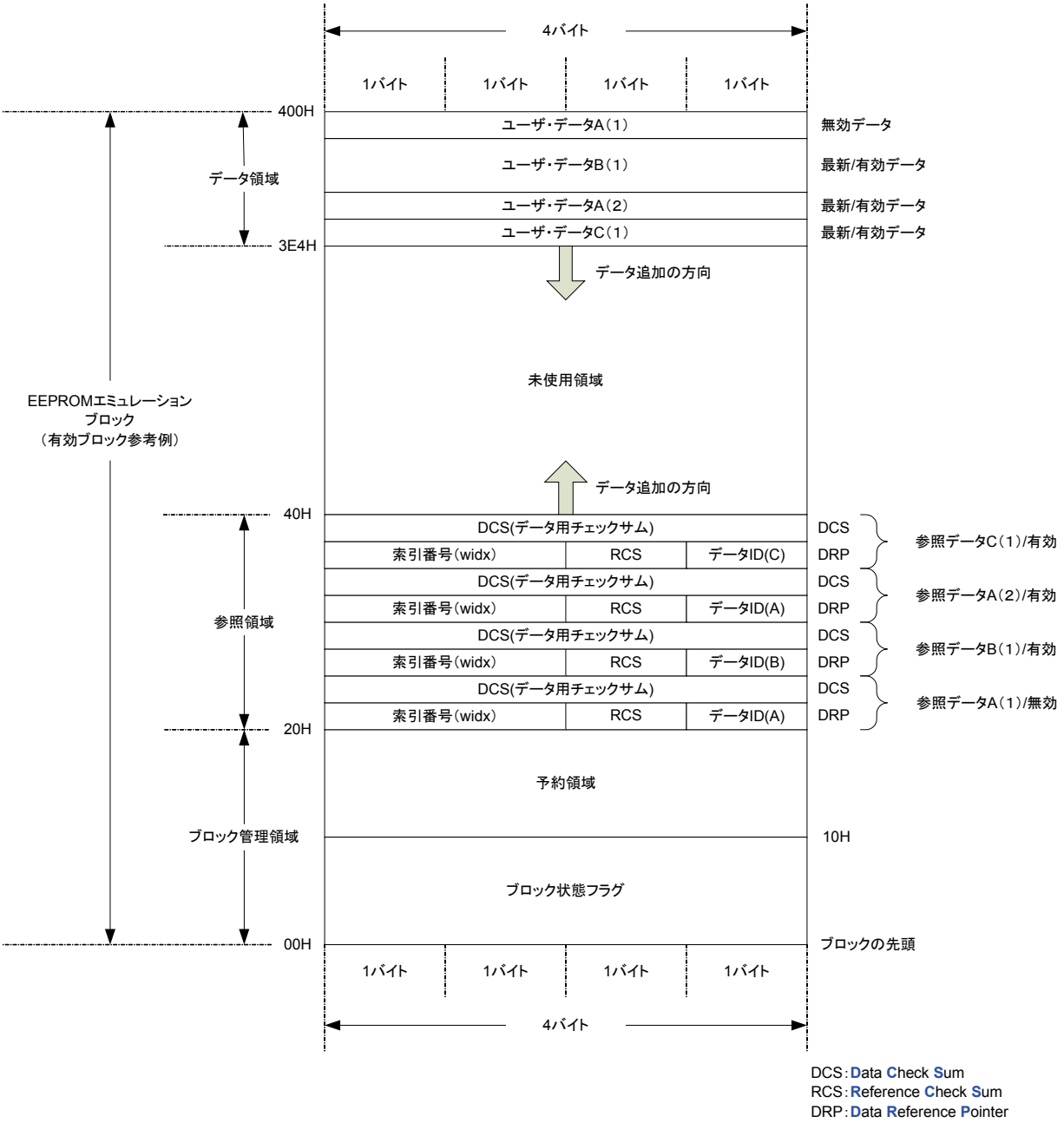


図 1−3　EEPROMエミュレーション・ブロックのデータ構造例（詳細）

・定義したサイズの異なるユーザ・データ（ユーザ・データA、ユーザ・データB、ユーザ・データC）を一定の順番で書き込んだ状態（書き込み順 : ユーザ・データA → ユーザ・データB → ユーザ・データA → ユーザ・データC ）の構造例となります。
ユーザ・データはアドレスの上位から、管理データは下位から順番に書き込まれ、最後に書き込まれたユーザ・データが有効データとなります。



DCS：Data Check Sum
RCS：Reference Check Sum
DRP：Data Reference Pointer

図 1−4　EEPROMエミュレーション・ブロックの有効ブロックの拡張例

・指定したデータを書き込むときに、使用している有効ブロックにデータを書き込むための十分な空き容量がない状態の場合は、有効ブロックを拡張して新しい有効ブロックに指定されたデータを書き込みます。

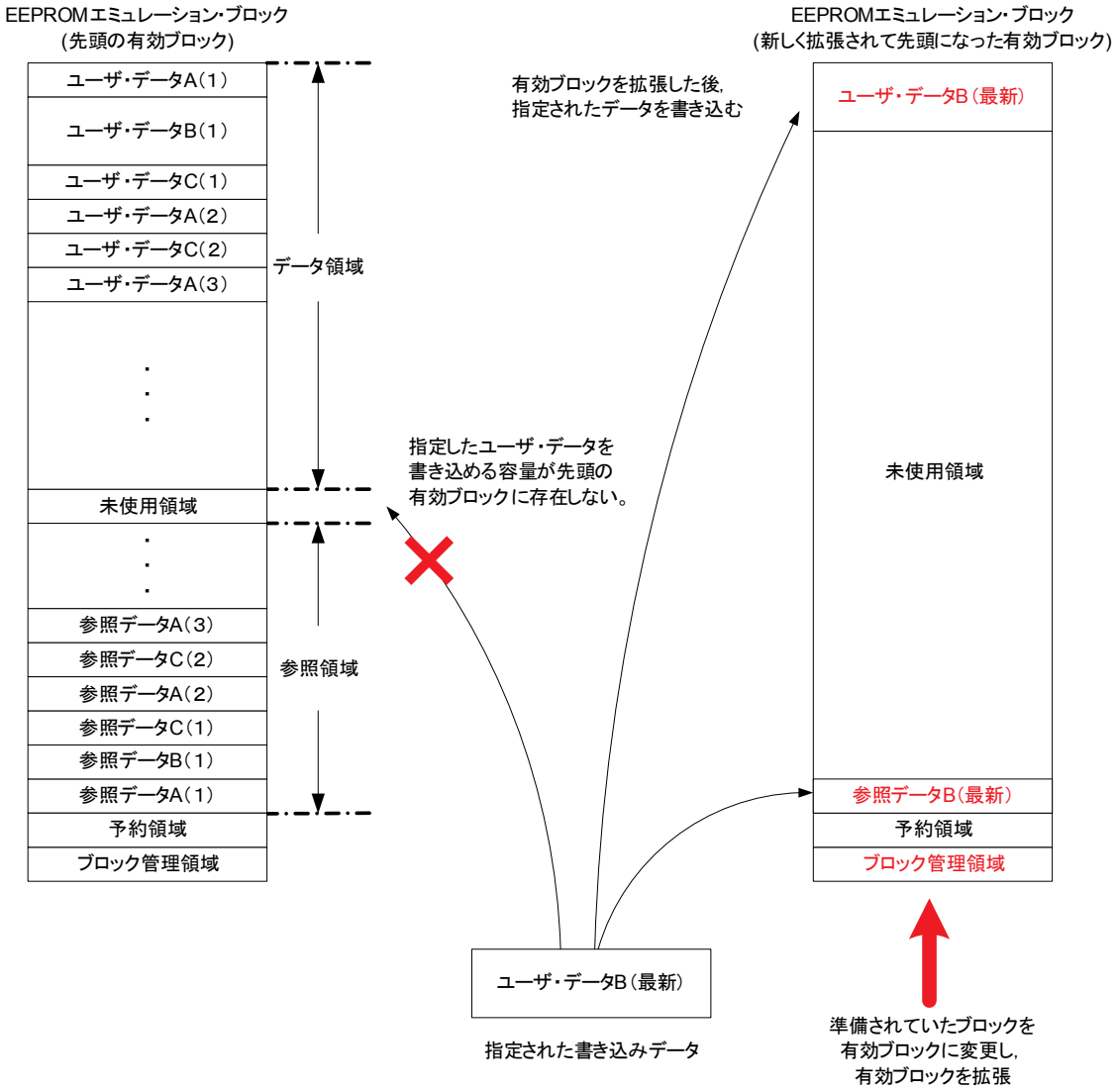
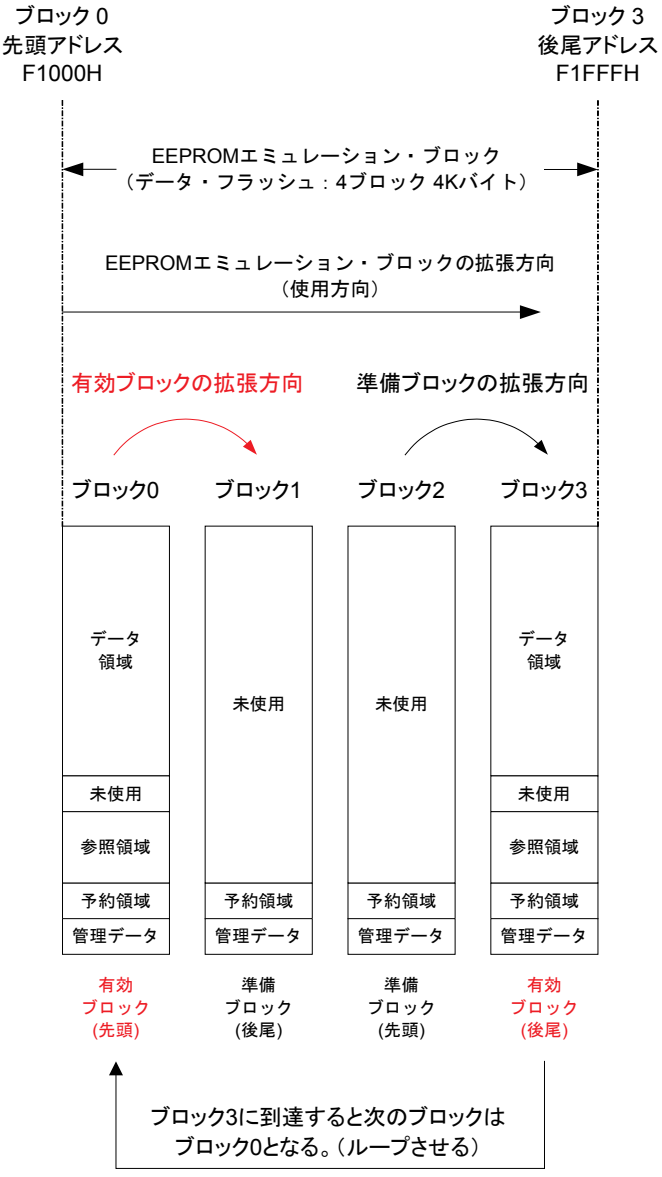


図 1−5　EEPROM エミュレーション・ブロックの使用方法（4ブロックの場合）

・EEPROMエミュレーションはデータ・フラッシュ・メモリのブロック0からブロック3を順番に有効ブロックや準備ブロックとして拡張し、最後のブロック3に到達すると次のブロックを最初のブロック0にすることでブロックをループさせながら使用しています。



有効ブロック：現在使用中のブロック
準備ブロック：使用できるブロックとして準備されている状態のブロック

図 1−6　(A)　EEPROM エミュレーション・ブロックの遷移例

・EEPROMエミュレーション・ブロックの初期化を実行した後、何も書き込みが行われていない場合のブロック例です。

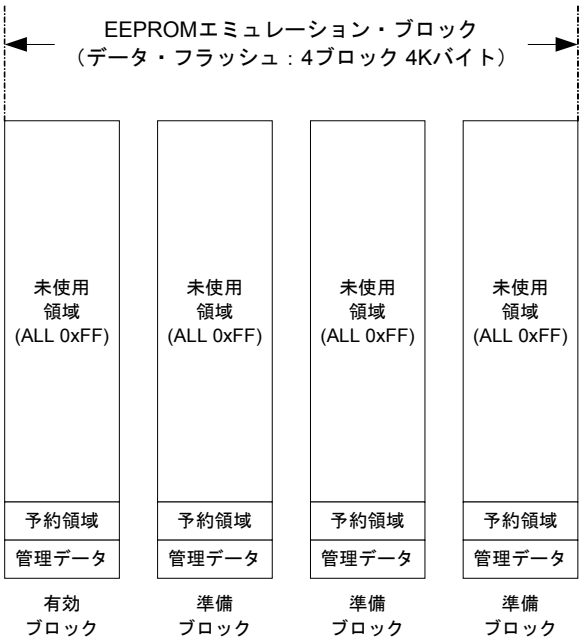


図 1−6 (B)　EEPROM エミュレーション・ブロックの遷移例

・書き込み処理のみを継続的に実行した場合、一定のブロック数(残りの準備ブロックが２ブロック以下)に達するまで有効ブロックが拡張されていきます。

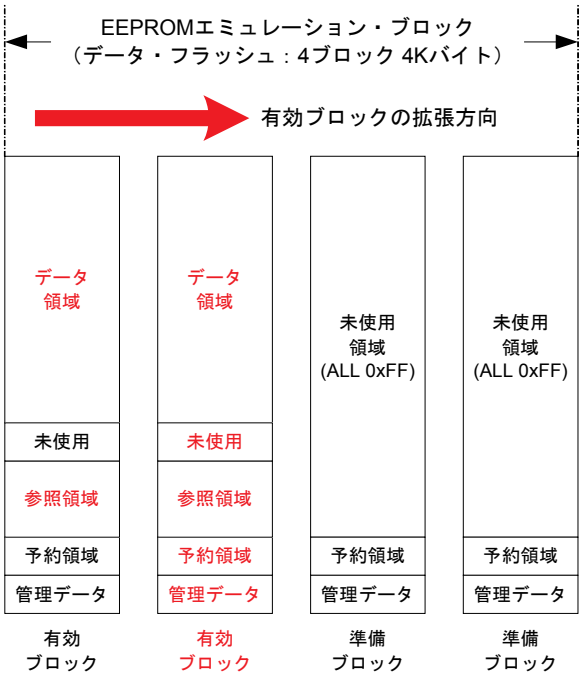


図 1−6 (C)　EEPROM エミュレーション・ブロックの遷移例

・EEPROMエミュレーションで書き込み処理のみを継続的に実行し、残りの準備ブロックが2ブロック以下になった状態でブロックの拡張が発生する場合には、一番古い有効ブロックに存在している有効データを最新の有効ブロックにコピーし、一番古い有効ブロックを消去します。

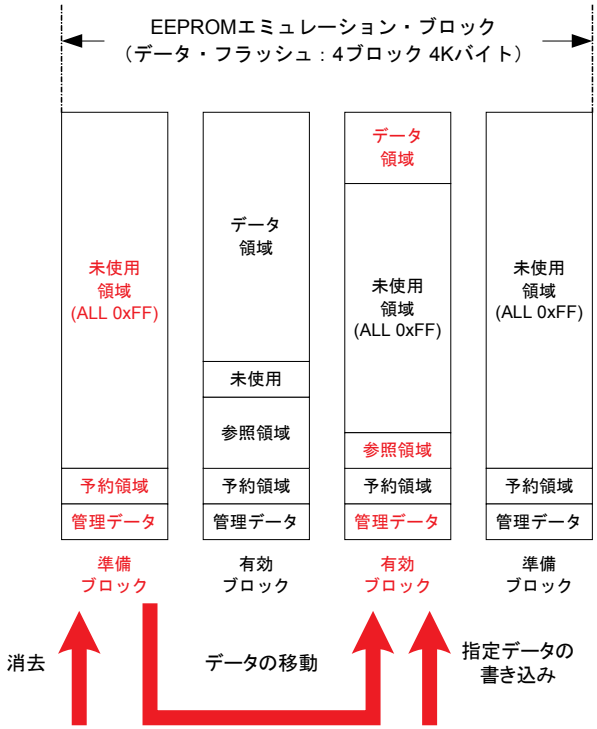


表 1−1　書き込みデータの設定項目とその使用範囲

設定項目	使用範囲	備考
ユーザ・データ長	1 ～ 255	—
格納ユーザ・データ数 注1	1 ～ 255	データ種別の数
データIDの範囲	1 ～ 255	—
EEPROMエミュレーション・ブロック数注2	4 ～ 255	—
ユーザ･データの推奨サイズ 注1	980×総ブロック数×1/4 −980/2バイト	書き込むときに付与される 管理用の参照データ分も含む

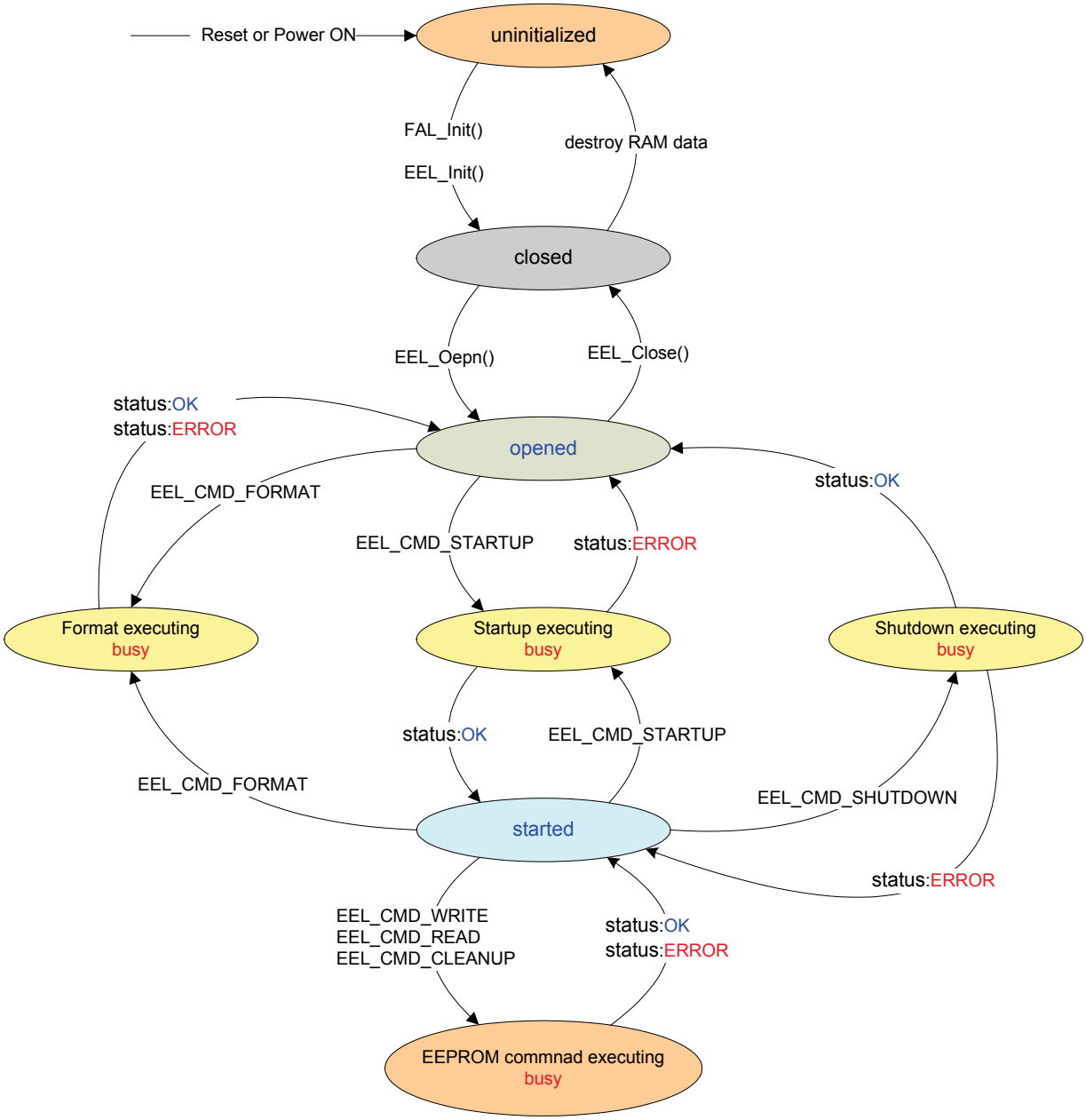
注 1. ユーザ・データは、すべてのユーザ・データがEEPROMエミュレーション・ブロックへ書き込まれるときに必要な合計サイズを総ブロック数の1/4以内に収められる状態にする必要があります。そのため、格納するユーザ・データのサイズによって格納ユーザ・データ数の使用範囲は変わります。また、合計サイズも管理用としてデータごとに付与される参照データ分のサイズも考慮に入れる必要があります。格納ユーザ・データ数や合計サイズの詳細については**1.2.4 格納ユーザ・データ数とユーザ・データの合計サイズ**を参照してください。

2. 搭載されているデータ・フラッシュ・メモリの最大ブロック数以上には設定できません。

1. 2　EEPROMエミュレーションの操作フロー

ユーザ・プログラムからEEPROMエミュレーションを使用する為にはEEPROMエミュレーション・ライブラリの初期化処理を行い、書き込みや読み込み等EEPROMエミュレーション・ブロックの操作を行う関数を実行する必要があります。全体の状態遷移図を図1－7に、基本的な機能を使用するための操作フローを図1－8に示します。EEPROMエミュレーションを使用する場合は、この流れに沿ってユーザ・プログラムに組み込んで下さい。

図 1－7　EPROMエミュレーションの状態遷移図



【状態遷移図の概要】

EEPROMエミュレーション・ライブラリを使用してデータ・フラッシュ・メモリを操作するためには、用意されている関数を順に実行し、処理を進める必要があります。

(1) uninitialized

Power ON、Reset時の状態です。また、フラッシュ・セルフ・プログラミング・ライブラリを実行した場合もこの状態に遷移します。

(2) closed

FAL_Init()、EEL_Init()関数を実行し、EEPROMエミュレーションを実行するためのデータを初期化した状態(データ・フラッシュ・メモリへの操作は停止状態)です。EEPROMエミュレーションを動作させた後にフラッシュ・セルフ・プログラミング・ライブラリやSTOPモード、HALTモードを実行する場合は、opened状態からEEL_Closeを実行し、この状態に遷移させてください。

(3) opened

closed状態からEEL_Openを実行し、データ・フラッシュ・メモリへの操作が可能になった状態です。EEL_Closeを実行し、closed状態に遷移するまでの間はフラッシュ・セルフ・プログラミング・ライブラリやSTOPモード、HALTモードは実行できません。

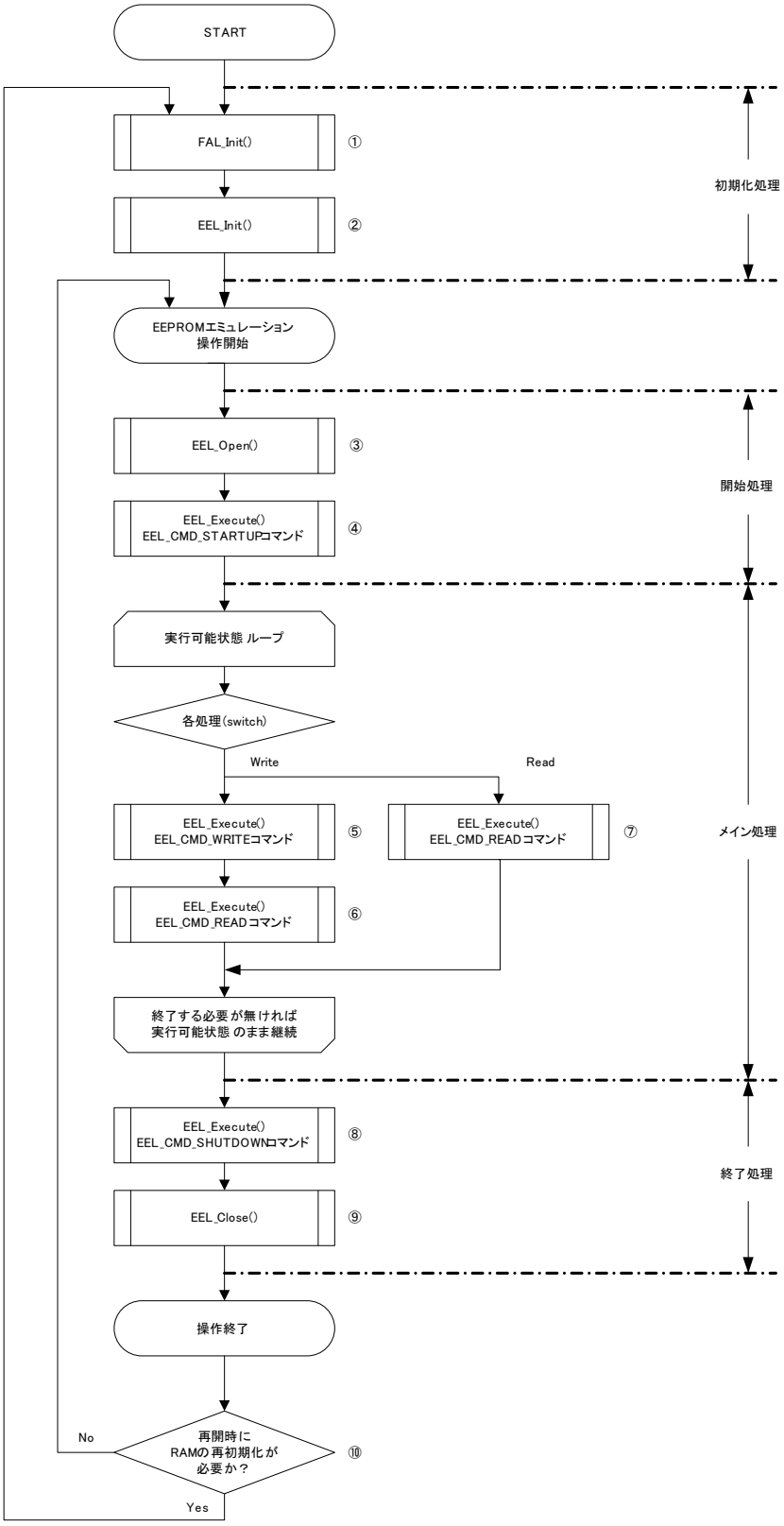
(4) started

opened状態からEEL_CMD_STARTUPコマンドを実行し、EEPROMエミュレーションが実行できるようになった状態です。この状態からEEPROMエミュレーションを使用した書き込みや読み込みを行います。

(5) busy

指定された各コマンドを実行している状態です。実行コマンドと終了状況によっては遷移する状態が変わる場合もあります。

図 1−8　EEPROMエミュレーション基本操作フロー(Enforcedモード使用時)



【基本操作フローの概要】

EEPROMエミュレーションはモード設定によってEEL_Execute関数の実行方法が異なります。モードには、Enforcedモード、Timeoutモード、Pollingモードの３種類があります。各モードの実行方法の違いに関しては、**3.2 EEPROMエミュレーション・ライブラリ関数**の「EEL_Execute関数」の説明を参照してください。

① データ・フラッシュ・ライブラリの初期化処理 (FAL_Init)
EEPROMエミュレーション・ライブラリを使用してデータ・フラッシュ・メモリにアクセスする場合は、データ・フラッシュ・ライブラリのパラメータ(RAM)を初期化する必要があるため、FAL_Init関数を事前に実行する必要があります。初期化終了後にフラッシュ・セルフ・プログラミング・ライブラリを実行した場合は再度本処理から処理を実行する必要があります。

② EEPROMエミュレーション・ライブラリの初期化処理 (EEL_Init)
EEPROMエミュレーション・ライブラリで使用するパラメータ(RAM)を初期化します。

③ EEPROMエミュレーション準備処理 (EEL_Open)
EEPROMエミュレーション実行するため、データ・フラッシュ・メモリを制御可能な状態(opened)にします。

④ EEPROMエミュレーション実行開始処理 (EEL_Execute : EEL_CMD_STARTUPコマンド)
EEPROMエミュレーションを実行可能状態(started)にします。

⑤ EEPROMエミュレーション・データ書き込み処理 (EEL_Execute : EEL_CMD_WRITEコマンド)
指定されたデータをEEPROMエミュレーション・ブロックへ書き込みます。

⑥ EEPROMエミュレーション・データ確認処理 (EEL_Execute : EEL_CMD_READコマンド)
データの読み出しを行い、元のデータと比較する事で正常に書き込みが行われたかを確認します。

⑦ EEPROMエミュレーション・データ読み込み処理 (EEL_Execute : EEL_CMD_READコマンド)
書き込みを行ったデータの読み出しを行います。

⑧ EEPROMエミュレーション実行停止処理 (EEL_Execute : EEL_CMD_SHUTDOWNコマンド)
EEPROMエミュレーションの動作を停止状態(opened)にします。

⑨ EEPROMエミュレーション終了処理 (EEL_Close)
EEPROMエミュレーション終了するため、データ・フラッシュ・メモリを制御出来ない状態(closed)にします。

⑩ EEPROMエミュレーション再実行前の確認
EEPROMエミュレーション終了後にフラッシュ・セルフ・プログラミングを実行する等、EEPROMエミュレーションを再実行するに当たってRAMの再初期化が必要な場合、FAL_Init関数から処理を再実行します。

1. 2. 1　EEPROMエミュレーション・ブロック

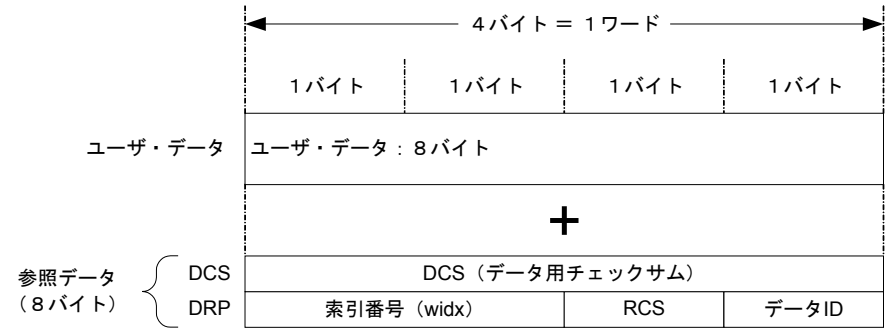
EEPROMエミュレーション･ライブラリ Pack01は､4ブロック以上のデータ･フラッシュ･メモリをEEPROMエミュレーション用のブロックとして使用します。

1. 2. 2　データ構造

データ・フラッシュへの書き込みはワード・サイズ（4バイト）で行われます。そのため、EEPROMエミュレーション・ライブラリがデータ・フラッシュ・メモリへの書き込みを行う際には、必ずワード・サイズ（4バイト）単位にデータ長を調整して書き込みを行います。

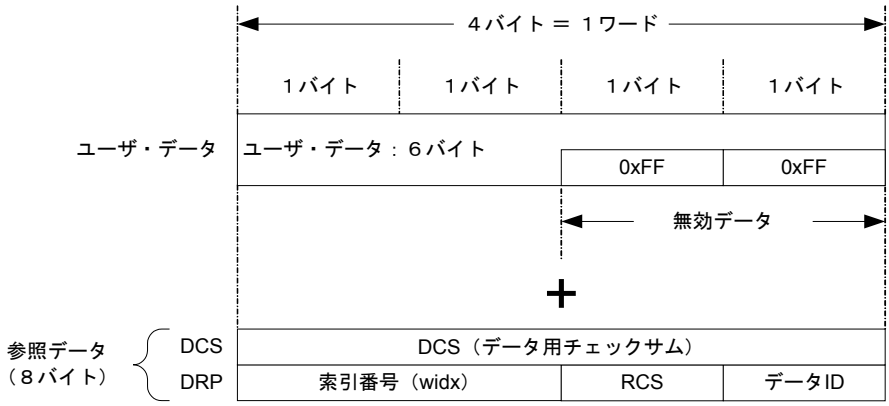
また、ユーザ・データの書き込みを行う際にはデータ管理用の参照データも同時に書き込まれるため、データを書き込むために必要なサイズは、管理用の参照データ2ワード（8バイト）分に、ワード・サイズに調整されたユーザ･データのサイズを足し合わせた容量が必要となります。図1－9から図1－11にデータ･フラッシュ･メモリへユーザ・データが書き込まれる時のデータ構造例を示します。

図 1－9　データ長とデータ構造例1（ユーザ・データが8バイトの場合）



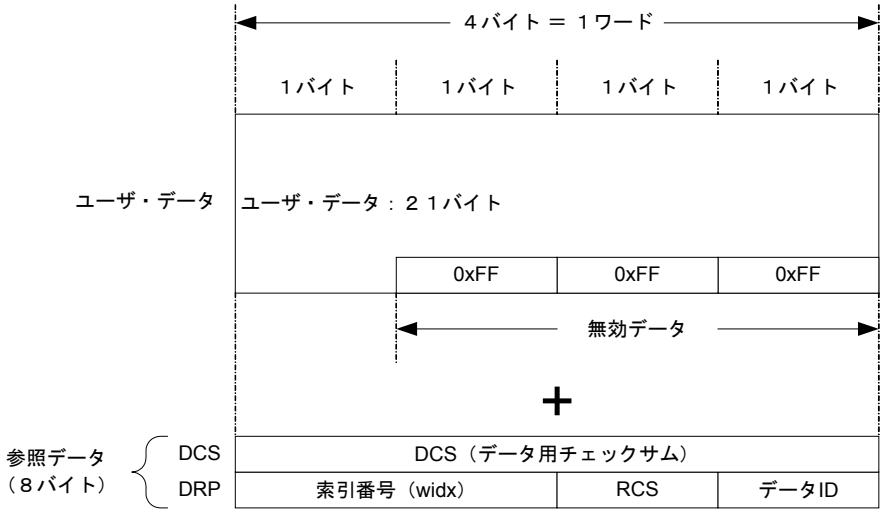
合計サイズ 4ワード(16バイト)＝参照データ 2ワード(8バイト) ＋ [ユーザ・データ(8バイト) / ワード・サイズ(4バイト)]
※ ユーザ・データをワード単位に変更。

図 1－10　データ長とデータ構造例2（ユーザ・データが6バイトの場合）



合計サイズ 4ワード(16バイト)＝参照データ 2ワード(8バイト) ＋ [ユーザ・データ(6バイト) / ワード・サイズ(4バイト)]
※ ユーザ・データをワード単位に変更。（端数は切り上げる）

図 1−11　データ長とデータ構造例3（ユーザ・データが21バイトの場合）



合計サイズ ８ワード (３２バイト) ＝ 参照データ ２ワード (８バイト) ＋ [ユーザ・データ(２１バイト) / ワード・サイズ(４バイト)]
※ ユーザ・データをワード単位に変更。（端数は切り上げる）

(1) DRP

Data Reference Pointerの略。記録されているユーザ・データのIDや参照位置を記録する領域です。

(2) 索引番号(widx)

ユーザ・データの索引番号(参照位置)です。

(3) RCS

Reference Check Sumの略。DRP用のチェックサム値(8bit)です。

(4) データID

EEPROMエミュレーションで使用するデータ固有のIDです。ユーザが指定したID[注]が記録されます。

(5) DCS

Data Check Sumの略。ユーザ・データと参照データ、双方合わせたチェックサム値(32bit)です。

注　指定するデータIDは事前にディスクリプタ・テーブルに登録しておく必要があります。
詳細については**2.4 ユーザ設定初期値**の項を参照ください。

1. 2. 3　ブロック状態フラグ

ブロック状態フラグは、ブロックの先頭からPフラグ、Aフラグ、Iフラグ、Xフラグの各4バイトずつ計16バイトのデータとして配置されます。このデータはEEPROMエミュレーション・ブロックの状態を示し、各フラグの組み合わせによりブロックの状態を示します。

図1－12に各フラグの配置状態を、表1－2に各フラグの組み合わせによる状態を示します。

図 1－12　ブロック状態フラグの配置場所

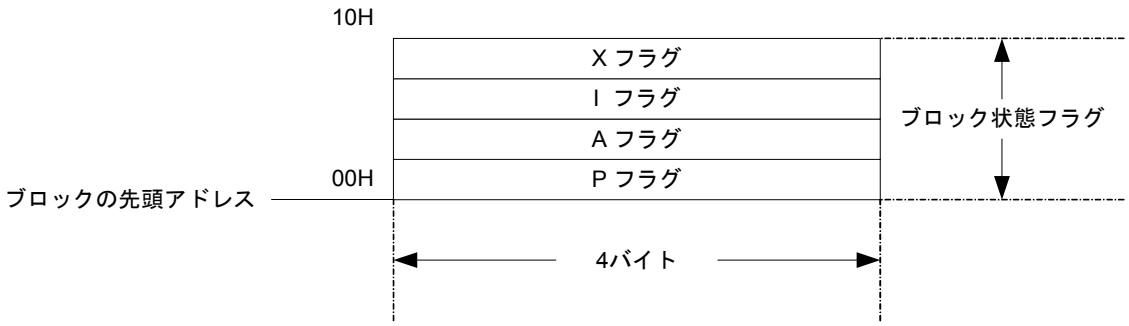


表 1－2　ブロック状態フラグの概要

ブロック状態フラグ				状　態	概　要
Pフラグ	Aフラグ	Iフラグ	Xフラグ		
55555555H	FFFFFFFFH	FFFFFFFFH	FFFFFFFFH	準備	使用できるブロックとして準備されている状態のブロック。
55555555H	55555555H	FFFFFFFFH	FFFFFFFFH	有効	現在使用中のブロック。
55555555H	55555555H	00000000H	FFFFFFFFH	無効	無効状態となったブロック。
上記以外のデータ			FFFFFFFFH		
－	－	－	FFFFFFFFH 以外	使用禁止[注]	使用禁止となったブロック。

注　一度使用禁止に設定されたブロックは再使用できません。

1. 2. 4　格納ユーザ・データ数とユーザ・データの合計サイズ

EEPROMエミュレーションで使用できるユーザ･データの合計サイズには制限があり、すべてのユーザ･データがEEPROMエミュレーション･ブロックに書き込まれる場合に必要なサイズを総ブロック数の1/4以内に収める状態にする必要があります。そのため、使用できる格納データ数は実際に格納するユーザ・データのサイズによって変わります。また、格納されるユーザ・データは一つのデータを複数のブロックに跨った形式で配置する事は出来ないため、ユーザ･データの書き込みに必要な合計サイズが１ブロックを超える場合には、1ブロックを超えた場合に使用されない可能性がある領域の最大サイズも考慮に入れる必要があります。

以下に実際にユーザ・データの書き込みで使用できるサイズ、およびユーザ・データの合計サイズの計算方法を、図1－13にユーザ・データの合計サイズが1ブロックを超えて複数ブロックになる場合のサイズ概念を示します。

・ユーザ・データの書き込みに使用できる1ブロック分の最大使用可能サイズ

データ・フラッシュ・メモリの1ブロックのサイズ	: 1024バイト
EEPROMエミュレーションでブロックの管理に必要なサイズ	: 32バイト
終端用の情報として必ず必要な空き容量（セパレータ）	: 12バイト

1ブロック分の最大使用可能サイズ ＝ 1024バイト－32バイト－12バイト ＝ 980バイト

★ ・最大サイズと推奨サイズ

最大サイズはEEPROMエミュレーション・ブロックの使用可能サイズの合計です。推奨サイズは瞬断等の影響によって正常に書き込みが出来なかった場合等を考慮し、最大サイズにマージンを設けた値です。全体から1ブロックの半分の容量を引いた値内で使用することを推奨しています。

最大サイズ ＝ 980バイト×EEPROMエミュレーション・ブロック数×1/4
推奨サイズ ＝ 最大サイズ－980/2

・ユーザ・データごとの書き込みサイズの計算方法 注

書き込まれる個々のユーザ・データのサイズ ＝ データ・サイズ（ワード長に調整済みのバイト・サイズ）
＋参照データ・サイズ（8バイト）

・ユーザ・データの基本合計サイズの計算方法

基本合計サイズ ＝（ユーザ・データ1＋8）＋（ユーザ・データ2＋8）・・・＋（ユーザ・データn＋8）

・ユーザ・データの基本合計サイズが1ブロック分の最大サイズを超える場合の合計サイズの計算方法

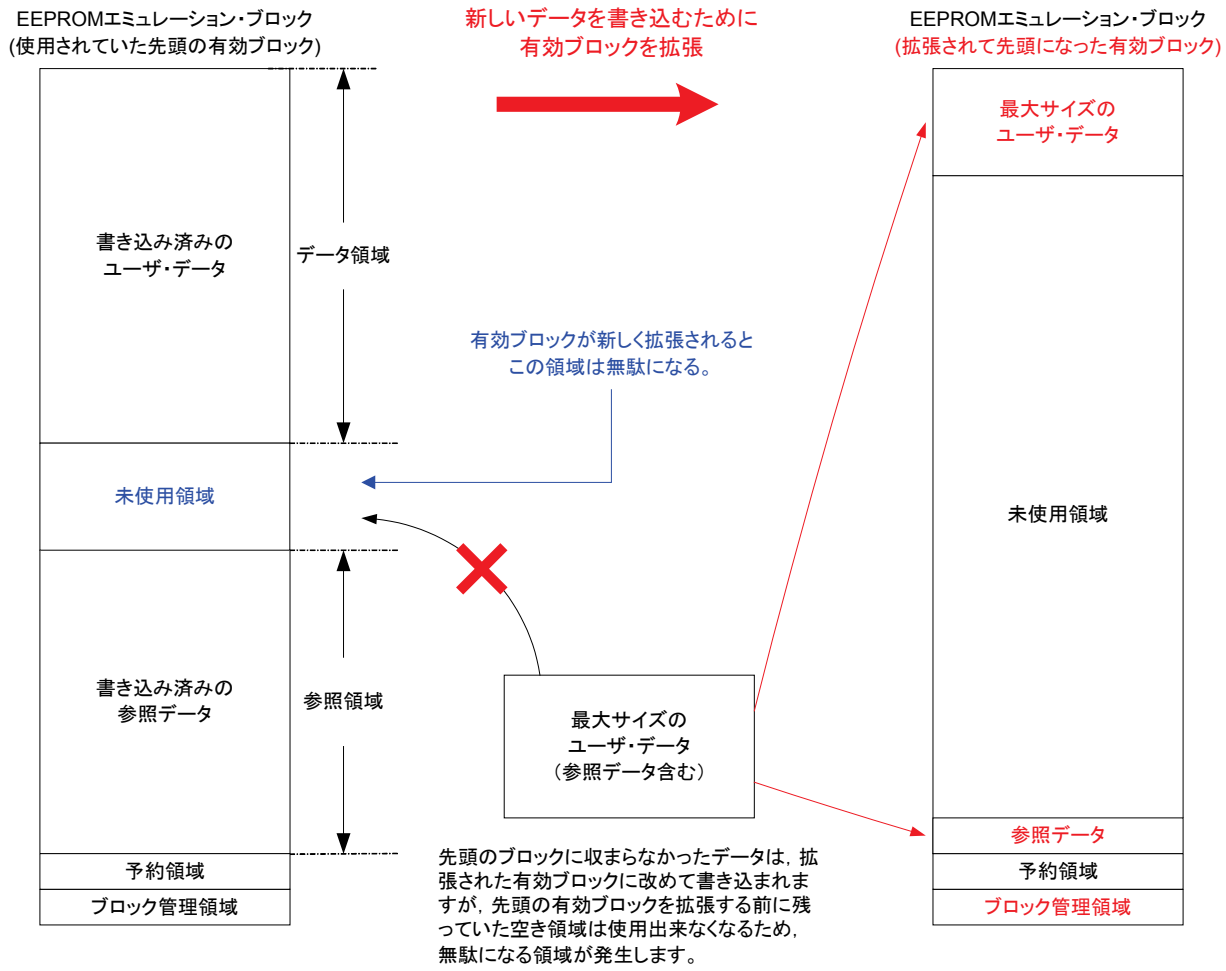
1ブロック分の最大使用可能サイズ980バイトよりユーザ・データの基本合計サイズが大きくなってしまう場合、有効ブロックを拡張する際に無駄になる可能性のある最大サイズを計算に入れる必要があります。

1ブロックを超える場合の合計サイズ ＝ 基本合計サイズ＋（（一番大きいユーザ・データのサイズ＋8）
－ EEPROMエミュレーションの最小書き込み単位（4バイト））
×（必要なブロック数－1）

注　　詳細については**1.2.2　データ構造**の項を参照してください。

図 1−13　ユーザ・データの合計サイズが複数ブロック(1ブロックを超えるサイズ)になる場合のサイズ概念

・ユーザ・データはすべてのデータを記録したときのサイズがEEPROMエミュレーション・ブロックの1/4以内に収まるようにする必要がありますが、ユーザ・データの合計サイズが1ブロックに収まらないサイズになった場合、有効ブロックの拡張時に無駄になる可能性がある最大サイズも計算に入れる必要があります。
無駄になる可能性のある最大サイズは一番大きなユーザ・データが書き込めなくなるサイズ分になります。



書き込みに必要な最大サイズ　　= 一番大きなユーザ・データのサイズ + 参照データ(8バイト)
無駄になる可能性のある最大サイズ = 書き込みに必要な最大サイズ - EEPROMエミュレーションの最小書き込み単位のサイズ(4バイト)
複数ブロックになる場合の合計サイズ = 基本合計サイズ + 無駄になる可能性のある最大サイズ×(必要なブロック数−1)

1. 3　EEPROMエミュレーション・ブロックの初期化

データ・フラッシュ・メモリをEEPROM エミュレーション・ライブラリで使用するためにはEEPROMエミュレーション・ブロックとして一度初期化を行う必要があります。初期化が必要なブロック状態の場合はEEPROMエミュレーション・ブロック上に有効ブロックが存在しないため、EEL_CMD_STARTUPコマンド実行時に「EEL_ERR_POOL_INCONSISTENT」エラーが発生します。このとき、EEPROMエミュレーション・ブロックを使用可能な状態にするためにEEL_CMD_FORMATコマンドを実行することによってブロックを初期化する必要があります。

また、データ・フラッシュ・メモリの領域が破壊されてしまった場合や、一度初期化を行った後にユーザ・データの追加等、初期設定値を変更する必要が生じた場合等、EEPROMエミュレーション・ブロックをこれまで使用していた状態のまま継続して使用する事が出来なくなったとき、あるいは任意のタイミングで最初の初期化を行いたい場合については、任意のタイミングでEEL_CMD_FORMATコマンドを実行する事によって初期化を行う事が可能です。

図1－14、図1－15にEEL_CMD_STARTUPコマンド実行時に「EEL_ERR_POOL_INCONSISTENT」エラーが発生したときの初期化フローとブロックの状態遷移例を、図1－16、図1－17に任意のタイミング初期化を行うときのフローとブロックの状態遷移例を示します。

図 1－14　「EEL_ERR_POOL_INCONSISTENT」エラー発生時の初期化フロー(Enforcedモード使用時)

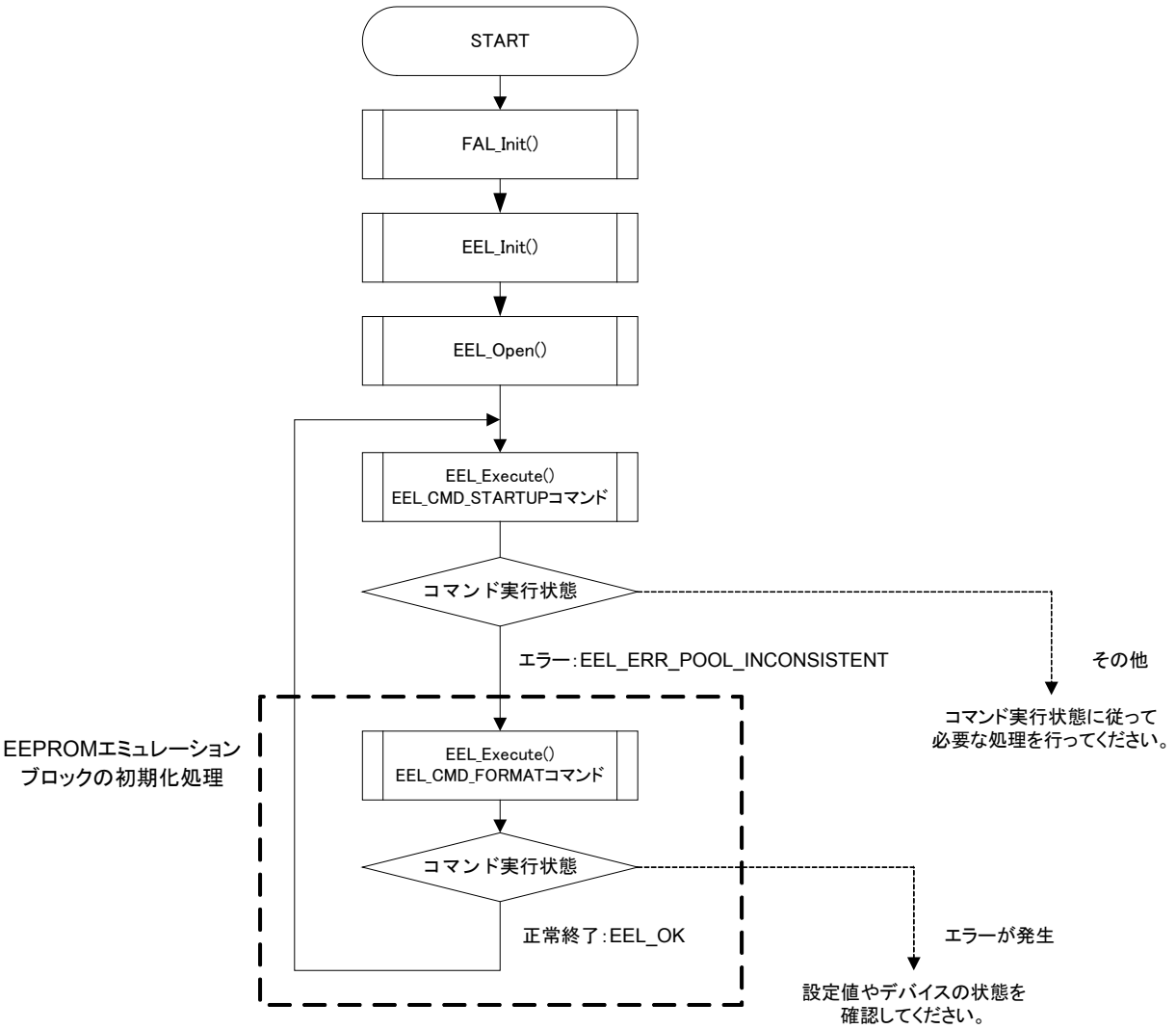


図 1－15　「EEL_ERR_POOL_INCONSISTENT」エラー発生時に初期化を行った時のブロック状態遷移例

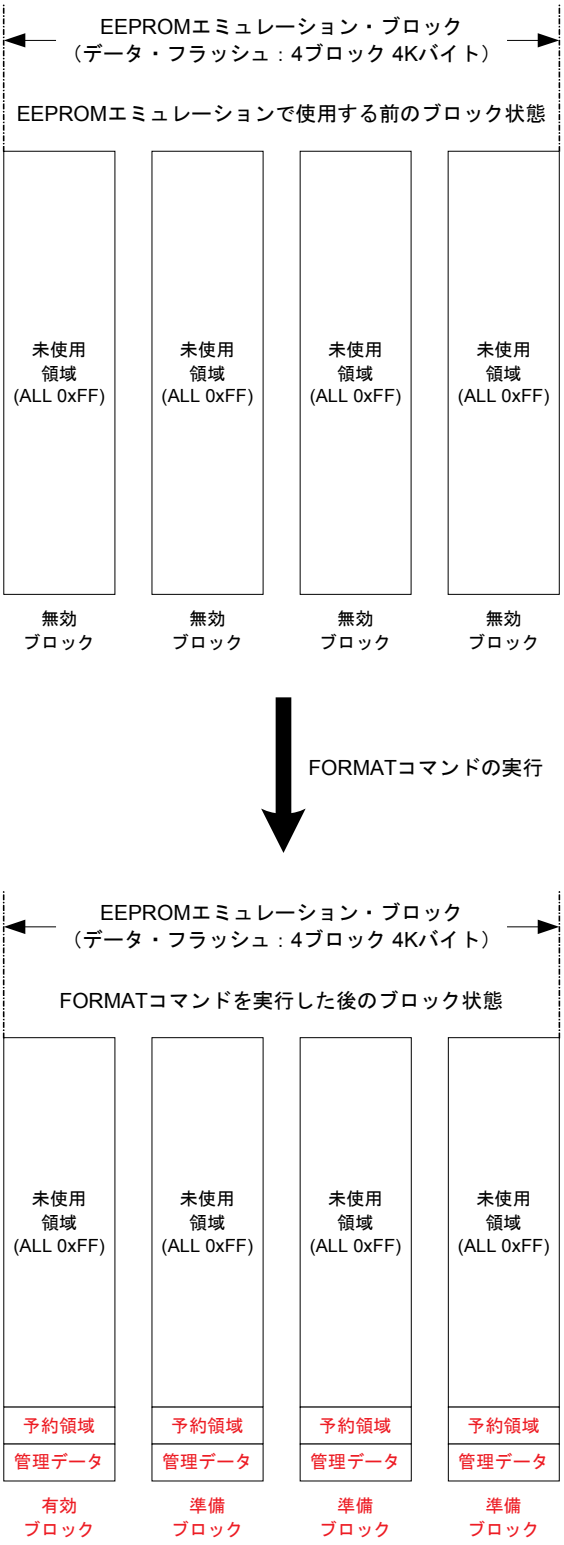


図 1−16　任意のタイミング初期化を行うときのフロー(Enforcedモード使用時)

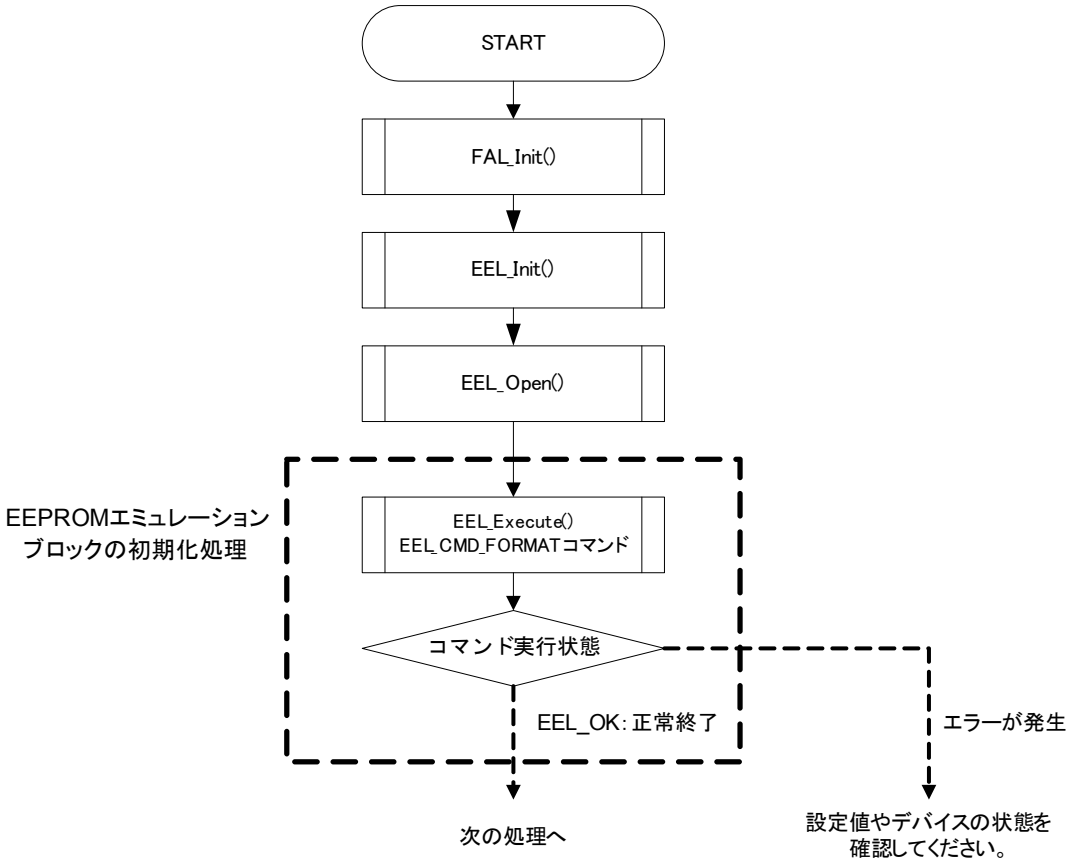
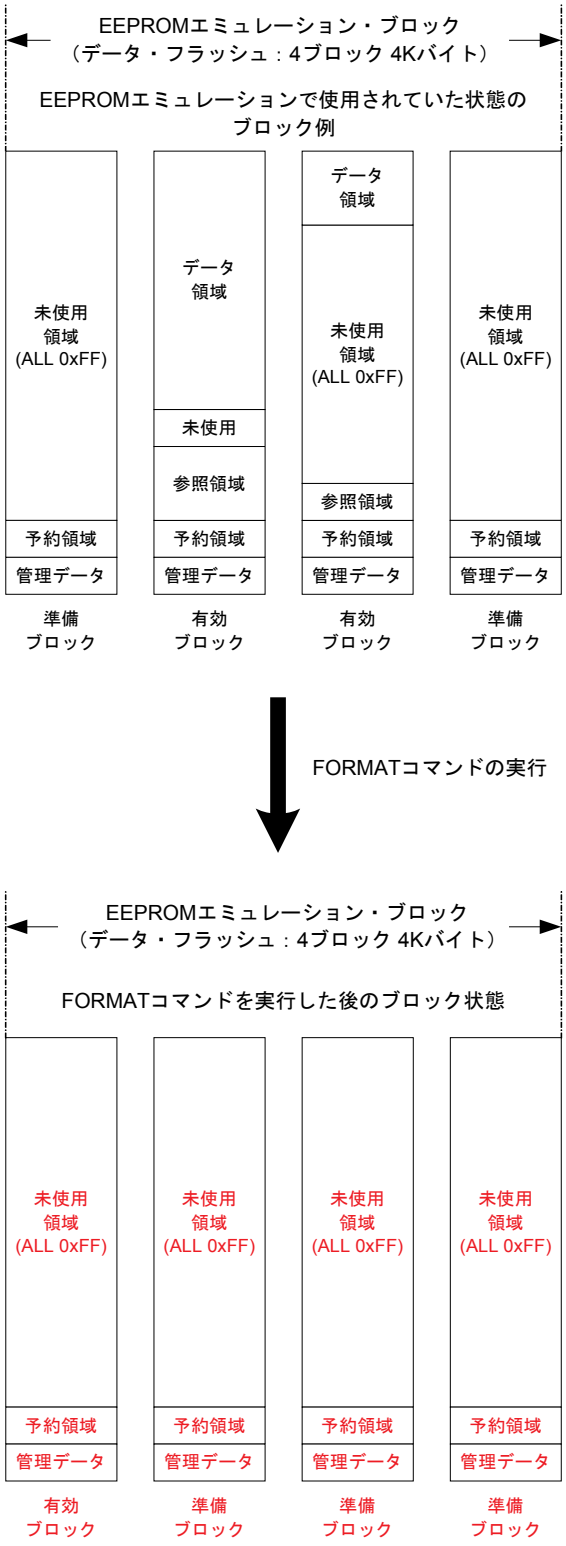


図 1－17　使用しているEEPROMエミュレーション・ブロックを任意のタイミングで初期化したときの状態遷移例



1. 4　EEPROMエミュレーション・ブロックの整理

EEPROMエミュレーション・ブロックへの書き込みを行っていくと、有効ブロックを拡張する度に準備ブロックが消費されて行く事になりますが、残りの準備ブロックが2ブロック以下で有効ブロックを拡張する必要が発生した場合には、指定データの書き込みを行う前に有効ブロックの拡張に加えて古いブロックに残っている有効データの移動と、古いブロックの消去処理が行われることになります。この消去処理を行う必要のあるタイミングで書き込みが発生した場合、書き込みにかかる処理時間に加えてデータの移動、および消去に必要な処理時間も別途発生する事になります。

この追加分の処理時間が許容できない場合は、システムに余裕があるタイミングでブロックのメンテナンスを行うことにより、緊急性の高い書き込みを行う必要が発生したタイミングでデータの移動や消去処理が同時に行われることを回避する事が可能になります。

メンテナンスを行うには、EEL_CMD_CLEANUPコマンドの実行によってブロックを整理する方法と、EEL_Handler関数のメンテナンス・モード処理を実行することによってメンテナンスを実行する方法の二通りがあります。

図1－18に有効ブロック拡張時にデータの移動と消去が発生した場合のブロック状態例、図1－19にデータ書き込み時にブロック状態によって処理時間が変わる時のイメージ図を示します。

図 1－18　有効ブロックを拡張時にデータの移動と消去が発生した場合のブロック状態例

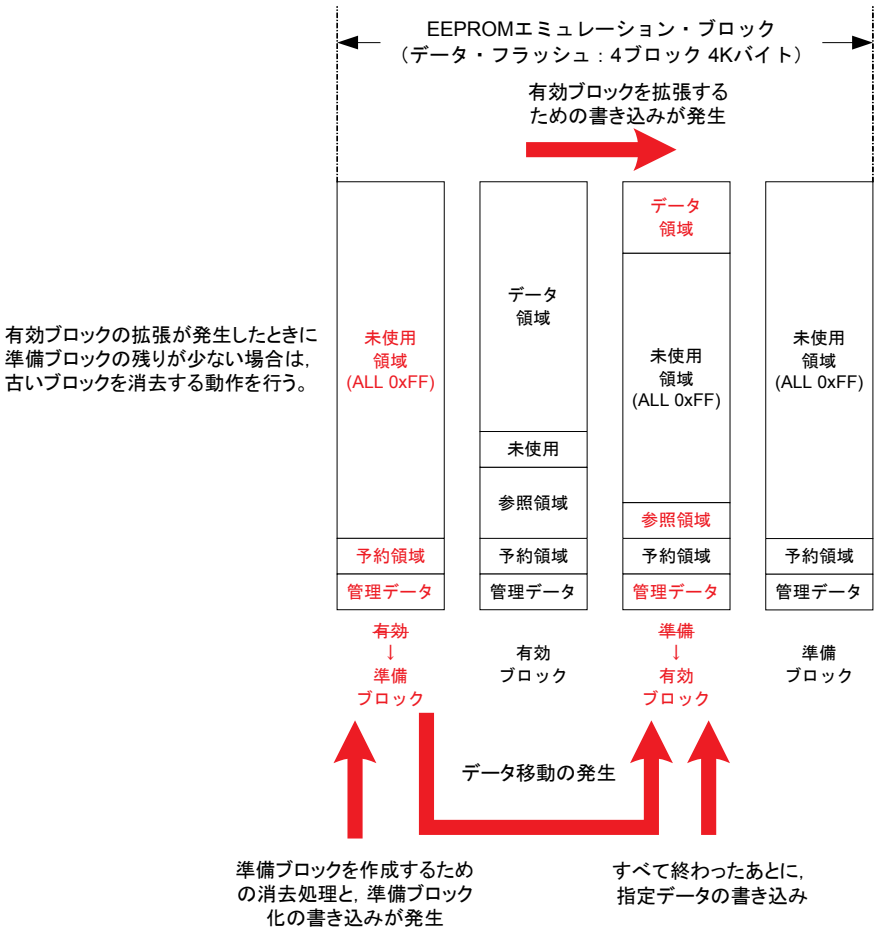
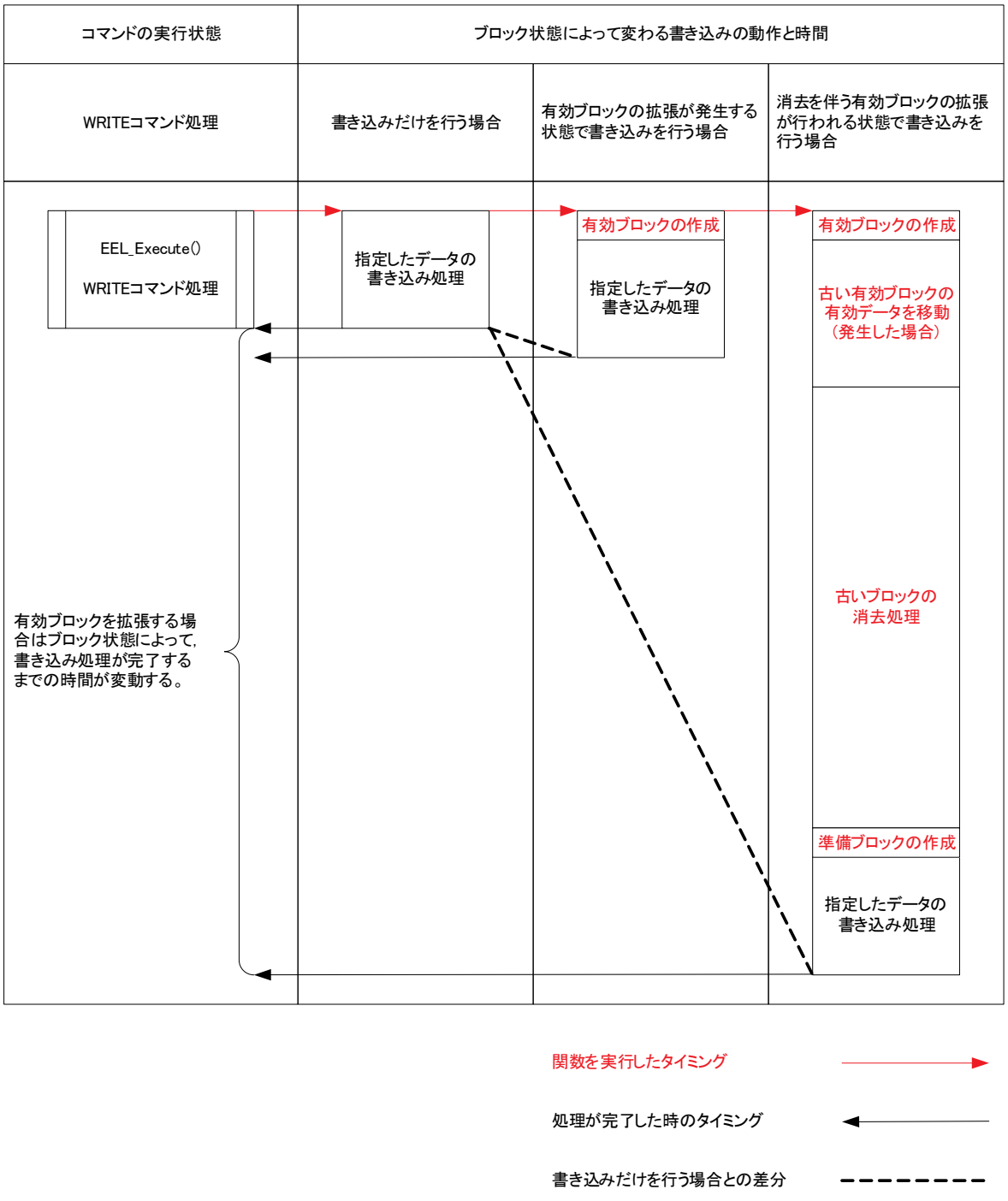


図 1－19　データ書き込み時にブロック状態によって処理時間が変わるときのイメージ図(Enforcedモード使用時)



1. 4. 1　EEL_CMD_CLEANUPコマンドによるブロックの整理

EEL_Execute関数からEEL_CMD_CLEANUPコマンドを実行することにより、ブロックの整理を行う事ができます。

本機能は、最新の有効ブロックに残り容量がある状態でもすべてのデータを新しく作成した最新の有効ブロックに移動させるため、書き換え可能回数がデータ格納に必要なブロック数分減少しますが、有効データのみの状態（無効となったデータが存在しない状態）にすることが可能です。

図1－20にEEL_CMD_CLEANUPコマンドを実行する場合のフロー、図1－21にEEL_CMD_CLEANUPコマンドを実行したときのブロック状態を示します。

図 1－20　EEL_CMD_CLEANUPコマンドの操作フロー(Enforcedモード使用時)

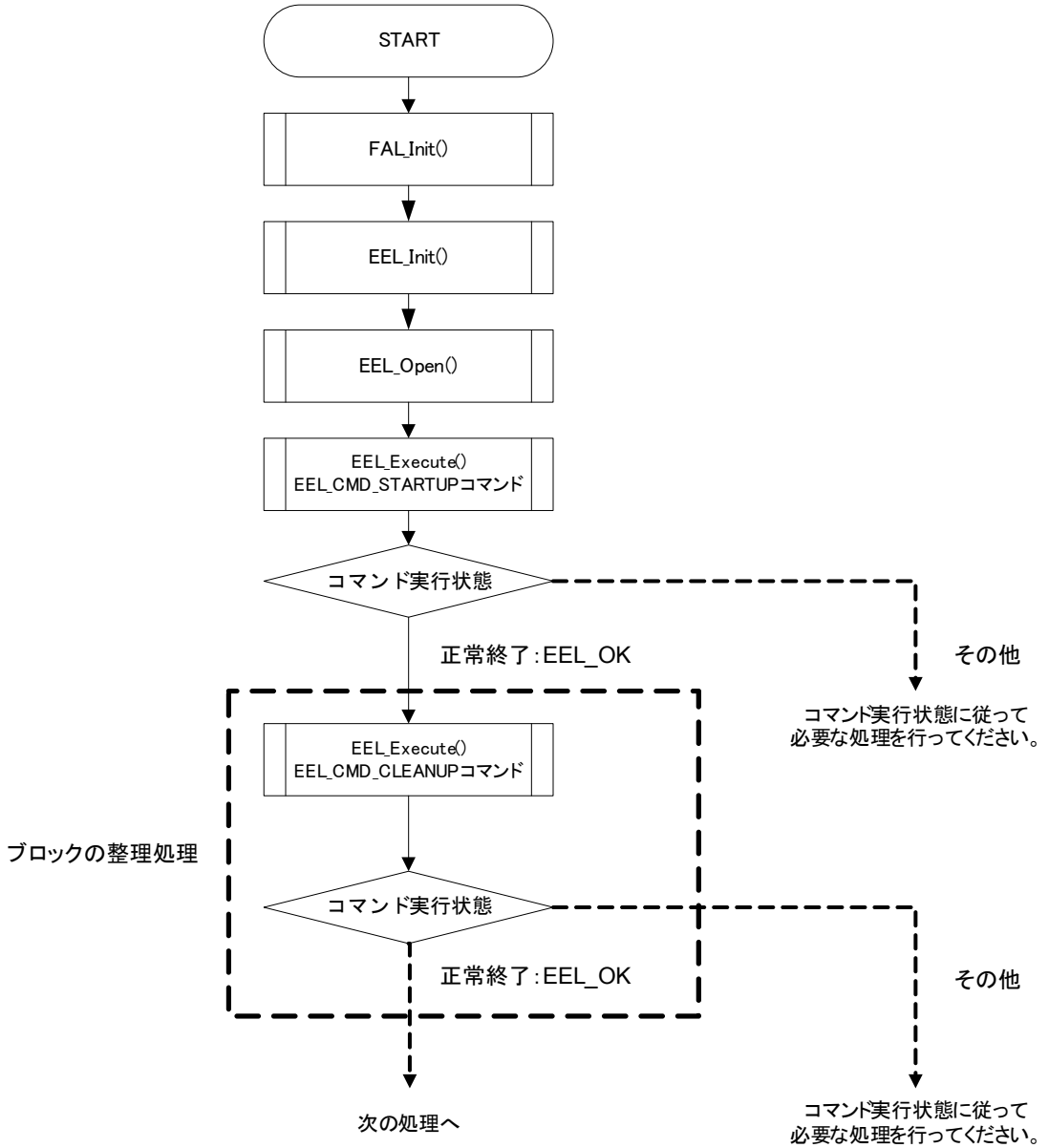
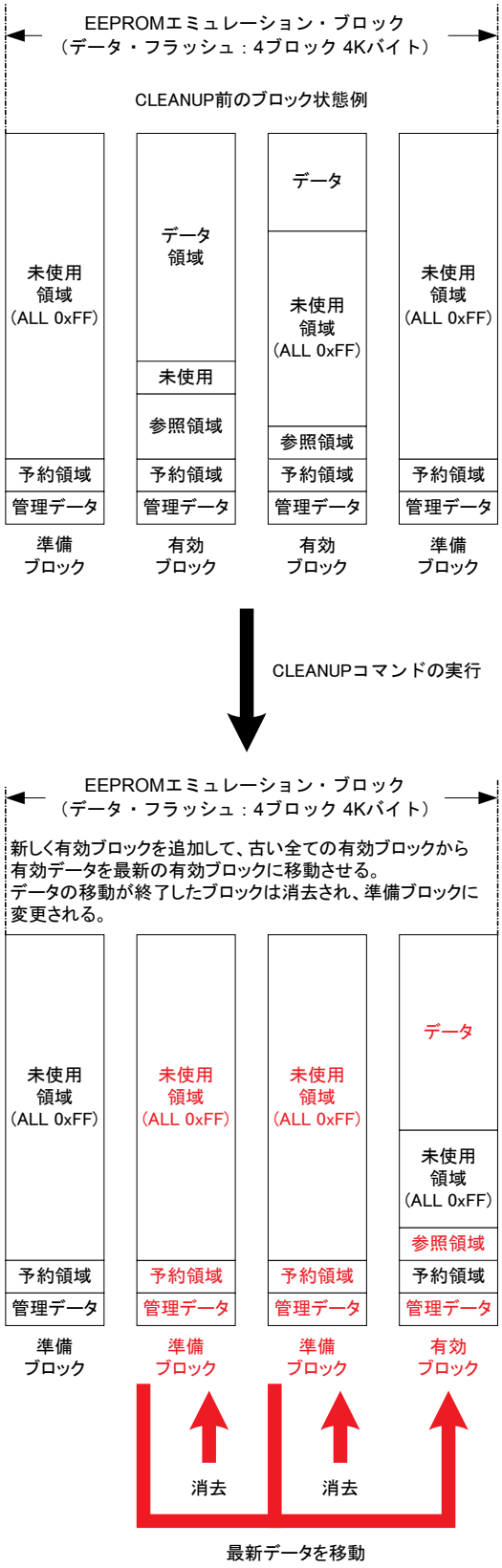


図 1−21　EEL_CMD_CLEANUPコマンドによるブロックの整理



1. 4. 2　EEL_Handler関数によるブロックの整理（メンテナンス・モード）

コマンドが実行されていない状態でEEL_Handler関数を実行すると自動的にブロックを整理する処理が動き始めます。これをメンテナンス・モードと言います。EEL_Handler関数は実行するコマンドがない状態で実行されるとブロックの状態を確認し、初期値で設定する「EEL_REFRESH_BLOCK_THRESHOLD」（**2. 4　ユーザ設定初期値**を参照）の値より有効ブロック数が多い場合はブロックの整理が必要と判断し、古い有効ブロックから必要なデータを最新ブロックへ移動させ、不要になったブロックを消去して準備ブロックに変更します。

メンテナンス・モードは、処理の中断が可能です。EEL_Execute関数で新たなコマンドが実行されるとメンテナンス・モードを一時停止させ、新たに指定されたコマンドを優先して実行します。メンテナンス・モードを実行中かどうかの判定については、別途EEL_GetDriverStatus関数を実行し、EEL_Handler関数の実行状態を確認する必要があります。EEL_Handle関数を4回実行してもすべて何も行っていない場合は整理が完了している状態（整理完了状態）になります。

図1-22、表1-3にEEL_Handler関数の実行状態の確認方法、図1-23～図1-26に実行フロー例、図1-27にメンテナンス・モードが終了した状態（整理完了状態）のブロック遷移を記載します。

図 1－22　EEL_Handler関数の実行状態確認フロー

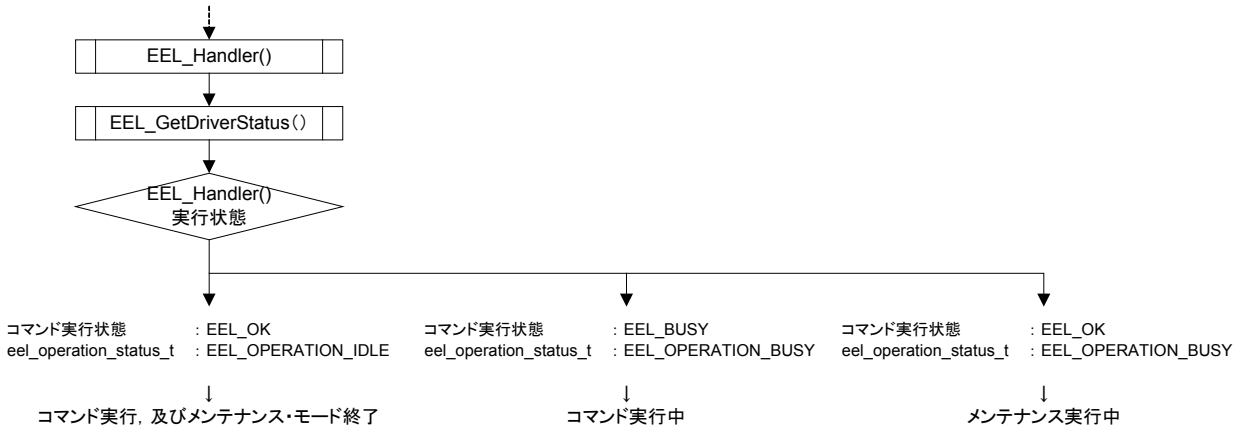


表 1－3　EEL_Handler関数の実行状態

EEL_Handler() コマンド実行状態	EEL_GetDriverStatus() eel_operation_status_tの状態	説明
EEL_OK	EEL_OPERATION_IDLE	コマンド実行、およびメンテナンス・モード終了状態 （メンテナンス・モードは一時的な終了も含む） EEL_Handler関数を4回実行してもすべての結果がこの状態の場合は整理完了
EEL_OK	EEL_OPERATION_BUSY	メンテナンス・モード実行中
EEL_BUSY	EEL_OPERATION_BUSY	コマンド実行中

図 1－23　メンテナンス・モードの実行フロー（整理完了状態になるまでの実行例）

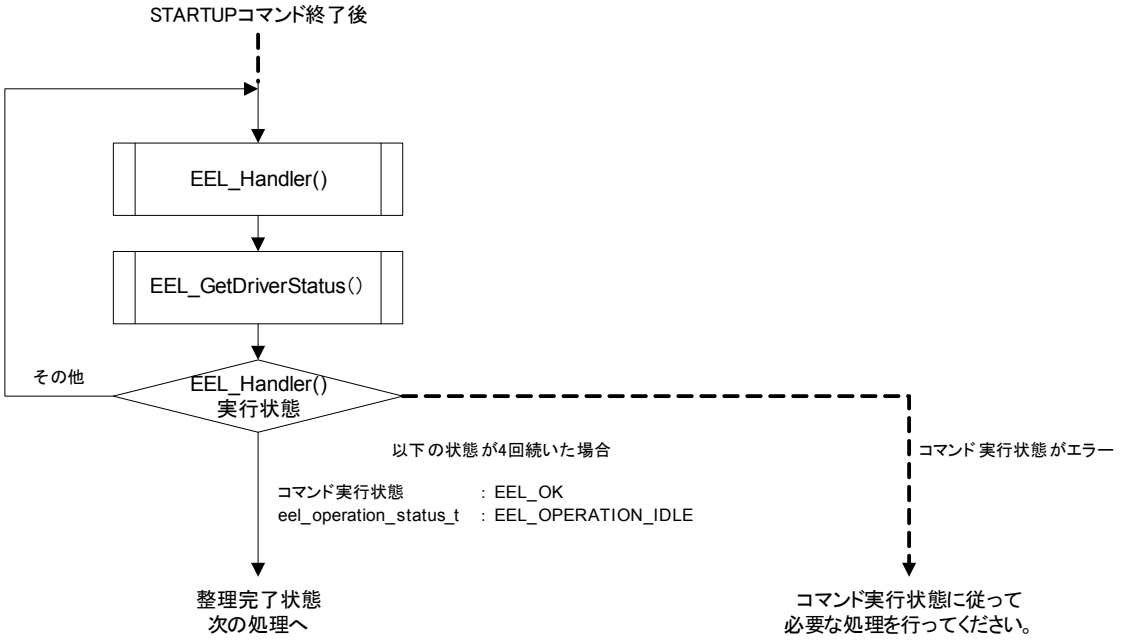


図 1－24　EEL_CMD_WRITEコマンド実行後に整理を行う場合の実行フロー（整理完了状態）

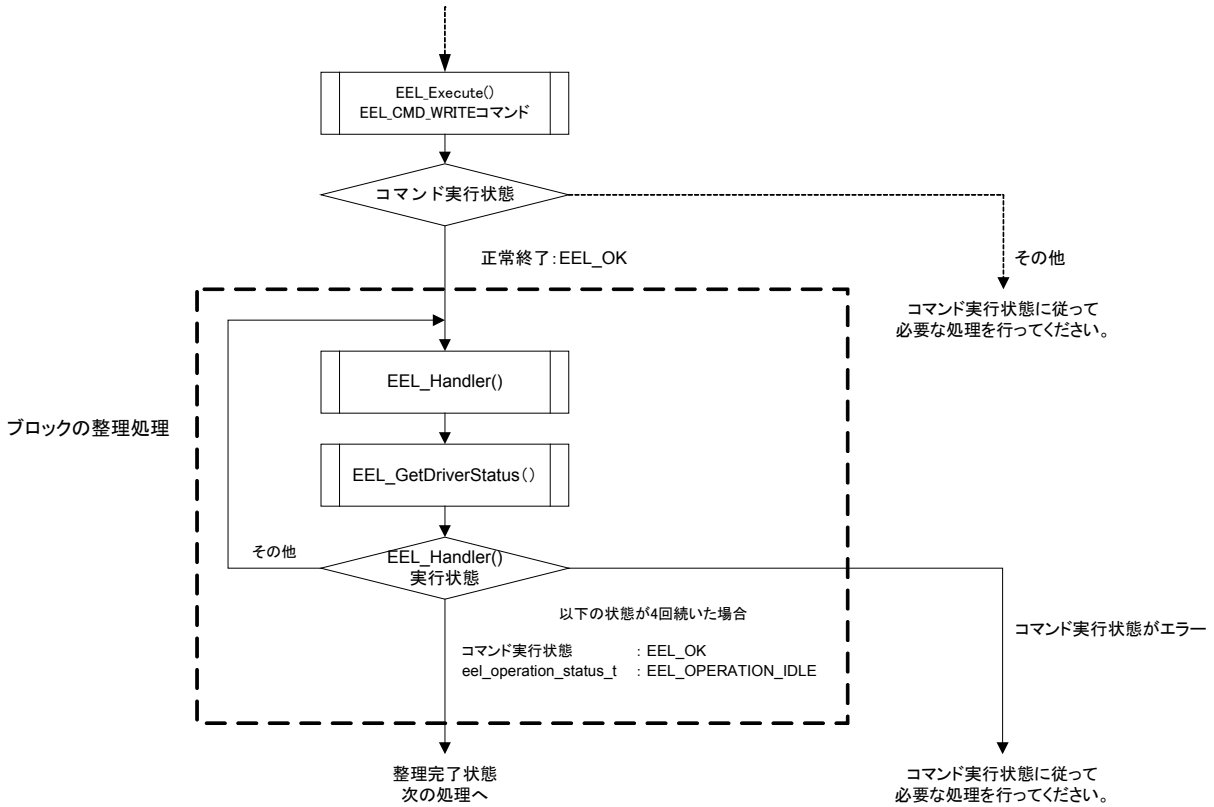


図 1−25　EEPROMエミュレーション開始時に整理を行う場合の実行フロー（整理完了状態）

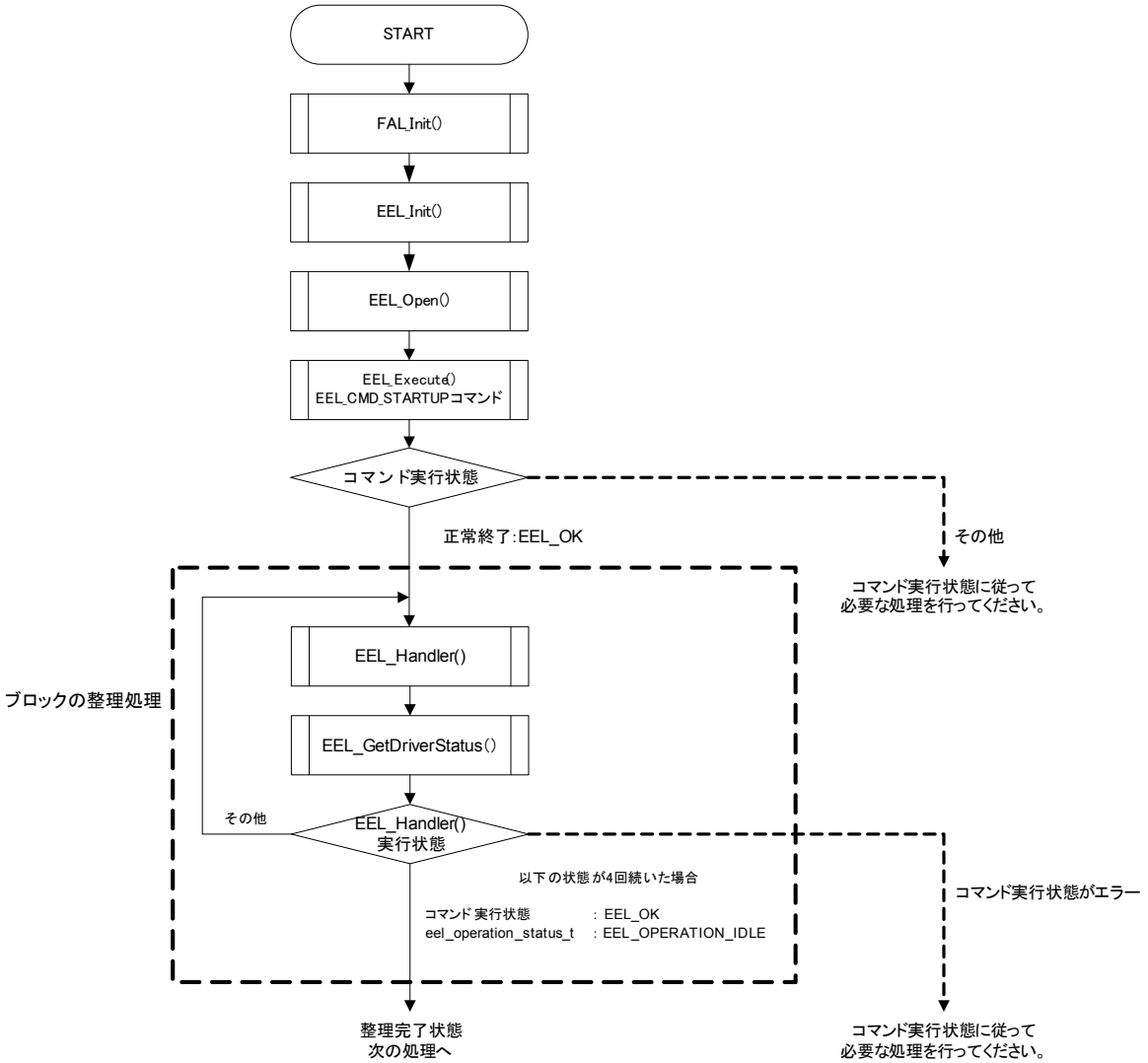


図 1−26　ポーリング・モードでコマンドとメンテナンス・モードを並行して実行するフロー例

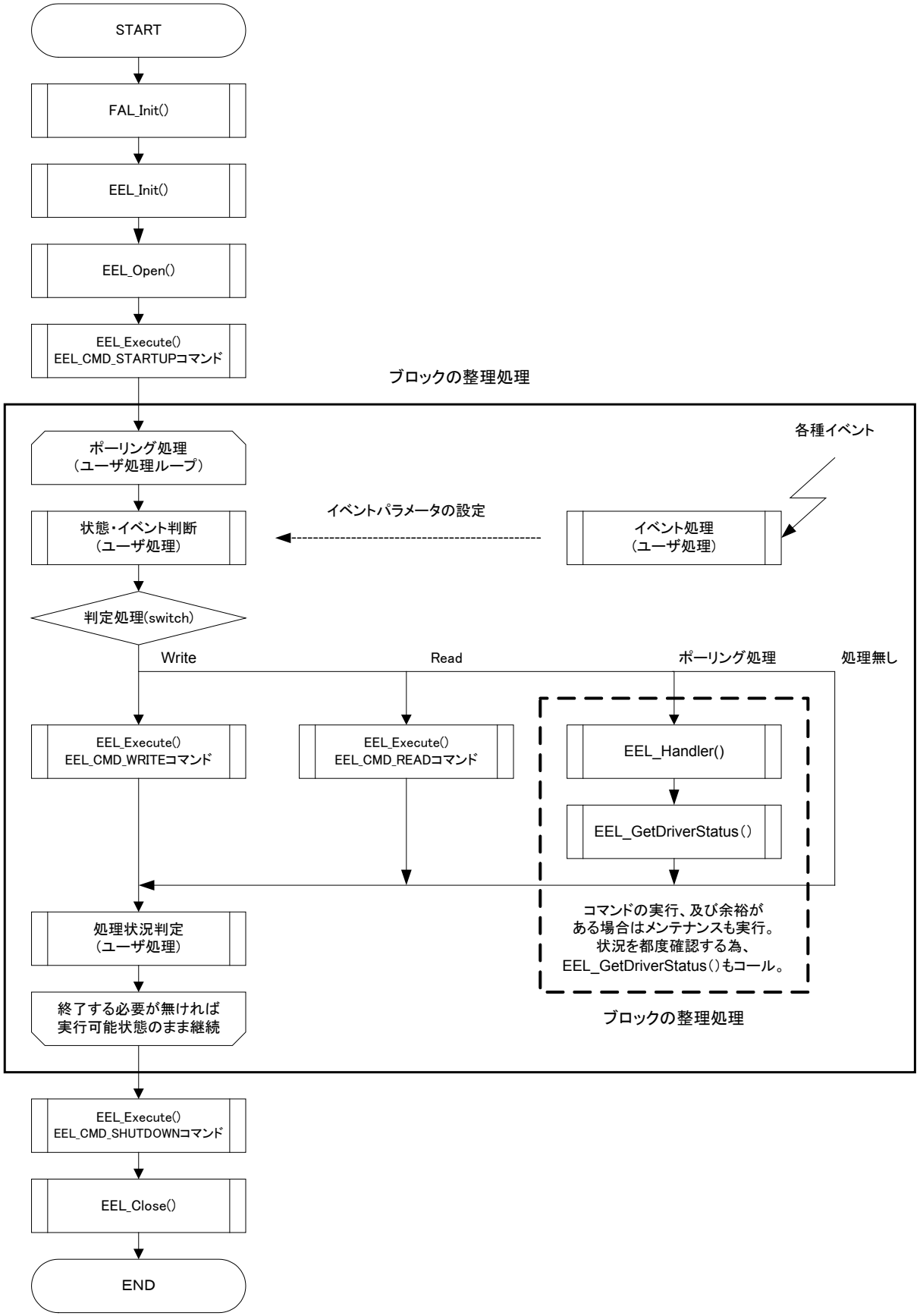
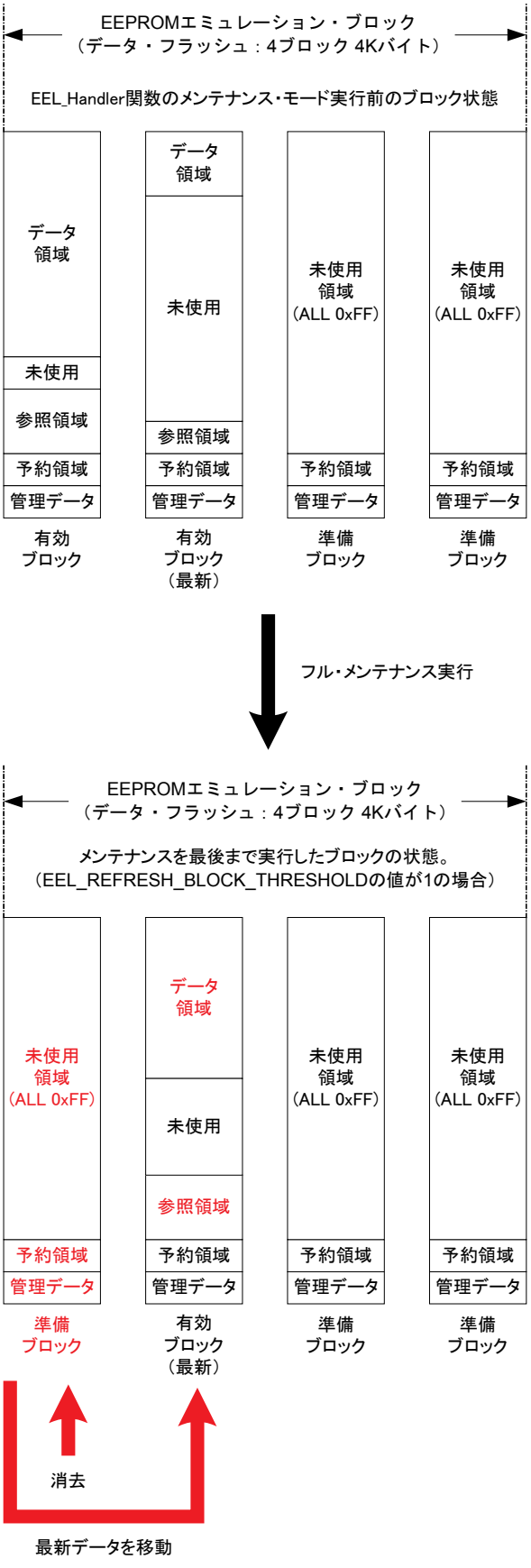


図 1－27　EEL_Handler関数のメンテナンス・モードによるブロックの整理例(整理完了状態)



第2章　EEPROMエミュレーションの使用方法

EEPROM エミュレーションは、4ブロック以上のデータ・フラッシュ・メモリを使用することにより、1～255 バイトのデータを最大255個[注] EEPROM エミュレーションによりデータ・フラッシュ・メモリに格納することができます。

このEEPROM エミュレーション・ライブラリをユーザ・プログラムに組み込み、そのプログラムを実行することにより、EEPROM エミュレーションを実現することができます。

注　格納可能なユーザ・データ数の詳細については**1.2.4 格納ユーザ・データ数とユーザ・データの合計サイズ**を参照してください。

2. 1　注意事項

EEPROM エミュレーションはマイコンに搭載しているデータ・フラッシュ・メモリを操作する機能を使用して実現しています。このため、次の点に注意する必要があります。

（1）　EEPROMエミュレーション実行中にフラッシュ・セルフ・プログラミング・ライブラリを実行しないでください。フラッシュ・セルフ・プログラミング・ライブラリを使用する場合は必ずEEL_Close関数まで実行し、EEPROMエミュレーションを終了状態にする必要があります。
フラッシュ・セルフ・プログラミング・ライブラリを実行後にEEPROMエミュレーションを使用する場合は、初期化関数(FAL_Init関数)から処理を行う必要があります。

（2）　EEPROMエミュレーション実行中にSTOP命令、およびHALT命令は実行しないでください。STOP命令、およびHALT命令を実行する必要がある場合は必ずEEL_Close関数まで実行し、EEPROMエミュレーションを終了させてください。

（3）ウォッチドック・タイマはEEPROMエミュレーション・ライブラリ実行中も停止しません。

（4）　EEPROMエミュレーション・ライブラリによるデータ・フラッシュ・メモリ操作中はデータ・フラッシュ・メモリを読み出せません。

（5）　EEPROMエミュレーション・ライブラリ関数、およびデータ・フラッシュ・ライブラリ関数で使用するデータ・バッファ(引数)やスタックを0xFFE20(0xFE20)以上のアドレスに配置しないでください。

（6）　EEPROMエミュレーションの実行中にデータ・トランスファ・コントローラ（DTC）を使用する場合は、DTCで使用するRAM領域をセルフRAM、および 0xFFE20(0xFE20)以上のアドレスに配置しないでください。

（7） EEPROMエミュレーションが終了するまで、EEPROMエミュレーションが使用するRAM領域（セルフRAM含む）を破壊しないでください。

（8） EEPROMエミュレーション・ライブラリ関数のうち、「EEL_TimeOut_CountDown」関数以外を割り込み処理内で実行しないでください。EEPROMエミュレーション・ライブラリは関数の多重実行に対応していないため、割り込み処理内で実行された場合は動作保証ができません。

（9）OS上でEEPROMエミュレーション・ライブラリを実行する場合は、EEPROMエミュレーション・ライブラリ関数のうち、「EEL_TimeOut_CountDown」関数以外を複数のタスクから実行しないでください。
EEPROMエミュレーション・ライブラリは関数の多重実行に対応していないため、複数のタスクで実行された場合は動作保証ができません。

（10）EEPROMエミュレーションを開始する前に高速オンチップ・オシレータを起動しておく必要があります。

（11）RL78マイクロコントローラのCPUの動作周波数と初期化関数（FAL_Init）で設定するCPUの動作周波数値について、以下の点に注意してください。

- RL78マイクロコントローラのCPUの動作周波数として4MHz未満の周波数を使用する場合は、1 MHz、 2 MHz、 3 MHz のみを使用することができます（1.5 MHz のように整数値にならない周波数は使用できません）。
- RL78マイクロコントローラのCPUの動作周波数として4 MHz以上[注1]の周波数を使用する場合は、RL78マイクロコントローラに任意の周波数を使用することができます。
- 高速オンチップ・オシレータの動作周波数ではありません。

注 1. 最大周波数については、対象となるRL78マイクロコントローラのユーザーズ・マニュアルをご参照ください。

（12）EEPROMエミュレーション・ライブラリの実行中にDFLCTL（データ・フラッシュ・コントロール・レジスタ）を操作しないでください。

（13）EEPROMエミュレーション・ライブラリで使用する引数(RAM)は一度初期化してください。初期化をしない場合、RAMパリティ・エラーが検出され、RL78マイクロコントローラにリセットが発生する可能性があります。
RAMパリティ・エラーについては、対象となるRL78マイクロコントローラのユーザーズ・マニュアルを参照してください。

（14）EEPROMエミュレーション・ライブラリで使用するSADDR領域は一度初期化してください。初期化をしない場合、RAMパリティ・エラーが検出され、RL78マイクロコントローラにリセットが発生する可能性があります。
RAMパリティ・エラーについては、対象となるRL78マイクロコントローラのユーザーズ・マニュアルを参照してください。

（15）データ・フラッシュ・メモリをEEPROMエミュレーションで使用するためには初回起動時にEEL_CMD_FORMATコマンドを実行し、データ・フラッシュ・メモリをEEPROMエミュレーション・ブロックとして使用できるように初期化をおこなう必要があります。

（16）EEPROMエミュレーション・ブロックや格納データ用に固定値で設定している初期値はEEPROMエミュレーション・ブロックの初期化後に変更しないでください。変更した場合、設定されているパラメータがブロックに書き込まれているデータとの整合が取れなくなり、EEPROMエミュレーションが正常に実行できなくなる恐れがあります。変更する必要がある場合はEEL_CMD_FORMATコマンドを実行し、EEPROMエミュレーション・ブロックを再初期化してください。

（17）EEPROMエミュレーション・ライブラリ Pack01を使用するためには、データ・フラッシュ・メモリが4ブロック以上必要です。データ・フラッシュ・メモリが4ブロック未満の場合、本ライブラリは使用できません。

2. 2　処理時間

この節では EEPROM エミュレーション・ライブラリ Pack01 の処理時間について記載します。

(1) 総合処理時間

総合処理時間は正常終了する場合の時間です。入力データに誤りがある、あるいはエラーなどにより異常終了する場合の処理時間は含みません。

図 2-1　総合処理時間の概念図

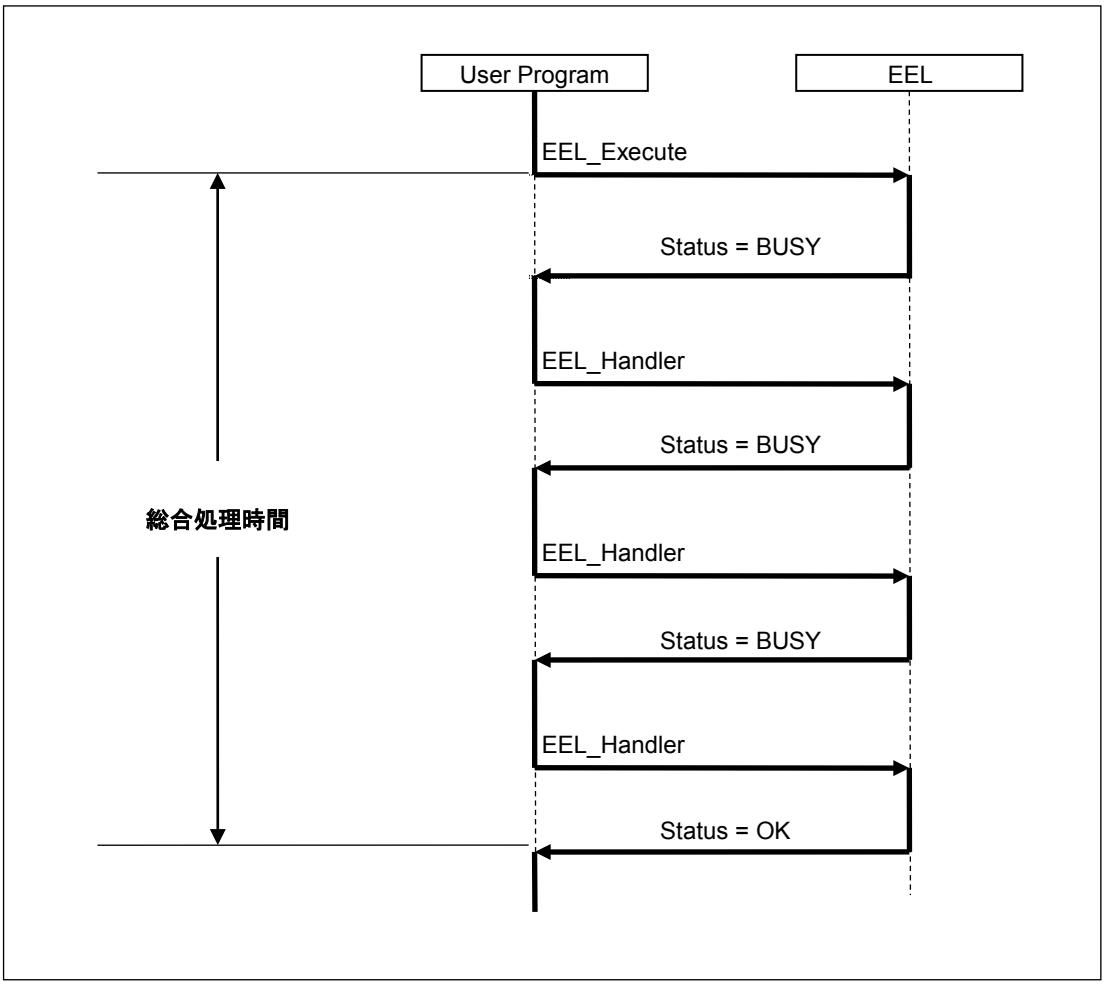


表 2-1 EEPROM エミュレーション・ライブラリ Pack01 総合処理時間

(全データが 1 ブロック内に格納可能な場合)

Funcstions		MAX time(Full Speed Mode)	MAX time(Wide Voltage Mode)
FAL_Init		2580 / fclk + 443 µs	2536 / fclk + 968 µs
EEL_Init		(2123 + 80 × Data Num) / fclk µs	(2123 + 80 × Data Num) / fclk µs
EEL_Open		87 / fclk + 12 µs	87 / fclk + 12 µs
EEL_Close		866 / fclk + 443 µs	822 / fclk + 968 µs
EEL_GetDriverStatus		47 / fclk µs	47 / fclk µs
EEL_GetSpace		57 / fclk µs	57 / fclk µs
EEL_GetVersionString		10 / fclk µs	10 / fclk µs
EEL_TimeOut_CountDown		30 / fclk µs	30 / fclk µs
EEL_Excute/EEL_Handler			
・EEL_CMD_FORMAT		(352494 / fclk + 330988) × Block Num µs	(311793 / fclk + 374134) × Block Num µs
・EEL_CMD_STARTUP 1.最小データ長が 1～4バイト (最小サイズ)	Data Num 8	(1082020 + 491768 × (Block Num - 2)) / fclk + 330988 µs	(1054886 + 491768 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 64	(1926490 + 4482900 × (Block Num - 2)) / fclk + 330988 µs	(1899356 + 4482900 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 128	(2891690 + 9044100 × (Block Num - 2)) / fclk + 330988 µs	(2864556 + 9044100 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 255	(4806090 + 18097500 × (Block Num - 2)) / fclk + 330988 µs	(4777956 + 18097500 × (Block Num - 2)) / fclk + 374134 µs
2.最小データ長が 13～16バイト[注]	Data Num 8	(771606 + 256268 × (Block Num - 2)) / fclk + 330988 µs	(744472 + 256268 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 64	(1189076 + 2363400 × (Block Num - 2)) / fclk + 330988 µs	(1161942 + 2363400 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 128	(1666676 + 4771500 × (Block Num - 2)) / fclk + 330988 µs	(1639542 + 4771500 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 255	(2613676 + 9550500 × (Block Num - 2)) / fclk + 330988 µs	(2586542 + 9550500 × (Block Num - 2)) / fclk + 374134 µs
3.最小データ長が 61～64バイト[注]	Data Num 8	(569540 + 101109 × (Block Num - 2)) / fclk + 330988 µs	(542406 + 101109 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 64	(705816 + 967779 × (Block Num - 2)) / fclk + 330988 µs	(678682 + 967779 × (Block Num - 2)) / fclk + 374134 µs
	Data Num 128	(861528 + 1958100 × (Block Num - 2)) / fclk + 330988 µs	(834394 + 1958100 × (Block Num - 2)) / fclk + 374134 µs

備考.　fclk　　：CPUクロック周波数(例：20Mhz時のfclk = 20)

Data Num　：登録データ数

Block Num　：EEPROMエミュレーション・ブロック数

注.　使用予定の最小データ長がない場合は、より小さい最小データ長の処理時間をご確認ください。

Funcstions	MAX (Full Speed Mode)	MAX (Wide Voltage Mode)
EEL_Excute/EEL_Handler		
・EEL_CMD_CLEANUP	(352494 / fclk + 330988) × Block Num + 4236841 / fclk + 223893 µs	(311793 / fclk + 374134) × Block Num + 4219942 / fclk + 851467 µs
・EEL_CMD_WRITE (最大データ長)	5763174 / fclk + 1024588 µs	5650473 / fclk + 1906134 µs
・EEL_CMD_READ	58563 / fclk µs	58563 / fclk µs
・EEL_CMD_SHUTDOWN	1674 / fclk µs	1674 / fclk µs

備考.　fclk　　：CPUクロック周波数(例：20Mhz時のfclk = 20)

Block Num ：EEPROMエミュレーション・ブロック数

2.3　ソフトウェア・リソース

EEPROMエミュレーション・ライブラリでは、該当プログラムをユーザ領域に配置するため、使用するライブラリ分の容量のプログラム領域、ライブラリ内で使用する変数、ワーク・エリア（セルフRAM）分のRAM領域を消費します。また、EEPROMエミュレーション・ライブラリはデータ・フラッシュ・ライブラリを使用するため、データ・フラッシュ・ライブラリで使用する領域も別途必要となります。

表 2−2、 2−3 に、必要となるソフトウェア・リソースの一覧[注 1,2]、図 2−2、2−3 に RAM の配置イメージ例を示します。

★ **表 2-2 EEPROM エミュレーション・ライブラリ Pack01 Ver.1.14 ソフトウェア・リソース**

項目	容量(バイト)	EEPROMエミュレーション・ライブラリ Pack01の使用領域[注1]
セルフRAM[注2]	0 ～ 1024[注2]	RL78ファミリ EEPROMエミュレーション・ライブラリ Pack01 で使用するセルフRAM領域は 各デバイス毎に異なります。詳細については、『RL78ファミリ セルフプログラミングライブラリ セルフRAMリスト(R20UT2943)』を参照してください。
スタック	100	セルフRAM、FFE20H-FFEFFH以外のRAM領域に配置可能
データ・バッファ[注3]	1 ～ 256	
リクエスタ	6	
ライブラリ検索領域	2	
SADDR RAM ワーク・エリア	SADDR : 11 (fdl:2) (eel:9)	ショート・アドレッシングRAM領域のみ配置可能
ライブラリ・サイズ	8200 (fdl:1500) (eel:6700)	セルフRAM、FFE20H-FFEFFH以外のプログラム領域に配置可能（ROM）
データ・テーブル	(n + 1) * 4 N = 格納データ数	
固定パラメータ領域 (デフォルト値)	72 (fdl:64) (eel:4)	
EEPROMエミュレーション・ブロック	4ブロック以上 (4kByte ～ Max)	データ・フラッシュ・メモリのみ使用可能 (コード・フラッシュ・メモリは使用不可)

★ 注 1.　『RL78 ファミリ セルフプログラミングライブラリ セルフ RAM リスト(R20UT2943)』に掲載の無い製品については、お問い合わせください。

2.　EEPROM エミュレーション・ライブラリ Pack01 でワーク・エリアとして使用する領域を本書、及びリリース・ノートではセルフ RAM と呼びます。セルフ RAM はマッピングされず、EEPROM エミュレーション・ライブラリ実行時に自動的に使用される（以前のデータは破壊される）領域のため、ユーザ設定等は必要ありません。EEPROM エミュレーション・ライブラリを使用していない状態の場合は、通常の RAM 空間として使用できます。

3.　データ・バッファは、EEPROM エミュレーション・ライブラリ内部処理で使用するワーク領域、または EEL_Execute 関数では設定するデータを配置する領域として使用します。必要となるサイズは、使用する関数によって異なります。

表 2-3 EEPROM エミュレーション・ライブラリ Pack01 関数のデータ・バッファ使用サイズ

関数名	バイト	関数名	バイト
FAL_Init	0	EEL_Handler注	0 ～ 256
EEL_Init	0	EEL_TimeOut_CountDown注	0
EEL_Open	0	EEL_GetDriverStatus	3
EEL_Close	0	EEL_GetSpace	2
EEL_Execute注	0 ～ 256	EEL_GetVersionString	0

注.　別途リクエスタ(6Byte)の領域を使用します。

図2−2 RAMの配置イメージ例1 / セルフRAM有り（RL78/G13：RAM 4KB/ROM 64KB製品）

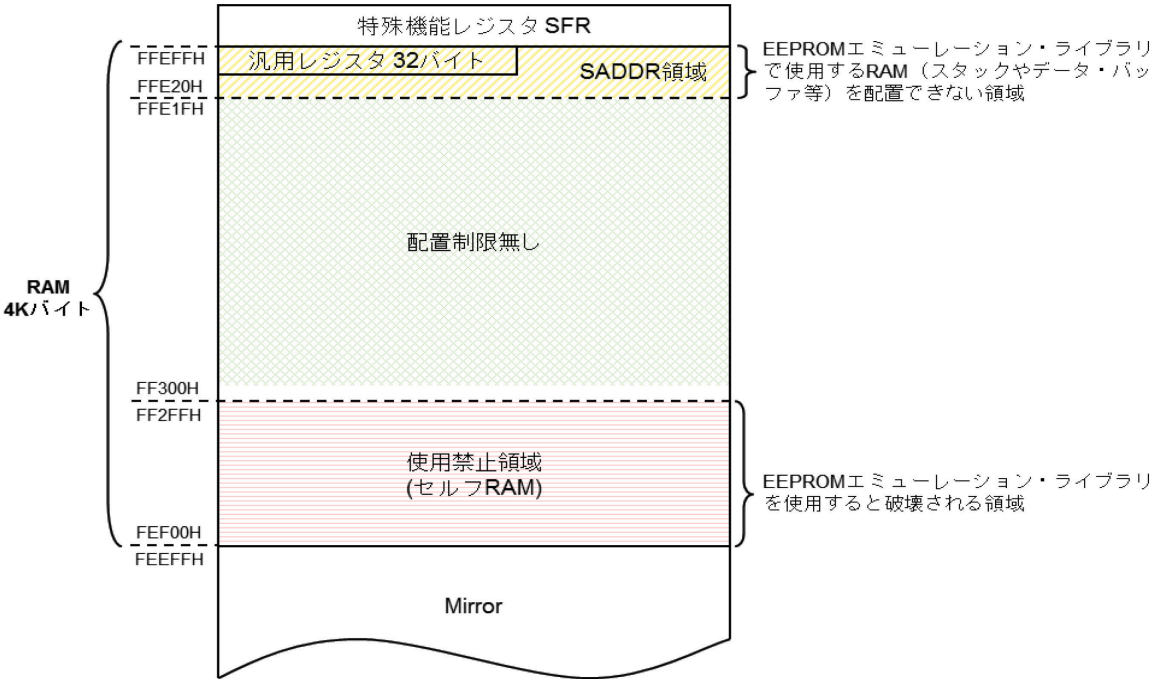
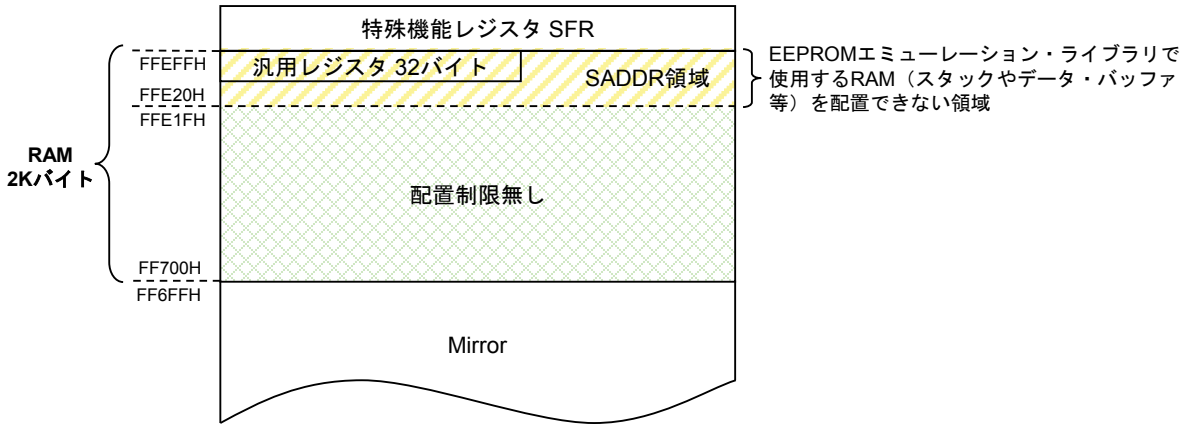


図2−3 RAMの配置イメージ例2 / セルフRAM無し（RL78/G13：RAM 2KB/ROM 32KB製品）



2. 4　ユーザ設定初期値

EEPROMエミュレーション・ライブラリの初期設定値として、次に示す項目を必ず設定する必要があります。また、EEPROMエミュレーション・ライブラリを実行する前に、高速オンチップ・オシレータを起動しておく必要があります。

・格納データのデータ数、および個々のデータIDとデータ・サイズ

＜ データ・フラッシュ・ライブラリ・ユーザ・インクルード・ファイル (fdl_descriptor.h) ＞注1,2

```
#define    FDL_SYSTEM_FREQUENCY   20000000        : (1) 動作周波数
#define    FDL_WIDE_VOLTAGE_MODE                  : (2) 電圧モード
#define    FAL_POOL_SIZE          4               : (3) FALプール・サイズ
#define    EEL_POOL_SIZE          0               : (4) EELプール・サイズ
```

＜ EEPROMエミュレーション・ライブラリ・ユーザ・インクルード・ファイル (eel_descriptor.h) ＞注1,2

```
#define   EEL_STORAGE_TYPE     'D'                : フラッシュ・タイプ（D : データ・フラッシュ）
#define   EEL_VAR_NO           5                  : 格納データ数
#define   EEL  REFRESH  BLOCK  THRESHOLD   1      : スレッショルド設定
```

＜ EEPROMエミュレーション・ライブラリ・ユーザ・プログラム・ファイル (eel_descriptor.c) ＞注1,2

```
__far const eel_u08     eel_descriptor[EEL_VAR_NO+1][4]    : (5) データの識別子（データID）、およびサイズ

__far const eel_u08     eel_refresh_bth_u08                = (eel_u08)EEL_REFRESH_BLOCK_THRESHOLD;
                                                           : (6) スレッショルド設定
__far const eel_u08     eel_storage_type_u08               = (eel_u08)EEL_STORAGE_TYPE;
                                                           : (7) フラッシュ・タイプ
__far const eel_u08     eel_var_number_u08                 = (eel_u08)EEL_VAR_NO;
                                                           : (8) 格納データ数
```

注 1. 使用しているマクロ、およびマクロ名はEEPROMエミュレーション・ライブラリ共通のパラメータとして使用していますので、数値以外は変更しないでください。

2. EEPROMエミュレーション・ブロック初期化後(EEL_CMD_FORMATコマンド実行後)は各値を変更しないでください。変更する場合はEEPROMエミュレーション・ブロックの再初期化を行ってください。

(1) CPUの動作周波数

RL78マイクロコントローラで使用されているCPUの動作周波数を設定します。[注]

設定値は以下の計算式によりFAL_Initの周波数パラメータへ設定されます（周波数は切り上げで計算されます。計算結果の小数点は切り捨てになります）。

FAL_InitのCPUの動作周波数設定値 = ((FDL_SYSTEM_FREQUENCY+999999)/1000000)

例1：FDL_SYSTEM_FREQUENCYが20000000(20MHz)の場合
((20000000+999999)/1000000) = 20.999999 = 20

例2：FDL_SYSTEM_FREQUENCYが4500000(4.5MHz)の場合
((4500000+999999)/1000000) = 5.499999 = 5

例3：FDL_SYSTEM_FREQUENCYが5000001(5.000001MHz)の場合
((5000001+999999)/1000000) = 6.000000 = 6

注　本設定はデータ・フラッシュ・メモリの制御に必要な値になります。本設定により、RL78マイクロコントローラのCPUの動作周波数が変わることはありません。
また、高速オンチップ・オシレータの動作周波数ではありません。

(2) 電圧モード [注1]

データ・フラッシュ・メモリの電圧モードを設定します。[注2]
FDL_WIDE_VOLTAGE_MODEが定義されていない場合　：フルスピード・モード
FDL_WIDE_VOLTAGE_MODEが定義されている場合　　：ワイド・ボルテージ・モード

注 1.　初期設定ではFDL_WIDE_VOLTAGE_MODEはコメントアウトされており、定義されていません。ワイド・ボルテージ・モードで使用する場合は、コメントアウトを外して定義されるように修正してください。
2.　電圧モードの詳細については、対象となるRL78マイクロコントローラのユーザーズ・マニュアルを参照ください。

(3) FALプール・サイズ

使用中のRL78マイクロコントローラに搭載されているデータ・フラッシュ・メモリのブロック数を設定してください。

(4) EELプール・サイズ [注]

FALプール・サイズと同様に使用中のRL78マイクロコントローラに搭載されているデータ・フラッシュ・メモリのブロック数を設定してください。

注　必ず4 (4ブロック)以上の値を設定してください。

(5) データの識別子(データID)、およびサイズ

データの識別子(データID)、およびサイズを規定するテーブルです。これをEELディスクリプタ・テーブルといいます。RL78 EEPROMエミュレーション・ライブラリ Pack01では、プログラム動作中に識別子を追加することができません。したがって、書き込みを行うデータをEELディスクリプタ・テーブルに事前に登録する必要があります。

図 2-4　EELディスクリプタ・テーブル（4件の異なったデータがある場合）

__far const eel_u08　eel_descriptor[格納データ数 ＋ 1][4]

データID (A)	ワード・サイズ	バイト・サイズ	0x01
データID (B)	ワード・サイズ	バイト・サイズ	0x01
データID (C)	ワード・サイズ	バイト・サイズ	0x01
データID (D)	ワード・サイズ	バイト・サイズ	0x01
0x00	0x00	0x00	0x00

データID

ユーザが指定するデータ ID です。

ワード・サイズ

書き込むデータのワード・サイズです。

バイト・サイズ

書き込むデータのバイト・サイズです。

RAM参照フラグ(0x01)

参照設定用のフラグです。登録データには 1 を設定してください。

終端領域(0x00)

終端情報として 0 を設定します。

(6) スレッショルド設定

メンテナンス・モードを実行した場合に整理基準となる有効ブロック数です。メンテナンス・モードは有効ブロック数がこの設定値を超えた場合に実行されます。設定するブロック数は有効データの格納に必要なブロック数[注]に＋1をした値を設定します。データ格納に必要な容量が1ブロックの最大使用可能サイズの半分以下の場合は＋1をする必要はありません。

注　有効データの格納に必要なブロック数の詳細については**1.2.4 格納ユーザ・データ数とユーザ・データの合計サイズ**を参照してください。

(7) フラッシュ・タイプ

変更する必要はありません。初期値のままご使用ください。

(8) 格納データ数

EEPROMエミュレーションで使用するデータ数を設定します。設定できる値は1～255の範囲です。

第3章　EEPROMエミュレーション機能

この章では、EEPROMエミュレーションで使用する関数の詳細について説明しています。

3.1　データ・フラッシュ・ライブラリ関数

データ・フラッシュ・メモリを使用してEEPROMエミュレーションを実行する場合には、EEPROMエミュレーション・ライブラリからデータ・フラッシュ・ライブラリを操作し、データ・フラッシュ・メモリに対して消去や書き込み等の処理を行います。そのため、EEPROMエミュレーションを実行する前にはデータ・フラッシュ・ライブラリを初期化し、データ・フラッシュ・メモリへアクセスするための設定、および準備を行う必要があります。表3－1にEEPROMエミュレーション・ライブラリを実行するために必要なデータ・フラッシュ・ライブラリの関数を示します。

表 3－1　EEPROMエミュレーションを使用する前に実行する必要のあるデータ・フラッシュ・ライブラリ関数

関数名	概　要
FAL_Init	データ・フラッシュ・ライブラリの初期化

FAL_Init

【概　要】
データ・フラッシュ・ライブラリの初期化処理

【書　式】
＜C 言語＞

```
fal_status_t   __far FAL_Init(const __far fal_descriptor_t* descriptor_pstr)
```

＜アセンブラ＞

```
CALL !_FAL_Init  または  CALL !!_FAL_Init
```

備考　データ・フラッシュ・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】

1. フラッシュ・セルフ・プログラミング・ライブラリ、およびEEPROMエミュレーション・ライブラリが未実行、あるいは終了していること。
2. 高速オンチップ・オシレータを起動しておくこと。

【機　能】
データ・フラッシュ・メモリへアクセスする為のパラメータを初期化します。

注意 1. EEPROM エミュレーションを開始するときには必ず本関数を実行し、データ・フラッシュ・メモリへのアクセスが開始できるようにしてください。
2. フラッシュ・セルフ・プログラミング・ライブラリ(FSL)とは排他利用となります。本関数を実行する前にフラッシュ・セルフ・プログラミング・ライブラリは必ず終了させてください。また、 EEPROM エミュレーション・ライブラリ実行中にフラッシュ・セルフ・プログラミング・ライブラリは使用しないでください。
3. 本関数を実行後にフラッシュ・セルフ・プログラミング・ライブラリを使用した場合には、使用する RAM を再初期化する必要がありますので、EEPROM エミュレーション・ライブラリ再開時には必ず本関数を実行してください。
4. 本関数を再度実行する場合には､必ず EEPROM エミュレーション･ライブラリを終了させてください。
5. 本関数で使用するテーブルは修正できません。必ず定義済みのテーブルをご使用ください。

【呼び出し後のレジスタ状態】

戻り値　：C
破壊レジスタ：AX（引数），BC（引数）

【引　数】

fal_descriptor_t[注1]の内容（ 定義済み固定値：ユーザ変更不可 ）

引数	型	説　明
fal_pool_first_addr_u32	fal_u32	データ・フラッシュ・メモリの開始アドレス
eel_pool_first_addr_u32	fal_u32	EEL[注2]で使用するデータ・フラッシュ・メモリの開始アドレス
user_pool_first_addr_u32	fal_u32	未使用
fal_pool_last_addr_u32	fal_u32	データ・フラッシュ・メモリの終了アドレス
eel_pool_last_addr_u32	fal_u32	EEL[注2]で使用するデータ・フラッシュ・メモリの終了アドレス
user_pool_last_addr_u32	fal_u32	未使用
fal_pool_first_block_u16	fal_u16	データ・フラッシュ・メモリの先頭ブロック番号
eel_pool_first_block_u16	fal_u16	EEL[注2]で使用するデータ・フラッシュ・メモリの先頭ブロック番号
user_pool_first_block_u16	fal_u16	未使用
fal_pool_last_block_u16	fal_u16	データ・フラッシュ・メモリの終了ブロック番号
eel_pool_last_block_u16	fal_u16	EEL[注2]で使用するデータ・フラッシュ・メモリの終了ブロック番号
user_pool_last_block_u16	fal_u16	未使用
fal_first_widx_u16	fal_u16	データ・フラッシュ・メモリのアクセス開始番号
eel_first_widx_u16	fal_u16	EEL[注2]で使用するデータ・フラッシュ・メモリのアクセス開始番号
user_first_widx_u16	fal_u16	未使用
fal_last_widx_u16	fal_u16	データ・フラッシュ・メモリのアクセス終了番号
eel_last_widx_u16	fal_u16	EEL[注2]で使用するデータ・フラッシュ・メモリのアクセス終了番号
user_last_widx_u16	fal_u16	未使用
fal_pool_wsize_u16	fal_u16	全ブロック分の領域サイズ(ワード単位：4バイト単位)
eel_pool_wsize_u16	fal_u16	EEL[注2]で使用する全ブロック分の領域サイズ(ワード単位：4バイト単位)
user_pool_wsize_u16	fal_u16	未使用
block_size_u16	fal_u16	1ブロック分の領域サイズ(バイト単位)
block_wsize_u16	fal_u16	1ブロックの領域サイズ(ワード単位：4バイト単位)
fal_pool_size_u08	fal_u08	データ・フラッシュ・メモリのブロック・サイズ
eel_pool_size_u08	fal_u08	EEL[注2]で使用するデータ・フラッシュ・メモリのブロック・サイズ
user_pool_size_u08	fal_u08	未使用
fx_MHz_u08	fal_u08	RL78マイクロコントローラの動作周波数の設定
wide_voltage_mode_u08	fal_u08	電圧モードの設定

注 1.　本定義済みテーブルはユーザで修正しないでください。

2.　EEL: EEPROMエミュレーション・ライブラリの略称です。

開発ツール	引数型／レジスタ	
	C言語	アセンブリ言語
RENESAS製Small and medium model	const __far fal_descriptor_t* descriptor_pstr	AX(0-15), BC(16-23)
RENESAS製Large model	const __far fal_descriptor_t* descriptor_pstr	AX(0-15), BC(16-23)

【戻り値】

型	シンボル定義	説　明
fal_status_t	FAL_OK	正常終了
	FAL_ERR_CONFIGURATION	初期化エラー。設定値に誤りがあるか、高速オンチップ・オシレータが起動していません。定義済みデータが変更されていないか、高速オンチップ・オシレータが起動しているかを確認してください。

備考　アセンブリ言語の戻り値は、Cレジスタに格納されます。

3. 2　EEPROMエミュレーション・ライブラリ関数

EEPROMエミュレーション・**ライブラリ**は、次に示すライブラリ関数で構成されています。

表3−2　EEPROMエミュレーション・ライブラリ関数一覧

関数名	概　要
EEL_Init	EEPROMエミュレーションで使用するRAMの初期化処理
EEL_Open	EEPROMエミュレーション準備処理
EEL_Close	EEPROMエミュレーション終了処理
EEL_Execute	EEPROMエミュレーション・コマンド実行処理
EEL_Handler	EEPROMエミュレーション・コマンド継続実行処理、またはブロック整理処理 ※Enforced モード以外でコマンド実行時
EEL_TimeOut_CountDown	EEPROMエミュレーション・コマンド実行タイム・カウント処理 ※Timeout モードのみ使用
EEL_GetDriverStatus	EEPROMエミュレーション・ライブラリ状態を取得
EEL_GetSpace	EEPROMエミュレーション・ブロックの空き状態数を取得
EEL_GetVersionString	EEPROMエミュレーション・ライブラリ（EEL）のバージョン情報を取得

EEL_Init

【概　要】
EEPROM エミュレーションで使用する RAM の初期化処理

【書　式】
＜C 言語＞

```
eel_status_t　__far EEL_Init (void)
```

＜アセンブラ＞

```
CALL !_EEL_Init または CALL !!_EEL_Init
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】

1. フラッシュ・セルフ・プログラミング・ライブラリ、およびEEPROMエミュレーション・ライブラリが未実行、あるいは終了していること。
2. FAL_Init関数を正常終了させていること。

【機　能】
EEPROM エミュレーションを実行する為のパラメータを初期化します。

注意 1. EEPROM エミュレーションを開始するときには必ず本関数を実行し、使用する RAM を初期化させてください。
2. フラッシュ・セルフ・プログラミング・ライブラリ(FSL)とは排他利用となります。本関数を実行する前にフラッシュ・セルフ・プログラミング・ライブラリは必ず終了させてください。また、 EEPROM エミュレーション・ライブラリ実行中にフラッシュ・セルフ・プログラミング・ライブラリは使用しないでください。
3. 本関数を実行後にフラッシュ・セルフ・プログラミング・ライブラリを使用した場合には、使用する RAM を再初期化する必要がありますので、EEPROM エミュレーション・ライブラリ再開時には必ず本関数を実行してください。
4. 本関数を再度実行する場合には、必ず EEPROM エミュレーション･ライブラリを終了させてください。

【呼び出し後のレジスタ状態】

戻り値　：C

【引　数】
なし

【戻り値】

型	シンボル定義	説　明
eel_status_t	EEL_OK	正常終了
	EEL_ERR_CONFIGURATION	初期化エラー。FAL_Init関数が実行されていないか、FAL_Init関数、およびEEL_Init関数で設定された値ではEELを実行できません。

備考　アセンブリ言語の戻り値は、Cレジスタに格納されます。

EEL_Open

【概　要】
EEPROM エミュレーション準備処理

【書　式】
＜C 言語＞

```
void　__far EEL_Open(void)
```

＜アセンブラ＞

```
CALL !_EEL_Open または CALL !!_EEL_Open
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】

1. FAL_Init、およびEEL_Init関数を正常終了させていること。
2. EEPROMエミュレーションを実行していた場合はEEL_Closeまでを実行して処理を終了(Closed状態)させていること。

【機　能】
データ・フラッシュ・メモリを操作可能状態に変更し、EEPROM エミュレーションを実行できる状態にします。

注　EEL_Open関数を実行し、EEPROMエミュレーションを開始状態(opened)に遷移させた後はフラッシュ・セルフ・プログラミング・ライブラリは実行できません。またSTOPモード、およびHALTモードも実行する事が出来なくなります。フラッシュ・セルフ・プログラミング・ライブラリや、STOPモード、HALTモードを実行する必要がある場合は、EEL_Close関数を実行し、EEPROMエミュレーションを終了状態(closed)に遷移させてください。

【呼び出し後のレジスタ状態】
レジスタは破壊されません。

【引　数】
なし

【戻り値】
なし

EEL_Close

【概　要】
EEPROM エミュレーション終了処理

【書　式】
＜C 言語＞

```
void   __far EEL_Close(void)
```

＜アセンブラ＞

```
CALL !_EEL_Close または CALL !!_EEL_Close
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】
EEPROMエミュレーションを実行していた場合は、EEL_CMD_SHUTDOWNコマンドでEEPROMエミュレーションを停止状態(Opened状態)にさせていること。

【機　能】
データ・フラッシュ・メモリを操作終了状態に変更し、EEPROM エミュレーションを実行できない状態にします。

【呼び出し後のレジスタ状態】
レジスタは破壊されません。

【引　数】
なし

【戻り値】
なし

EEL_Execute

【概　要】
EEPROM エミュレーション実行関数

【書　式】
＜C 言語＞

```
void   __far EEL_Execute( eel_request_t* request )
```

＜アセンブラ＞

```
CALL !_EEL_Execute または CALL !!_EEL_Execute
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】

FAL_Init、EEL_Init、およびEEL_Open関数を正常終了させていること。

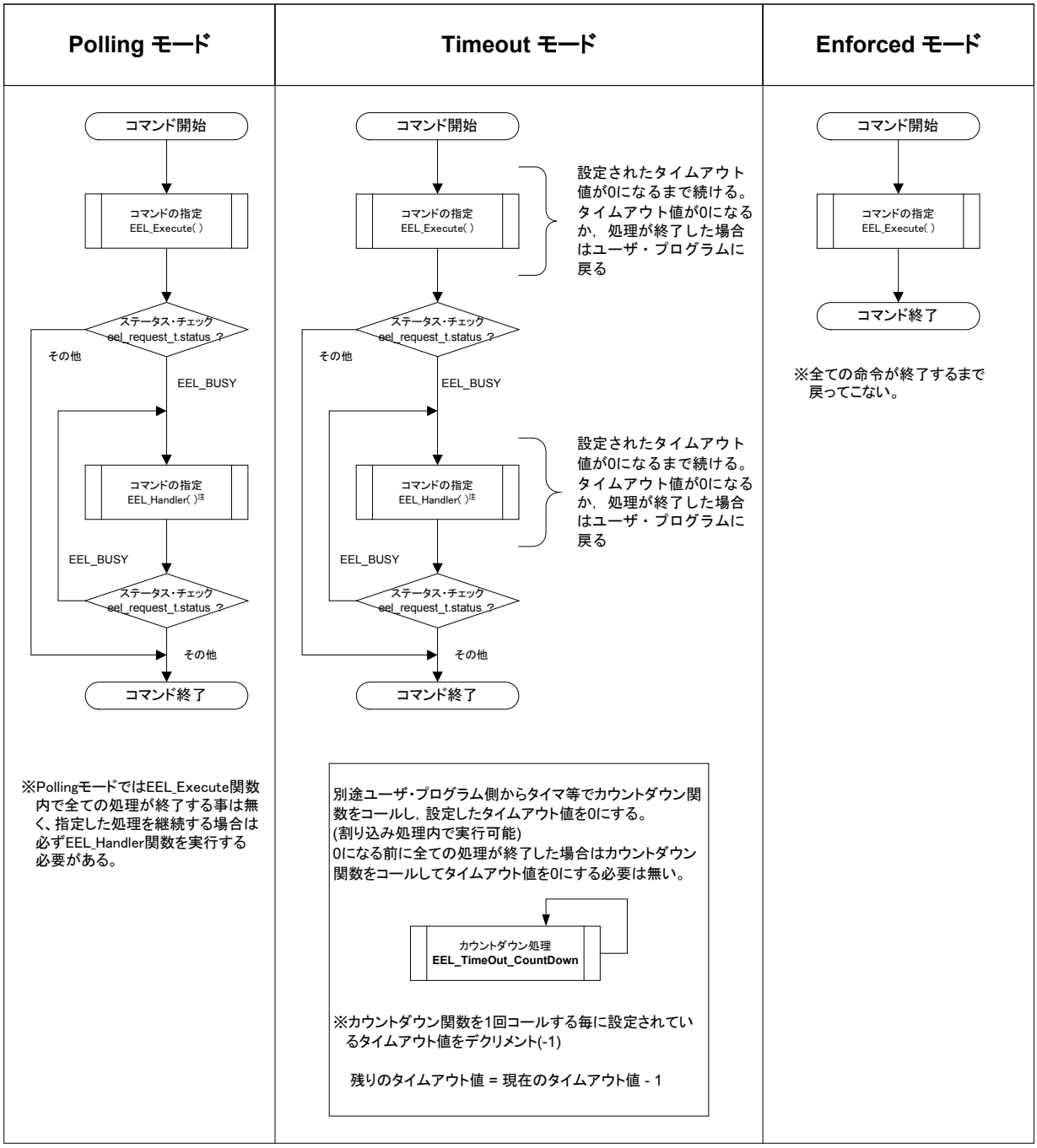
【機　能】
EEPROM エミュレーションを操作する為の各処理をコマンド形式で本関数の引数に設定させ、処理を実行させます。EEL_Execute 関数は実行する際に実行モードを設定し、EEPROM エミュレーションの実行方法を選択する事ができます。各モードの状態については表 3−3、図 3−1 に記載します。

表 3−3　各モード時のコマンド実行方法

実行モード	説　明
Enforced モード	指定されたコマンドをEEL_Execute関数のみですべての処理を実行します。コマンドの処理が終了するまで関数から戻りません。
Timeout モード注	指定されたコマンドをEEL_Execute関数で設定するタイムアウト値がEEL_TimeOut_CountDown関数によって0に設定されるまで処理を継続して実行します。タイムアウト時に残処理があった場合はEEL_Handler関数で処理を継続します。 タイムアウト値が0になる前にすべての処理が終了した場合は関数を終了し、ユーザ・プログラムに戻りますが、このモードを使用する場合はEEL_TimeOut_CountDown関数を使用してタイムアウト値を必ず 0 に出来るように使用してください。
Polling モード注	指定されたコマンドを一定の処理単位ごとに実行します。EEL_Execute関数実行後は、EEL_Handler関数で処理を継続します。

注　実行モードはEEL_Handler実行時にも再指定可能です。

図 3−1　各モード時のコマンド実行方法



注　　実行モードはEEL_Handler実行時にも再指定可能です。

【呼び出し後のレジスタ状態】
破壊レジスタ：AX（引数）

【引　数】

eel_request_tの内容

引数	型	説　明
eel_request_t.address_pu08	eel_u08 *	書き込み、読み込みデータ設定用データ・バッファ注
eel_request_t.identifier_u08	eel_u08	実行コマンドの設定パラメータ
eel_request_t.timeout_u08	eel_u08	タイムアウト値(コマンド実行モードの設定)
eel_request_t.command_enu	eel_command_t	実行させるコマンド
eel_request_t.status_enu	eel_status_t	コマンド実行状態

注　対象パラメータが必要なコマンドの場合のみ設定します。また、データ・バッファのサイズについては、書き込み、読み込みデータのバイト・サイズ分をご用意ください。

タイムアウト値(timeout_u08)の設定内容一覧

タイムアウト値	説　明
0xFF	Enforcedモードで処理を実行
0x01～0xFE	Timeoutモードで処理を実行
0x00	Pollingモードで処理を実行

実行コマンド(eel_command_t)一覧

コマンド	説　明
EEL_CMD_STARTUP	ブロックの状態を確認し、EEPROMエミュレーションを開始(started)状態にします。すべてのブロックが有効ブロックであった場合、強制的に一番古い有効ブロックを消去し、準備ブロックを作成します EEPROMエミュレーション・ブロックの初期化(消去)注2を行う場合に使用する EEL_CMD_FORMATコマンド以外については、必ず本コマンドを事前に実行し、正常終了させてください
EEL_CMD_WRITE注1	EEPROMエミュレーション・ブロックへ指定データの書き込みを行います ※実行には以下の引数の設定が必要です ・eel_request_t.address：書き込みデータが保存されているRAM領域の先頭アドレスを指定 ・eel_request_t.identifier：書き込みデータのデータID指定
★ EEL_CMD_READ注1	EEPROMエミュレーション・ブロックから指定したデータIDの最新データを読み込みます ※実行には以下の引数の設定が必要です ・eel_request_t.address：読み込みデータを保存するRAM領域の先頭アドレスを指定 ・eel_request_t.identifier：読み込みデータのデータID指定
EEL_CMD_CLEANUP注1,注3	EEPROMエミュレーション・ブロックの最新データのみを新規に作成した有効ブロックに移動させ、その他不要なブロックはすべて初期化(消去)注2します すべてのブロックが有効ブロックになっている場合は、強制的に一番古い有効ブロックを消去して準備ブロックを作成した後、処理を実行します

コマンド	説　明
EEL_CMD_FORMAT[注4]	EEPROMエミュレーション・ブロックを記録されていたデータも含め、すべて初期化(消去)[注2]します。EEPROMエミュレーションを最初に使用する場合に必ず使用します。また、EEPROMエミュレーション･ブロックに異常が発生(有効ブロックがなくなる等)した場合や、初期値(変更できない固定値)を修正する場合にも本コマンドを使用し、ブロック全体を初期化する必要があります 処理終了後は結果に関わらず必ず停止状態(Opened)に遷移しますので、EEPROMエミュレーションを継続して使用する場合は、EEL_CMD_STARTUPコマンドを実行してください
EEL_CMD_SHUTDOWN[注1]	EEPROMエミュレーションを停止状態(Opened)にします

注 1.　EEL_CMD_STARTUPコマンドを正常に終了させてからコマンドを実行してください。
2.　使用禁止に設定されているブロックに対しては初期化(消去)は行いません。
3.　詳細動作については**1.4.1　EEL_CMD_CLEANUPコマンドによるブロックの整理**を参照してください。
4.　詳細動作については**1.3　EEPROMエミュレーション・ブロックの初期化**を参照してください。

開発ツール	引数型／レジスタ	
	C言語	アセンブリ言語
RENESAS製Small and medium model	eel_request_t* request	AX(0-15)
RENESAS製Large model	eel_request_t* request	AX(0-15)

コマンド実行状態(eel_status_t)一覧

コマンド実行状態	説　明	対応コマンド
EEL_OK	正常終了	すべてのコマンド
EEL_BUSY	コマンド実行中	すべてのコマンド
EEL_ERR_INITIALIZATION	初期化エラー： FAL_Init()、EEL_Init()、およびEEL_Open関数が正常に完了していません	すべてのコマンド
EEL_ERR_ACCESS_LOCKED	EEPROMエミュレーション・ロック・エラー： EEPROMエミュレーションが実行できない状態です。EEL_CMD_STARTUPコマンドを正常に終了させてください	EEL_CMD_STARTUP以外のコマンド
EEL_ERR_COMMAND	コマンド・エラー： 存在しないコマンドが指定されています	－
EEL_ERR_PARAMETER	パラメータ・エラー： コマンドの設定パラメータに誤りがあります。設定したパラメータを見直してください	すべてのコマンド
EEL_ERR_REJECTED	リジェクト・エラー： 別コマンドが実行中です	すべてのコマンド
EEL_ERR_NO_INSTANCE	識別子エラー： 指定されたデータはディスクリプタ・テーブルに存在しないか、対象データが記録されていません。	EEL_CMD_READ

コマンド実行状態	説　明	対応コマンド
EEL_ERR_POOL_FULL	プール・フル・エラー： データを書き込める領域が存在しない。EEL_CMD_STARTUPかEEL_CMD_CLEANUPコマンドで強制的に一番古いブロックを消去する事によって状態を回復させることができる可能性がありますが、状態が回復しても一部のデータが消えてしまう可能性があります	EEL_CMD_WRITE
EEL_ERR_POOL_INCONSISTENT	EEPROMエミュレーション・ブロック不整合エラー： EEPROMエミュレーション・ブロックが不定状態(有効ブロックがない等)です。EEL_CMD_FORMATコマンドを実行し、EEPROMエミュレーション・ブロックを初期化してください	EEL_CMD_STARTUP
EEL_ERR_POOL_EXHAUSTED	EEPROMエミュレーション・ブロック消耗エラー： 継続して使用できるEEPROMエミュレーション・ブロックがなくなりました。EEPROMエミュレーションを終了してください	EEL_CMD_READ EEL_CMD_SHUTDOWN 以外のコマンド
EEL_ERR_INTERNAL	内部エラー： 予期しないエラーが発生しました。デバイス状態を確認してください。また、EEL_CMD_SHUTDOWN コマンド実行時に発生した場合は、EEL_Close関数を実行し、EEPROMエミュレーションを終了させてください	EEL_CMD_READ EEL_CMD_WRITE 以外のコマンド

【戻り値】
　なし

EEL_Handler

【概　要】
EEL_Execute を Enforced モード以外で実行した場合の EEPROM エミュレーション継続実行処理、またはメンテナンス･モード実行処理

【書　式】
＜C 言語＞

```
void　__far EEL_Handler(eel_u08 timeout_u08);
```

＜アセンブラ＞

```
CALL !_EEL_Handler または CALL !!_EEL_Handler
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】
FAL_Init、EEL_Init、およびEEL_Open関数を正常終了させていること。

【機　能】
EEL_Execute 関数で指定された EEPROM エミュレーションの処理を、本関数で継続実行します。[注1] また、コマンドが実行されていない状態で本関数を実行することにより、メンテナンス・モードによる EEPROM エミュレーション・ブロックの整理を実行することが可能です。[注2]

注 1. EEL_Handler関数のコマンド実行状態はEEL_Execute関数の引数で使用された「eel_request_t* request」に設定されます。その為、EEL_Handlerを使用する場合、「eel_request_t* request」変数は開放しないでください。
2. メンテナンス・モードについては**1.4.2 EEL_Handler関数によるブロック整理**の項を参照してください。

【呼び出し後のレジスタ状態】
破壊レジスタ：AX（引数）

【引　数】

タイムアウト値(timeout_u08)の設定内容一覧[注]

タイムアウト値	説　明
0x01～0xFF	Timeoutモードで処理を継続実行 (EEL_TimeOut_CountDown関数を使用してタイムアウト値を 0 にする必要があります)
0x00	Pollingモードで処理を継続実行

注　各モードの詳細についてはEEL_Execute関数の説明を参照してください。

開発ツール	引数型／レジスタ	
	C言語	アセンブリ言語
RENESAS製Small and medium model	eel_u08 timeout_u08	X
RENESAS製Large model	eel_u08 timeout_u08	X

【戻り値】
EEL_Execute 関数で引数として用意された「eel_request_t* request」の領域に実行後のステータス情報が格納されます。

eel_request_tに入力されるデータ

引数	型	説　明
eel_request_t.status	eel_status_t	コマンド実行状態

EEL_TimeOut_CountDown

【概　要】
EEPROM エミュレーション・コマンド実行タイム・カウント処理

【書　式】
＜C 言語＞

```
void   __far EEL_TimeOut_CountDown(void)
```

＜アセンブラ＞

```
CALL !_EEL_TimeOut_CountDown  または  CALL !!_EEL_TimeOut_CountDown
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】
FAL_Init、EEL_Init、およびEEL_Open関数を正常終了させていること。

【機　能】
EEPROM エミュレーションをタイム・アウト・モードでコマンドを実行する場合に使用します。
本関数を実行すると、EEPROM エミュレーションのコマンド実行時に設定したタイムアウト値をデクリメント(-1)し、タイムアウト値が 0 になると、EEL_Execute、および EEL_Handler 関数のループ処理が終了します。
タイムアウト値の扱いについては EEL_Execute、および EEL_Handler 関数の項を参照ください。

【呼び出し後のレジスタ状態】
レジスタは破壊されません。

【引　数】
なし

【戻り値】
なし

EEL_GetDriverStatus

【概　要】
EEPROM エミュレーション・ライブラリ状態を取得

【書　式】
＜C 言語＞

```
void   __far EEL_GetDriverStatus(__near eel_driver_status_t *driverStatus_pstr)
```

＜アセンブラ＞

```
CALL !_EEL_GetDriverStatus  または  CALL !!_EEL_GetDriverStatus
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】
FAL_Init、EEL_Init関数を正常終了させていること。

【機　能】
EEPROM エミュレーション・ライブラリの状態を取得します。EEL_Handler 関数の状態と EEPROM エミュレーションの状態を確認する事が可能です。

【呼び出し後のレジスタ状態】
レジスタは破壊されません。

【引　数】

eel_driver_status_tの内容

引数	型	説　明
eel_driver_status_t.operationStatus_enu	eel_operation_status_t	EEL_Handler実行状態
eel_driver_status_t.accessStatus_enu	eel_access_status_t	EEPROMエミュレーションの状態
eel_driver_status_t.backgroundStatus_enu	eel_status_t	ライブラリ状態（ユーザ使用不可）

eel_operation_status_tの内容

型	説　明
EEL_OPERATION_PASSIVE	コマンドを実行中以外でEEL_CMD_STARTUPコマンドが正常に終了していない
EEL_OPERATION_IDLE	コマンド、およびメンテナンス・モード[注]実行中以外
EEL_OPERATION_BUSY	コマンド、あるいはメンテナンス・モード[注]実行状態

注　メンテナンス・モードについては**1.4.2 EEL_Handler関数によるブロック整理**の項を参照してください。

eel_access_status_tの内容

型	説　明
EEL_ACCESS_LOCKED	データの書き込み、読み込みが実行できない状態(Opened，Closed)
EEL_ACCESS_UNLOCKED	データの書き込み、読み込みが実行できる状態(started)

開発ツール	引数型／レジスタ	
	C言語	アセンブリ言語
RENESAS製Small and medium model	__near eel_driver_status_t *driverStatus_pstr	AX(0-15)
RENESAS製Large model	__near eel_driver_status_t *driverStatus_pstr	AX(0-15)

【戻り値】
なし

EEL_GetSpace

【概　要】
EEPROM エミュレーション・ブロックの空き容量（ワード単位）を取得

【書　式】
＜C 言語＞

```
eel_status_t　__far EEL_GetSpace(__near eel_u16* space_pu16)
```

＜アセンブラ＞

```
CALL !_EEL_GetSpace または CALL !!_EEL_GetSpace
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

【事前設定】
FAL_Init、EEL_Init、EEL_Open関数、およびEEL_CMD_STARTUPコマンドを正常終了させていること。

【機　能】
EEPROM エミュレーション・ブロックの空き容量（ワード単位）を取得。

【呼び出し後のレジスタ状態】
戻り値　：C
破壊レジスタ　：AX（引数）

【引　数】

型	説　明
__near eel_u16*	現在の有効ブロック、および準備ブロックの合計空き容量の情報が入力されているアドレス (2バイト領域) ： データはワード単位(1ワード = 4バイト)

開発ツール	引数型／レジスタ	
	C言語	アセンブリ言語
RENESAS製Small and medium model	__near eel_u16* space_pu16	AX(0-15)
RENESAS製Large model	__near eel_u16* space_pu16	AX(0-15)

【戻り値】

型	シンボル定義	説　明
eel_status_t	EEL_OK	取得成功
	EEL_ERR_INITIALIZATION	EEL_Initが実行されていません
	EEL_ERR_ACCESS_LOCKED	EEL_CMD_STARTUPコマンドが正常に終了していません
	EEL_ERR_REJECTED	コマンド実行中

備考　　アセンブリ言語の戻り値は、Cレジスタに格納されます。

EEL_GetVersionString

【概　要】
EEPROM エミュレーション・ライブラリ（EEL）のバージョン情報を取得

【書　式】
＜C 言語＞

```
__far eel_u08*　__far EEL_GetVersionString(void)
```

＜アセンブラ＞

```
CALL !_EEL_GetVersionString または CALL !!_EEL_GetVersionString
```

備考　EEPROMエミュレーション・ライブラリを00000H-0FFFFH に配置する場合は“！”、それ以外の場合は“！！”で呼び出してください。

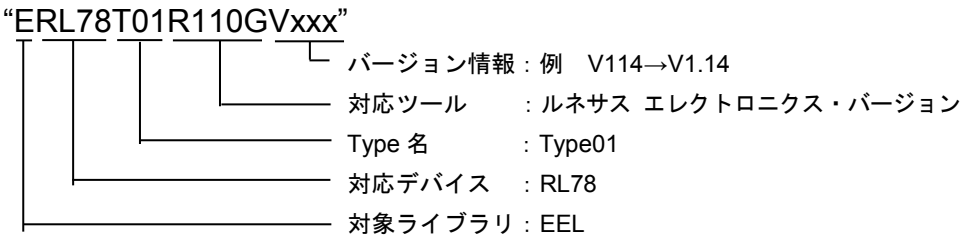
【事前設定】
FAL_Init、EEL_Init関数を正常終了させていること。

【機　能】
なし

【呼び出し後のレジスタ状態】
戻り値　：BC，DE
破壊レジスタ　：BC，DE

【引　数】
なし

【戻り値】

型	説　明
eel_u08*	EEPROMエミュレーション・ライブラリ（EEL）のバージョン情報が入力されているアドレス (24bitアドレス領域) 例：EEPROMエミュレーション・ライブラリのPack01 V1.14の場合（ASCIIコード） 

付録A　改版履歴

A. 1　本版で改訂された主な箇所

（1/1）

箇　所	内　容	分類
全般		
－	対応デバイスを削除	(c)
－	対象MCUについてのリスト参照説明を追加	(e)
第1章 EEPROMエミュレーション概要		
p.2	図1-1、図1-2 フラッシュ・データ・ライブラリをデータ・フラッシュ・ライブラリに訂正	(a)
p.15	説明に関する記述を訂正	(a)
第2章 EEPROMエミュレーションの使用方法		
p.36	表2-2 表題にバージョンを追加	(c)
p.36	表2-2 セルフRAMの使用領域に関する記述を変更	(c)
p.36	表2-2 注1で問合せに関する注意事項を変更	(c)
p.36	表2-2 注2の記載を削除	(c)
p.36	表2-2 注4の記載を削除	(c)
第3章 EEPROMエミュレーション機能		
p.52	説明に関する記述を変更	(c)

備考　表中の「分類」により、改訂内容を次のように区分しています。

（a）：誤記訂正、（b）：仕様（スペック含む）の追加／変更、（c）：説明、注意事項の追加／変更、

（d）：パッケージ、オーダ名称、管理区分の追加／変更、（e）：関連資料の追加／変更

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。なお、適用箇所は各版での章を示します。

（1/1）

版　数	内　容	適用箇所
Rev.1.02	フラッシュ・ライブラリのドキュメントの扱いをアプリケーション・ノート（旧版R01AN0351）からユーザーズ・マニュアルへ変更	全般
	表紙に対応ZIPファイル名、リリース版を明記	
	注意事項、処理時間、並びにソフトウェアリソースの内容を使用上の留意点から本書に移動。また、それに伴って前述の内容に対する本書の参照先を変更	
	対応デバイスを追加	
	高速OCOの表記を削除。高速オンチップ・オシレータに表記統一	
	動作周波数の記載について、説明毎に表記方法が異なっていた部分をCPUの動作周波数に表記統一	
	セパレータ容量の誤りを修正、及び最大サイズの計算方法の修正	第1章 EEPROMエミュレーション概要
	高速オンチップ・オシレータの周波数に関する注意事項を追加	第2章 EEPROMエミュレーションの使用方法
	RAMパリティ・エラーに関する注意事項を追加	
	データ・フラッシュ・コントロール・レジスタ（DFLCTL）の注意事項を追加	
	データ・フラッシュ・メモリのブロック数に関する注意事項を追加	
	処理時間に関する項目を追加（使用上の留意点から処理時間に関する記述を本書に移動）	
	リソースに関する項目を追加（使用上の留意点からリソースに関する記述を本書に移動と説明を変更	
	高速オンチップ・オシレータの注意事項を追加	
	Fal_Init関数の戻り値の説明を追記	第3章 EEPROMエミュレーション機能
	EEL_Execute関数の引数の説明を追記	
	EEL_GetVersionString関数の戻り値の説明を変更	

RL78 ファミリ　ユーザーズマニュアル
EEPROM エミュレーション・ライブラリ Pack01

発行年月日　　2016 年 1 月 29 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）

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

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）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

RL78 ファミリ

ルネサス エレクトロニクス株式会社

R01US0054JJ0103