User's Manual

16

RL78ファミリ

EEPROMエミュレーション・ライブラリ Pack02

日本リリース版

ZIPファイル名：JP_R_EEL_RL78_P02_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.1.01　2016.02

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　マイクロコントローラのEEPROMエミュレーション・ライブラリ Pack02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　マイクロコントローラのデータ・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り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。

口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。

口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5章　ユーザ・インタフェースを参照してください。

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０ｘＸＸＸＸ

目次

第1章　概説

1.1　概　要

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。

EEPROMエミュレーション・ライブラリは、ユーザ・プログラムによってデータ・フラッシュを書き換えるための無償ソフトウエアです。

本ユーザーズ・マニュアル内では、FDLの処理を含めEELの処理として記述します。

なお、本ユーザーズ・マニュアルは、必ず、本EEPROMエミュレーション・ライブラリのパッケージに添付されているリリースノート、及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。

1.2　対象デバイス

本EEPROMエミュレーション・ライブラリの対象デバイス最新情報は、販売会社、特約店へお問い合わせください。

1.3　用　語

このユーザーズ·マニュアルで使用する用語について、その意味を次に示します。

・Pack
EEPROMエミュレーション・ライブラリの種類を表す識別名です。必ずご使用のデバイスに対応したPackをご使用ください。

・EEL
EEPROMエミュレーション・ライブラリの略称です。
なお、以降、本ユーザーズ・マニュアル内ではRL78 EEPROMエミュレーション・ライブラリ Pack02をEELと呼称します。

・FDL
データ・フラッシュ・ライブラリの略称です。

・FSL

フラッシュ・セルフ・プログラミング・ライブラリの略称です。

・EEL関数

EELが提供する関数の総称です。

・FDL関数

FDLが提供する関数の総称です。

・FSL関数

FSLが提供する関数の総称です。

・ブロック番号

フラッシュ・メモリのブロックを示す番号です。

・EELブロック

EEPROMエミュレーション・ライブラリがアクセスするブロックの略称です。なお、以降、本ユーザーズ・マニュアル内ではEEPROMエミュレーション・ブロックをEELブロックと呼称します。

・CF

コード・フラッシュ

・DF

データ・フラッシュ

第2章　EEPROMエミュレーション

2. 1　EEPROMエミュレーションの仕様

ユーザ・プログラムから、EELが提供するEEL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。

EELでは、1バイトの識別子（データID：1～64）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（識別子は最大64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3ブロック以上（推奨）[注]の領域を使用します。このブロックをEEL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小さいアドレスから、ユーザ・データはブロックの大きいアドレスから対象ブロックに書き込みが行われます。

注　EEPROMエミュレーションを行う場合、最低限2ブロック以上のブロックが必要です。2ブロックと指定した場合、１度でも書き込みエラーが発生した時に、以降はそれまで正常に書き込めたデータの読み出しのみ可能で、書き込みを継続して行うことができません。次にEELでデータの書き込みを行う場合は対象2ブロックをフォーマットする必要があり、それまで書き込んでいたデータは全て消去されます。また、システムにより電圧低下など、不慮の状態が発生する可能性もありますので、対象ブロックに3ブロック以上をご指定いただくことを推奨いたします。

2. 2　機能概要

EEL では以下のような特徴を含む基本的な書き込み/読み出し機能等が提供されています。

- データは 64 個設定可能
- データのサイズは 1～255 バイトに設定可能
- BGO(Back Ground Operation)に対応
- 管理データ用メモリ消費量
  （EEL ブロックあたり 10 バイト、EEL ブロックに書き込まれるデータあたり 2 バイト）
- ★ EEL_CMD_WRITE または、EEL_CMD_REFRESH 実行中の CPU リセットによる中断に対して、EEL_CMD_REFRESH で復旧
- ブロック・ローテーション（データ・フラッシュ使用頻度の均衡化）

弊社では、本資料の対象である本EELの他のEELとして、RL78 EEPROMエミュレーション・ライブラリ Pack 01(EEL Pack01)も提供しています。本EELは、EEL Pack01からリソース消費量を縮小して、以下の機能差分があります。 (機能の詳細については、RL78 マイクロコントローラEEPROMエミュレーション・ライブラリ Pack 01 ★ ユーザーズ・マニュアル R01US0054JJを参照）。

項目	EEL　Pack01	EEL
ユーザ・データ長	1 ～ 255	1～255
格納ユーザ・データ数 注1	1 ～ 255	1～64
データIDの範囲	1 ～ 255	1～64
EEPROMエミュレーション・ブロック数注2	4 ～ 255	3～255
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	1014/2バイト
EnforceモードおよびTimeOutモード	対応	非対応
バックグラウンド・メンテナンス処理	対応	非対応
データID番号	任意設定可能	任意設定不可
データ内容に対する自動チェックサム	対応	非対応

注1　ユーザ・データは、すべてのユーザ・データがEELブロックへ書き込まれるときに必要な合計サイズを1ブロックの1/2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4.2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。

注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。

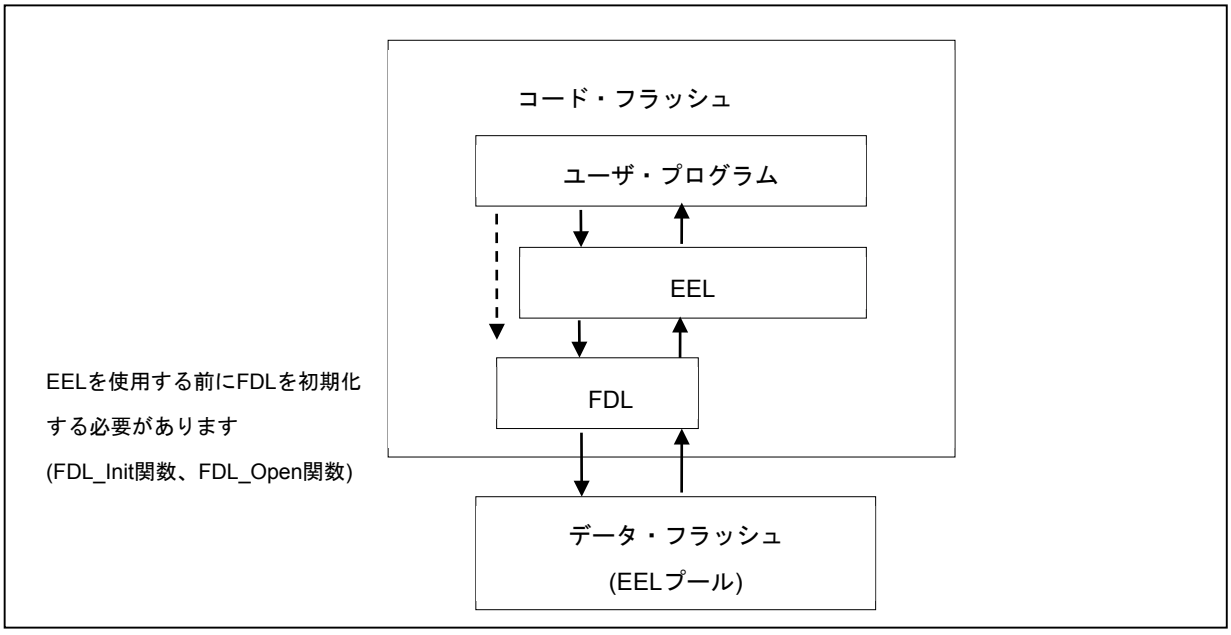
2. 3　EELアーキテクチャ

この章では、ユーザがEELを使用してデータ・フラッシュ(EELプール)の書換えを行う上で必要なEELのアーキテクチャについて説明します。

2. 3. 1　システム構成

EELは、ユーザが定義するデータ・フラッシュ領域にアクセスする為の、インタフェースです。
「図 2−1　システム構成」の矢印が操作の流れです。

図 2−1　システム構成



2. 3. 2　EELプール

EELプールはユーザによって定義されるEELがアクセス可能なデータ・フラッシュ領域です。ユーザ・プログラムからのデータ・フラッシュへのアクセスは、EEL経由でのEELプールへのアクセスのみ許可されます。

必ず対象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数をEELプールのブロック数に設定してください。なお、設定方法につきましては、「4.3 ユーザ設定初期値」をご参照ください。

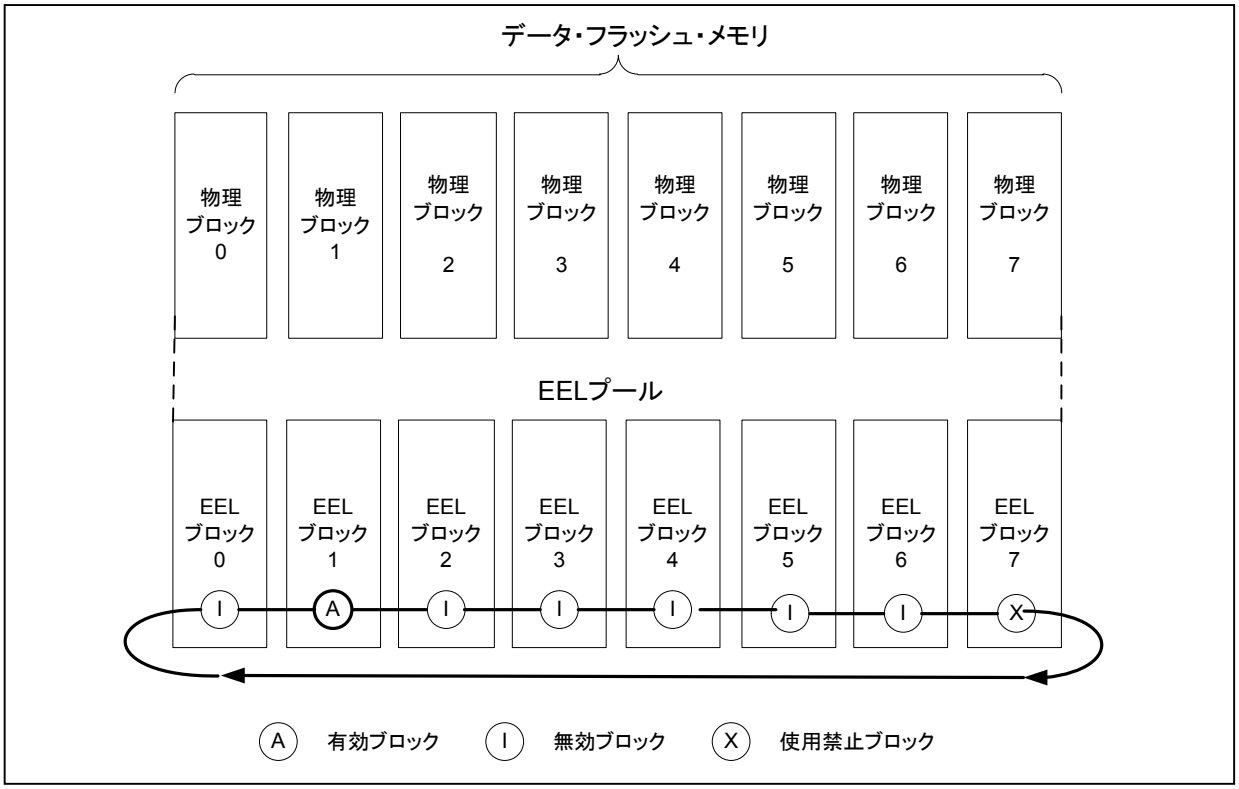
EELではEELプールを1024バイトのブロックに分割します。各ブロックには状態があり、これがブロックの現在の使用状態を示しています。

状態	内容
有効	1つのEELブロックが有効となり、定義済みのデータを格納します。有効ブロックはEELプールに割り当てられたデータ・フラッシュ・ブロック群を循環します。
無効	無効ブロックにはデータは格納されません。EELブロックはEELによって無効とされるか、消去ブロックの場合は無効となります。
使用禁止	機能動作が失敗してデータ・フラッシュの故障の可能性が判明した場合は、EELは該当ブロックを使用禁止とし、その後はそのブロックをEEPROMエミュレーションには使用しません。

8KBのデータ・フラッシュを有するデバイスのEELプール構成例を「図2−2　EELプール構成」に示します。

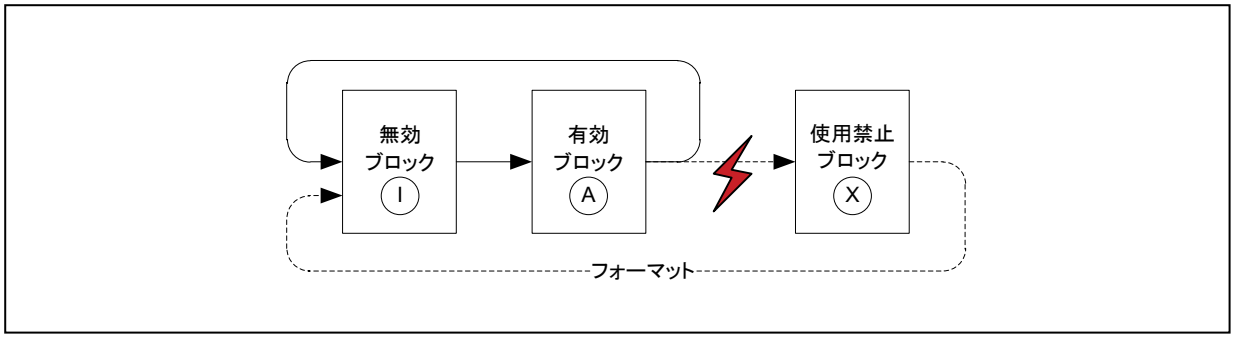
有効ブロック（例ではブロック1）に書き込み可能領域がなくなり追加データの格納ができなくなった時（writeコマンドの失敗）には、新規の有効ブロックが循環的に選定され、その時点で有効なデータ群が新規の有効ブロックにコピーされます。このプロセスは「リフレッシュ」と呼ばれます。EEL_CMD_REFRESHコマンド実行後に元の有効ブロックは無効となり、1つの有効ブロックのみが存在します。このプロセスにおいて使用禁止ブロック（例ではブロック7）は無視され、次の有効ブロック選定の候補にはなりません。

図 2−2　EELプール構成



EELプール内のEELブロックのライフサイクルのイメージを「図2−3　EELブロックのライフサイクル」に示します。通常動作時では、EELブロックは有効状態と無効状態の間を行き来します。EELブロックへのアクセス中にエラーが発生した場合には、障害が起きたEELブロックは使用禁止ブロックとなります。このブロックはライフサイクルに再投入されることはありません。ただし、ユーザがEELプール全体をフォーマットすることによりそのブロックの修復を試みることは可能ですが、この際に既存データの内容はすべて消去されます。

図 2−3　EELブロックのライフサイクル



EELプールには、以下のような4種類の状態があります。

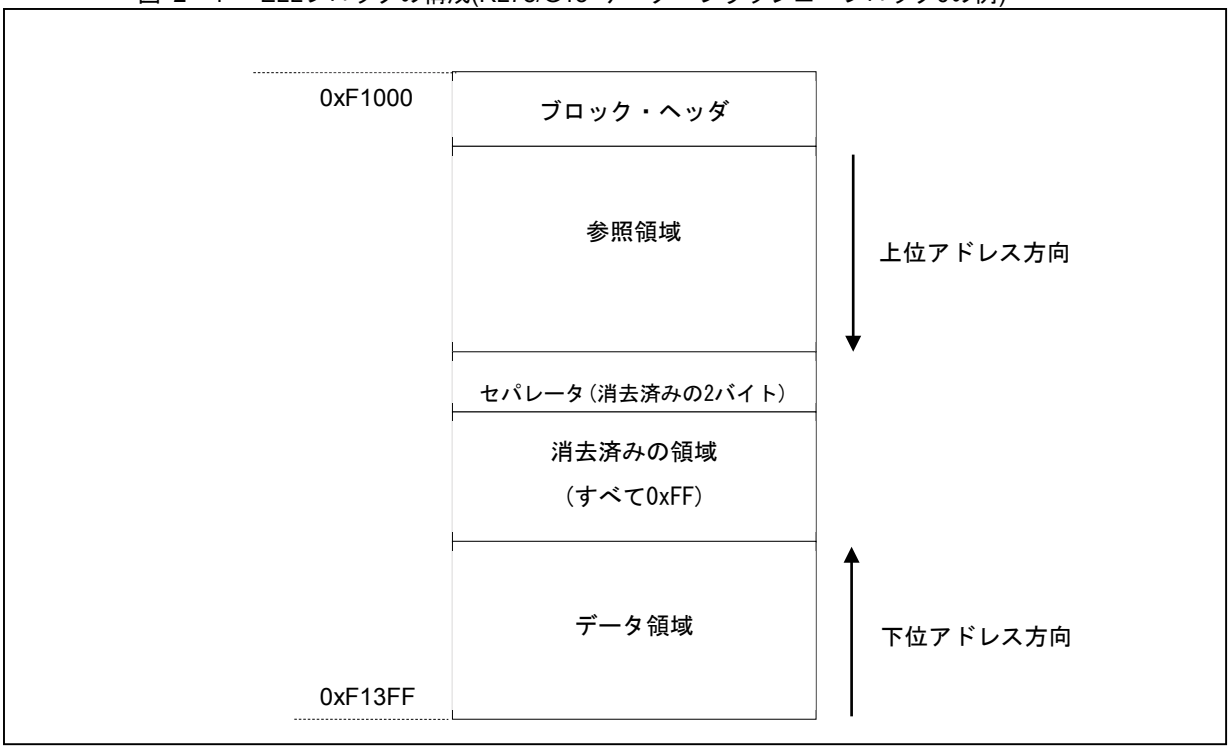
★ 表 2−1　EELプールの状態一覧

状態	説明
プール動作可能	EEL動作時の通常状態です。すべてのコマンドが実行可能です。
プール・フル	使用中の有効ブロックは、書き込みを行うための空き容量が足りません。リフレッシュの実行が必要であることを示しています。
プール消耗	継続して使用できるEELブロックがなくなりました（EELの動作には、最低でも使用禁止でないブロックが2つ必要です）。
プール不整合	プールの状態に不整合があり、EELブロック内に存在するデータ構造が、ユーザが設定した構造と一致しません。EELブロックが不定状態(有効ブロックがない等)です。

2. 3. 3　EELブロック構造

EELが使用するEELブロック構造を「図2−4 ELLブロックの構成(RL78/G13 データ・フラッシュ・ブロック0の例)に示します。EELブロックは、ブロック・ヘッダ、参照領域、データ領域の3つの利用領域からなります。

★ 図 2−4　ELLブロックの構成(RL78/G13 データ・フラッシュ・ブロック0の例)



★ 表 2−2　EELブロックの構成一覧

名称	説明
ブロック・ヘッダ	EELプール内のブロック管理に必要なブロック状態の情報が格納されています。8バイトの固定サイズです。
参照領域	データの管理に必要な参照データが格納されています。データが書き込まれると、上位アドレス方向に拡大します。
データ領域	ユーザ・データが格納されています。データが書き込まれると、下位アドレス方向に拡大します。

参照領域とデータ領域の間には、消去済み領域があります。データが更新される（データの書き込みが行われる）たびに、この領域は減少します。しかし、参照領域とデータ領域の間には、領域の分離とブロック管理のために最低でも2バイトの未使用領域が常に残されています。これは「図2−4 ELLブロックの構成(RL78/G13 データ・フラッシュ・ブロック0の例)」ではセパレータとして示されています。

EELブロック・ヘッダの詳細を「2.3.4　EELブロック・ヘッダ」に、参照領域とデータ領域に格納されるデータの構造は「2.3.5 格納データの構造」にて説明します。

2. 3. 4　EELブロック・ヘッダ

ブロック・ヘッダの構成を「図2-5　EELブロック・ヘッダの構成」に示します。8バイトで構成され、その中の4バイトはシステムで予約されています。

図 2−5　ELLブロック・ヘッダの構成

ブロック内の相対アドレス		
0x0000	A	N
0x0001	B	0xFF - N
0x0002	I	0x00
0x0003	X	0x00
0x0004	-	予約
0x0005	-	予約
0x0006	-	予約
0x0007	-	予約

ブロック状態フラグは、ブロックの先頭からAフラグ、Bフラグ、Iフラグ、Xフラグの各1バイトずつ計4バイトのデータとして配置されます。各フラグの組み合わせによりEELブロックの状態を示します。「図2-5　EELブロック・ヘッダの構成」に各フラグの配置状態を、「表2-3　ブロック状態フラグの説明」に各フラグの組み合わせによる状態を示します。

表 2−3　ブロック状態フラグの説明

ブロック状態フラグ				状　態	概　要
Aフラグ	Bフラグ	Iフラグ	Xフラグ		
0x01	0xFE	0xFF	0xFF	有効	現在使用中のブロック。 EEL_CMD_REFRESHコマンドを実行後、新しい有効ブロックのAフラグには0x02が設定されます。
0x02	0xFD	0xFF	0xFF		現在使用中のブロック。 EEL_CMD_REFRESHコマンドを実行後、新しい有効ブロックのAフラグには0x03が設定されます。
0x03	0xFC	0xFF	0xFF		現在使用中のブロック。 EEL_CMD_REFRESHコマンドを実行後、新しい有効ブロックのAフラグには0x01が設定されます。
上記以外のデータ		0xFF	0xFF	無効	無効状態となったブロック。
−		0xFF以外	0xFF		
−		−	0xFF以外	使用禁止	使用禁止となったブロック。

2. 3. 5　格納データの構造

EELブロックにユーザ・データを書き込むときの格納データの構造を示します。データは、レコード開始（SoR）フィールド、レコード終了（EoR）フィールド、データ・フィールドの3つの部分から成ります。EELディスクリプタ・テーブルを使って、EEL内部で使用するデータを設定できます。各データは識別番号（ID）によって参照され、1～255バイトまでのサイズが設定可能です。（EELディスクリプタ・テーブルのフォーマットの正確な仕様は4.3に記載されています。）

データが書き込まれるたびにEELブロック内に格納データが増加し、複数の格納データが存在することになりますが、参照されるのは最新の格納データのみです。

SoRとEoRは、データの管理に必要な参照データを構成します。参照データとユーザ・データは有効ブロック内の別々のフィールドに格納されますが、これらのフィールドはそれぞれ参照領域、データ領域と呼ばれます。データの全体構造の概要を「図2-6　格納データの構造」に示します。

図 2－6　格納データの構造

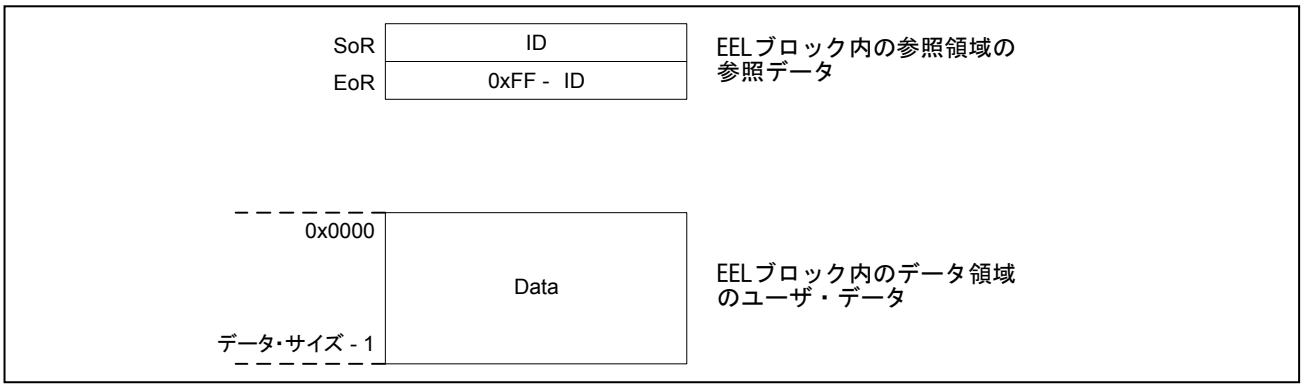


表 2－4　データの各領域の説明

名称	説明
SoRフィールド （Start of Record）	1バイトのSoRフィールドにはデータのデータIDが格納されています。 このフィールドは、書き込み処理の開始を示します。消去済みセルのパターンを避けるためにデータIDの0x00と0xFFは使用されません。
EoRフィールド （End of Record）	1バイトのEoRフィールドには0xFF – データIDの値が格納されています。 このフィールドは、書き込み処理の正常終了を示します。デバイスのリセット等により書き込みが不完全である場合には、対応する格納データはEELから無視されます。
データ・フィールド	データ･フィールドにはユーザ･データが格納されています。1～255バイトの範囲となります。 サイズが2バイト以上のデータの場合、小さいアドレスのデータが、データ・フィールドの小さいアドレス側に格納されます(図2-7で確認してください)。

格納データは、SoR　→　データ・フィールド　→　EoR　の順番でEELブロックへの書き込みが行われます。また、書き込み処理が正常に終了しなかった場合、ひとつ前のデータが有効となります。

注1　各格納データによって消費される参照データの合計サイズは2バイトです。EEL_GetSpace関数を用いてデータの書き込み前に空き容量を確認する際には、このことを考慮する必要があります。

注2　ユーザ・データにチェックサムは追加されません。チェックサムが必要な場合、ユーザ・データにチェックサムを付加し、判定する等ユーザ・プログラムで対応してください。

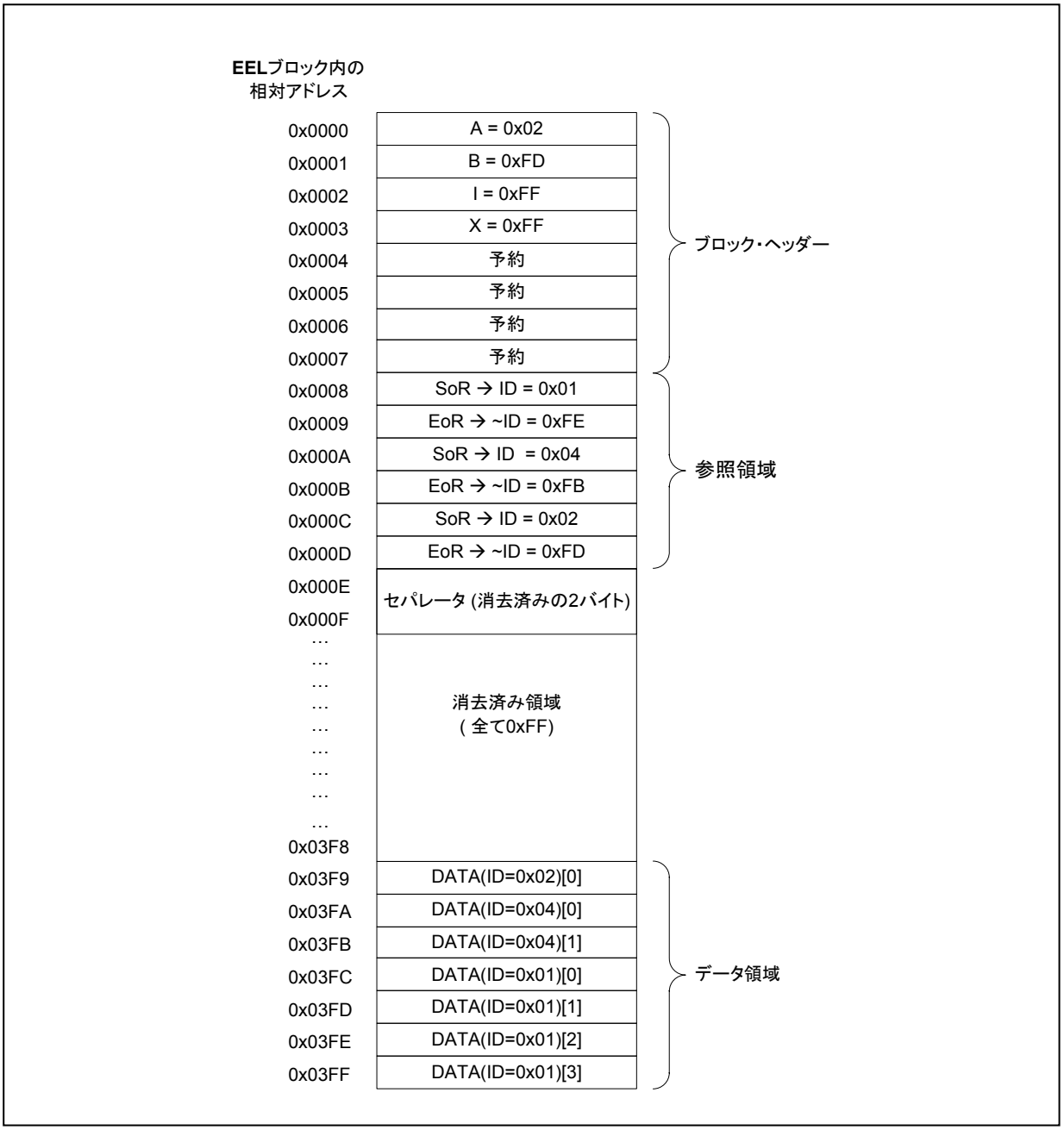
2. 3. 6　EELブロックの概要

「図2-7　有効なEELブロックの例」に、複数の格納データを含むEELブロックの例を示します。:

- •データID 0x01　：データ長　4バイト
- •データID 0x02　：データ長　1バイト
- •データID 0x03 は定義されているが、書き込まれていません。
- •データID 0x04　：データ長　2バイト

データは、データID 0x01→データID 0x04→データID 0x02の順に書かれています。　この例では、データID 0x03のデータはまだ書き込まれていません。

図 2-7　有効なEELブロックの例



第3章　EEL機能

この章では、EEPORMエミュレーションを実行する上で必要なEELの機能について説明します。

3. 1　EEL関数一覧 / EEL_Execute関数 各コマンド機能

「表 3−1　EEL関数」に、EELが提供しているEEL関数を示します。

表 3−1　EEL関数

EEL関数名	機能概要
FDL_Init	FDLの初期化
FDL_Open	FDLの準備処理
FDL_Close	FDLの終了処理
EEL_Init	EELの初期化
EEL_Open	EELの準備処理
EEL_Close	EELの終了処理
EEL_Execute	各コマンドによるデータ・フラッシュ操作の実行 コマンド：EEL_CMD_STARTUP EEL_CMD_WRITE EEL_CMD_READ EEL_CMD_REFRESH EEL_CMD_VERIFY EEL_CMD_FORMAT EEL_CMD_SHUTDOWN
EEL_Handler	実行中のEELを制御
EEL_GetSpace	EELブロックの空き容量の確認処理
EEL_GetVersionString	EELのバージョン情報の取得

以下に、EEL_Execute関数において実行できる各コマンドの機能の内容について説明します。

3. 1. 1　EEL_CMD_STARTUPコマンド【スタートアップ処理】

EEL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。

3. 1. 2　EEL_CMD_SHUTDOWNコマンド【シャットダウン処理】

EEPROMエミュレーションを停止状態(opened)にします。

3. 1. 3　EEL_CMD_REFRESHコマンド【リフレッシュ処理】

有効ブロック（コピー元ブロック）からEELプールの次のブロック（コピー先ブロック）に対し、消去処理後に各データの最新の格納データをコピーします。これにより、コピー先ブロックが新たな有効ブロックとなります。

3. 1. 4　EEL_CMD_FORMATコマンド【フォーマット処理】

EEL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)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。

3. 1. 5　EEL_CMD_WRITEコマンド【書き込み処理】

EELブロックへ指定データの書き込みを行います。

3. 1. 6　EEL_CMD_READコマンド【読み出し処理】

EELブロックから指定データの読み出しを行います。

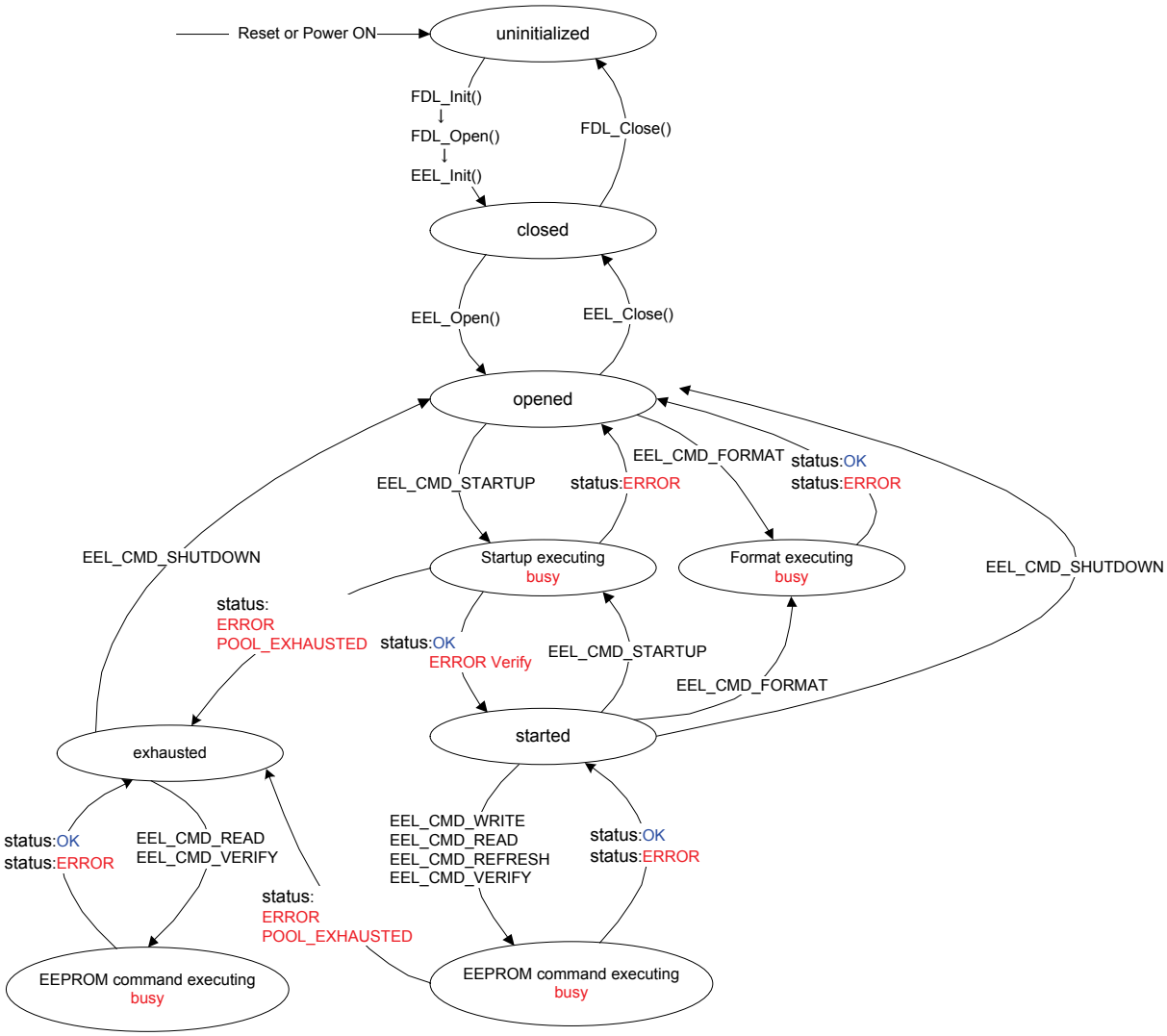
3. 1. 7　EEL_CMD_VERIFYコマンド【ベリファイ処理】

有効ブロックの信号レベルを確認するためのベリファイ(内部ベリファイ)を行います。

3. 2　状態遷移

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EELの初期化処理を行い、書き込みや読み出し等EEL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3-1　状態遷移図」に、基本的な機能を使用するための操作フローを「図3-2 EEL基本フローチャート」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 3-1　状態遷移図



注意1　EEL_Close関数およびEEL_Init関数はどの状態からも呼び出し可能です。ただし、これによってEELのあらゆる処理は中断し、予測不可能な挙動となる可能性があります。

★ 注意2　EEL_CMD_FORMATコマンドを開始した場合、“EEL_Handler関数”を実行して、必ず終了を確認してください。

【状態遷移図の概要】

EEL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。

(2) closed

FDL_Init関数、FDL_Open関数、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FSLやSTOPモード、HALTモードを実行する場合は、opened状態からEEL_Close関数を実行し、この状態に遷移させてください。

(3) opened

closed状態からEEL_Open関数を実行し、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関数を実行し、closed状態に遷移するまでの間はFSL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出しを行います。

(5)exhausted

opened状態およびstarted状態から、コマンド実行中に継続して使用できるEELブロックがなくなった状態です。この状態では、EEL_CMD_READコマンド、EEL_CMD_VERIFYコマンド、EEL_CMD_SHUTDOWNのみ実行できます。

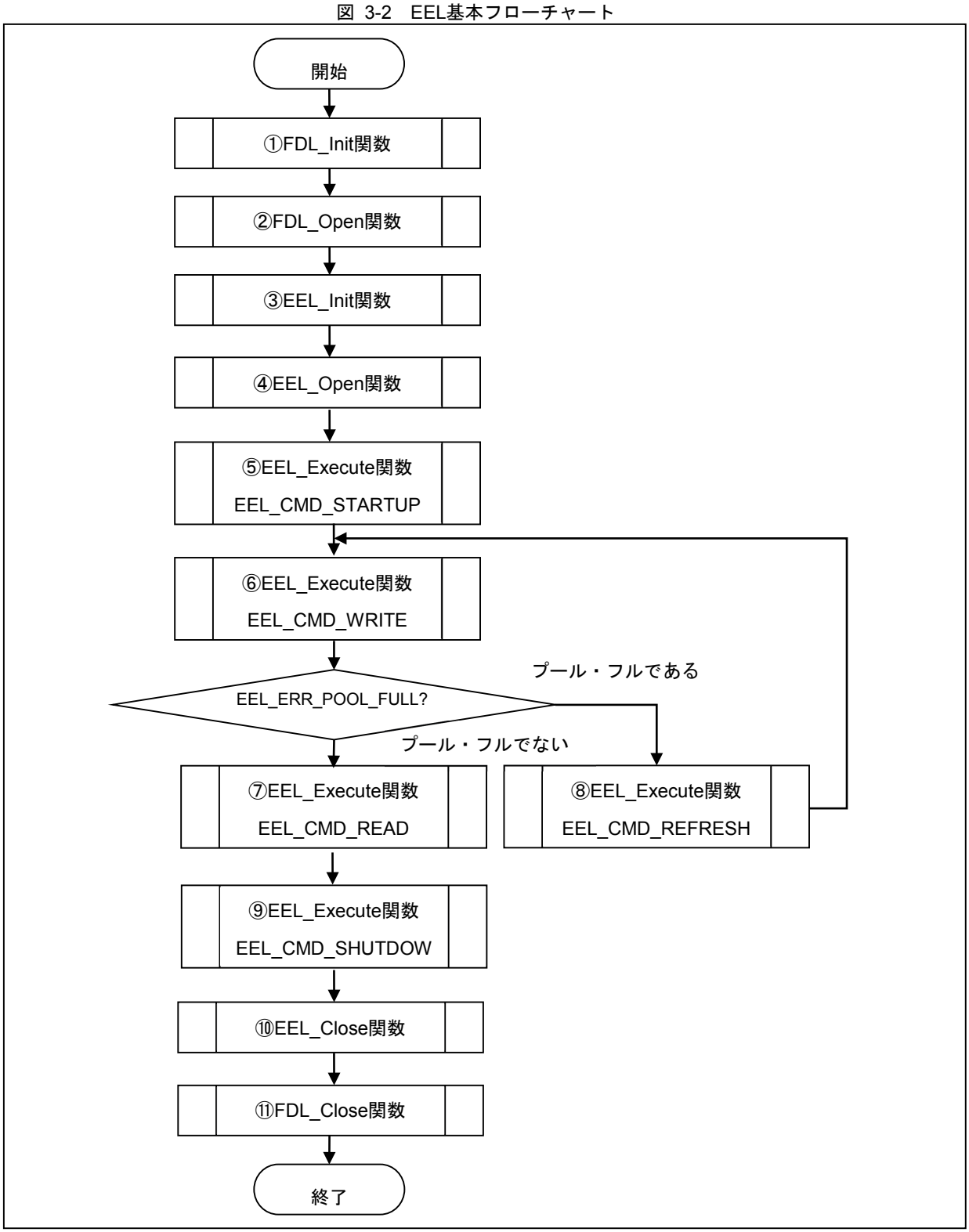
(6) busy

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。

3. 3　基本フローチャート

「図 3-2　EEL基本フローチャート」に、EELを用いてデータ・フラッシュを操作（書き込み、読み出し等）する際の基本手順を示します。

図 3-2　EEL基本フローチャート



注意1　EEPROMエミュレーションを初めて使用する場合、必ずEEL_CMD_FORMATコマンドを実行して下さい。

注意2　上記フローはエラー処理を省略しています。

【基本操作フローの概要】

① FDLの初期化処理 (FDL_Init関数)
EEL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、FDLのパラメータ(RAM)を初期化する必要があるため、FDL_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FSL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 FDLの準備処理(FDL_Open関数)
データ・フラッシュ・コントロール・レジスタ（DFLCTL）をデータ・フラッシュ・メモリへのアクセス許可状態（DFLEN = 1）に設定します。

③ EELの初期化処理 (EEL_Init関数)
EELで使用するパラメータ(RAM)を初期化します。

④ EEPROMエミュレーション準備処理 (EEL_Open関数)
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。

⑤ EEPROMエミュレーション実行開始処理 (EEL_Execute関数 : 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。

⑥ EEPROMエミュレーション・データ書き込み処理 (EEL_Execute関数 : EEL_CMD_WRITEコマンド)
指定されたデータをEELブロックへ書き込みます。

⑦ EEPROMエミュレーション・データ読み出し処理 (EEL_Execute関数 : EEL_CMD_READコマンド)
指定されたデータをEELブロックから読み出します。

⑧ EEPROMエミュレーション・リフレッシュ処理 (EEL_Execute関数 : EEL_CMD_REFRESHコマンド)
有効ブロック（コピー元ブロック）からEELプールの次のブロック（コピー先ブロック）に対し、消去処理後に各データの最新の格納データをコピーします。これにより、コピー先ブロックが新たな有効ブロックとなります。

⑨ EEPROMエミュレーション実行停止処理 (EEL_Execute関数 : 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。

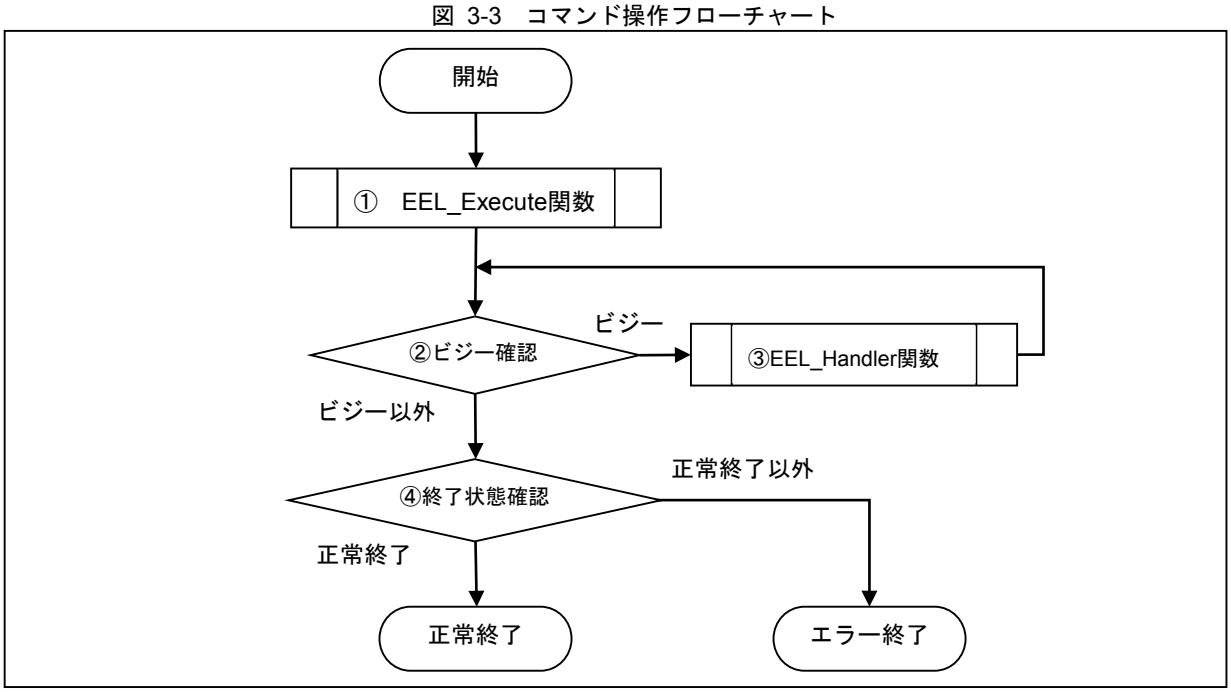
⑩ EEPROMエミュレーション終了処理 (EEL_Close関数)
EEPROMエミュレーション終了するため、データ・フラッシュ・メモリを制御出来ない状態(closed)にします。

⑪ FDLの終了処理(FDL_Close関数)
データ・フラッシュ・コントロール・レジスタ（DFLCTL）をデータ・フラッシュ・メモリへのアクセス禁止状態（DFLEN = 0）に設定します。

3. 4　コマンド操作フローチャート

「図 3-3　コマンド操作フローチャート」に、EELを用いてデータ・フラッシュを操作（書き込み、読み出し等）する際の基本手順を示します。

図 3-3　コマンド操作フローチャート



① EEL_Execute関数

データ・フラッシュ操作を実行します。

② ビジー確認

リクエスト・ストラクチャー（eel_request_t）のstatus_enuを確認します。

EEL_BUSYであれば、引き続きデータ・フラッシュ操作を実行します。EEL_BUSY以外であればエラー終了してください。

③ EEL_Handler関数

実行中のEELを制御します。EEL_Handler関数を繰り返し実行することによってデータ・フラッシュ操作を進行させます。

★ ④ 終了状態確認

EEL _OKであれば、正常終了です。EEL_OK以外であればエラー終了してください。

3. 5　BGO（Back Ground Operation）機能

EEL_Execute関数は、コマンドの処理を開始させたのち、制御を直ちにユーザ・プログラムに戻します。
つまり、データ・フラッシュ操作中も、ユーザ・プログラムを動作させることが可能です。
本機能をBGO（Back Ground Operation）と呼びます。

★　データ・フラッシュに対する書き込み、読み出し等のコマンドは、初回の処理はEEL_Execute関数の呼び出しで実行されますが、2回目以降の処理はEEL_Handler関数の呼び出しをトリガとして実行されます。よって、EEL_Handler関数の呼び出しを複数回実行する必要があります。また、複数に分割されているデバイスのハードウエア処理それぞれにおいて、完了してから次のトリガがかかるまでの間、処理は保留状態となりますので、EEL_Handler関数の呼び出し間隔が長くなりますと全体の処理時間が延びることになります。

なお、EEL_Execute関数から要求された処理が正常に完了したか否かについては、ユーザ・プログラムからEEL_Handler関数を呼び出して、処理の実行状況を確認することが必要となります。

EEL_CMD_SHUTDOWNコマンドは、EEL_Handler関数の呼び出しは必要ありません。ただし、図3-3で示されている、コマンド操作フローチャートを踏襲することを推奨します。

第4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、3ブロック以上（推奨）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64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。

EEL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4.2 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。

4. 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。

表 4－1　注意事項一覧(1/2)

No	注意事項
1	全てのEELのコードと定数は、同一の64KBのフラッシュ・ブロック内に配置する必要があります。
2	FDL_Init関数による初期化は、FDL_Open関数、FDL_Close関数、およびすべてのEEL関数群を実行する前に行う必要があります。
3	EEL_Init関数によるEELの初期化は、すべてのEEL関数群を実行する前に行う必要があります。
4	EEL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関数およびFDL_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。
6	ウォッチドック・タイマはEEL実行中も停止しません。
7	リクエスト・ストラクチャー（eel_request_t）は偶数アドレスに配置しなければなりません。
8	コマンド実行中はリクエスト・ストラクチャー（eel_request_t）を破壊しないでください。
9	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。
10	コマンドを実行する前に、リクエスト・ストラクチャー（eel_request_t）のすべてのメンバは一度初期化する必要があります。リクエスト・ストラクチャー（eel_request_t）内に使用されないメンバがある場合には、そのメンバに任意の値を設定してください。初期化されない場合、RAMパリティ・エラーによって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。詳細については、お使いのRL78製品の「ユーザーズ・マニュアル: ハードウエア編」を参照してください。
11	EELは多重実行に対応していないため、EEL関数を割り込み処理内で実行しないで下さい。

表 4－1　注意事項一覧(2/2)

No	注意事項
12	FDL_Close関数およびEEL_Close関数の実行後は、要求済みコマンドおよび実行中のコマンドは停止し、それを再開することはできません。FDL_Close関数およびEEL_Close関数を呼び出す前に、実行中のコマンドをすべて終了させてください。
13	EEPROMエミュレーション実行中にFSLを実行しないでください。FSLを使用する場合は必ずEEL_Close関数およびFDL_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。FSL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(FDL_Init関数)から処理を行う必要があります。
14	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。また、外部クロックを使用時も、高速オンチップ・オシレータは起動しておく必要があります。
15	EEL関数、およびFDL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。
16	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。
17	EEPROMエミュレーションが終了するまで、EEPROMエミュレーションで使用するRAM領域（セルフRAM含む）を破壊しないでください。
18	EELではユーザ・データにチェックサムを追加しません。チェックサムが必要な場合、ユーザ・データにチェックサムを付加し、判定する等ユーザ・プログラムで対応してください。
19	FDLディスクリプタもしくはEELディスクリプタが変更された場合、EEPROM エミュレーションの実行を継続することはできません。このような場合には、FDLとEELの初期化だけでなく、EEL_CMD_FORMATコマンドによるEELプールのフォーマットを行う必要があります。ただし、データの追加の場合は、実行を継続することができます。
20	EEL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。
21	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EEL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。
22	EELを使用するためには、データ・フラッシュ・メモリが3ブロック以上を推奨します。
23	他のEEL/FDLを使用するユーザ・プログラム等でEELブロックを破壊しないください。
24	EELは多重実行に対応していません。OS上でEEL関数を実行する場合は、複数のタスクからEEL関数を実行しないで下さい。
25	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（FDL_Init関数）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、1MHz、 2MHz、 3MHz 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]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　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。

4. 2　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EELブロックに書き込まれる場合に必要なサイズを1ブロックの1/2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。

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　　: 8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）　　: 2バイト

1ブロックの最大使用可能サイズ ＝ 1024バイト － 8バイト － 2バイト ＝ 1014バイト

【最大サイズと推奨サイズ】

データはすべて1ブロック内に収める必要があります。そのため、最大サイズは1ブロックの最大使用可能サイズですが、以下の関係式を満たすことを推奨します。全データを1回は更新できるようにするため、1ブロックの最大使用可能サイズの半分の容量内で使用することを推奨しています。

最大サイズ ＝　ユーザ・データの基本合計サイズ　＋　最大のデータ・サイズ　+　2　≤　1014
（全データを書き込み後、一番サイズが大きなデータを1回更新できることを想定）

推奨サイズ ＝　1014　/　2
（全データを書き込み後、全データを1回更新できることを想定）

【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＋参照エントリ・サイズ（2バイト）

注　詳細については2.3.5　格納データの構造の項を参照してください。

【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 ＋ 2）＋（ユーザ・データ2 ＋ 2）・・・＋（ユーザ・データn ＋ 2）

4. 3　ユーザ設定初期値

EELの設定値初期として、次に示す項目を必ず設定する必要があります。また、EEL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。外部クロックを使用時も、高速オンチップ・オシレータは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のデータ・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    32000000      : (1) 動作周波数
#define   FDL_WIDE_VOLTAGE_MODE                 : (2) 電圧モード
#define   FDL_POOL_BLOCKS         0             : (3) FDLプール・サイズ
#define   EEL_POOL_BLOCKS         4             : (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_VAR_NO      8                     : (5)格納データ数
```

＜ 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2

```
__far const eel_u08 eel_descriptor[EEL_VAR_NO+2] =   : (6)識別子(データID)の
{                                                      データ・サイズ
 (eel_u08)(EEL_VAR_NO),    /* variable count   */  ¥
 (eel_u08)(sizeof(type_A)),  /* id=1            */  ¥
 (eel_u08)(sizeof(type_B)),  /* id=2            */  ¥
 (eel_u08)(sizeof(type_C)),  /* id=3            */  ¥
 (eel_u08)(sizeof(type_D)),  /* id=4            */  ¥
 (eel_u08)(sizeof(type_E)),  /* id=5            */  ¥
 (eel_u08)(sizeof(type_F)),  /* id=6            */  ¥
 (eel_u08)(sizeof(type_X)),  /* id=7            */  ¥
 (eel_u08)(sizeof(type_Z)),  /* id=8            */  ¥
 (eel_u08)(0x00),          /* zero terminator    */  ¥
};
```

注1　使用しているマクロは、EEL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。

注2　EEL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EELブロックの再初期化(EEL_CMD_FORMATコマンド実行)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1]
設定値は以下の計算式によりFDL_Init関数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）。

FDL_Init関数の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1　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2]

データ・フラッシュ・メモリの電圧モードを設定します。[注3]
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2　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。
注3　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) FDLプール・サイズ

0を設定してください。

(4) EELプール・サイズ[注4]

必ず対象デバイス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EELプールのブロック数に設定してください。
注4　3 (3ブロック)以上の値を設定してください(推奨)

(5) 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1～64の範囲です。

(6)各データ識別子(データID)のデータのサイズ

各識別子のデータの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。EELでは、プログラム動作中に識別子を追加のみすることができます。書き込みを行うデータは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 4-1　EELディスクリプタ・テーブル（8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 ＋ 2]

EEL_VAR_NO
データID1のバイト・サイズ
データID2のバイト・サイズ
データID3 のバイト・サイズ
データID4のバイト・サイズ
データID5のバイト・サイズ
データID6のバイト・サイズ
データID7のバイト・サイズ
データID8のバイト・サイズ
0x00

・EEL_VAR_NO

ユーザが指定するEELで使用するデータの数です。

・データIDxのバイト・サイズ

ユーザが指定する各ユーザ・データのバイト・サイズです。

・終端領域(0x00)

終端情報として0を設定します。

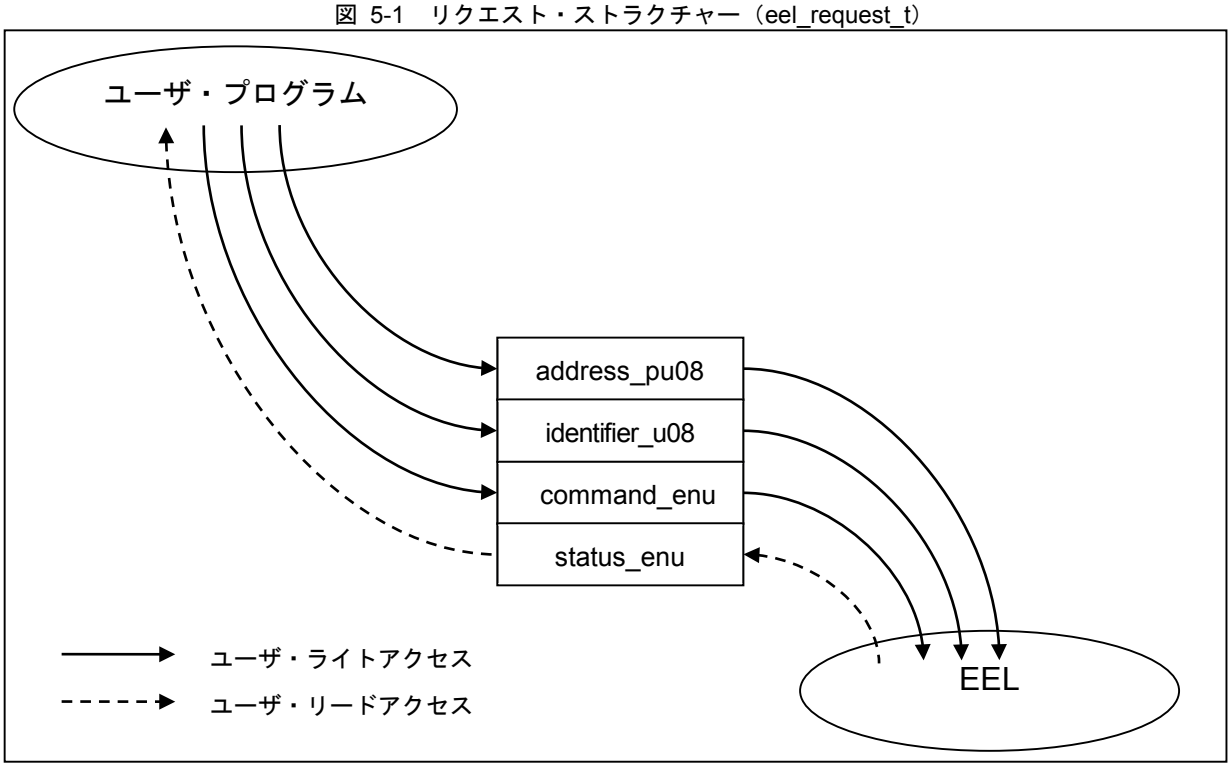
第5章　ユーザ・インタフェース

5.1　リクエスト・ストラクチャー（eel_request_t）設定

データ・フラッシュへの書き込み、読み出し等の基本操作は一つの実行関数で実行されます。実行関数へリクエスト・ストラクチャー（eel_request_t） 経由でコマンドやデータIDをEELへ受け渡します。また、逆にEELの状態、エラー情報をリクエスト・ストラクチャー（eel_request_t）経由で取得します。

以降ユーザによるリクエスト・ストラクチャー（eel_request_t）への書き込みアクセスを「**ユーザ・ライトアクセス**」と呼び、読み出しアクセスを「**ユーザ・リードアクセス**」と呼びます。

図 5-1　リクエスト・ストラクチャー（eel_request_t）



リクエスト・ストラクチャー（eel_request_t） はファイル eel_types.h に記述されています。ユーザによる変更は禁止です。

【リクエスト・ストラクチャー（eel_request_t）の定義】

```
typedef struct
{
    __near eel_u08*         address_pu08;
           eel_u08          identifier_u08;
           eel_command_t    command_enu;
           eel_status_t     status_enu;
} eel_request_t;
```

図 5-2　リクエスト・ストラクチャー（eel_request_t）変数配置

__near eel_u08*　address_pu08	
eel_u08 identifier_u08	eel_command_t command_enu;
eel_status_t status_enu;	

bit0　　Bit15

注意　リクエスト・ストラクチャー（eel_request_t）は偶数アドレスに配置する必要があります。

5. 1. 1　ユーザ・ライトアクセス

（1）address_pu08

EEL_CMD_WRITEコマンド、EEL_CMD_READコマンド時に使用するデータ・バッファの先頭アドレスを設定します。

対応コマンド名（マクロ名）	設定値
EEL_CMD_WRITE	データ・バッファ[注1] の先頭アドレス
EEL_CMD_READ	データ・バッファ[注2]の先頭アドレス

注1　ユーザの書き込むデータが配置されているバッファ

注2　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

（2）identifier_u08

各コマンドで使用するデータIDを設定します。設定方法の詳細は「5.4　EEL関数の説明EEL_Execute」のページをご参照ください。

対応コマンド名（マクロ名）	設定値
EEL_CMD_WRITE	書き込みデータのID指定
EEL_CMD_READ	読み出しデータのID指定

（3） command_enu

共通実行関数へ設定するコマンド

コマンド名（マクロ名）	説明
EEL_CMD_STARTUP	スタートアップ処理
EEL_CMD_WRITE	書き込み処理
EEL_CMD_READ	読み出し処理
EEL_CMD_REFRESH	リフレッシュ処理
EEL_CMD_VERIFY	ベリファイ処理
EEL_CMD_FORMAT	フォーマット処理
EEL_CMD_SHUTDOWN	シャットダウン処理

5. 1. 2　ユーザ・リードアクセス

- status_enu

EELの状態、エラー情報。各関数において発生する可能性のある状態、エラーにつきましては「5.4　EEL関数の説明EEL_Execute」の各関数をご参照ください。

5. 2　EEL関数の呼び出し

EEL関数をC 言語、および、アセンブリ言語で記述されたユーザ・プログラムから呼び出す方法を以下に示します。

- C 言語

EEL関数を C 言語で記述されたユーザ・プログラムから呼び出す場合、通常の C 言語関数と同様の方法で呼び出しを行うことにより、 EEL関数のパラメータはEELに引き数として渡され、該当処理が実行されます。

- アセンブリ言語

EEL関数をアセンブリ言語で記述されたユーザ・プログラムから呼び出す場合、ユーザが開発環境として使用するC コンパイラ・パッケージの関数呼び出し規約にしたがった処理（パラメータの設定、戻り番地の設定等）を行ったのち呼び出しを行うことにより、 EEL関数のパラメータはEELに引き数として渡され、該当処理が実行されます。

備考1　EELが提供している EEL関数をユーザ・プログラムから呼び出す場合、以下に示した標準ヘッダ・ファイルの定義（インクルード処理）を行う必要があります。

C言語用

fdl.h：　FDLヘッダ・ファイル

fdl_types.h：FDL定義設定・ヘッダ・ファイル

eel.h：EELヘッダ・ファイル

eel_types.h：：EEL定義設定・ヘッダ・ファイル

アセンブラ言語用

fdl.inc：　FDLヘッダ・ファイル

eel.inc：EELヘッダ・ファイル

eel_types.inc：　EEL定義設定・ヘッダ・ファイル

備考2　EEL_Init関数以外のEEL関数については、EEL_Init関数を呼び出す前に呼び出した場合、動作は保証されません。

備考3　FDL_Init関数以外のEEL関数については、FDL_Init関数を呼びだす前に呼び出した場合、動作は保証されません。

5. 3　データ・タイプ

EELが提供しているEEL関数を呼び出す際に指定する各種パラメータのデータ・タイプを示します。

マクロ名	説明
eel_u08	符号なし8ビット整数（unsigned char）
eel_u16	符号なし16ビット整数（unsigned short）
eel_u32	符号なし32ビット整数（unsigned long）

5. 4　EEL関数の説明

次項からEELが提供しているEEL関数について、以下の記述フォーマットにしたがって解説します。

名称

【機　能】

この関数の機能概要を示します。

【書　式】

＜C言語＞

この関数をC言語で記述されたユーザ・プログラムから呼び出す際の書式を示します。

＜アセンブラ＞

この関数をアセンブリ言語で記述されたユーザ・プログラムから呼び出す際の書式を示します。

【事前条件】

この関数の事前条件を示します。

【事後条件】

この関数の事後条件を示します。

【注意】

この関数の注意事項を示します。

【呼び出し後のレジスタ状態】

この関数を呼び出した後のレジスタの状態を示します。

【引　数】

この関数の引数を示します。

【戻り値】

この関数からの戻り値を示します。

FDL_Init

【機　能】
FDL の初期化処理

【書　式】
＜C 言語＞

```
fdl_status_t __far FDL_Init(const __far fdl_descriptor_t* descriptor_pstr)
```

＜アセンブラ＞

```
CALL !FDL_Init または CALL !!FDL_Init
```

備考FDL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。

【事前条件】

1. FSL、およびEEL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。

【事後条件】

FDL_Open関数を実行してください。

【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. FSLとは排他利用となります。本関数を実行する前にFSLは必ず終了させてください。また、 EEL実行中にFSLは使用しないでください。
3. 本関数を実行後にFSL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、EEL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EELを終了させてください。
5. 本関数で使用するディスクリプタ・テーブルは修正できません。必ず定義済みのディスクリプタ・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】

引数	型	説　明
descriptor_pstr	fdl_descriptor_t* (far)	ディスクリプタ・テーブルへのポインタ

【戻り値】

型	シンボル定義	説　明
fdl_status_t	FDL_OK	正常終了
	FDL_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。

FDL_Open

【機　能】
FDL の準備処理

データ・フラッシュ・コントロール・レジスタ（DFLCTL）をデータ・フラッシュ・メモリへのアクセス許可状態（DFLEN = 1）に設定します。

【書　式】
＜C 言語＞

```
void __far FDL_Open(void)
```

＜アセンブラ＞

```
CALL !FDL_Open  または  CALL !!FDL_Open
```

備考 FDL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。

【事前条件】
FDL_Init関数を正常終了させていること。

【事後条件】
EEL_Init関数を実行してください。

【注意】
なし

【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

FDL_Close

【機　能】

FDLの終了処理

データ・フラッシュ・コントロール・レジスタ（DFLCTL）をデータ・フラッシュ・メモリへのアクセス禁止状態（DFLEN = 0）に設定します。動作中のすべてのEEL処理が停止します。

【書　式】

＜C 言語＞

```
void __far FDL_Close(void)
```

＜アセンブラ＞

```
CALL !FDL_Close  または  CALL !!FDL_Close
```

備考FDL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。

【事前条件】

FDL_Init関数、FDL_Open関数およびEEL_Init関数、EEL_Open関数、EEL_Close関数を正常に終了させていること。

【事後条件】

なし

【注意】

なし

【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_Init

【機　能】
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　EEL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. FSL、およびEELが未実行、あるいは終了していること。
2. FDL_Init関数およびFDL_Open関数を正常終了させていること。

【事後条件】

EEL_Open関数を実行してください。

【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. FSLとは排他利用となります。本関数を実行する前にFSLは必ず終了させてください。また、 EEL実行中にFSLは使用しないでください。
3. 本関数を実行後にFSL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、EEL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EEL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。FDL_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。 4. 3 ユーザ設定初期値を参考に設定値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。

EEL_Open

【機　能】
EEPROM エミュレーション準備処理
EEPROM エミュレーションを実行できる状態にします。

【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　EEL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。

【事前条件】

1. FDL_Init関数、FDL_Open関数、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関数およびFDL_Close関数を実行して EEPROMエミュレーションを終了させてください。

【事後条件】
なし

【注意】
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FSL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。FSLや、STOPモード、HALTモードを実行する必要がある場合は、EEL_Close関数とFDL_Close関数を実行し、EEPROMエミュレーションをuninitialized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

【機　能】
EEPROM エミュレーション終了処理
EEPROM エミュレーションを実行できない状態にします。

【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　　EEL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。

【事前条件】
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。

【事後条件】
FDL_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。

【注意】
なし

【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

【機　能】

EEPROMエミュレーション実行関数

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。

【書　式】

＜C 言語＞

```
void __far EEL_Execute(__near eel_request_t* request_pstr);
```

＜アセンブラ＞

```
    CALL !EEL_Execute  または  CALL !!EEL_Execute
```

備考　EEL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。

【事前条件】

FDL_Init関数、FDL_Open関数およびEEL_Init関数、EEL_Open関数を正常に終了させていること。

【事後条件】

1. リクエスト・ストラクチャー（eel_request_t）のstatus_enuがEEL_BUSY中は、繰り返しEEL_Handler関数を実行してください。
2. EEL_Execute関数は、コマンドの処理を開始させたのち、制御を直ちにユーザ・プログラムに戻します。コマンドの処理の継続はEEL_Handlerの実行によって行われます。そのため、コマンドの処理が完了するまで、EEL_Handler関数を継続的に実行しなければいけません。

【注意】

なし

【呼び出し後のレジスタ状態】

破壊レジスタ：AX（引数）

【引　数】

引数	型	説　明
request_pstr	eel_request_t*　(near)	リクエスト・ストラクチャー（eel_request_t）のポインタ

リクエスト・ストラクチャーの内容

メンバ	型	説　明
eel_request_t.address_pu08	eel_u08 * (near)	書き込み、読み出しデータ設定用データ・バッファのポインタ注
eel_request_t.identifier_u08	eel_u08	実行コマンドの設定パラメータ
eel_request_t.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、読み出しデータのバイト・サイズ分をご用意ください。

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。有効ブロックが2個あった場合等は、不正なブロックを無効ブロックに変更します。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	EEL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。 ・address_pu08：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 ・identifier_u08：書き込みデータのデータID指定
★ EEL_CMD_READ注1	EELブロックから指定したデータIDの最新データを読み出します。 ※実行には以下の引数の設定が必要です。 ・address_pu08：読み出したデータを保存するRAM領域の先頭アドレスを指定 ・identifier_u08：読み出すデータのデータID指定
EEL_CMD_VERIFY注1,2	有効ブロックの信号レベルを確認するためのベリファイ(内部ベリファイ)を行います。このベリファイは、フラッシュ・メモリのセルの信号レベルが適正であるかを確認します。
EEL_CMD_REFRESH注1,3	EELブロックの有効ブロック（コピー元ブロック）からEELプールの次のブロック（コピー先ブロック）に対し、消去処理後に、各データの最新の格納データをコピーします。これにより、コピー先ブロックが新たな有効ブロックとなります。
EEL_CMD_FORMAT	EEL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)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、EEL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、ディスクリプタ・テーブル等の値(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。

注2　書き込まれたデータを読み出し、比較する処理ではありません。比較する場合はユーザ・プログラムで EEL_Execute(EEL_CMD_READ)関数を使用しデータを読み出して、比較してください。

注3　EEL_CMD_REFRESHコマンドを実行することで、消去処理が実行されます。

EEL_Execute／EEL_Handlerのコマンド実行状態(eel_status_t)一覧(1/2)

コマンド実行状態	カテゴリ	説　明	対応コマンド
EEL_OK	説明	正常終了	すべてのコマンド
	原因	なし	
	対処方法	なし	
EEL_BUSY	説明	コマンド実行中	すべてのコマンド
	原因	なし	
	対処方法	状態が変化するまでEEL_Handler呼び出してください。	
EEL_ERR_POOL_FULL	説明	プール・フル・エラー	EEL_CMD_WRITE
	原因	データを書き込める領域が存在しない。	
	対処方法	EEL_CMD_REFRESHを実行し、書き込みを再実行してください。	
EEL_ERR_INITIALIZATION	説明	初期化エラー	すべてのコマンド
	原因	FDL_Init関数、FDL_Open関数、EEL_Init関数、およびEEL_Open関数が正常に完了していません。	
	対処方法	FDL_Init関数、FDL_Open関数、EEL_Init関数、およびEEL_Open関数を正常に完了させてください。	
EEL_ERR_ACCESS_LOCKED	説明	EEPROMエミュレーション・ロック・エラー	EEL_CMD_STARTUP、EEL_CMD_FORMAT以外のコマンド
	原因	EEPROMエミュレーションが実行できない状態です。	
	対処方法	EEL_CMD_STARTUPコマンドを正常に終了させてください。	
EEL_CMD_UNDEFINED		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_VERIFY	説明	EEL_CMD_STARTUPコマンド実行時： ブロック・ヘッダもしくは最後に書き込まれた格納データのベリファイ(内部ベリファイ)処理中にエラーが発生しました。 EEL_CMD_VERIFYコマンド実行時: 有効ブロックのベリファイ(内部ベリファイ)処理中にエラーが発生しました。	EEL_CMD_STARTUP EEL_CMD_VERIFY
	原因	フラッシュ・メモリのセルで信号レベルが適正でないものがありました。	
	対処方法	EEL_CMD_REFRESHコマンドを実行してください。	
EEL_ERR_PARAMETER	説明	パラメータ・エラー	すべてのコマンド
	原因	コマンドの設定パラメータに誤りがあります。	
	対処方法	設定したパラメータを見直してください。	
EEL_ERR_REJECTED	説明	リジェクト・エラー	すべてのコマンド
	原因	別コマンドが実行中です。	
	対処方法	EEL_Handler関数を呼び出して実行中のコマンドを終了させてください。	

EEL_Execute／EEL_Handlerのコマンド実行状態(eel_status_t)一覧(2/2)

コマンド実行状態	カテゴリ	説　明	対応コマンド
EEL_ERR_NO_INSTANCE	説明	データ未書き込みエラー	EEL_CMD_READ
	原因	指定された識別子のデータが書き込まれていない。	
	対処方法	EEL_CMD_WRITEコマンドで指定された識別子にデータを書いてください。	
EEL_ERR_POOL_INCONSISTENT	説明	EELブロック不整合エラー	EEL_CMD_STARTUP
	原因	EELブロックが不定状態(有効ブロックがない等)です。	
	対処方法	EEL_CMD_FORMATコマンドを実行し、EELブロックを初期化してください。	
EEL_ERR_POOL_EXHAUSTED	説明	EELブロック消耗エラー	EEL_CMD_STARTUP EEL_CMD_FORMAT EEL_CMD_REFRESH EEL_CMD_WRITE
	原因	継続して使用できるEELブロックがなくなりました。	
	対処方法	EEPROMエミュレーションを終了してください。 EEL_CMD_FORMATコマンドを実行し修復(既存のデータはすべて消去されます)を試行、もしくは既存データの読み出しの実行可能。	
EEL_ERR_INTERNAL	説明	内部エラー	EEL_CMD_SHUTDOWN以外のコマンド
	原因	予期しないエラーが発生しました。	
	対処方法	デバイス状態を確認してください。	

【戻り値】
なし

EEL_Handler

【機　能】

EEPROMエミュレーション継続実行処理

★　EEL_Execute関数で指定されたEEPROMエミュレーションの処理を、進行させ終了を確認する関数です。

【書　式】

＜C 言語＞

```
void __far EEL_Handler(void);
```

＜アセンブラ＞

```
CALL !EEL_Handler  または  CALL !!EEL_Handler
```

備考　EEL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。

【事前条件】

1. FDL_Init関数、FDL_Open関数およびEEL_Init関数、EEL_Open関数を正常に終了させていること。
2. EEL_Execute関数を実行し[注]、リクエスト・ストラクチャー（eel_request_t）のstatus_enuがEEL_BUSYとなっていること。

注意　EEL_CMD_SHUTDOWNコマンドにはEEL_Handler関数の実行は必要ありません。
ただし、図3-3で示されている、コマンド操作フローチャートを踏襲することを推奨します。

【事後条件】

リクエスト・ストラクチャー（eel_request_t）のstatus_enu がEEL_BUSY中は、繰り返し実行してください。また、コマンド実行期間以外で、EEL_Handler関数を実行した場合、eel_request_tのstatus_enuは更新されません。

【注意】

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関数を使用する場合、「eel_request_t* request」変数は開放しないでください。EEL_Handler関数が設定するコマンドの実行状態に関しては、EEL_ExecuteのExecute／Handlerのコマンド実行状態(eel_status_t)一覧を参照してください。

【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_GetSpace

【機　能】
EEL ブロックの空き容量を取得

【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　EEL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】
FDL_Init関数、FDL_Open関数およびEEL_Init関数、EEL_Open関数、EEL_Execute(EEL_CMD_STARTUP)関数を正常に終了させていること。

【事後条件】
なし

【注意】
1. EELプールがプール消耗状態の場合、空き容量には常にゼロが戻ります。
2. エラー値が戻る場合、空き容量の情報は変化しません。

【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】

引数	型	説　明
space_pu16	eel_u16*　(near)	現在の有効ブロックの空き容量の情報が入力されるアドレス

【戻り値】

型	シンボル定義	説　明
eel_status_t	EEL_OK	正常終了
	EEL_ERR_INITIALIZATION	EEL_Init関数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

【機　能】
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 EEL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。

【事前条件】
なし

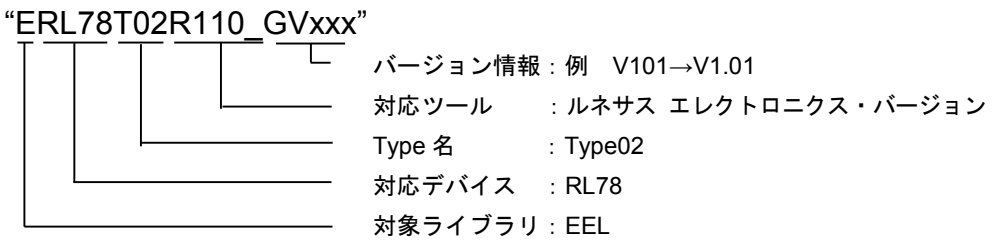
【事後条件】
なし

【注意】
なし

【呼び出し後のレジスタ状態】
なし

【引　数】
なし

【戻り値】

型	説　明
eel_u08* (far)	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Pack02 V1.01の場合（ASCIIコード） 

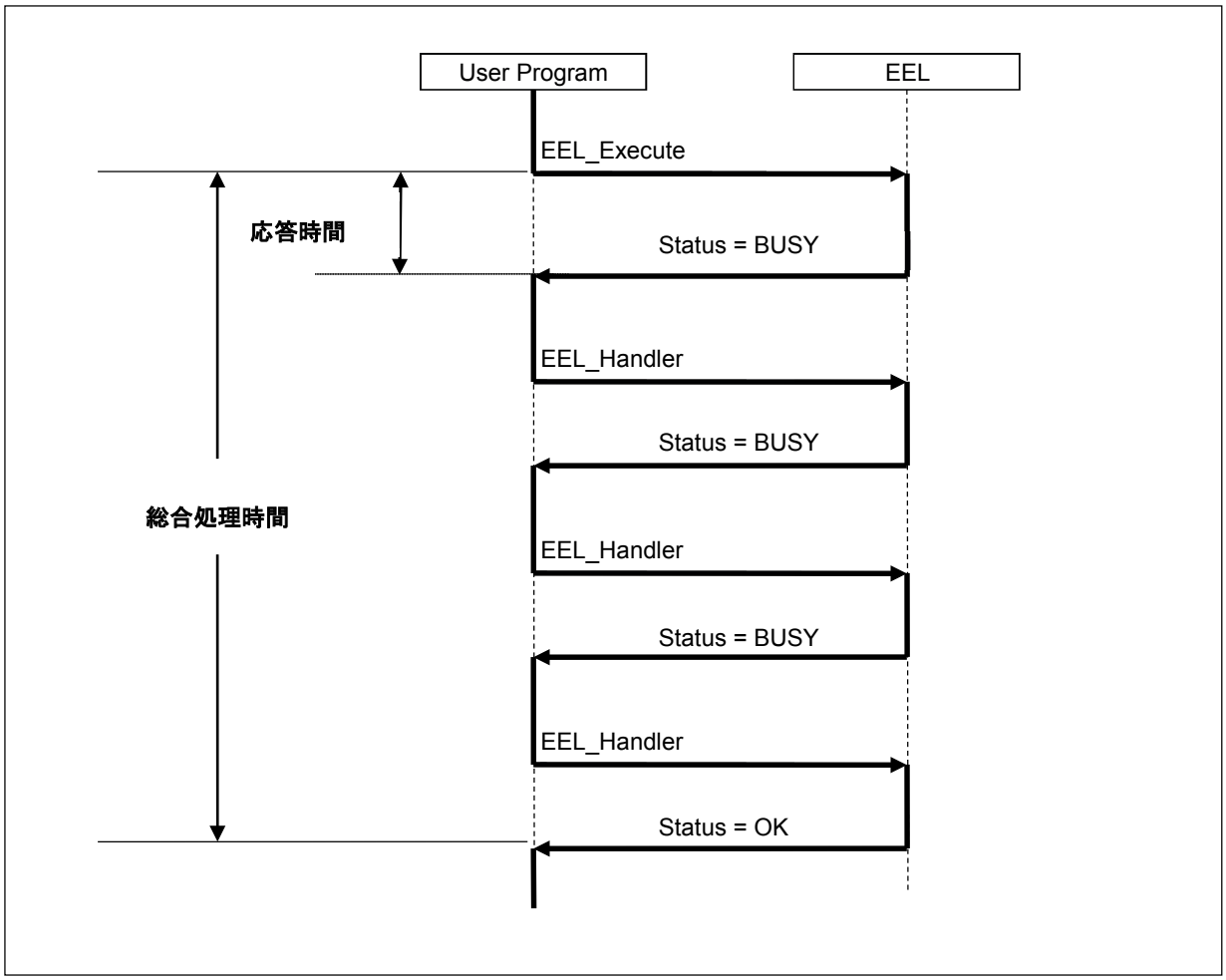
第6章　ソフトウェア・リソースと処理時間

6. 1　処理時間

この節ではEELの処理時間について記載します。

図6-1処理時間の概念図にEEL関数の応答時時間および総合処理時間の概念を示します。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、あるいはエラー等により異常終了する場合の処理時間は含みません。EEL_Handler関数の実行による遅延時間も考慮されていません。EEL_Handler関数の呼び出し間隔が長くなった場合、最大総合処理時間を超える可能性があります。

図 6-1　処理時間の概念図



★　表 6−1　EEPROM エミュレーション・ライブラリPack02 各EEL関数の応答時間

関数	最大時間 (フルスピード・モード)	最大時間(ワイド・ボルテージ・モード)
FDL_Init	1199 / fcpu µs	1199 / fcpu µs
FDL_Open	27 / fcpu + 14 µs	27 / fcpu + 14 µs
FDL_Close	836 / fcpu + 444 µs	791 / fcpu + 969 µs
EEL_Init	3268 / fcpu µs	3268 / fcpu µs
EEL_Open	14 / fcpu µs	14 / fcpu µs
EEL_Close	17 / fcpu µs	17 / fcpu µs
EEL_GetSpace	47 / fcpu µs	47 / fcpu µs
EEL_GetVersionString	14 / fcpu µs	14 / fcpu µs
EEL_Execute	320 / fcpu µs	320 / fcpu µs
EEL_Handler	4582 / fcpu µs	4582 / fcpu µs

★　表 6−2　EEPROM エミュレーション・ライブラリPack02　総合処理時間

関数	最大時間 (フルスピード・モード)	最大時間(ワイド・ボルテージ・モード)
EEL_Execute / EEL_Handler		
EEL_CMD_STARTUP	(280530 + 235 * Block Num) / fcpu + 1612 µs	(277604 + 235 *　Block Num) / fcpu + 8798 µs
EEL_CMD_FORMAT	(67102 + 288981 *　Block Num) / fcpu + (266627 *　Block Num) µs	(67102 + 256218 * Block Num) / fcpu + (303359 * Block Num) µs
EEL_CMD_REFRESH 1. 正常に終了	5163828 / fcpu + 774424 µs	5072479 / fcpu + 1421917 µs
EEL_CMD_REFRESH 2. Block Num − 1まで REFRESH処理が失敗	(1554000 + 7538406 * (Block Num - 1)) / fcpu + (1548404 * (Block Num - 1)) µs	(1554000 + 7355752 * (Block Num - 1)) / fcpu + (2842866 * (Block Num - 1)) µs
EEL_CMD_VERIFY	30869 / fcpu + 4126 µs	19605 / fcpu + 29754 µs
EEL_CMD_WRITE	303387 / fcpu + 111858 µs	289240 / fcpu + 253342 µs
EEL_CMD_READ	5102 / fcpu µs	5102 / fcpu µs
EEL_CMD_SHUTDOWN	219 / fcpu µs	219 / fcpu µs

備考.　fcpu　：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

6. 2　ソフトウェア・リソース

EEL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、EELはFDLを使用するため、FDLで使用する領域も別途必要となります。

表6－2 、6－3に、必要となるソフトウェア・リソースの一覧[注1,2]、図6－2、6－3にRAMの配置イメージ例を示します。

★　表 6－3　EEPROMエミュレーション・ライブラリ Pack02 Ver.1.01 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 Pack02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 Pack02 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	80	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 ～ 255	
リクエスト・ストラクチャー	5	
SADDR RAM ワーク・エリア	SADDR : 3 (fdl:2) (eel:1)	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	3400 (fdl:600) (eel:2800)	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	3~66	
固定パラメータ領域 (デフォルト値)	14 (fdl:10) (eel:2)	
EEPROMエミュレーション・ブロック	3ブロック以上 (3kByte ～ 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。

注2　EEL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、EEL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。EEL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。

注3　データ・バッファは、EEL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 6－4　EEL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
FDL_Init	0	EEL_Close	0
FDL_Open	0	EEL_Execute[注]	0 ～ 255
FDL_Close	0	EEL_Handler[注]	0 ～ 255
EEL_Init	0	EEL_GetSpace	2
EEL_Open	0	EEL_GetVersionString	0

注　別途リクエストストラクチャー(5Byte)の領域を使用します。

図 6-2　RAMの配置イメージ例1 / セルフRAM有り（RL78/G13：RAM 4KB/ROM 64KB製品）

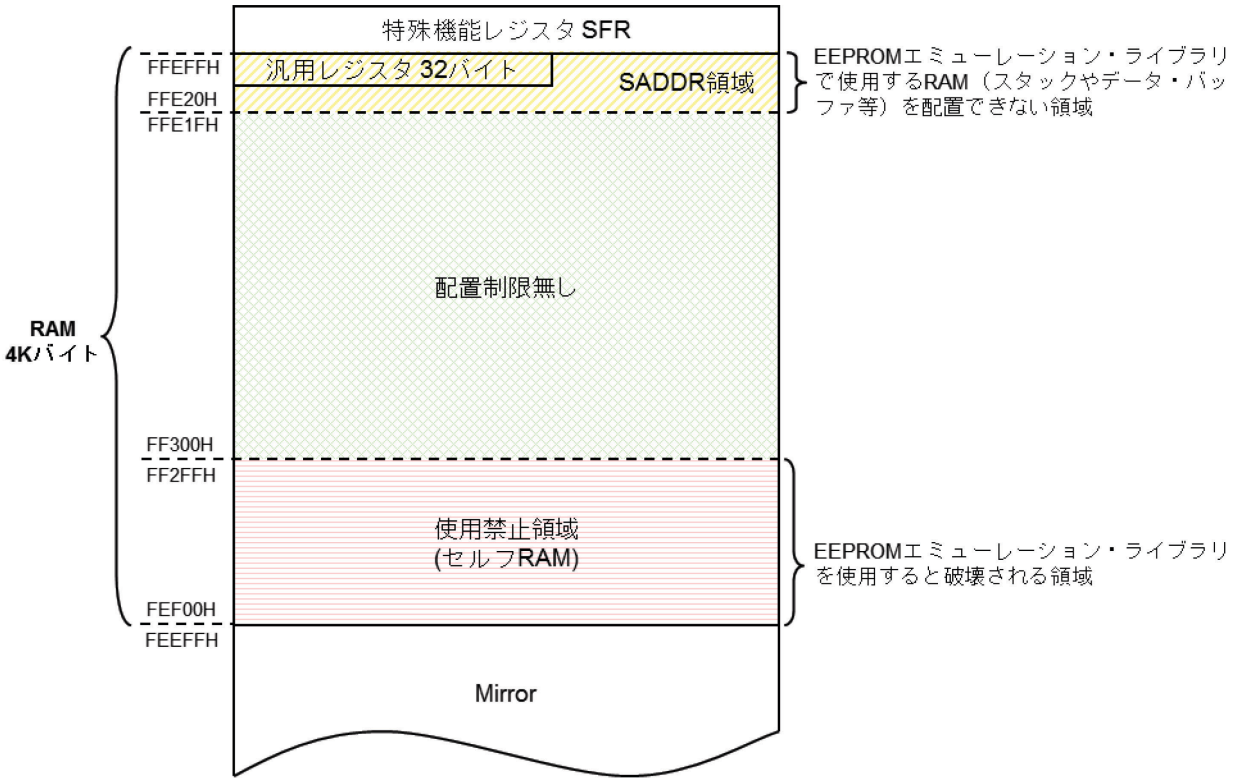
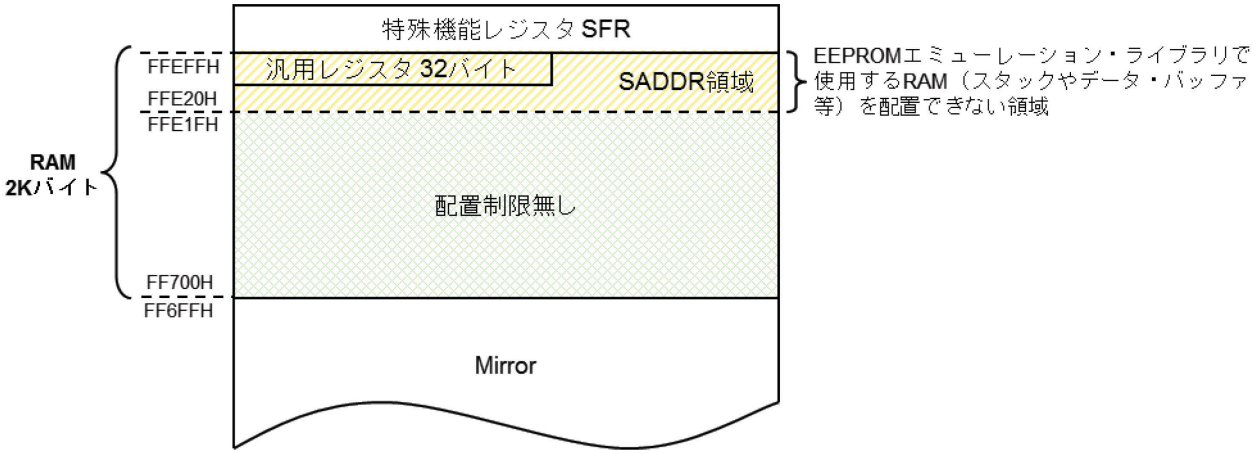


図 6-3　RAMの配置イメージ例2 / セルフRAM無し（RL78/G13：RAM 2KB/ROM 32KB製品）



6. 2. 1　セクション

EEL/FDLでは使用する関数、定数、変数が指定されたセクションに割り当てられています。

以下がEEL/FDLによって定義されているセクションです。

表 6－5　　EEL/FDLが使用するセクション

セクション名	説明
FDL_CODE	FDLのコードセクションです。FDLのプログラムが配置されています
FDL_SDAT	FDLの変数データセクションです。FDL内部において使用される変数データが配置されています ショート・アドレッシングRAM領域に配置してください
FDL_CNST	FDLの定数データセクションです。FDLにおいて使用される定数データが配置されています
EEL_CODE	EELのコードセクションです。EELのプログラムが配置されています
EEL_SDAT	EELの変数データセクションです。EEL内部において使用される変数データが配置されています ショート・アドレッシングRAM領域に配置してください
EEL_CNST	EELの定数データセクションです。EELにおいて使用される定数データが配置されています

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（ページ)	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2章 EEPROMエミュレーション		
p.3	2.2 機能概要に関する記述を変更	(c)
p.7	表2-1 説明に関する記述を訂正	(a)
p.7	図2-4 説明に関する記述を変更	(c)
p.7	表2-2 説明に関する記述を変更	(c)
第3章 EEL機能		
P.13	図3-1 注意2の記述を変更	(c)
P.17	終了状態確認の項目番号を訂正	(a)
P.18	説明に関する記述を変更	(c)
第4章 EEPROMエミュレーションの使用方法		
P.21	説明に関する記述を訂正	(a)
第5章 ユーザ・インタフェース		
P.36	説明に関する記述を変更	(c)
P.39	説明に関する記述を変更	(c)
第6章 ソフトウェア・リソースと処理時間		
p.43	表6-1 項目を変更	(c)
p.43	表6-2 項目を変更	(c)
p.44	表6-3 表題、項目を変更	(c)
p.44	表6-3 セルフRAMの使用領域に関する記述を変更	(c)
p.44	表6-3 注1でお問合せに関する記述を変更	(c)
p.44	表6-3 注2の記載を削除	(c)
p.44	表6-3 注4の記載を削除	(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.1.00	新規作成	全般

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 Pack02

発行年月日　　2016 年 2 月 29 日　Rev.1.01

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

R01US0068JJ0101