お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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。
5. 輸出に際しては、「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。本資料に記載されている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の目的で使用しないでください。また、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に使用することができません。
6. 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。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されてい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生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   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、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社が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。

H8S/2215 グループ

0.35μm F-ZTAT ソフトウェア ECC 書き込み

目次

1. はじめに 2
2. ソフトウェアECCの概要 3
2.1 ソフトウェアECCの目的 3
2.2 ソフトウェアECCの機能 3
3. ソフトウェアECCのアルゴリズム 5
4. サンプルプログラム 7
4.1 プログラムの構成と関数仕様 7
4.2 使用例 12
4.2.1 CreateFlashToUser( )関数の使用例 12
4.2.2 CreateUserToFlash ( )関数の使用例 13
4.3 ユーザデータの扱い 14
4.4 ソースプログラムリスト 15
5. 注意事項 21

1. はじめに

F-ZTAT* マイコンは基板実装後に書き換え可能なフラッシュメモリを内蔵しています。内蔵フラッシュメモリは通常その書き換え回数が100回まで保証されています。本アプリケーションノートは、内蔵フラッシュメモリをデータ領域として使用する場合に限って書き換え回数を10000回まで保証するための方法を説明しています。お客様のシステムにおいて、0.35μmF-ZTAT マイコンの内蔵フラッシュメモリの一部をデータ領域として使用され、その書き換え回数が100回を越えるような使用が必要となるシステムの開発に際して、ご参考として役立てていただけるようにまとめました。本アプリケーションノートはサンプルを説明したものであり、実際に本内容をご使用になる場合は必ず動作確認の上、ご使用くださいますようお願いいたします。

【注】* F-ZTAT（Flexible Zero Turn Around Time）は（株）ルネサステクノロジの商標です。

本アプリケーションノートで対象とするF-ZTAT マイコンの内蔵フラッシュメモリの使用例を図1.1に示します。

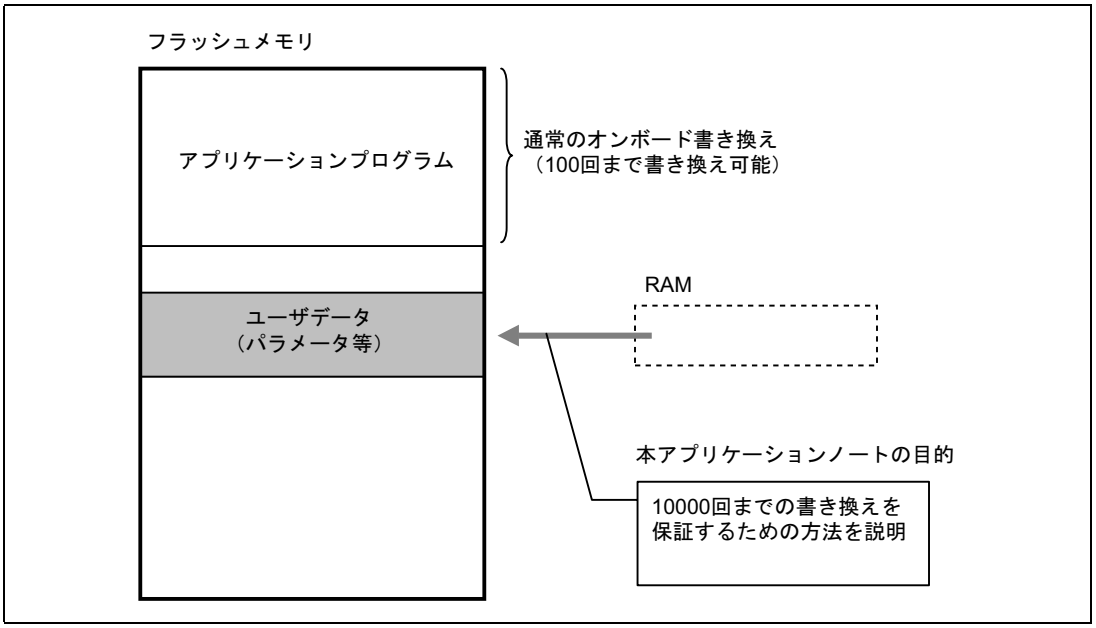


図 1.1 フラッシュメモリの使用例

2. ソフトウェア ECC の概要

2.1 ソフトウェア ECC の目的

0.35μmF-ZTATに内蔵されているフラッシュメモリは通常の書き換え回数が100回までと制限されています。この書き換え回数の制限はフラッシュメモリの保持特性（いわゆるリテンション）による制限です。一般的にフラッシュメモリは書き込み／消去回数（W/Eサイクル）が増えるとデータの保持特性が劣化しますが、この劣化はフラッシュメモリのメモリセルのすべてのビットに対して発生するわけではありません。フラッシュメモリセル単体のリテンションによるフェイル率は非常に小さいものです。
そこで、フラッシュメモリへの書き込みデータに対してエラー訂正コード（ECC：エラーコレクションコード）を付加して、書き込みデータ中のビットエラーの検出とその訂正を行うことができるようにすることにより、フラッシュメモリの書き換え回数制限を大幅に緩和することができます。ECCの付加およびビットエラーの検出と訂正はいずれもソフトウェアによって実現します。このソフトウェアECCによりフラッシュメモリの書き換え回数を10000回まで保証するものです。ただし、書き換え回数を10000回まで保証する領域は8kバイトまでとなります。この方法を適用するにあたっては本アプリケーションノートの記載内容に十分ご注意ください。

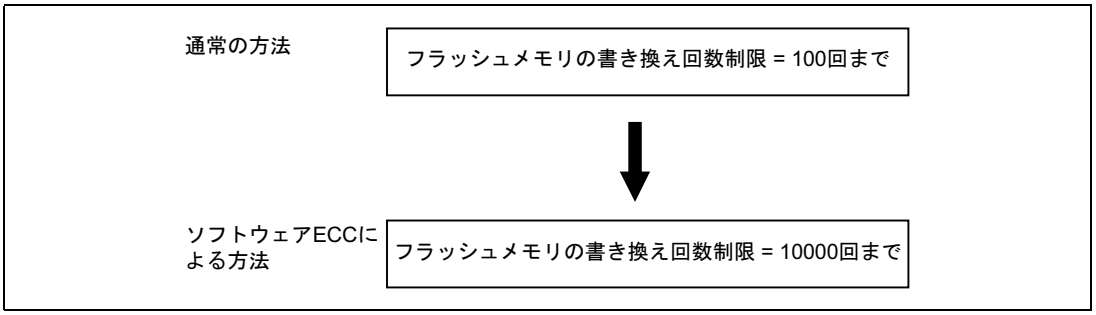


図 2.1 ソフトウェア ECC によるフラッシュメモリ書き換え回数の保証

2.2 ソフトウェア ECC の機能

ソフトウェアECCによるエラー訂正（エラーコレクション）機能における訂正レートはユーザデータ32ビットに対して 1 ビットのエラー検出およびエラー訂正です。32ビットユーザデータに対するECCは 6 ビット必要となりますが、バイト境界を考慮して32ビットユーザデータに対するECCは 1 バイト（下位 6 ビットが有効）とします。ECCのビットエラーを含めると訂正レートは38ビットに対して 1 ビットのエラー検出およびエラー訂正といえます。ユーザデータとECCの配置はフラッシュメモリのメモリマットの物理的配置を考慮する必要があることから、図2.2に示すフォーマットにしてください。このフォーマットでない場合には規定の書き換え回数が保証されません。

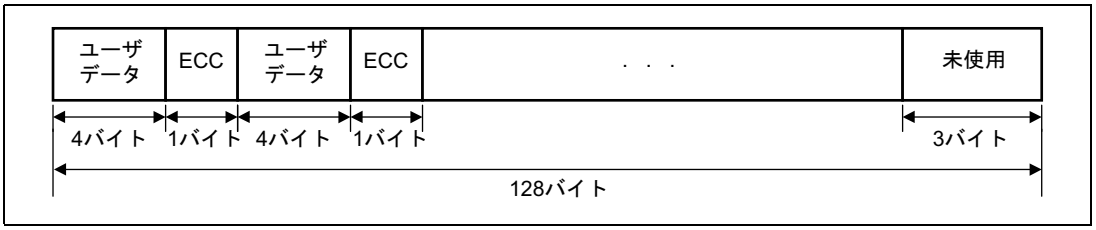


図 2.2 フラッシュメモリへの書き込みデータフォーマット

図2.2に示すように、エラー訂正を行う4バイト（32ビット）のユーザデータとそれに対する 1 バイトのECCを連続して配置する必要があります。ECCはソフトウェアによって生成します。フラッシュメモリの書き込み単位は128バイトであり、その単位でのユーザデータは最大100バイトとなります。フラッシュメモリへは図2.2のフォーマットで書き込みを行うことが必須です。また、フラッシュメモリから読み込んだデータ（書き込み時と同一フォーマット）に対して、ソフトウェアECC機能によりつぎのいずれかのエラー検出およびエラー訂正が可能です。

- 4バイトのユーザデータに対して 1 ビットのエラー検出およびそのビットのエラー訂正
- 1バイトのECCに対して 1 ビットのエラー検出およびそのビットのエラー訂正

ソフトウェアECC機能の位置付けを図2.3に示します。本アプリケーションノートではフラッシュメモリへ書き込み／消去そのものについては対象としていませんので、実際にフラッシュメモリへの消去・書き込みを実施するにあたっては各製品のハードウェアマニュアルに記載の手順に従ってください。

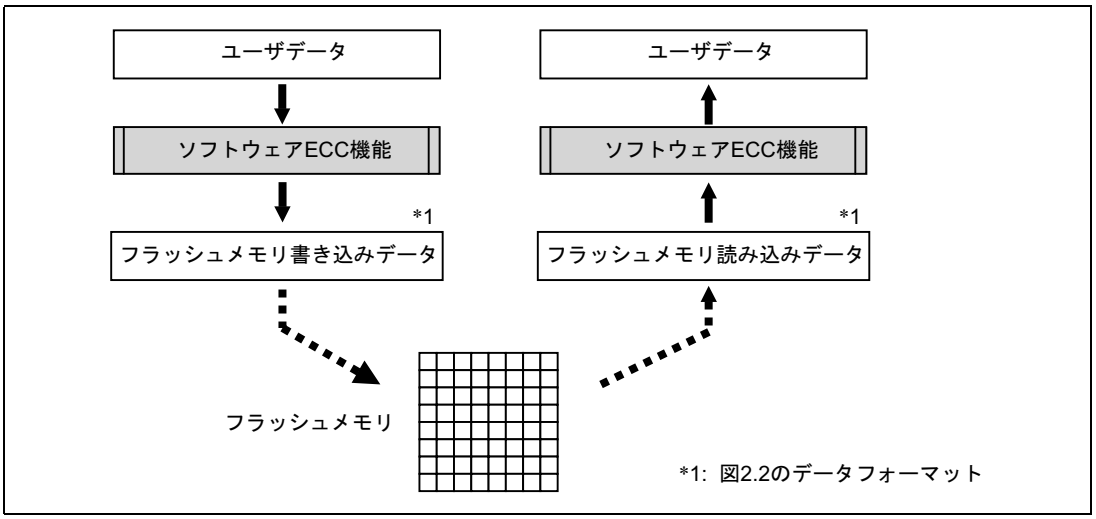


図 2.3　ソフトウェア ECC 機能の位置付け

3. ソフトウェア ECC のアルゴリズム

ソフトウェアECCは単一誤り訂正符号であるハミング符号を用います。ハミング符号における情報ビットと検査ビットの関係を図3.1に示します。

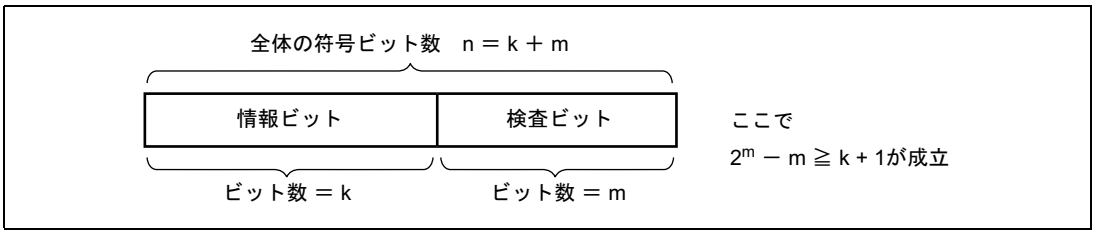


図 3.1　(n,k)ハミング符号の構成

ハミング符号において情報ビット数=32とした場合、m=6とすると$2^m-m \geqq k+1$が成立します。32ビットの情報ビットに対して6ビットの検査ビットを付加することで単一誤り訂正が可能になります。つまり(38,32)ハミング符合を適用することになります。情報ビットが32ビットのユーザデータに対応し、検査ビットが6ビットのECCに対応します。ECCは6ビットですがバイト境界を考慮して1バイト（下位6ビットのみ有効）にして適用します。

ここでは説明を簡単にするために、(7,4)ハミング符合を使って例を示します。(7,4)ハミング符合は図3.2で示されます。また、検査ビット生成のための定義（パリティ）を表3.1のようにします。

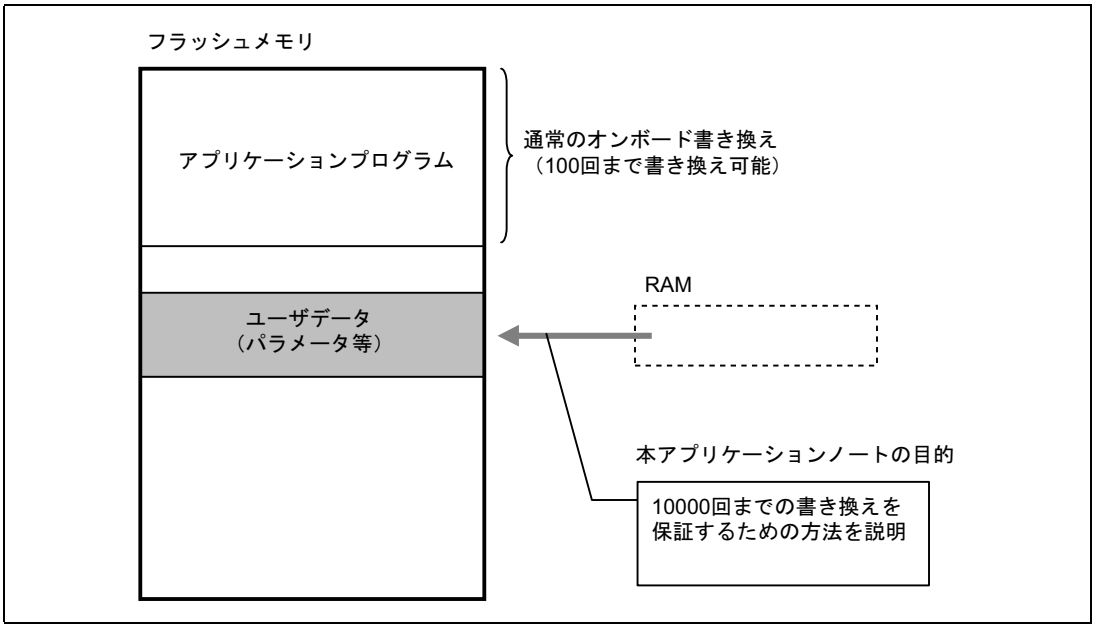


図 3.2　(7,4)ハミング符号の構成

表 3.1 パリティ定義

誤りの位置	S_1	S_2	S_3	16 進表記
x_1	1	1	1	0x07
x_2	1	1	0	0x06
x_3	1	0	1	0x05
x_4	0	1	1	0x03
c_1	1	0	0	0x04
c_2	0	1	0	0x02
c_3	0	0	1	0x01

情報ビットがB'1101とします。情報ビットの各ビット位置に対応するパリティをXOR（排他的論理和）します。x_1=1,x_2=1,x_3=0,x_4=1なので0x07,0x06,0x03をXORすると0x02になります。この値を検査ビットとしますので、c_1=0,c_2=1,c_3=0（これをCとします）にします。

- 情報ビットがB'1111に変化した場合（x_3が0→1に変化）
  B'1111に対する検査ビットを求めるとc_1'=1,c_2'=1,c_3'=1（これをC'とします）になります。
  C≠C'なのでビット変化が発生したことがわかります。ここでCとC'のXORをとった値をsとすると、
  s_1=1,s_2=0,s_3=1になります。表3.1からこの値を見つけるとx_3がビット誤りを発生したことがわかりますので情報ビットをB'1101に訂正します。

- 情報ビットがB'1001に変化した場合（x_2が1→0に変化）
  B'1001に対する検査ビットを求めるとc_1'=1,c_2'=0,c_3'=0（これをC'とします）になります。
  C≠C'なのでビット変化が発生したことがわかります。ここでCとC'のXORをとった値をsとすると、
  s_1=1,s_2=1,s_3=0になります。表3.1からこの値を見つけるとx_2がビット誤りを発生したことがわかりますので情報ビットをB'1101に訂正します。

- 検査ビットがB'110に変化した場合（c_1が0→1に変化）
  B'1101に対する検査ビットを求めるとc_1'=0,c_2'=1,c_3'=0（これをC'とします）になります。
  Cはc_1=1,c_2=1,c_3=0に変化しているのでC≠C'となりビット変化が発生したことがわかります。ここでCとC'のXORをとった値をsとすると、s_1=1,s_2=0,s_3=0になります。表3.1からこの値を見つけるとc_1がビット誤りを発生したことがわかりますので検査ビットをB'010に訂正します。

- 検査ビットがB'000に変化した場合（c_2が1→0に変化）
  B'1101に対する検査ビットを求めるとc_1'=0,c_2'=1,c_3'=0（これをC'とします）になります。
  Cはc_1=0,c_2=0,c_3=0に変化しているのでC≠C'となりビット変化が発生したことがわかります。ここでCとC'のXORをとった値をsとすると、s_1=0,s_2=1,s_3=0になります。表3.1からこの値を見つけるとc_2がビット誤りを発生したことがわかりますので検査ビットをB'010に訂正します。

- 情報ビットおよび検査ビットが変化しなかった場合
  同様の手順でC'を求めるとC= C'なのでビット誤りが発生しなかったことがわかります。

4. サンプルプログラム

本章ではソフトウェアECCを実現するサンプルプログラムについて説明します。サンプルプログラムは動作確認済ですが、サンプルプログラムをユーザシステムに組み込んで使用される場合には、動作環境の相違も考えられますので改めて動作確認していただく必要があります。

4.1 プログラムの構成と関数仕様

サンプルプログラムは表4.1に示す2つのファイルで構成しています。

表 4.1 サンプルプログラムの構成ファイル

No.	ファイル名	内　　容
1	FlashECC.h	関数プロトタイプ宣言および関数の戻り値となる定数を定義したヘッダファイル
2	FlashECC.c	ソフトウェア ECC を実現する関数（4 つの関数で構成）

各関数の機能概要を表4.2に示しますが詳細は関数仕様をご参照ください。また関数の階層構造を図4.1に示します。

表 4.2　各関数の機能概要

No.	関　数　名	機　　能　　概　　要
1	CreateUserToFlash( )	ユーザデータ（最大で 100 バイト）から 4 バイトごとに 1 バイトの ECC を付加したフラッシュメモリへの書き込みデータ *1) を作成する *1) 図 2.2 のフォーマットのデータ
2	CreateFlashToUser( )	フラッシュメモリからの読み込みデータ *2) からユーザデータ（最大で 100 バイト）を作成する。読み込みデータ中に 1 ビットのエラーが検出された場合はそのエラー訂正を行う *2) 図 2.2 のフォーマットのデータ
3	GenerateECC( )	４バイトのユーザデータに対して 1 バイトの ECC を算出する
4	CheckECC( )	４バイトのユーザデータおよび 1 バイトの ECC に対して 1 ビットのエラーの検出を行う １ビットのエラーが検出された場合には当該ビットを訂正する もし２ビット以上のエラーがあった場合にはエラー検出はできるがエラー訂正はできない

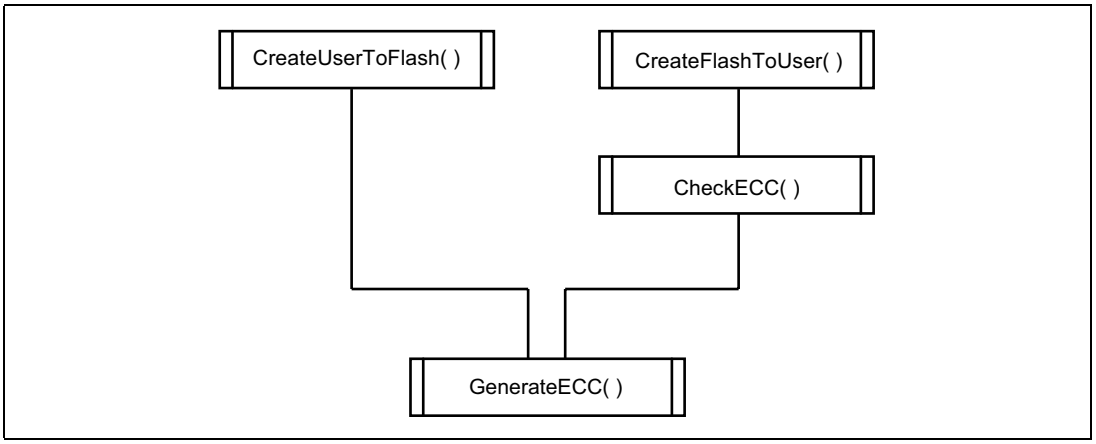
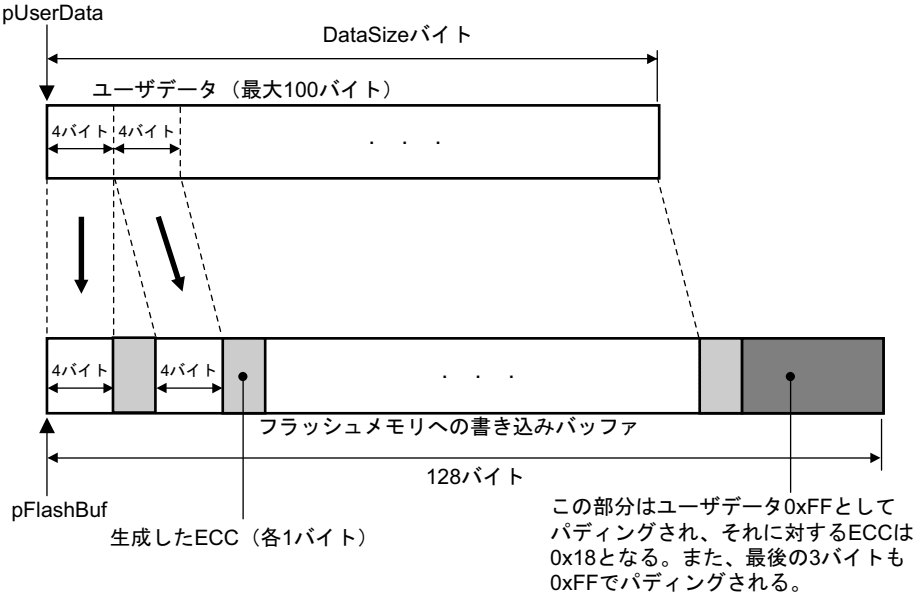
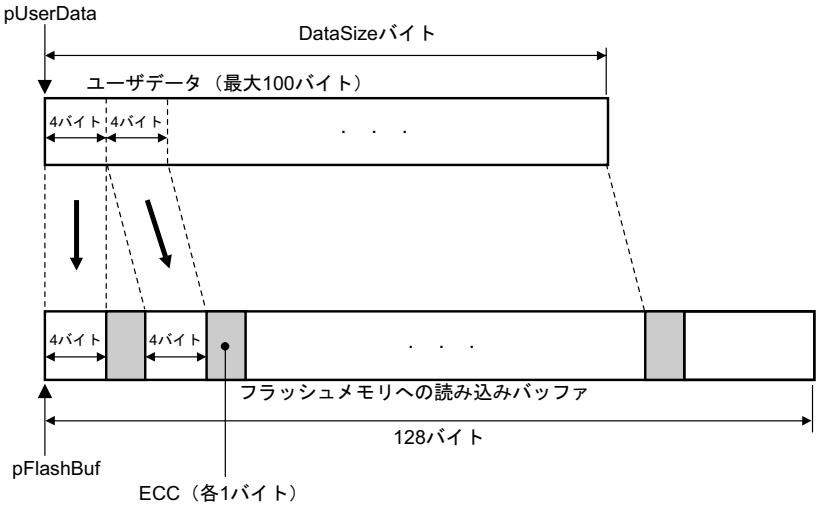


図 4.1　関数の階層構造

次頁以降に各関数の関数仕様を記述します。

フラッシュメモリへの書き込みデータ作成	
void　CreateUserToFlash(unsigned char *pUserData, unsigned char *pFlashBuf, unsigned char DataSize)	
機　能	ユーザデータから 4 バイトごとに 1 バイトの ECC を付加したフラッシュメモリへの書き込みデータ *1) を作成する。 *1) 図 2.2 に示したフォーマット
引　数	pUserData　ユーザデータへのポインタ ユーザデータは連続した DataSize バイト分の領域であること pFlashBuf　フラッシュメモリへの書き込みバッファへのポインタ バッファは 128 バイトの領域があること DataSize　ユーザデータのバイト数、1≦DataSize≦100 であること
戻り値	なし
説　明	 ユーザデータ長が 4 の倍数バイトでない場合は、最後のデータは 4 バイトに満たない分を 0xFF で埋めたデータとして扱う。書き換え回数 10000 回を保証するのは 8k バイトまでとなるため、128 バイト単位の書き込みバッファ×64 個分（＝8k バイト）がソフトウェア ECC を適用可能である。

フラッシュメモリ読み込みデータからのユーザデータ作成	
unsigned char　CreateFlashToUser (unsigned char *pUserData, unsigned char *pFlashBuf, unsigned char DataSize)	
機　能	フラッシュメモリからの読み込みデータ *1) からユーザデータを作成する。 読み込みデータ中の 4 バイト単位のユーザデータおよびその ECC ごとにエラー訂正処理（1 ビットのエラー検出とその訂正）を行う。エラー訂正処理はフラッシュメモリからの読み込みデータに対して行い、エラー訂正した後にユーザデータを作成する。 *1) 図 2.2 に示したフォーマット
引　数	pUserData　ユーザデータへのポインタ ユーザデータは連続した DataSize バイト分の領域があること pFlashBuf　フラッシュメモリからの読み込みバッファへのポインタ バッファは 128 バイトの領域があること DataSize　ユーザデータのバイト数、1≦DataSize≦100 であること
戻り値	ビットエラー未検出であった場合　：ECC_NOERROR ビットエラー検出とその訂正が行われた場合：ECC_REPAIRED 複数ビットのエラーが検出された場合　：ECC_FAILED
説　明	 読み込みバッファ中の 4 バイト単位のユーザデータ中に 1 ビットのエラーがあった場合（4 バイトについて 1 ビットのエラー）または ECC1 バイト中に 1 ビットのエラーが検出された場合には、そのエラー訂正をフラッシュメモリからの読み込みバッファ上で行った後にユーザデータを作成する。したがってフラッシュメモリからの読み込みバッファおよびユーザデータのいずれもエラー訂正された状態となる。このケースでは戻り値が ECC_REPAIRED となる。 もし、読み込みバッファ中の 4 バイト単位のユーザデータおよびその ECC 1 バイトを含めた 5 バイト中に 2 ビット以上のエラーがあった場合には、その時点で処理を中止する。このケースでは戻り値が ECC_FAILED となり、 正しいユーザデータは作成されない。ただし、フラッシュメモリの書き換え回数制限内では通常このエラーは発生しない。読み込みバッファ中にビットエラーが全くなかった場合には戻り値は ECC__NOERROR となる。 ビットエラーの検出は 4 バイトのユーザデータとその ECC 1 バイトを含めた 5 バイト（ECC の上位 2 ビットは無効であるので実際に有効なのは 38 ビットになる）ごとに行われ、1 ビットのエラー検出とその訂正が可能である。

4 バイトユーザデータに対する ECC 算出	
unsigned char　GenerateECC(unsigned long *pDataItem)	
機　能	4 バイトのユーザデータに対して 1 バイトの ECC（エラーコレクションコード）を算出する。
引　数	pDataItem　ユーザデータに対するポインタ ユーザデータは 4 バイトであること
戻り値	ECC
説　明	引数で渡された 4 バイトのユーザデータに対して ECC を算出する。算出した ECC は戻り値で返される。

4 バイトユーザデータに対するビット訂正	
unsigned char　CheckECC(unsigned long *pDataItem, unsigned char *pECC)	
機　能	4 バイトのユーザデータとそれに対する 1 バイトの ECC（エラーコレクションコード）からビットエラーの検出とそのエラー訂正を行う。
引　数	pDataItem　ユーザデータに対するポインタ ユーザデータは 4 バイトであること pECC　ECC に対するポインタ ECC は 1 バイトであること
戻り値	ビットエラー未検出であった場合　：ECC_NOERROR ビットエラー検出とその訂正が行われた場合：ECC_REPAIRED 複数ビットのエラーが検出された場合　：ECC_FAILED
説　明	引数で渡された 4 バイトのユーザデータとそれに対する ECC からビットエラーの有無を検出する。ユーザデータまたは ECC に 1 ビットのビットエラーが検出された場合にはこのビットエラーを訂正する。訂正は引数で渡された領域に対して行うので、ビットエラー訂正が行われた場合には pDataItem または pECC で示される領域のデータが更新される。このケースでは戻り値が ECC_REPAIRED となる。 もし、ユーザデータと ECC を含めた 5 バイト中（ECC の上位 2 ビットは無効であるので実際に有効なのは 38 ビットになる）に 2 ビット以上のエラーがあった場合にはビットエラー訂正は行われない。このケースでは戻り値が ECC_FAILED となる。ただし、フラッシュメモリの書き換え回数制限内では通常このエラーは発生しない。 ユーザデータおよび ECC にビットエラーが全くなかった場合には戻り値は ECC_NOERROR となる。

4.2 使用例

サンプルプログラムではCreateFlashToUser( )関数とCreateUserToFlash ( )関数がユーザシステムとのインタフェース関数となります。これらの関数の簡単な使用例を示します。

4.2.1 CreateFlashToUser( )関数の使用例

```
#include   <string.h>
#include   “FlashECC.h”

#define    Param_Adrs  0x0030000              // フラッシュメモリ上のユーザデータアドレス
#define    Param_Size  (unsigned char)92      // フラッシュメモリ上のユーザデータバイト数
unsigned char      FlashBuf[128];             // フラッシュメモリからの読み込みバッファ

unsigned char      bUserParam1[11];           // これらの変数を RAM 上のユーザデータとす
る
unsigned char      bReserved;                 // 境界調整のためのダミー変数も含めておく
unsigned long      lUserParam2[20];           // 全体で 92 バイトの領域となる

void  sample1(void)
{
unsigned char   *pUserData;                   // ユーザデータへのポインタ
unsigned char    bRtnCode;                    // 関数の戻り値

                                              // フラッシュメモリからユーザデータ読み込み
  strncpy((const char *)FlashBuf,(const char *) Param_Adrs,(size_t) Param_Size);
                                              // 読み込みデータからユーザデータ作成
bRtnCode = CreateFlashToUser(bUserParam1, FlashBuf, (unsigned char)Param_Size);
if(bRtnCode == ECC_NOERROR)
{
                                              // ビットエラー検出なし
}
else if(bRtnCode == ECC_REPAIRED)
{
                                              // ビットエラー検出および訂正あり
}
else
{
                                              // 複数ビットエラー検出
}
     .
     .
     .
}
```

4.2.2 CreateUserToFlash ( )関数の使用例

```
#include   “FlashECC.h”

#define    Param_Size  (unsigned char)92      // フラッシュメモリ上ユーザデータバイト数
unsigned char      FlashBuf[128];             // フラッシュメモリへの書き込みバッファ

unsigned char      bUserParam1[11];           // これらの変数を RAM 上のユーザデータとす
る
unsigned char      bReserved;                 // 境界調整のためのダミー変数も含めておく
unsigned long      lUserParam2[20];           // 全体で 92 バイトの領域となる

void  sample2(void)
{
  unsigned char    *pUserData;                // ユーザデータへのポインタ

      .
      .       (RAM 上のユーザデータ設定)
      .
                                              // ユーザデータから書き込みデータ作成
CreateUserToFlash (bUserParam1, FlashBuf, (unsigned char)Param_Size);

      .
      .       (FlashBuf の 128 バイトをフラッシュメモリに書き込み)
      .
}
```

【注意】　実際にはフラッシュ書き込み処理は RAM 上で実行する必要があります。

4.3 ユーザデータの扱い

フラッシュメモリへの書き込み単位は128バイトであり、このうち100バイトをユーザデータとして使用することができますが、ユーザデータが100バイトを越える場合には100バイト単位に分割する必要があります。例を示します。

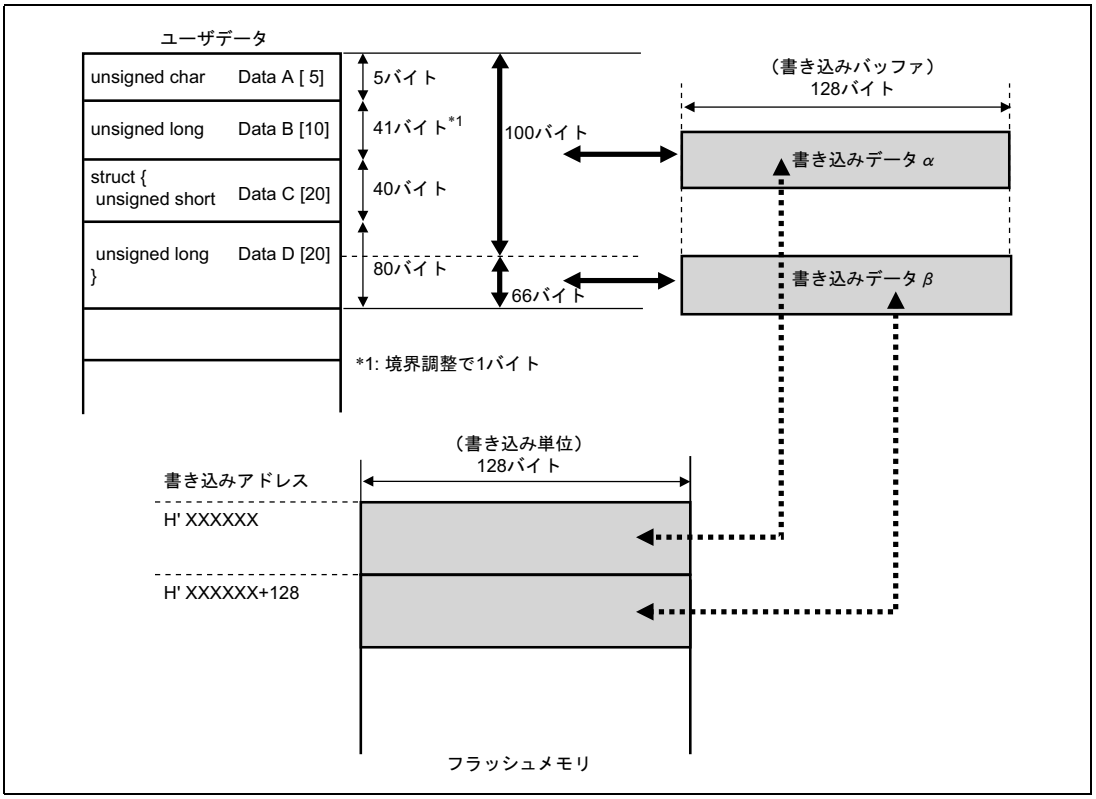


図 4.2 ユーザデータ扱い例

4.4 ソースプログラムリスト

```
/************************************************************************/
/*                                                                      */
/*  FILE       :FlashECC.h                                              */
/*  DATE       :Sep 1, 2002                                             */
/*  DESCRIPTION :Header file of software ECC functions for Flash memory */
/*  CPU TYPE   :H8S/2215                                                */
/*                                                                      */
/************************************************************************/
#ifndef    _FLASHECC_H_
#define    _FLASHECC_H_

/************************************************************************/
/*                                                                      */
/*  Function prototypes                                                 */
/*                                                                      */
/************************************************************************/
void          CreateUserToFlash(  unsigned char *pUserData,
                                  unsigned char *pFlashBuf,
                                  unsigned char  DataSize);

unsigned char   CreateFlashToUser(unsigned char *pUserData,
                                  unsigned char *pFlashBuf,
                                  unsigned char  DataSize);

unsigned char   GenerateECC(unsigned long *pDataItem);

unsigned char   CheckECC   (unsigned long *pDataItem, unsigned char *pECC);

/************************************************************************/
/*                                                                      */
/*  Return-values of functions                                          */
/*                                                                      */
/************************************************************************/
enum
{
   ECC_NOERROR ,                /* No error was detected.               */
   ECC_REPAIRED,                /* One bit repair was completed.        */
   ECC_FAILED                   /* Multiple bits error was detected.    */
};

#endif                          /*_FLASHECC_H_                          */
```

```
/*****************************************************************************/
/*                                                                           */
/*  FILE           :FlashECC.c                                               */
/*  DATE           :Sep 1, 2002                                              */
/*  DESCRIPTION    :Software ECC functions for Flash memory                  */
/*  CPU TYPE       :H8S/2215                                                 */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
#include   "FlashECC.h"

/*****************************************************************************/
/*                                                                           */
/*  The T-table for ECC generation                                           */
/*                                                                           */
/*****************************************************************************/
static const unsigned char  T[38] =
{                                /* values for the 32-bit data item          */
   0x03, 0x05, 0x06, 0x07, 0x09, 0x0A, 0x0B, 0x0C,
   0x0D, 0x0E, 0x0F, 0x11, 0x12, 0x13, 0x14, 0x15,
   0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D,
   0x1E, 0x1F, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26,
                                 /* values for the 6-bit ECC item            */
   0x01, 0x02, 0x04, 0x08, 0x10, 0x20
};

/*****************************************************************************/
/*                                                                           */
/*  C Prototype: void  CreateUserToFlash(unsigned char *pUserData,           */
/*                                       unsigned char *pFlashBuf,           */
/*                                       unsigned char  DataSize);           */
/*                                                                           */
/*  Function   : This routine makes the data to be write into Flash memory   */
/*               including ECC values from user's data.                      */
/*                                                                           */
/*  Parameters: pUserData is a pointer to user's data. Maximal length of     */
/*               User's data must be 100 bytes.                               */
/*               pFlahBuf is a pointer to the data area to be write into     */
/*               Flash memory. The data area must be in RAM and needs 128    */
/*               bytes area.                                                 */
/*               DataSize is a byte-length of user's data. Its value may     */
/*               not exceed 100.                                             */
/*                                                                           */
/*  Returns    : None.                                                       */
/*                                                                           */
/*****************************************************************************/
```

```
void  CreateUserToFlash(   unsigned char *pUserData,
                           unsigned char *pFlashBuf,
                           unsigned char  DataSize)
{
   unsigned char   i;                          /* a loop counter                */
   unsigned char   ecc;                        /* an ECC value                  */
   unsigned char   len;                        /* a temporary length            */
   unsigned char   fil;                        /* byte-length for fill up       */
   union                                       /* a data with zero padding      */
   {
      unsigned long   lword;
      unsigned char   bytes[4];
   }  upad;
   /*-----------------------------------------*/
   /* Initialize local variables               */
   /*-----------------------------------------*/
   upad.lword = 0xFFFFFFFF;
   fil = 128 - (((DataSize + 3) >> 2) * 5);
   /*-----------------------------------------*/
   /* Generate an ECC and a Flash data buffer  */
   /*-----------------------------------------*/
   while(0 < DataSize)
   {
      if(4 <= DataSize)   len = 4;
      else                len = DataSize;
      for(i=0; i<len; i++)  upad.bytes[i] = *(pUserData++);
      ecc = GenerateECC((unsigned long *)(&upad));
      for(i=0; i<4; i++)     *(pFlashBuf++) = upad.bytes[i];
      *(pFlashBuf++) = ecc;
      DataSize -= len;
   }
   /*-----------------------------------------*/
   /* Fill up a Flash data buffer              */
   /*-----------------------------------------*/
   for(i=0; i<fil;i++)
   {
      if((i % 5) == 4)    *(pFlashBuf++) = 0x18;
      else                *(pFlashBuf++) = 0xFF;
   }
   return;
}
```

```
/*****************************************************************************/
/*                                                                           */
/*  C Prototype: unsigned char  CreateFlashToUser(unsigned char *pUserData,  */
/*                                                unsigned char *pFlashBuf,  */
/*                                                unsigned char  DataSize);  */
/*                                                                           */
/*  Function   : This routine makes user's data from the data including ECC  */
/*               values which are read from Flash memory into RAM.           */
/*                                                                    */
/*  Parameters : pUserData is a pointer to user's data. Maximal length of   */
/*               User's data must be 100 bytes.                              */
/*               pFlahBuf is a pointer to the data area which are read from  */
/*               Flash memory into RAM. The data area must be 128 bytes.     */
/*               DataSize is a byte-length of user's data. Its value may     */
/*               not exceed 100.                                             */
/*                                                                           */
/*  Returns    : If no error was detected, return-value will be ECC_NOERROR.  */
/*               If one bit repair for user's data or an ECC value was       */
/*               detected, return-value will be ECC_REPAIRED.                */
/*               If multiple bits error was detected, return-value will be   */
/*               ECC_FAILED.                                                 */
/*                                                                           */
/*****************************************************************************/
unsigned char  CreateFlashToUser( unsigned char *pUserData,
                                  unsigned char *pFlashBuf,
                                  unsigned char  DataSize)
{
    unsigned char   i;                      /* a loop counter                */
    unsigned char   ecc;                    /* an ECC value                  */
    unsigned char   len;                    /* a temporary length            */
    unsigned char   rtn;                    /* a return-value                */
    union                                   /* a data with zero padding      */
    {
        unsigned long   lword;
        unsigned char   bytes[4];
    }  upad;
    /*---------------------------------------*/
    /* Initialize local variables            */
    /*---------------------------------------*/
    upad.lword = 0xFFFFFFFF;
    rtn = ECC_NOERROR;
    /*---------------------------------------*/
    /* Generate user's data from a Flash buffer */
    /*---------------------------------------*/
    while(0 < DataSize)
    {
        if(4 <= DataSize)   len = 4;
        else                len = DataSize;
        for(i=0; i<4; i++)  upad.bytes[i] = *(pFlashBuf++);
        switch(CheckECC((unsigned long *)(&upad),pFlashBuf))
        {
            case ECC_REPAIRED:
                for(i=0; i<4; i++)  *(pFlashBuf - 4 + i) = upad.bytes[i];
```

```
            rtn = ECC_REPAIRED;
            /*--- break statement no need --*/
         case ECC_NOERROR:
            for(i=0; i<len; i++)  *(pUserData++) = upad.bytes[i];
            pFlashBuf++;
            break;
         case ECC_FAILED:
            return(ECC_FAILED);
      }
      DataSize -= len;
   }
   return(rtn);
}

/*****************************************************************************/
/*                                                                           */
/*  C Prototype: unsigned char  GenerateECC(unsigned long *pDataItem)        */
/*                                                                           */
/*  Function   : This routine returns an 8-bit ECC value calculated from a   */
/*               supplied 32-bit data item.                                  */
/*                                                                           */
/*  Parameters : pDataItem is a pointer to 32-bit data value for which an    */
/*                ECC value is required.                                     */
/*                                                                           */
/*  Returns    : The return value is an 8-bit ECC value. Only lower 6-bit of  */
/*               an ECC are valid.                                           */
/*                                                                           */
/*****************************************************************************/
unsigned char  GenerateECC(unsigned long *pDataItem)
{
   unsigned char   i;                      /* a loop counter               */
   unsigned char   ecc;                    /* an ECC value                 */
   unsigned long   mask;                   /* a bit mask value             */
   /*--------------------------------------*/
   /* Initialize local variables           */
   /*--------------------------------------*/
   ecc  = 0x00;
   mask = 0x00000001;
   /*--------------------------------------*/
   /* Calculate for each bit of a 32-bit data  */
   /*--------------------------------------*/
   for(i=0; i<32; i++)
   {
      if((*pDataItem & mask) != 0)  ecc ^= T[i];
      mask <<= 1;
   }
   return(ecc);
}
```

```
/*****************************************************************************/
/*                                                                           */
/*  C Prototype: unsigned char  CheckECC(     unsigned long *pDataItem,      */
/*                                            unsigned char *pECC);          */
/*                                                                           */
/*  Function   : This routine checks the validity of a 32-bit data item and  */
/*               an 8-bit ECC value. If an error is detected, a repair is    */
/*               performed. The repair can be in operation for only one bit  */
/*               error.                                                      */
/*                                                                           */
/*  Parameters : pDataItem is a pointer to the 32-bit data value.            */
/*               pECC is a pointer to the 8-bit original ECC value.          */
/*                                                                           */
/*  Returns    : If no error was detected, return-value will be ECC_NOERROR. */
/*               If one bit repair for user's data or an ECC value was       */
/*               detected, return-value will be ECC_REPAIRED.                */
/*               If multiple bits error was detected, return-value will be   */
/*               ECC_FAILED.                                                 */
/*                                                                           */
/*****************************************************************************/
unsigned char  CheckECC(unsigned long *pDataItem,
                        unsigned char *pECC)
{
    unsigned char   i;                      /* a loop counter          */
    unsigned char   ecc;                    /* an ECC value            */
    unsigned long   mask;                   /* a bit mask value        */
    /*-----------------------------------------*/
    /* Initialize a local variable             */
    /*-----------------------------------------*/
    mask = 0x00000001;
    /*-----------------------------------------*/
    /* Check an ECC and repair a data an ECC   */
    /*-----------------------------------------*/
    ecc = GenerateECC(pDataItem);
    if(ecc != *pECC)
    {
        for(i=0; i<sizeof(T); i++)
        {
            if(T[i] == (ecc ^ *pECC))
            {
                if(32 <= i)  *pECC       ^= (unsigned char)mask;
                else         *pDataItem  ^= mask;
                return(ECC_REPAIRED);
            }
            if(i == 31)    mask   = 0x00000001;
            else           mask <<= 1;
        }
        return(ECC_FAILED);
    }
    else
    {
        return(ECC_NOERROR);
    }
}
```

5. 注意事項

ソフトウェアECCを適用するにあたっての注意事項を記述します。前章までの記述と重複している点もあります。

1. メモリマットの物理的配置からフラッシュメモリへの書き込みデータフォーマットは図2.2のフォーマットとしてください。
2. エラー訂正は32ビットのユーザデータに対して 1 ビットのエラー検出とその訂正です。32ビットのユーザデータに対して2ビット以上のエラーが起こった場合、エラー検出はできますがエラー訂正はできません。
3. フラッシュメモリの書き込み単位（128バイト）中にはユーザデータは100バイトのみ保持できます。
4. 書き換え回数10000回を保証するのは8kバイト（フラッシュメモリ上での配置は任意）のみです。ユーザデータおよびそのECCを含めて8kバイトですので、図2.2のデータフォーマットを64個分使用することができます。これ以外の領域については通常の書き換え回数100回までが保証されます。
5. ユーザデータは4バイトごとに分割されてフラッシュメモリに書き込まれることになります。ユーザデータは連続していることを前提としていますので、ユーザデータ中に境界調整による空き領域が発生しているケースではその領域について注意してください。
6. 4バイトごとにECCが挿入されるために、命令コードには使用できません。つまりECCが挿入されているフラッシュメモリ上の領域をメモリマッピングされたプログラムエリアとして使用することはできません。
7. ソフトウェアECCの処理時間によりフラッシュメモリ読み込み時間が問題となるようなシステムの場合はリセット時にフラッシュメモリ上のユーザデータをすべてRAM上に展開するなどの方策をおとりください。

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。