RX ファミリ

RAM ソフトエラー診断例

R01AN3361JJ0100
Rev.1.00
2016.08.10

要旨

本アプリケーションノートでは、2重化したRAMアクセス方法とビット演算によるSRAMのソフトエラー診断について説明します。

動作確認デバイス

この API は下記のデバイスをサポートしています。

- RX64M グループ
- RX71M グループ

本アプリケーションノートを他のルネサスマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 概要

近年、メモリ容量の増加やプロセスの微細化により、SRAMのソフトエラーが重要な問題となっています。ソフトエラーはSRAMのビットが反転させるエラーで、LSIのパッケージ内の不純物に含まれるα線や宇宙線として降り注ぐ中性子に起因する確率的に発生する現象です。

本アプリケーションノートでは、2重化したRAMアクセス方法（以下、ダブルRAM[1]）とビット演算によるSRAMのソフトエラー診断について説明します。ビット演算はユーザの安全データとデータフラッシュ上に格納された乱数（繰り返し、または、定数）のビットパターンで実行します。その後、ビット演算を施したデータをダブルRAM領域に書き込みます。ビット演算には排他的論理和を適用することで、ビットの0と1の数を均等化します。その結果、ビットが0または1に固定されたエラーの検出が容易になり、エラー検出率が向上します。また、ビットパターンを更新する場合、機能安全の観点からダブルRAMを使用します。

1.1 RAM ソフトエラー診断例

このサンプルプログラムはプロジェクト形式で提供しており、複数の FIT（Firmware Integration Technology）モジュールを使用した SRAM のソフトエラー診断例として使用できます。

1.2 関連ドキュメント

[1] Functional safety of electrical/electronic/programmable electronic safety-related systems, IEC61508, Edition 2.0, Apr, 2010

[2] Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, JESD89A, Oct 2006, JEDEC Solid State Technology Association.

[3] RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology, Rev.3.31, Document No. R01AN1685JJ0331, May 19, 2016

[4] RX Family Flash Module Using Firmware Integration Technology, Rev.1.60, Document No. R01AN2184EU0160, Nov 17, 2015

[5] RX ファミリ オープンソースFATファイルシステム M3S-TFAT-Tiny モジュール Firmware Integration Technology, Rev.3.02, Document No. R20AN0038EJ0302, Mar 01, 2015

[6] Renesas USB MCU USB Basic Host and Peripheral Driver Using Firmware Integration Technology, Rev.1.11, Document No. R01AN2025JJ0111, Sep 30, 2015

[7] Renesas USB MCU USB Host Mass Storage Class Driver (HMSC) Using Firmware Integration Technology, Rev.1.11, Document No. R01AN2026JJ0111, Sep 30, 2015

[8] Renesas Starter Kit+ for RX64M, ユーザーズマニュアル、Rev.1.20, Document No. R20UT2590JG0102, Jun 25, 2015

[9] Renesas Starter Kit+ for RX71M, ユーザーズマニュアル、Rev.1.00, Document No. R20UT3217JG0100, Jan 23, 2015

1.3 ハードウェアの構成

このサンプルプログラムは RX64M/71M の RAM とデータフラッシュを使用します。RAM の対象は、ECC エラー訂正機能なしの領域（以下、RAM）のみで、ECCRAM は対象外です。

詳細に関しては「RX64M/71M グループユーザーズマニュアル ハードウェア編」を参照ください。

1.4　ソフトウェアの構成

このサンプルプログラムは、共通の初期設定をボードサポートパッケージモジュール[3]として提供する複数の FIT モジュールを使用したアプリケーション層とミドルウェア層での実行例です。図 1.1にソフトウェアの典型的な構成と機能概要を示します。アプリケーションはデータフラッシュアクセス、RAM アクセス、ダブル RAM 処理、USB メモリにアクセスするストレージシステムで構成します。ダブル RAM とビット操作（db_ram_if.c）は乱数生成の初期値（以下、SEED）とビットパターンの生成、更新、消去、ユーザデータとビットパターンのビット操作、ダブル RAM の実行、非安全データの読み出しと書き込み、安全データの読み出しと書き込みを行います。フラッシュドライバ（フラッシュ API）[4]は、SEED とビットパターンのデータフラッシュからの消去とデータフラッシュへの書き込みを行います。FAT ファイルシステム（M3S-TFAT-Tiny）[5]は USB ホストドライバ[1]を使用し USB メモリ内のデータ[2]をファイルとして管理します。USB ホストドライバ[6], [7]は USB メモリに論理ブロック単位でアクセスします。

[1]USB ホストドライバの診断は対象外です。
[2]USB メモリから読み出した安全データは保証されたデータと仮定しています。

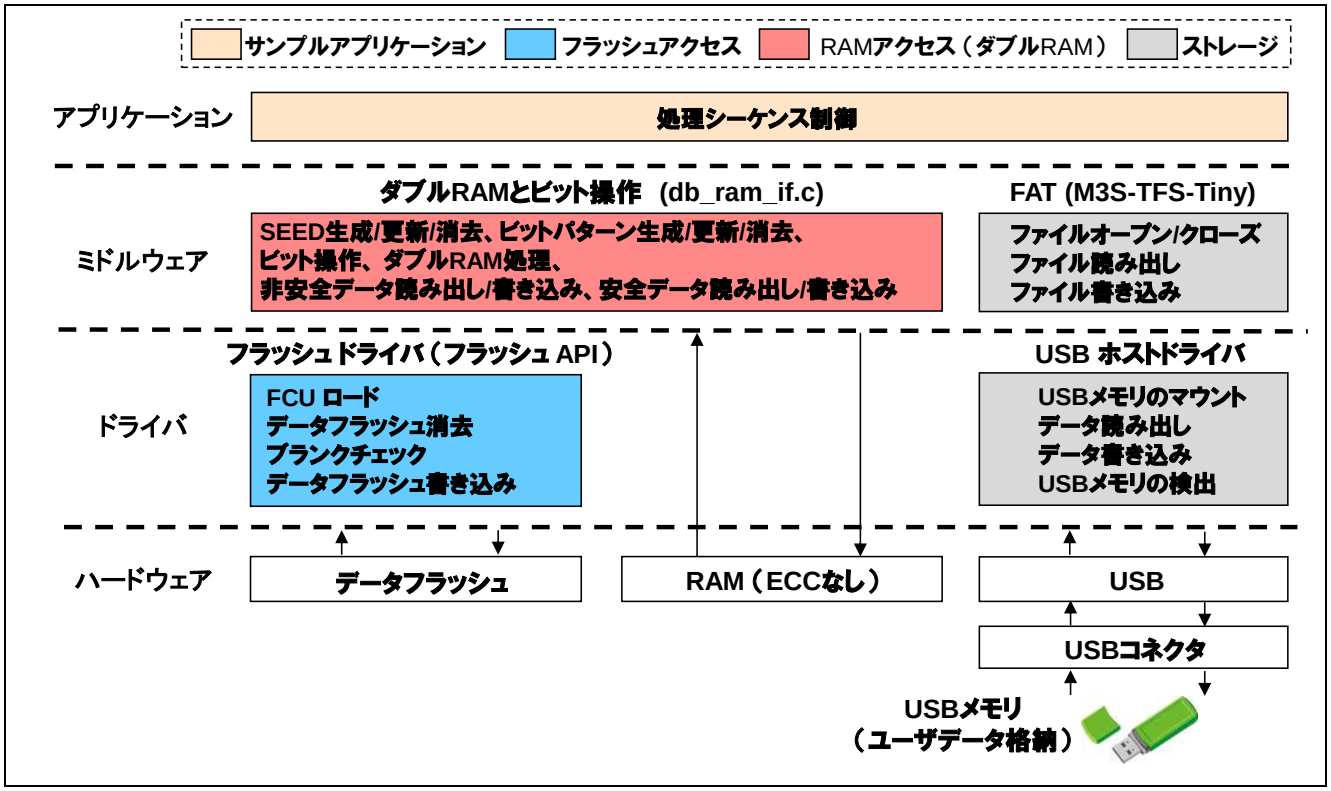

図1.1　ソフトウェア構成

1.5　ファイル構成

このサンプルプログラムのコードは demo_src とその下位階層のフォルダに格納しています。図 1.2にプロジェクトのソースファイルとヘッダファイルの構成を示します。非安全データと安全データのサンプルは、2セット分を USB メモリに格納しています。FIT モジュール（BSP、フラッシュドライバ、FAT ファイルシステム、USB ドライバ）の詳細は、個々の FIT モジュールのドキュメントを参照してください。

```
demo_src: メイン動作
|  sample_main.c
|  sample_main.h
|
+ --- db_ram: ダブルRAMとビット操作
|      db_ram.c
|      db_ram.h
|
+ --- tfat_if: ファイルシステムとUSBドライバ間のインタフェース
|      file_if.c
|      file_if.h
|      r_data_file.c
|      r_data_file.h
|      r_tfat_drv_if.c ;USBドライバインタフェース
|
+ --- usb_if: USBホスト メモリアクセス制御
|      r_usb_hmsc_defep.c
|      usb_memory_access.c
|
+ --- usr: LED制御
|      led.c
|      led.h
|
+ --- usb_memory_sample: USBメモリのデータサンプル
|      + --- set1, set2 ;  サンプル（2セット）
|      + --- NON: SRC_X.txt ; 非安全データファイル(X = 1,2,3)
|      + --- SAFE: SRC_X.txt ; 安全データファイル(X = 0,1,2,3,4,5,6,7)

r_bsp: BSP (Board Support Package) FITモジュール

r_config: FITモジュールの構成設定
|      r_bsp_config.h
|      r_bsp_interrupt_config.h
|      r_flash_rx_config.h
|      r_usb_basic_config.h
|      r_usb_hmsc_config.h

r_tfat_rx: FATファイルシステム (M3S-TFS-Tiny) FITモジュール
|      r_tfat_lib.h ;FATライブラリ　ヘッダファイル
|      + --- lib:  FATライブラリ　格納フォルダ
|              r_mw_version.h ; ミドルウェア　バージョン情報
|              r_stdint.h ; 整数型　定義
|              tfat_rx600_big.lib ; ライブラリ（ビッグエンディアン用）
|              tfat_rx600_little.lib ; ライブラリ（リトルエンディアン用）
|

r_usb_basic: USBドライバ（USB基本処理） FITモジュール

r_usb_hmsc: USBドライバ（ホストマスストレージクラス） FITモジュール
```

図1.2　ファイル構成

1.6 関数の概要

アプリケーション層の関数を表 1.1に、ダブル RAM とビット操作に関連する API 関数を表 1.2にそれぞれ示します。

表1.1 アプリケーション層の関数

関数	内容
main()	このプロジェクトのメイン処理
led_init()	ユーザ LED の初期化
led_ctrl()	ユーザ LED 表示の更新
prm_init()	処理パラメータの初期化
usb_memory_start()	USB メモリタスクの開始
Sample_Task()	サンプルアプリケーションのタスク処理
file_start()	USB メモリの開始
file_read()	ファイル読み出し処理
file_write()	ファイル書き込み処理
file_stop()	ファイル操作の停止と処理結果の保存
file_err()	ファイル操作でのエラーから復帰

表1.2 API 関数（ダブル RAM とビット操作）

関数	内容
FlashInit()	フラッシュドライバ（フラッシュ API）の初期化
load_seed()	データフラッシュから SEED のロード
upd_seed()	データフラッシュの SEED の更新
crt_bitpat()	ビットパターンの生成
load_bitpat()	データフラッシュからビットパターンのダブル RAM 処理によるロード
upd_bitpat()	データフラッシュのビットパターンのダブル RAM 処理による更新
ers_bitpat()	データフラッシュのビットパターンの消去
exec_bitop()	ビット操作の実行
RAM_Init()	RAM の初期化
RAM_Write()	非安全データの RAM への書き込み
RAM_SafetyWrite()	安全データの RAM への書き込み
RAM_Read()	非安全データの RAM からの読み出し
RAM_SafetyRead()	安全データの RAM からの読み出し

2. 機能情報

本サンプルプログラムは以下の要件で開発しています。

2.1 ハードウェア要件

サンプルプログラムは、使用する MCU が以下の機能をサポートしている必要があります。

- RAM[1]
- データフラッシュ
- USB

[1]ECC エラー訂正機能なしの領域のみを使用。

2.2 ハードウェアリソース要件

サンプルプログラムで使用するドライバが必要とする周辺回路ハードウェアについて説明します。特に明記されていない限り、周辺回路はドライバで制御し、ユーザアプリケーションから直接制御し、使用することはできません。

2.2.1 RAM

この例では RAM を使用し、その領域に非安全データと安全データを格納します。

2.2.2 データフラッシュ

この例では、SEED とビットパターンの格納にデータフラッシュを使用します。

2.2.3 USB チャネル

この例では、USB 2.0 FS ホスト/ファンクションを使用します。この周辺回路は、テストデータの USB メモリからの読み出し、処理結果の USB メモリへの書き込みに必要です。

2.3 ソフトウェア要件

サンプルプログラムは以下の FIT モジュールを使用しています。

- r_bsp
- r_flash_rx
- r_tfat_rx
- r_usb_basic
- r_usb_hmsc

2.4 制限事項

サンプルプログラムは以下の制限事項があります。

- USB ドライバの診断は対象外です。
- USB メモリから読み出した安全データは保証されたデータと仮定しています。

2.5 サポートされているツールチェイン

サンプルプログラムは次のツールチェインでテストと動作確認を行っています。

- Renesas RX Toolchain v2.04.01

2.6 ヘッダファイル

すべての関数呼び出しは、プロジェクトとともに提供されているヘッダファイル sample_main.h, led.h, file_if.h, r_usb_basic_if.h db_ram.h, r_frash_rx_if.h のうち 1 個のファイルをインクルードすることで行われます。

2.7 整数型

このプロジェクトでは ANSI C99 を使用し、整数型は stdint.h で定義しています。

2.8 コンパイル時の設定

このプロジェクトのコンフィグレーションオプションは sample_main.h、db_ram.h、 r_data_file.h で設定します。オプション名と設定値を以下の表に記載します。

構成設定	
#define NUM_REQ - Default value = 8	安全データのレコード数を設定します [1]。 • 1～8 を設定してください。
#define MAX_DAT_SIZE - Default value = 16*1024	テストデータファイルの最大サイズを設定します。 • 16*1024（16KB）を設定してください。
#define READ_DIR_NON - Default value "NON"	非安全テストデータファイルを格納するディレクトリ名を指定します。
#define READ_DIR_SAFE - Default value "SAFE"	安全テストデータファイルを格納するディレクトリ名を指定します。
#define READ_FILE - Default value "SRC"	テストデータファイルの文字列を指定します。 この文字列に、アクセス領域と関連付けられた番号と拡張子である".txt"を接続し、ファイル名となります。 例：“SRC_0.txt”, “SRC_1.txt”, , “SRC_7.txt”
#define WRITE_DIR_NON - Default value "NCMP"	実行後の非安全データファイルを格納するディレクトリ名を指定します。
#define WRITE_DIR_SCMP - Default value "SCMP"	実行後の安全データファイルを格納するディレクトリ名を指定します。
#define RESULT_FILE - Default value “RESULT"	安全データファイルのサイズ、生レコード領域の先頭アドレス、反転レコード領域の先頭アドレスを保存した実行結果ファイルの名前を指定します。 この文字列と拡張子である".txt"を接続しファイル名となります。 例：“RESULT.txt”
#define UPD_NXT_SEED - undefined	動作完了後、次回に使用する SEED を更新するか、しないかを設定します。 • 定義した場合、現在の処理完了後、SEED を更新します。
#define FORCE_ERASE_PRM - undefined	動作完了後、データフラッシュに書き込まれたビットパターンを消去するか、残すかを選択します。 • 定義した場合、データフラッシュからビットパターンを消去します。

構成設定	
#define BIT_PAT_MODE #define BIT_PAT_RAN (0) #define BIT_PAT_SEQ (1) #define BIT_PAT_CNT (2) - Default value = 0	ビットパターンを指定します。 • 0 を指定した場合、ビットパターンは乱数です。 • 1 を指定した場合、ビットパターンは繰り返しデータです。 • 2 を指定した場合、ビットパターンは定数です。
#define CONST_PAT - Default value = 0x00	定数のパターンを設定します。 この設定は、ビットパターンとして定数（BIT_PAT_CNT）を指定している場合のみ有効です。
#define SEED_BLOCK_ADDR - Default value = 0x00100000 (FLASH_DF_BLOCK_0)	SEED の値を格納するアドレスを設定します。
#define NUM_BIT_SEG - Default value = 8	ビットパターンのセグメント数を設定します。 このバージョンでは 8（デフォルト値）を設定してください。
#define BIT_SEG_SIZE - Default value = 1024	ビットパターンのセグメントサイズを設定します。 このバージョンでは 1024（デフォルト値）を設定してください。
#define NUM_NON_BLK - Default value = 3	非安全データの領域数を設定します。 このバージョンでは 3（デフォルト値）を設定してください。
#define NON1_AREA_SIZE - Default value = 16*1024 (16KB)	非安全データ領域（No.1）のサイズを設定します。 このバージョンでは 16*1024（デフォルト値）を設定してください。
#define NON2_AREA_SIZE - Default value = 16*1024 (16KB)	非安全データ領域（No.2）のサイズを設定します。 このバージョンでは 16*1024（デフォルト値）を設定してください。
#define NON3_AREA_SIZE - Default value = 16*1024 (16KB)	非安全データ領域（No.3）のサイズを設定します。 このバージョンでは 16*1024（デフォルト値）を設定してください。
#define NUM_RAW_REC - Default value = 8	安全データの生レコードの数を設定します。 このバージョンでは 8（デフォルト値）を設定してください。
#define NUM_INV_REC - Default value = 8	安全データの反転レコードの数を設定します。 このバージョンでは 8（デフォルト値）を設定してください。
#define RAW_REC_SIZE - Default value = 1024 (1KB)	安全データの生レコードのサイズを設定します。 このバージョンでは 1024（デフォルト値）を設定してください。
#define INV_REC_SIZE - Default value = 1024 (1KB)	安全データの反転レコードのサイズを設定します。 このバージョンでは 1024（デフォルト値）を設定してください。
#define KND_BIT_OP #define OP_NONE (0) #define OP_EXOR (1) - Default value = 1 (OP_EXOR)	ビット操作の種別を指定します。 • 0 を指定した場合、ビット操作を実行しません。 • 1 を指定した場合、排他的論理和の操作を実行します。
#define FILESIZE - Default value = 2048	FAT ファイルシステムのバッファサイズを設定します。 2048 を設定してください。

[1] ダブル RAM 処理の回数となります。

2.9 データ構造

サンプルプログラムの関数で使用するデータ構造について説明します。このプロジェクトのデータ構造は sample_main.h と db_ram.h で宣言しています。

```
/* USB access state */
typedef enum
{
    APL_START = 0, /* Operation start state */
    APL_NREAD,     /* Non safety data read state */
    APL_NRAM,      /* Non safety data RAM access state */
    APL_NWRITE,    /* Non safety data write state */
    APL_SREAD,     /* Safety data read state */
    APL_SRAM,      /* Safety data RAM access state */
    APL_SWRITE,    /* Safety data write state */
    APL_STOP,      /* Operation stop state */
} APLState;
```

```
/* Kind of RAM area */
typedef enum
{
    AREA_NON1 = 0, /* Non safety data area No.1 */
    AREA_NON2,     /* Non safety data area No.2 */
    AREA_NON3,     /* Non safety data area No.3 */
    AREA_SAFE,     /* Safety data area */
} RAMArea;
```

```
/* Data access information structure */
typedef struct
{
    uint16_t size; /* Data size */
    int8_t *src;   /* Address of read data from USB */
    int8 t *dst;   /* Address of write data to USB */
    int8_t area;   /* Kind of RAM area */
} ACCInfo;
```

```
/* Bit pattern segment address */
const flash_block_address_t bp_addr[NUM_BIT_SEG] =
{ /* Refer to "r_flash_rx/src/targets/rx64m/r_flash_rx64m.h or
rx71m/r_flash_rx71m.h". */
    FLASH_DF_BLOCK_384, /* 0x00106000 to 0x001063FF (1KB) */
    FLASH_DF_BLOCK_400, /* 0x00106400 to 0x001067FF (1KB) */
    FLASH_DF_BLOCK_416, /* 0x00106800 to 0x00106BFF (1KB) */
    FLASH_DF_BLOCK_432, /* 0x00106C00 to 0x00106FFF (1KB) */
    FLASH_DF_BLOCK_448, /* 0x00107000 to 0x001073FF (1KB) */
    FLASH_DF_BLOCK_464, /* 0x00107400 to 0x001077FF (1KB) */
    FLASH_DF_BLOCK_480, /* 0x00107800 to 0x00107BFF (1KB) */
    FLASH_DF_BLOCK_496, /* 0x00107C00 to 0x00107FFF (1KB) */
};
```

```
/* Bit pattern for double RAM segment address */
const flash_block_address_t bpc_addr[NUM_BIT_SEG] =
{ /* Refer to "r_flash_rx/src/targets/rx64m/r_flash_rx64m.h or
rx71m/r_flash_rx71m.h". */
    FLASH_DF_BLOCK_512, /* 0x00108000 to 0x001083FF (1KB) */
    FLASH_DF_BLOCK_528, /* 0x00108400 to 0x001087FF (1KB) */
    FLASH_DF_BLOCK_544, /* 0x00108800 to 0x00108BFF (1KB) */
    FLASH_DF_BLOCK_560, /* 0x00108C00 to 0x00108FFF (1KB) */
    FLASH_DF_BLOCK_576, /* 0x00109000 to 0x001093FF (1KB) */
    FLASH_DF_BLOCK_592, /* 0x00109400 to 0x001097FF (1KB) */
    FLASH DF BLOCK 608, /* 0x00109800 to 0x00109BFF (1KB) */
    FLASH_DF_BLOCK_624, /* 0x00109C00 to 0x00109FFF (1KB) */
};
```

2.10 戻り値

サンプルプログラムの関数の戻り値を示します。このプロジェクトの戻り値は db_ram.h と file_if.h にプロトタイプ宣言と共に定義しています。

```
/* Double RAM operation return value */
typedef enum
{
    DBRAM_ERR_FLERASE = -8,  /* Flash erase error */
    DBRAM_ERR_FLWRITE = -7,  /* Flash write error */
    DBRAM_ERR_FLVERIFY = -6, /* Flash verify error */
    DBRAM_ERR_FLDBRAM = -5,  /* Flash double RAM error */
    DBRAM_ERR_PARAM = -4,    /* Parameter error */
    DBRAM_ERR_BOP = -3,      /* Bit operation error */
    DBRAM_ERR_CMP = -2,      /* Double RAM compare error */
    DBRAM_ERR = -1,          /* General error */
    DBRAM_OK = 0,
} dbram_t;
```

```
/* File access return value */
typedef enum
{
    FLIF_ERR = -1, /* General error */
    FLIF_OK = 0,
} flif_t;
```

3.　サンプルプログラムの仕様

3.1　　環境とプログラムの実行

サンプルプログラムは、RX64M/71M RSKボード[1]とUSBメモリを使用します。

実行手順の概要を以下に説明します。

- RX64M /71M RSK ボード（以後、RSK ボード）のコードフラッシュに実行コードを書き込んでください。
- USB メモリを RSK ボードの USB ポートに接続してください。USB メモリは RAM への書き込んだ、または、RAM からの読み出した非安全データと安全データで構成するテストデータを格納します。ここで、安全データにはダブル RAM とビット操作を適用します。
- テストデータのサンプルをソースプロジェクトの demo_src/usb_memory_sampl/set1 と set2 フォルダに準備していますので、そのどちらかを USB メモリのルートにコピーすれば使用頂けます。
- RSK ボードに電源を投入してください。
- RSK ボードは FAT ファイルシステムと USB ホストの初期化と開始、ビットパターンの生成、ダブル RAM によるビットパターンの更新、及び SEED のロードが完了すると、ユーザ LED が”1”パターンに点灯します（LED3: オフ、LED2: オフ、LED1: オフ、LED0: オン）。
- RSK ボードのユーザスイッチである”SW1”を押すと、RSK ボードは3.2節で説明する動作を開始します。
- 各領域への安全データのダブル RAM 処理が完了する毎に、ユーザ LED は最後にアクセスしたレコード番号（0～7）を示します。RAM 領域の詳細については3.2節を参照ください。
- RSK ボードのユーザスイッチである”SW2”を押すと、次の領域へのダブル RAM 処理を開始します。
- 動作シーケンスが正常に完了した場合、ユーザ LED が全て点灯します（LED3: オン、LED2: オン、LED1: オン、LED0: オン）。
- 処理中にエラーが発生した場合、ユーザ LED がエラー内容に応じ、下記のように点灯します。

  フラッシュアクセスエラー – LED3: ON, LED2: OFF, LED1: OFF, LED0: OFF（“8”パターン）

  安全データアクセスエラー – LED3: ON, LED2: OFF, LED1: OFF, LED0: ON（“9”パターン）

  非安全データアクセスエラー – LED3: ON, LED2: OFF, LED1: ON, LED0: OFF（“A”パターン）

  SEED 生成エラー – LED3: ON, LED2: OFF, LED1: ON, LED0: ON （“B”パターン）

  ビットパターン消去エラー – LED3: ON, LED2: ON, LED1: OFF, LED0: OFF （“C”パターン）

[1] 製品名は Renesas Starter Kit+ for RX64M [8] または Renesas Starter Kit+ for RX71M [9]です。

図 3.1に動作環境の一例を示します。

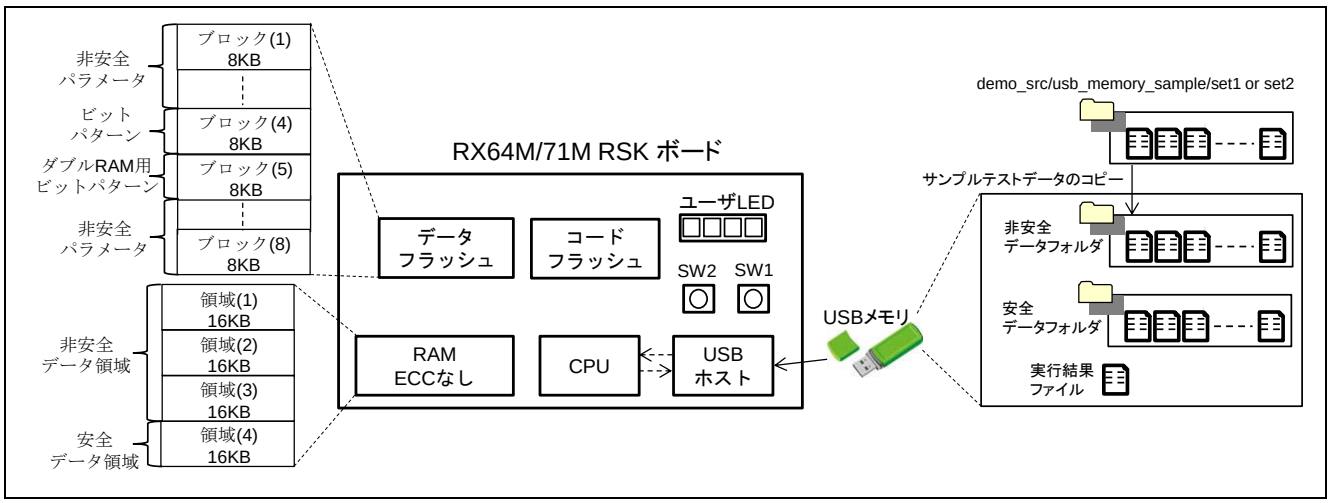

図3.1　動作環境

3.2　動作シーケンス

この例の動作シーケンスを安全データのレコード数[1]を 8 として説明します。

USB メモリに格納されるデータの内容を図 3.2に示します。USB メモリには NON フォルダ、NCMP フォルダ、SAFE フォルダ、SCMP フォルダ、RESULT.txt ファイルがあります。NON フォルダは SRC_1.txt から SRC_3.txt で構成する非安全テストデータを格納します。NCMP フォルダは、通常の書き込みと読み出しを実行後、比較した CMP_1.txt から CMP_3.txt で構成するデータを格納します。SAFE フォルダは SRC_0.txt から SRC_7.txt で構成する安全テストデータを格納します。SCMP フォルダは、ダブル RAM 処理を実行後、比較した CMP_0.txt から CMP_7.txt で構成するデータを格納します。SAFE フォルダは SRC_0.txt から SRC_7.txt で構成する受信データを格納します。処理が正常に終了した場合、SAFE（NON）フォルダと SCMP（NCMP）フォルダのデータの内容は互いに一致します。例として、図 3.2に非安全テストデータの SRC_3.txt と安全テストデータの内容を、それぞれ示します。RESULT.txt は試行回数、安全テストデータのサイズ、安全テストデータの生レコードと反転レコードのアドレスを転送性能結果として格納します。

[1] ”NUM_REC”で指定する。

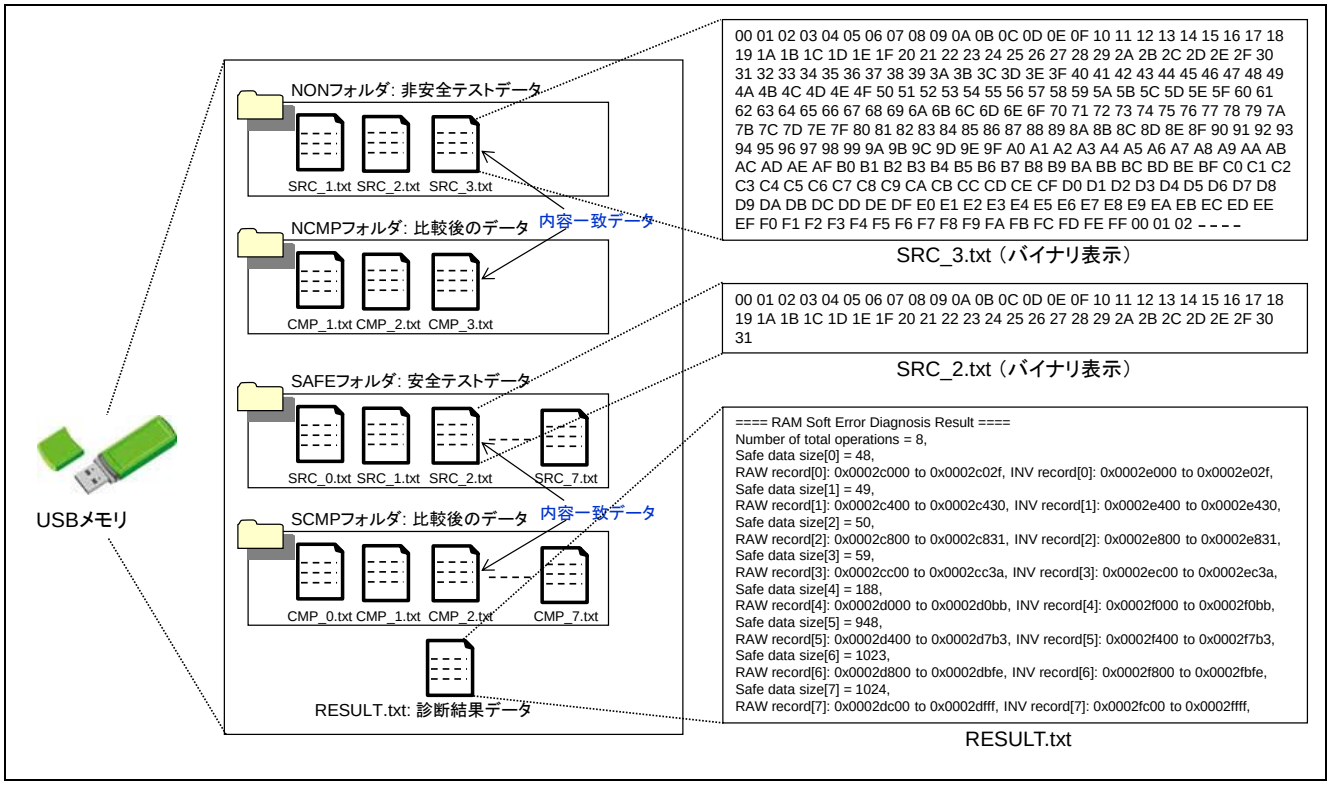

図3.2　USB メモリの内容

この例では 16K バイトのデータフラッシュと 64K バイトの RAM を使用します。RX64M/71M のデータフラッシュの消去単位と書き込み単位は、それぞれ 64 バイトと 4 バイトです[1]。

データフラッシュは 8 分割しブロックと定義します（ブロック(1)～ブロック(8)）。さらに、ブロックを 1K バイト単位で分割しセグメントと定義します（セグメント(0)～セグメント(7)）。1 ブロックをビットパターン格納領域に割り当て、他の 1 ブロックをダブル RAM 用ビットパターン格納領域に割り当て、残りのブロックを非安全用パラメータ格納領域に割り当てます。

RAM 領域で最後の 16K バイトを安全データ格納領域に割り当て領域(4)とし、残りの 16K バイト単位で 3 個の領域を非安全データ格納領域に割り当てます（領域(1)～領域(4)）。領域(4)の前半 8K バイトを生データ領域に割り当て（生レコード(0)～生レコード(7)）、後半 8K バイトをビット反転データ領域に割り当てます（反転レコード(0)～反転レコード(7)）。それらを 1K バイト単位で分割し生レコード、または、反転レコードと定義します。

1 バイトの安全データ（以下、データ A とする）を書き込み後、読み出した場合の処理概要を、データ配置と関連付け、図 3.3で説明する。ここで、データ A は 7 番目の生レコードの N バイト目（1 <= N <= 1024）、ビット操作は排他的論理和（以下、XOR）を適用するとする。

(1) データフラッシュ上のビットパターンを更新します。その後、ダブル RAM により更新したビットパターンを読み出し、正常に更新されたことを確認します。

(2) データ A とビットパターン(6, N)との XOR を実行し、実行後のデータ A をデータ A’とします。

(3) データ A’を 7 番目の生レコードの N バイト目に書き込みます。

(4) データ A’をビット反転し、実行後のデータ A’をデータ A”とします。

(5) データ A”を 2 番目の反転レコードの（1024 – N + 1）バイト目に書き込みます。

(6) データ A”を 2 番目の反転レコードの（1024 – N + 1）バイト目から読み出します。

(7) データ A”のビット反転処理をします。

(8) データ A’を 7 番目の生レコードの N バイト目から読み出します。

(9) (7)のデータ A”をビット反転したデータと(8)の読み出したデータ A’を比較します。比較の結果、互いに一致しない場合、RAM のソフトエラーとして検出する。

(10) データ A’とビットパターン(6, N)との XOR を実行します。実行後のデータは XOR の特性によりデータ A に戻ります。そして、ユーザアプリケーションにデータ A を返します。

[1] より詳細な内容については、RX64M/71M のユーザーズマニュアルを参照ください。

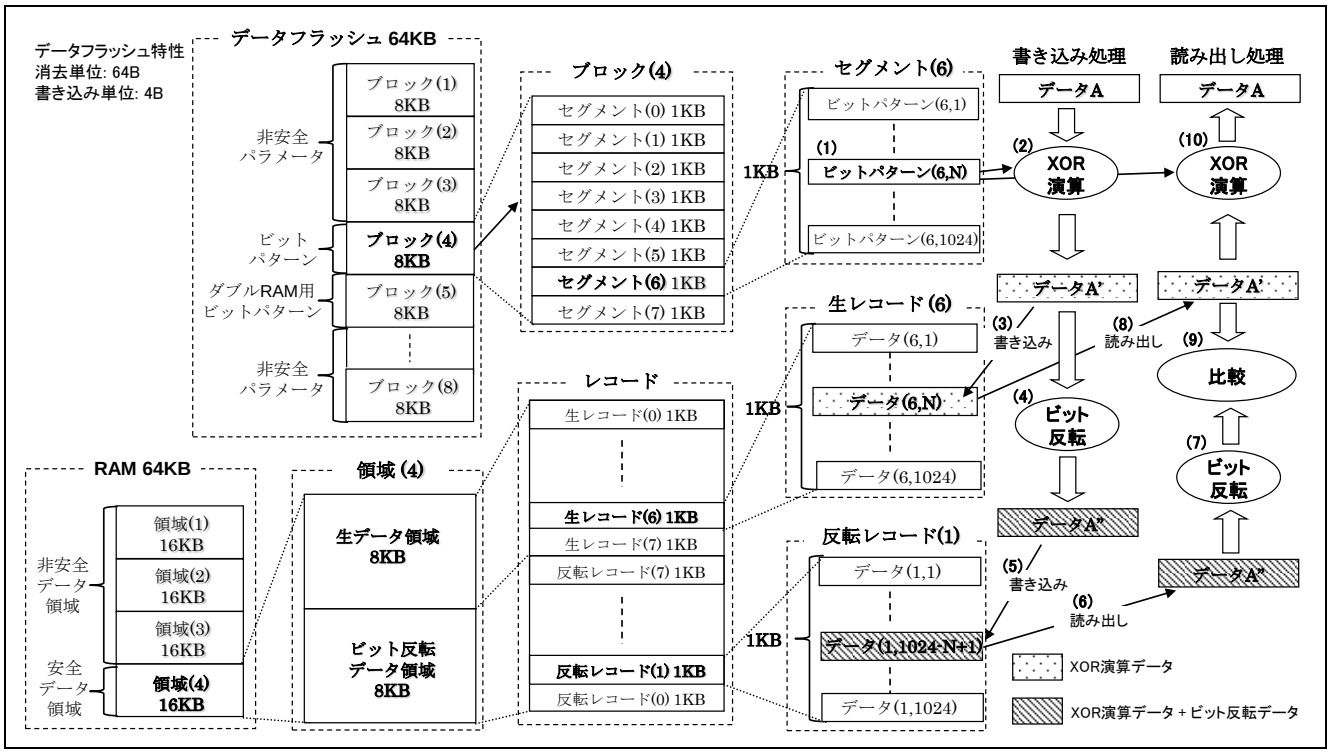

図3.3　処理概要とデータ配置

3.3　データの偏り改善

一般に、ユーザアプリケーションの安全データは”0”と”1”で平均化しておらず、”0”または”1”で固定されたRAMのエラーを効率よく検出するのは困難となります。そこで、XOR演算によりRAM上の安全データのデータパターンを均等化し、エラーの検出率を向上させます。

図3.3と同様な処理において、ユーザアプリケーションの偏りのあるデータをXOR演算により改善する例を図3.4に示します。0x00から0xFFを256バイトでの繰り返すビットパターンのデータをセグメント(6)に格納します。この場合、P_Nを現在のビット値を保持する確率、P_Iを現在のビット値を反転する確率とすると、ビットパターンのb’0とb’1の数はともに4096個であり、これらビットパターンと安全データのXOR演算により、P_NとP_Iは近似的に1/2となります。ユーザアプリケーションデータのb’0とb’1の数が、それぞれN個と（8192 – N)個とし、XOR演算を行うと、b’0とb’1の数はともに近似的に4096個になり、確率的にはデータの偏りが改善します。ここで、N = 0, 1, 2,,, 8192とする。

P_NとP_Iはデータサイズが大きいほど、より1/2に近くなるので、ユーザアプリケーションのデータサイズに比例し、エラー検出率は向上します。

乱数から生成したビットパターンを使用した場合もb’0とb’1の数は近似的に等しいので、同程度の結果が予測できます。

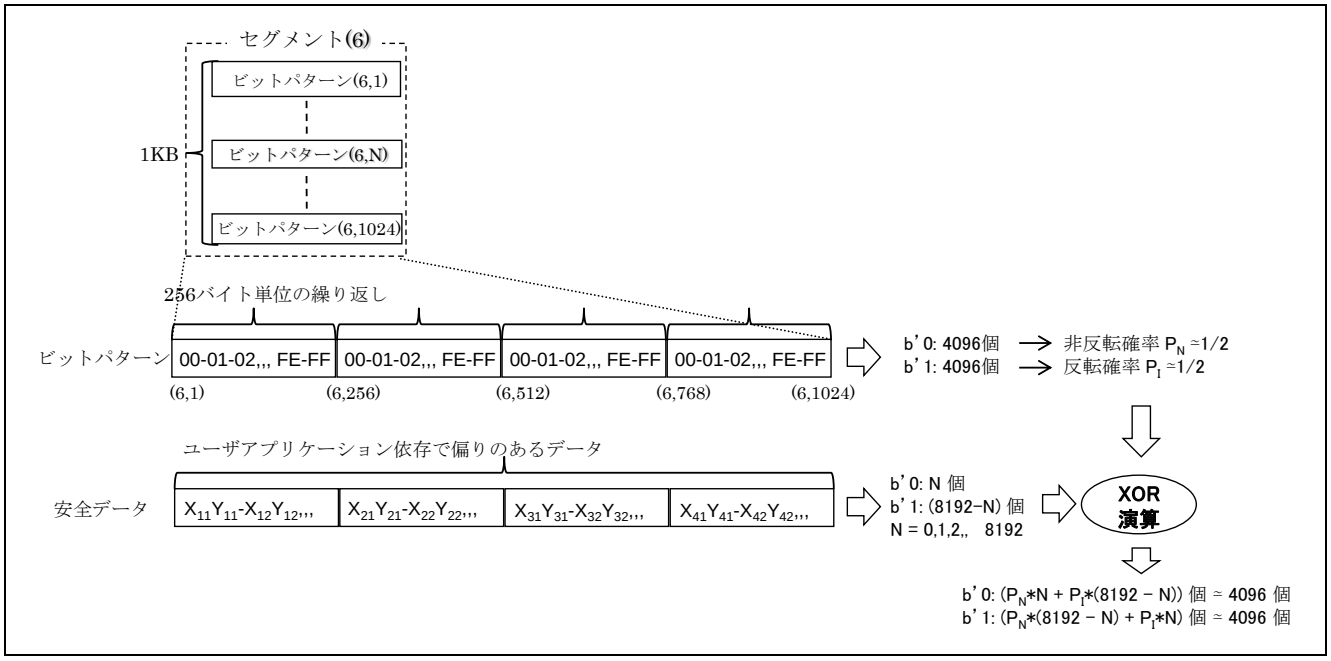

図3.4　XOR による偏り改善

3.4　安全データの処理性能（測定例）

RX64MとRX71Mの安全データアクセス時の処理性能を、典型的な例として表3.1と表3.2にそれぞれ示します。これらの処理時間はビット操作ありで、安全データの書き込み関数（RAM_SafetyWrite関数）、または、安全データの読み出し関数（RAM_SafetyRead関数）の実行時間により測定した結果です。また、安全データはSAFEフォルダに格納した安全データを使用しています。

測定結果は測定条件と環境により異なることにご注意ください。

[1] Renesaemo_src/usb_memory_sampl/set1 と set2 フォルダに格納。

表3.1　RX64M 処理時間

- RX64M 測定条件

ボード	動作周波数	コード領域	データ領域	コンパイラ	最適化設定
RX64M RSK ボード (R0K50564MC001BR)	120MHz	内蔵ROM Read: 1サイクル	内蔵RAM Read/Write: 1サイクル	CC-RX V2.04.01	デフォルト設定 - 最適化レベル: 2 - 最適化方法: サイズ優先 - モジュール間最適化: なし

- set1（ビット操作あり）

データサイズ (バイト)	48	49	50	59	188	948	1023	1024
書き込み (ns)	22,150	22,616	23,083	27,141	85,008	426,216	459,941	460,333
読み出し(ns)	22,891	23,375	23,858	28,066	87,941	441,025	475,916	476,325

- set2（ビット操作あり）

データサイズ (バイト)	150	1024	256	127	991	992	723	27
書き込み (ns)	67,983	460,333	115,533	57,675	445,575	445,966	325,250	12,775
読み出し(ns)	70,316	476,325	119,533	59,658	461,050	461,458	336,541	13,200

表3.2　RX71M 処理時間

- RX71M 測定条件

ボード	動作周波数	コード領域	データ領域	コンパイラ	最適化設定
RX71M RSK ボード (R0K50571MC000BR)	240MHz	内蔵ROM Read: 2サイクル	内蔵RAM Read/Write: 1サイクル	CC-RX V2.04.01	デフォルト設定 - 最適化レベル: 2 - 最適化方法: サイズ優先 - モジュール間最適化: なし

- set1（ビット操作あり）

データサイズ (バイト)	48	49	50	59	188	948	1023	1024
書き込み (ns)	17,983	18,350	18,733	22,033	69,250	347,516	375,000	375,333
読み出し(ns)	18,316	18,708	19,083	22,458	70,666	354,875	382,950	383,300

- set2（ビット操作あり）

データサイズ (バイト)	150	1024	256	127	991	992	723	27
書き込み (ns)	55,350	375,333	94,141	46,933	363,283	363,625	265,150	10,316
読み出し(ns)	56,483	383,300	96,108	47,891	370,983	371,333	270,766	10,491

3.5　ソフトウェア動作フロー

この節では、サンプルプログラムのソフトウェア動作フローを説明します。

図 3.5は初期化から無限ループまでの処理を示します。サンプルアプリケーションのタスク（Sample_Task 関数）は無限ループ内の図 3.6に示す USB アプリケーションタスクを経由し呼ばれます。図 3.7と図 3.8はアプリケーションの状態により管理される個々のタスク構造を示します。ダブル RAM とビット操作に関連する処理フローは図 3.9から図 3.20に示します。

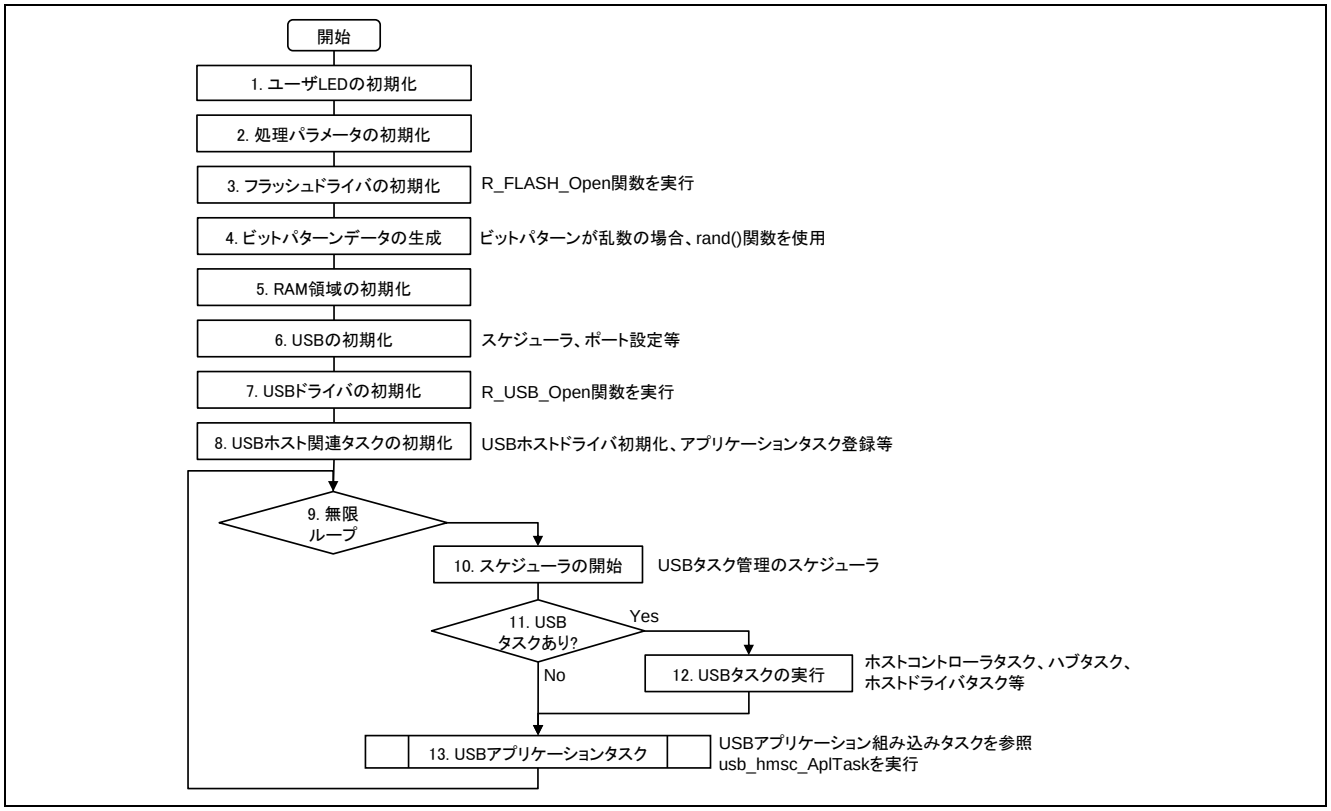

図3.5　初期化処理

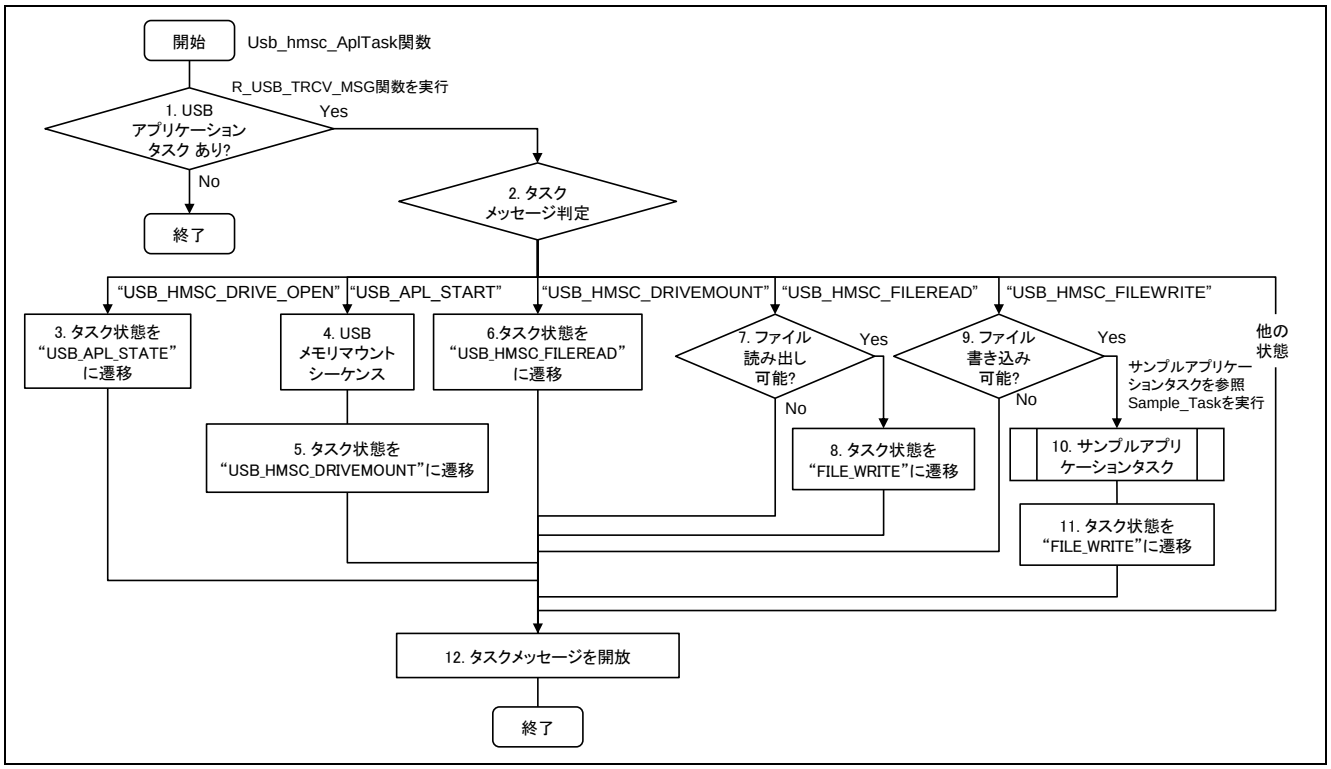

図3.6　USB アプリケーション組み込みタスク

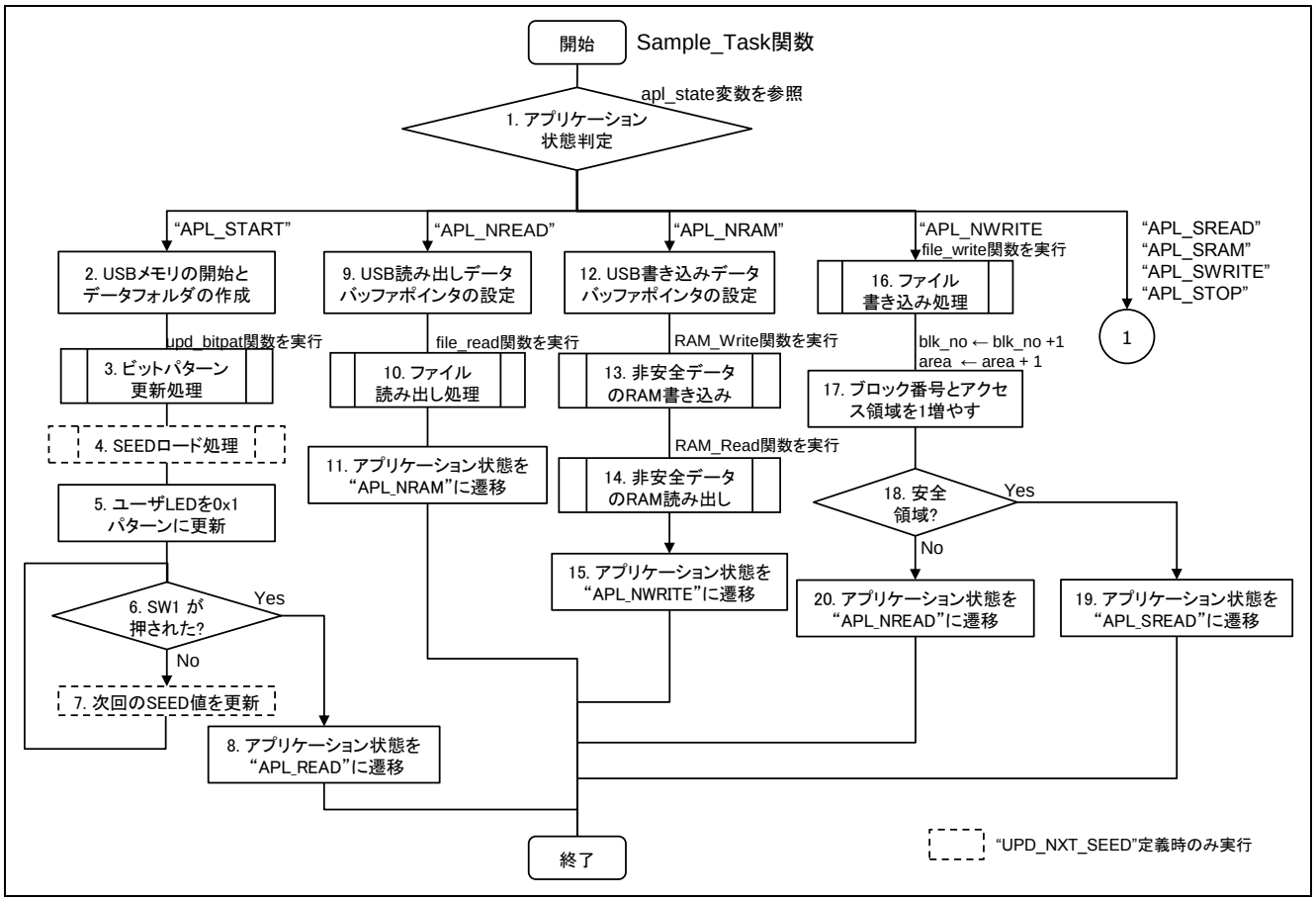

図3.7　サンプルアプリケーションタスク(1)

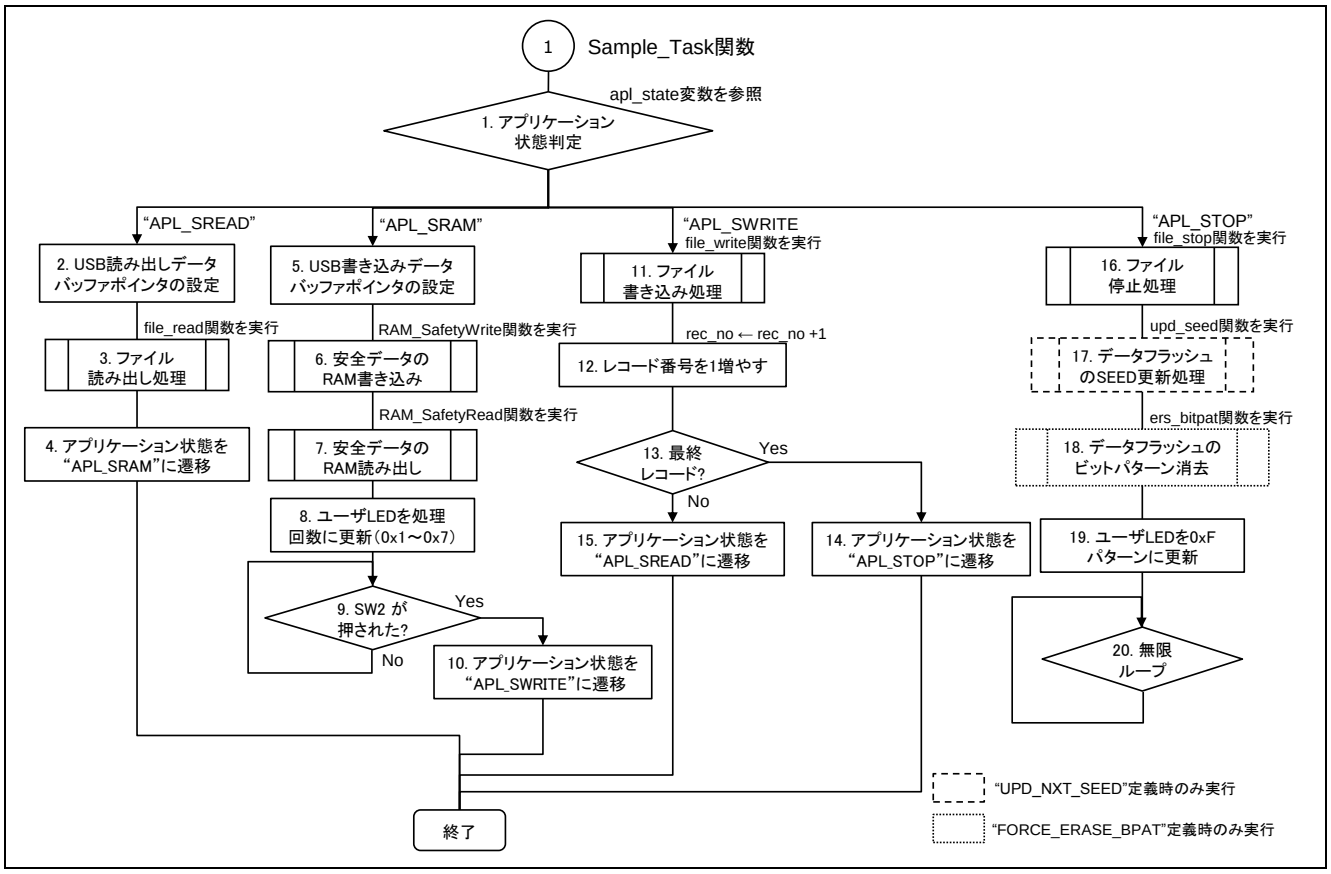

図3.8　サンプルアプリケーションタスク(2)

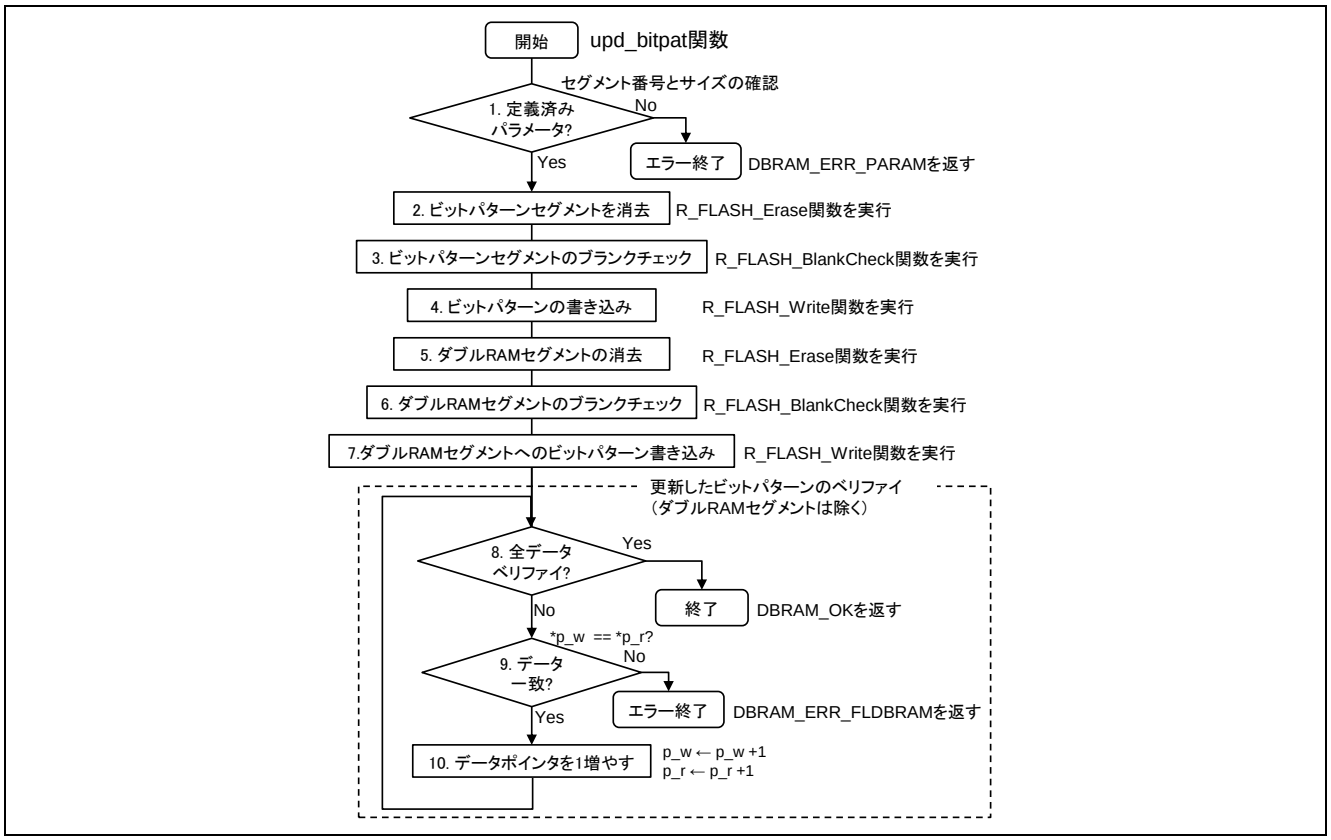

図3.9　ビットパターン更新処理

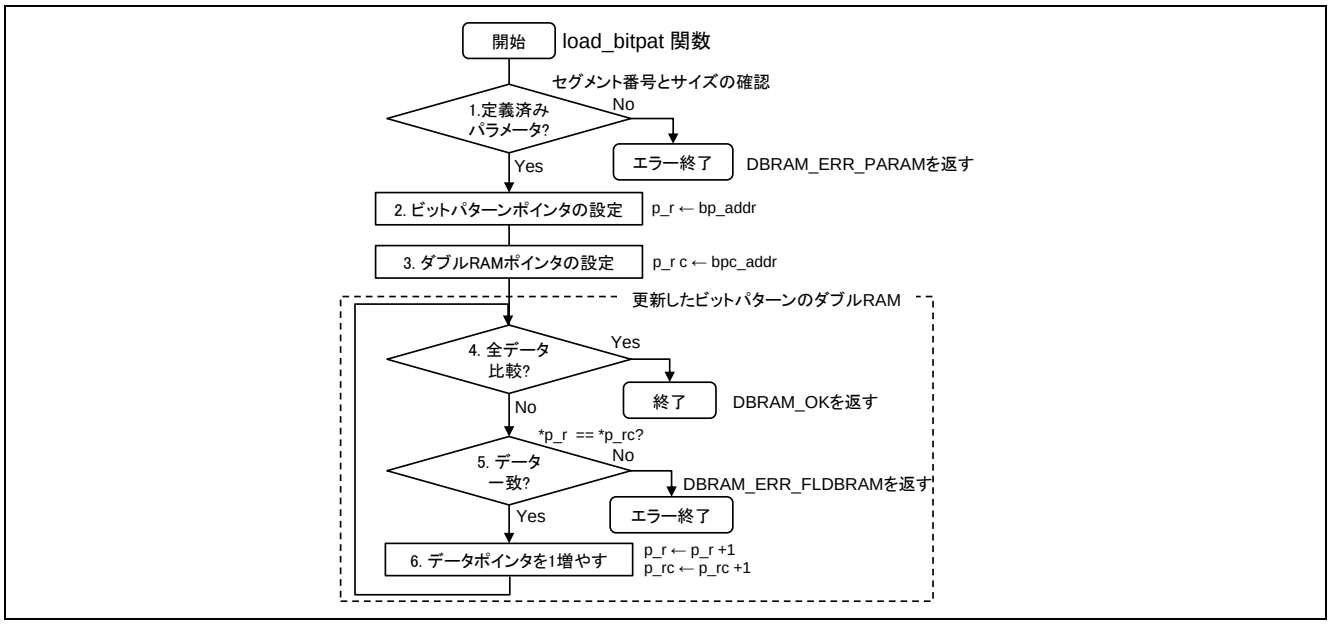

図3.10　ビットパターンロード処理

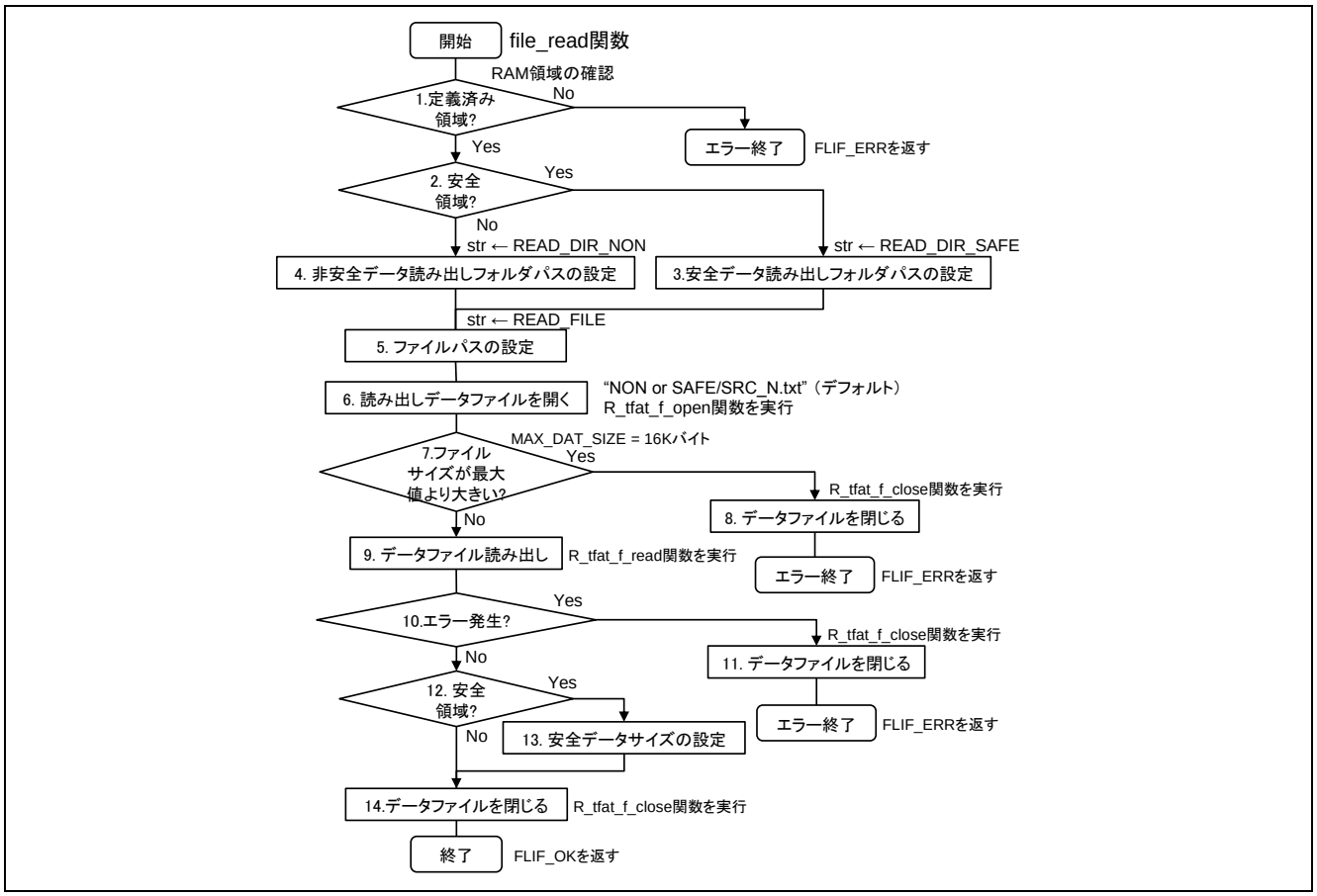

図3.11　ファイル読み出し処理

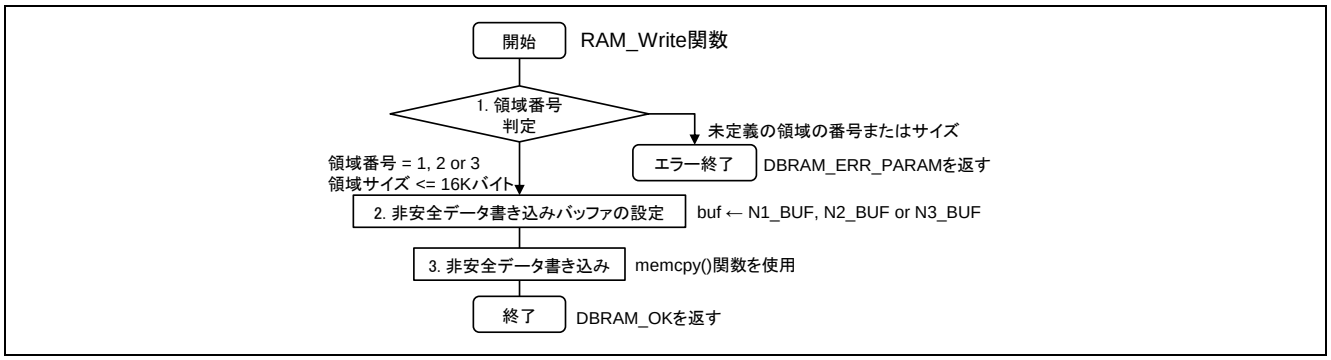

図3.12　非安全データの書き込み処理

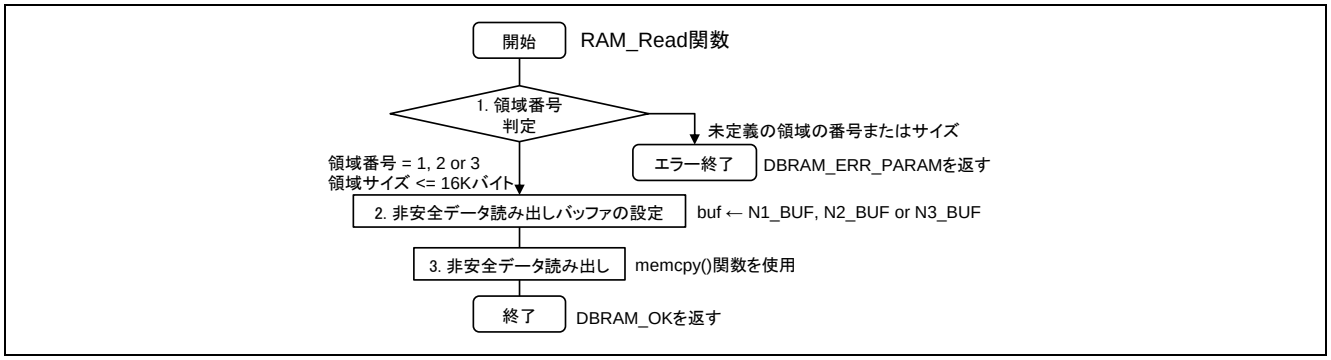

図3.13　非安全データの読み出し処理

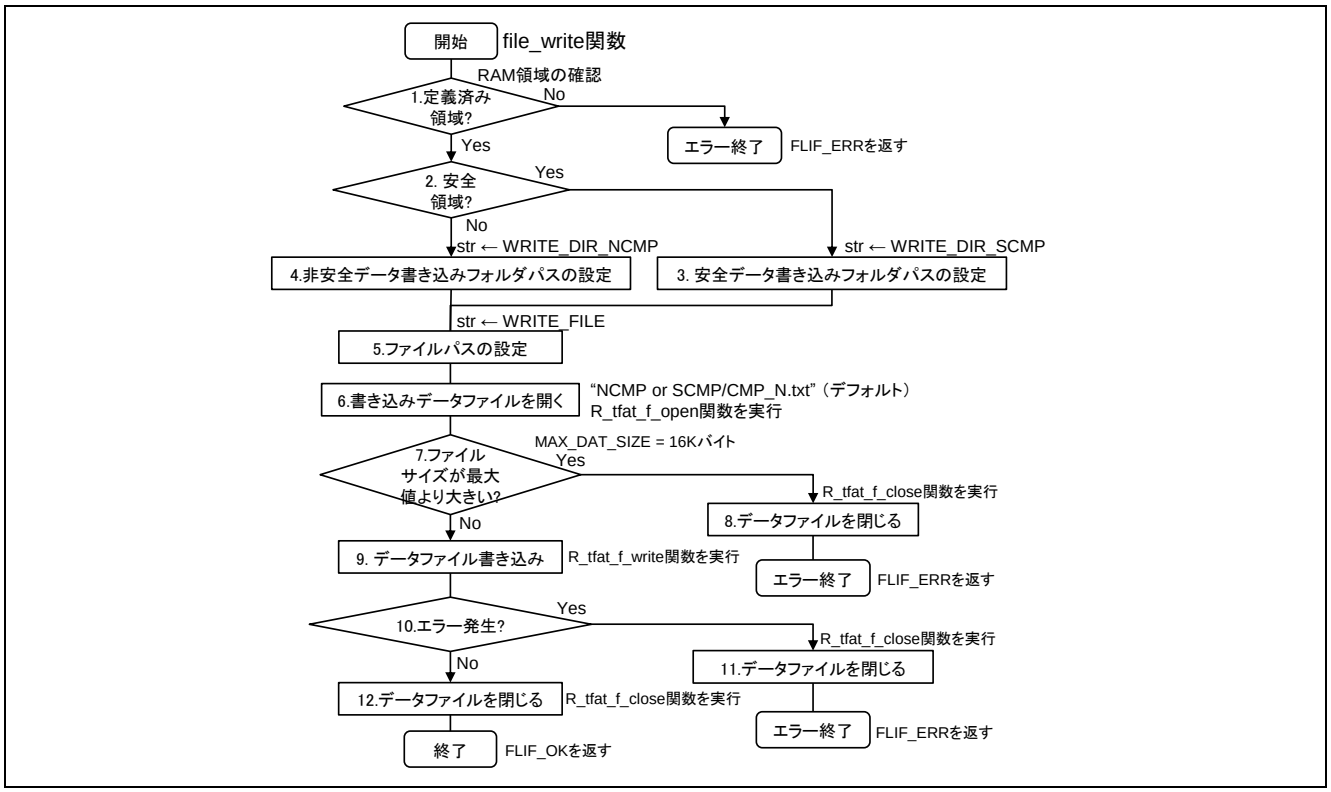

図3.14　ファイル書き込み処理

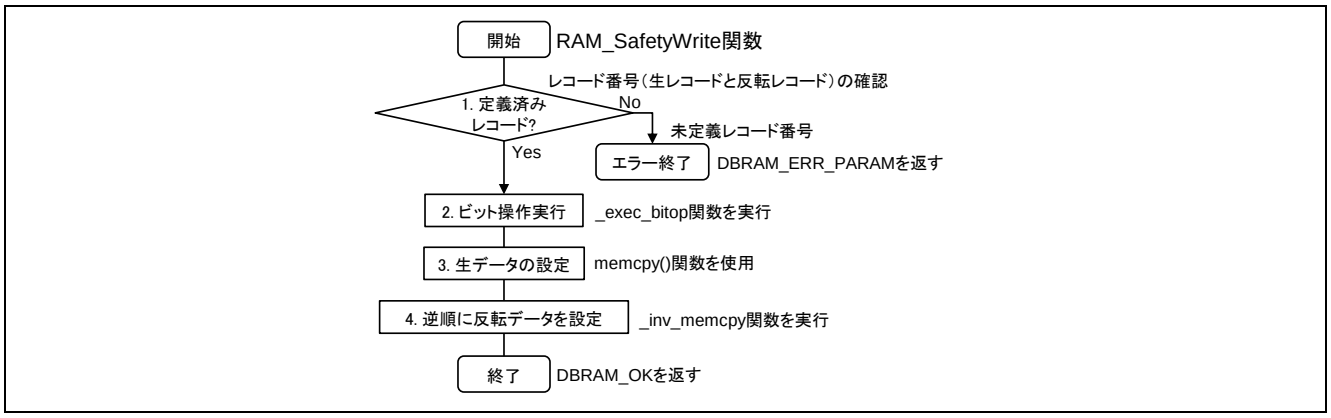

図3.15　安全データの書き込み処理

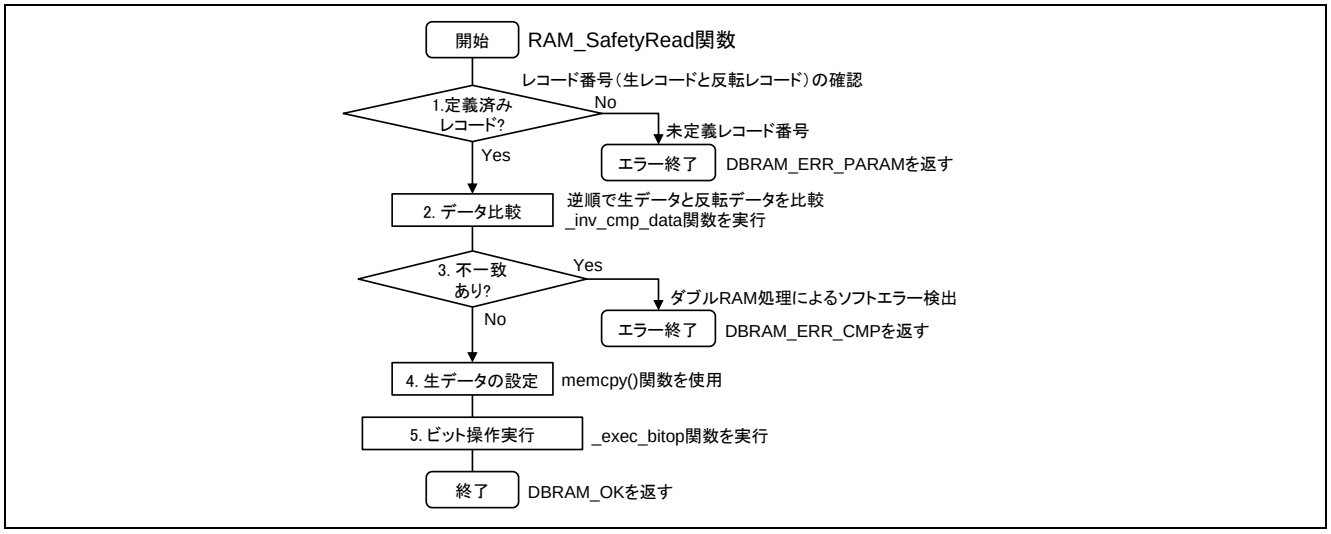

図3.16　安全データの読み出し処理

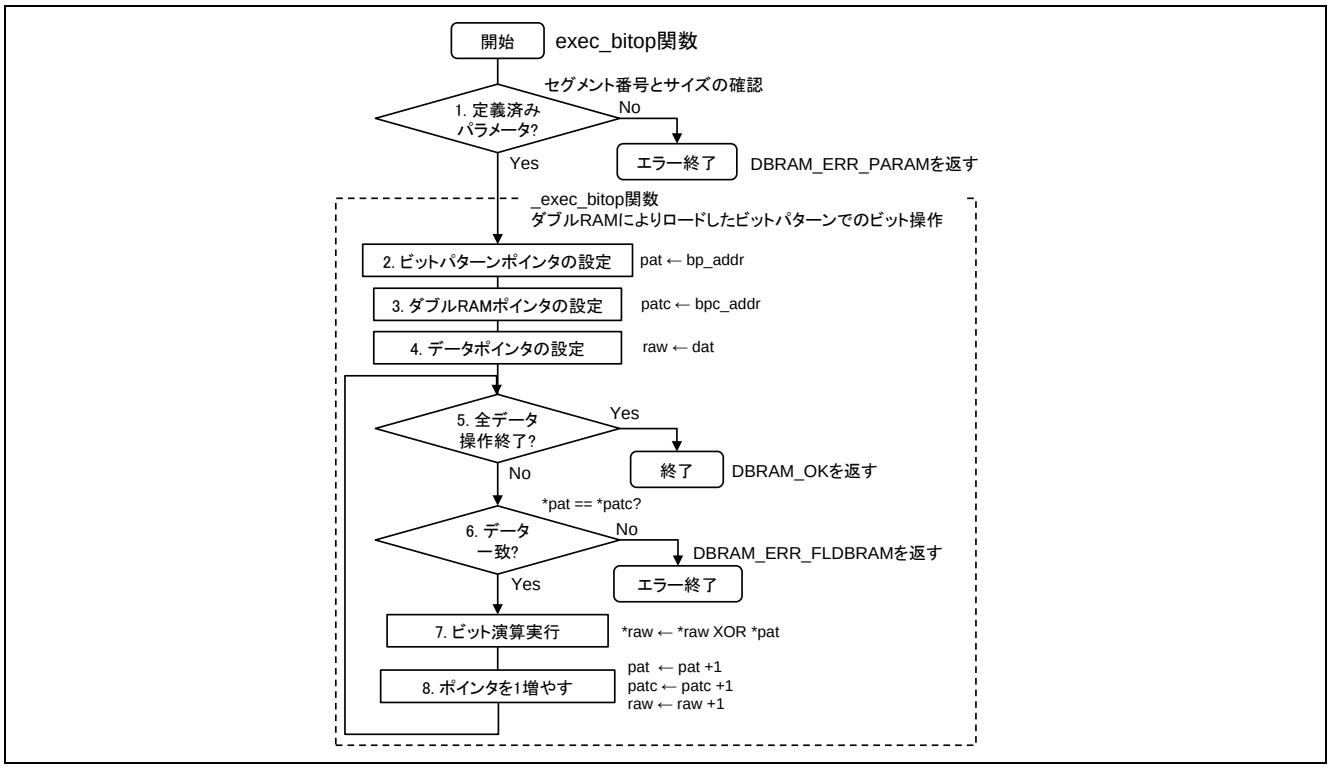

図3.17　ビット操作処理

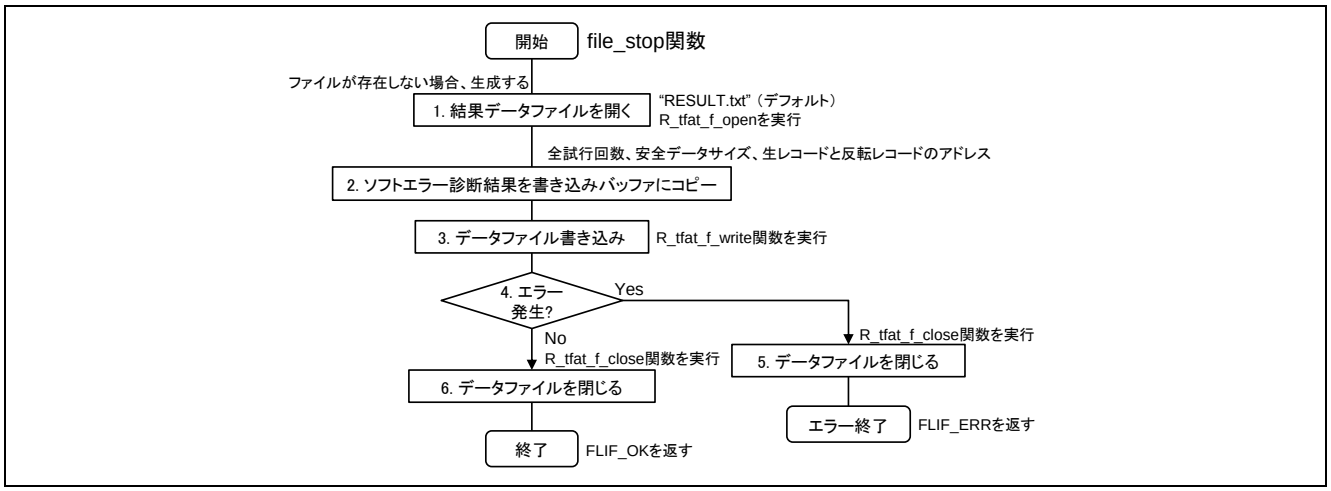

図3.18　ファイル停止処理

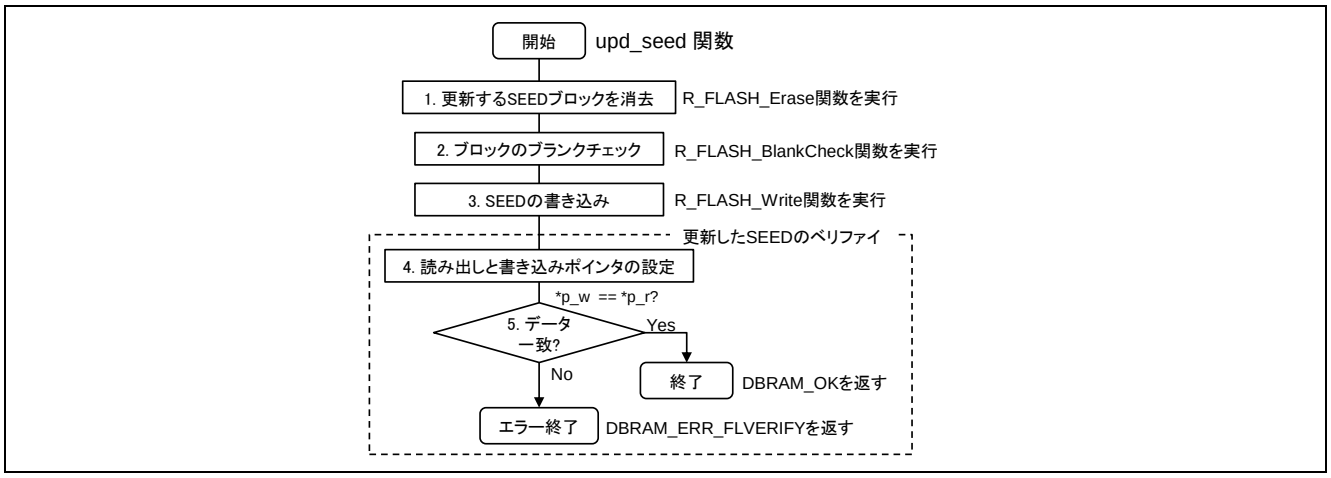

図3.19　SEED 更新処理

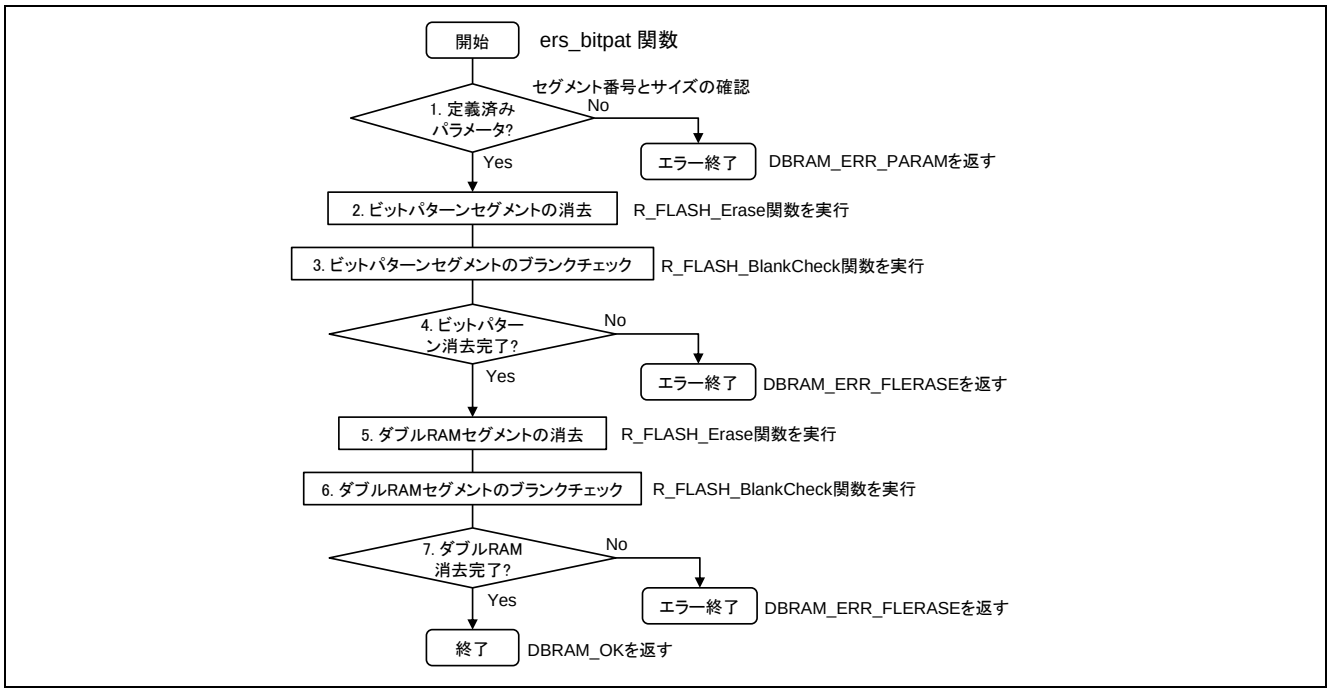

図3.20　ビットパターン消去処理

3.6 ボードの設定

サンプルプログラムの動作には RX64M/71M RSK ボードに実装された 2 個のジャンパをデフォルト設定から変更する必要があります。RX64M/71M RSK ボードの型名が R0K50564MC001BR または R0K5RX71MC010BR の場合、ジャンパ設定を表 3.3に示します。また、RX71M RSK ボード型名が R0K50571MC000BR の場合、ジャンパ設定を表 3.4に示します。

表3.3 ジャンパ設定

- USB アクセスの設定

ジャンパ	ボードのデフォルト設定	サンプル動作の設定	使用機能
J2	2-3	**1-2**	**USBでホストモードを有効化**
J6	1-2	**2-3**	**USB USB0VBUSEN**

表3.4 ジャンパ設定

- USB アクセスの設定

ジャンパ	ボードのデフォルト設定	サンプル動作の設定	使用機能
J1	2-3	**1-2**	**USBでホストモードを有効化**
J3	1-2	**2-3**	**USB USB0VBUSEN**

4. API 関数

4.1 FlashInit ()

この関数はフラッシュドライバ（フラッシュ API）を初期化します。

フォーマット

dbram_t FlashInit (void);

パラメータ

なし

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR: エラーが発生しました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

フラッシュドライバ（フラッシュ API）を初期化します。

- ビットパターン指定が乱数（BIT_PAT_RAN == BIT_PAT_MODE）の場合、データフラッシュから SEED をロードし、srand()関数により乱数生成の初期値として設定します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include <stdlib.h>
#include "db_ram.h"
#include "r_flash_rx_if.h"

static int8_t BPT_BUF[BIT_SEG_SIZE]; /* Bit pattern data buffer */

    dbram_t db_ret;

    /* Initialize flash API and set initial seed */
    db_ret = FlashInit();
    if (DBRAM_OK != db_ret)
    {
        goto Err_end; /* error */
    }

    /* Create bit pattern data */
    crt_bitpat(BPT_BUF, sizeof(BPT_BUF));

    /* Initializes RAM area */
    RAM_Init();

    /* Update bit pattern to 7th segment */
    db_ret = upd_bitpat(6, (uint8_t*)BPT_BUF, BIT_SEG_SIZE);
    if (DBRAM_OK != db_ret)
```

```
    {
        goto Err_end; /* error */
    }

    return;
```

注意

この関数はビットパターン指定が乱数の場合、システム起動後、少なくとも 1 度実行する必要があります。

4.2 load_seed ()

この関数はデータフラッシュから SEED をロードします。

フォーマット

uint32_t load_seed(void);

パラメータ

なし

戻り値

SEED の値

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

データフラッシュから SEED をロードします。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include "db_ram.h"

    dbram_t db_ret;
    uint32_t seed;

    /* Load seed from data flash */
    seed = load_seed();

    /* Update bit pattern to 7th segment */
    db_ret = upd_seed(&next_seed);
    if (DBRAM_OK != db_ret)
    {
        goto Err_end; /* error */
    }

    return;
```

注意

この関数はビットパターン指定が乱数の場合のみ使用します。

4.3 upd_seed ()

この関数はデータフラッシュの SEED を更新します。

フォーマット

dbram_t upd_seed(uint32_t *dat);

パラメータ

dat – 更新する SEED の値

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_FLERASE: フラッシュ消去エラーが発生しました。

DBRAM_ERR_FLWRITE: フラッシュ書き込みエラーが発生しました。

DBRAM_ERR_FLVERIFY: フラッシュベリファイエラーが発生しました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

データフラッシュの SEED を更新します。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例は「4.2 load_seed ()」に説明しています。

注意

この関数はビットパターン指定が乱数の場合のみ使用します。

4.4 crt_bitpat ()

この関数はビットパターンを生成します。

フォーマット

void crt_bitpat(int8_t *dat, int32_t size);

パラメータ

dat – ビットパターンデータ

size – ビットパターンサイズ

戻り値

なし

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

ビットパターンを生成します。ビットパターン指定により、以下の動作を実行します。

- ビットパターン指定が乱数（BIT_PAT_RAN == BIT_PAT_MODE）の場合、rand()関数により乱数を生成します。
- ビットパターン指定が繰り返しデータ（BIT_PAT_SEQ == BIT_PAT_MODE）の場合、0x00 から 0xFF を 256 バイト単位で繰り返すデータを生成します。
- ビットパターン指定が定数（BIT_PAT_CNT == BIT_PAT_MODE）の場合、定数パターン（CONST_PAT）により設定した定数を生成します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例は「4.1 FlashInit ()」に説明しています。

注意

生成するビットパターンサイズ以上のバッファ領域を確保する必要があります。

4.5 load_bitpat ()

この関数はデータフラッシュからビットパターンのダブル RAM 処理によりロードします。

フォーマット

dbram_t load_bitpat(int32_t seg_no, uint8_t *dat, int32_t size);

パラメータ

seg_no – セグメント番号

dat – ビットパターンデータ

size – ビットパターンサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_FLDBRAM: フラッシュダブル RAM エラーが発生しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

データフラッシュからビットパターンのダブル RAM 処理によりロードします。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include <stdio.h>
#include "db_ram.h"

static uint8_t BUF[4]; /* Bit pattern data buffer */

    dbram_t db_ret;

    /* Load bit pattern from 3rd segment,  */
    db_ret = load_bitpat(2, (uint8_t*)BUF, 4);
    if (DBRAM_OK != db_ret)
    {
        goto Err_end; /* error */
    }

    printf("Bit pattern data = %8x¥n", BUF);

    return;
```

注意

なし。

4.6 upd_bitpat ()

この関数はデータフラッシュのビットパターンをダブル RAM 処理により更新します。

フォーマット

dbram_t upd_bitpat(int32_t seg_no, uint8_t *dat, int32_t size);

パラメータ

seg_no – セグメント番号

dat – ビットパターンデータ

size – ビットパターンサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

DBRAM_ERR_FLERASE: フラッシュ消去エラーが発生しました。

DBRAM_ERR_FLWRITE: フラッシュ書き込みエラーが発生しました。

DBRAM_ERR_FLVERIFY: フラッシュベリファイエラーが発生しました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

データフラッシュのビットパターンをダブル RAM 処理により更新します。以下の動作を実行します。

- ビットパターンのセグメントを消去とブランクチェック後、更新するビットパターンを書き込みます。
- ダブル RAM のセグメントを消去とブランクチェック後、更新するビットパターンをダブル RAM セグメントに書き込みます。
- 更新したビットパターンをベリファイします（ダブル RAM セグメントはベリファイしません）。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例は「4.1 FlashInit ()」に説明しています。

注意

なし。

4.7 ers_bitpat ()

この関数はデータフラッシュのビットパターンを消去します。

フォーマット

dbram_t ers_bitpat(int32_t seg_no, int32_t num_seg);

パラメータ

seg_no – 消去セグメント番号

num_seg – 消去セグメント数

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

DBRAM_ERR_FLERASE: フラッシュ消去エラーが発生しました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

データフラッシュのビットパターンをダブル RAM のブロックとともに消去します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include <stdio.h>
#include "db_ram.h"

    dbram_t db_ret;

    /* Erase bit pattern of 4th segment,  */
    db_ret = ers_bitpat(3, 1);
    if (DBRAM_OK != db_ret)
    {
        goto Err_end; /* error */
    }

    printf("Bit pattern erased of 4th segment¥n");

    return;
```

注意

この関数はビットパターンデータに加え、ダブル RAM データも消去します。

4.8 exec_bitop ()

この関数はビット操作を実行します。

フォーマット

dbram_t exec_bitop(int32_t seg_no, int8_t *dat, uint16_t size);

パラメータ

seg_no – セグメント番号

dat – ビット操作データ

size – ビット操作データサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_FLDBRAM: フラッシュダブル RAM エラーが発生しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

ダブル RAM 処理により読み出したビットパターンを使用し、ビット操作を実行します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include <string.h>
#include "db_ram.h"

static int8_t USER_BUF[1024];

#pragma section _RAW_AREA
static int8_t R_BUF[NUM_RAW_REC][RAW_REC_SIZE]; /* Raw data area (8*1024) byte */
#pragma section

#pragma section _INV_AREA
static int8_t I_BUF[NUM_INV_REC][INV_REC_SIZE]; /* Invert data area (8*1024) byte
*/
#pragma section

    int32_t ret;
    int32_t *dat;

    /* Compare 1st record raw data with inverted data by 1KB */
    ret = inv_comp_data(R_BUF[0], &(I_BUF[0][BIT_SEG_SIZE-1]), 1024);
    if ((-1) != ret)
    {
       ErrPtr = ret; /* set error byte */
        goto Err_end; /* compare error */
    }
```

```
    /* Set user buffer pointer */
    dat = USER_BUF;

    /* Set raw data */
    memcpy(dat, R_BUF[0], 1024);

    /* Execute bit operation */
    _exec_bitop(rec_no, dat, size);

    return;
```

注意

ビット操作の種別がビット操作なし（OP_NONE == KND_BIT_OP）の場合、ビット操作を実行しません。

4.9 RAM_Init ()

この関数は RAM を初期化します。

フォーマット

void RAM_Init(void);

パラメータ

なし

戻り値

なし

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

この関数は以下の RAM 領域を初期化します。

- 非安全領域：領域(1)、領域(2)、領域(3)。
- 安全領域：生データ領域とビット反転データ領域。

リエントラント

この関数はリエントラントです。

使用例

この関数の使用例は「4.1 FlashInit ()」に説明しています。

注意

なし。

4.10 RAM_Write ()

この関数は非安全データを RAM に書き込みます。

フォーマット

dbram_t RAM_Write(int32_t area_no, int8_t *dat, uint16_t size);

パラメータ

area_no – 領域番号

dat – 書き込みデータ

size – 書き込みデータサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

非安全データを指定した領域番号の RAM に指定したサイズだけ書き込みます。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include <stdio.h>
#include "db_ram.h"

static int8_t W_BUF[1024]; /* write buffer */
static int8_t R_BUF[1024]; /* read buffer */

   dbram_t db_ret;
   int32_t i;

   /* Write non safe data to Area(1) by 1KB */
   db_ret = RAM_Write(1, W_BUF, sizeof(W_BUF));
   if (DBRAM_OK != db_ret)
   {
      printf("Write error occurred¥n");
      goto Err_end;
   }

   /* ==== Add user operation ==== */

   /* Read non safe data from Area(1) by 1KB */
   db_ret = RAM_Read(1, R_BUF, sizeof(R_BUF));
   if (DBRAM_OK != db_ret)
   {
      printf("Read error occurred¥n");
      goto Err_end;
   }
```

```
/* Compare read and write data by 1KB */
for (i = 0; i < 1024; i++)
{
    if (R_BUF[i] != W_BUF[i])
    {
          printf("Data error detected at %d¥n", i);
        goto Err_end; /* compare error */
    }
}

printf("Non safe data access completed¥n");

return;
```

注意

なし。

4.11 RAM_SafetyWrite ()

この関数は安全データを RAM に書き込みます。

フォーマット

dbram_t RAM_SafetyWrite(int32_t rec_no, int8_t *dat, uint16_t size);

パラメータ

rec_no – レコード番号

dat – 書き込みデータ

size – 書き込みデータサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_FLDBRAM: フラッシュダブル RAM エラーが発生しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

安全データを指定したレコードの RAM に指定したサイズだけ書き込みます。以下の動作を実行します。

- ダブル RAM 処理によりロードしたビットパターンを使用しビット操作を実行します。
- ビット操作したデータを生レコードに書き込みます。
- ビット操作後、ビット反転したデータを反転レコードに書き込みます。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例を以下に示します。

```
#include <stdio.h>
#include "db_ram.h"

static int8_t W_BUF[128]; /* write buffer */
static int8_t R_BUF[128]; /* read buffer */

    dbram_t db_ret;
    int32_t i;

    /* Write safe data to record(6)by 128B */
    db_ret =  RAM_SafetyWrite (6, W_BUF, sizeof(W_BUF));
    if (DBRAM_OK != db_ret)
    {
       printf("Write error occurred¥n");
       goto Err_end;
    }

    /* ==== Add user operation ==== */

    /* Read safe data from record(6)by 128B */
```

```
db_ret = RAM_SafetyRead(6, R_BUF, sizeof(R_BUF));
if (DBRAM_OK != db_ret)
{
   printf("Read error occurred¥n");
   goto Err_end;
}

/* Compare read and write data by 128B */
for (i = 0; i < 1024; i++)
{
    if (R_BUF[i] != W_BUF[i])
   {
         printf("Data error detected at %d¥n", i);
       goto Err_end; /* compare error */
    }
}

printf("Safe data access completed¥n");

return;
```

注意

なし。

4.12 RAM_Read ()

この関数は非安全データを RAM から読み出します。

フォーマット

dbram_t RAM_Read(int32_t area_no, int8_t *dat, uint16_t size);

パラメータ

area_no – 領域番号

dat – 読み出しデータ

size – 読み出しデータサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

非安全データを指定した領域番号の RAM から指定したサイズだけ読み出します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例は「4.10 RAM_Write ()」に説明しています。

注意

なし。

4.13 RAM_SafetyRead ()

この関数は安全データを RAM から読み出します。

フォーマット

dbram_t RAM_SafetyRead(int32_t rec_no, int8_t *dat, uint16_t size);

パラメータ

rec_no – レコード番号

dat – 読み出しデータ

size – 読み出しデータサイズ

戻り値

DBRAM_OK: 処理は正常に終了しました。

DBRAM_ERR_FLDBRAM: フラッシュダブル RAM エラーが発生しました。

DBRAM_ERR_PARAM: パラメータに誤りがありました。

DBRAM_ERR_CMP: ダブル RAM 処理でソフトエラーを検出しました。

プロパティ

db_ram.h にプロトタイプ宣言しています。

説明

安全データを指定したレコードの RAM から指定したサイズだけ読み出します。以下の動作を実行します。

- 生レコードから読み出したデータと反転レコードから逆のアドレス順に読み出しビット反転したデータを比較します。そして、不一致がある場合、ソフトエラーとして検出します。
- 生レコードから読み出したデータをユーザのバッファにコピーします。
- ダブル RAM 処理により読み出したビットパターンを使用し、ビット操作を実行します。

リエントラント

関数はリエントラントです。

使用例

この関数の使用例は「4.11RAM_SafetyWrite ()」に説明しています。

注意

なし。

5. 参考資料

ユーザーズマニュアル: ハードウェア

RX64M グループユーザーズマニュアル ハードウェア編 Rev.1.00（R01UH0377JJ）
RX71M グループユーザーズマニュアル ハードウェア編 Rev.1.00（R01UH0493JJ）
最新版はルネサス エレクトロニクスのウェブサイトからダウンロードできます。

ユーザーズマニュアル: ソフトウェア

RX ファミリ RXv2 命令セットアーキテクチャ ユーザーズマニュアル ソフトウェア編（R01US0071JJ）
最新版はルネサス エレクトロニクスのウェブサイトからダウンロードできます。

技術情報/ニュース

最新版はルネサス エレクトロニクスのウェブサイトからダウンロードできます。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.08.10	—	初版発行

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。

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。

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/