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 ファミリ

## 調歩同期式シリアル通信による ユーザプログラムモードフラッシュメモリ書き換え

### 要旨

マスタ側フラッシュメモリ上にある書き換えデータを，スレーブ側のフラッシュメモリに書き込みます。書き換えデータの転送は，調歩同期式シリアル通信で行ないます。

### 動作確認デバイス

H8S/2268F

### 目次



**注意事項：レジスタ名は，プログラム内の設定名を使用しています。**

## 1. 仕様

(1) ユーザプログラムモードによるフラッシュメモリ書き換えを行ないます。
(2) マスタ側フラッシュメモリ上にある書き換えデータを，スレーブ側のフラッシュメモリに書き込みます。
(3) 書き換えデータの転送は，調歩同期式シリアル通信で行ない，SCI チャネル 0 (SCI_0) を使用します。
(4) マスタ側のスイッチ 0 (SW0) が ON になると, マスタ側からスレーブ側にフラッシュメモリ書き換え開始コマンドを送信し，スレーブ側フラッシュメモリの書き換えを開始します。
(5) マスタ側，スレーブ側共に，フラッシュメモリ書き換え動作中は LED1 が消灯，LED2 が点灯し，フラッシュメモリ書き換え動作終了後は LED1 が点灯，LED2 が消灯します。
(6) マスタ側の $\overline{\text{IRQ0}}$ 端子にスイッチ 0 (SW0) が接続されています。
(7) マスタ側は，出力ポートに LED1 と LED2 が接続されています。
(8) スレーブ側は，P10 出力端子に LED1 が接続，P11 出力端子に LED2 が接続されています。
(9) オンボード書き換え回路の構成例を図 1 に示します。

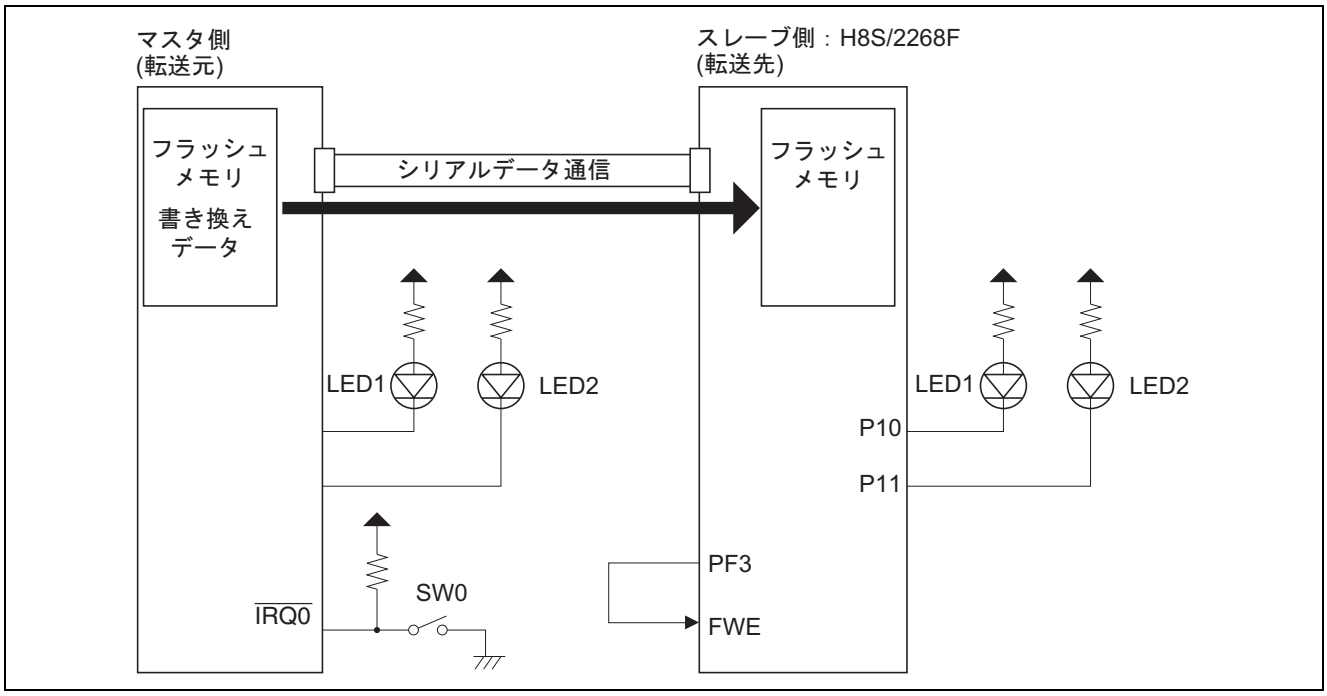



**図 1　オンボード書き換え回路構成例**

## 2. 適用条件

H8S ファミリ調歩同期式シリアル通信によるユーザプログラミングモードフラッシュメモリ書き換え(アプリケーションノート)のコンパイル条件です。

尚､コンパイラのバージョンやソースプログラムの作り方によっては的確な時間(wait 等)が得られない場合がありますので、コンパイルによって出力されたコードは必ずチェックする様、お願い致します。

**表1　適用条件**

| 項目 | 内容 |
|---|---|
| **動作周波数** | **入力クロック**: 10MHz<br>**システムクロック**: 10MHz<br>**周辺モジュールクロック**: 10MHz |
| **動作モード** | **モード** 7 (MD2 = 1, MD1 = 1, FEW = 0) |
| **オンボードプログラミングモード** | **ユーザプログラムモード** (MD2 = 1, MD1 = 1, FEW = 1) |
| C/C++**コンパイラ** | **ルネサス テクノロジ製**<br>H8S, H8/300 SERIES C/C++ Compiler Ver6.01.01 |
| **コンパイルオプション** | -cpu = 2000a, -code = machinecode, -optimize = 1, -regparam = 3<br>-speed = (register, shift, struct, expression) |

**表2　セクション設定**

| アドレス | セクション名 | 説明 |
|---|---|---|
| H'000000 | CV1 | **リセットルーチン** |
| H'001000 | P | **メインプログラム領域** |
| H'000400 | PASSCI | **調歩同期式シリアル通信プログラム領域** |
| H'001000 | DSMPL1 | **サンプルデータテーブル** 1 |
| H'004000 | DSMPL2 | **サンプルデータテーブル** 2 |
| H'007FF6 | DSMPL3 | **サンプルデータテーブル** 3 |
| H'008000 | PCPYFZRAM | **書き込み/消去プログラムの** RAM **転送プログラムの格納領域** |
| H'008100 | FZTAT<br>PFZTAT<br>DFZTAT<br>FZEND | **書き込み/消去プログラム領域** * |
| H'FFB000 | RAM<br>PRAM<br>DRAM | **書き込み/消去プログラムの** RAM **転送先** * |

**【注】** * ROM **化支援オプションの指定**
ROM **から** RAM **へプログラムを転送して，**RAM **上から実行するには，リンカの** ROM **化支援オプションを設定する必要があります。本タスク例の** ROM **化支援オプション設定例を以下に示します。**
rom = PFZTAT = PRAM,
DFZTAT = DRAM

# 3. 詳細仕様説明

## 3.1 オンボードプログラミング動作条件

- デバイス：HD64F2268 (H8S/2268F)
- CPU 動作：ユーザプログラムモード
- 動作電圧：3.3V
- 動作周波数：10MHz

## 3.2 オンボードプログラミングモード

- ユーザプログラムモード
  スレーブ側 MCU のフラッシュメモリには，書き込み/消去制御プログラム，書き換え開始コマンド受信プログラム，RAM 転送プログラム，FWE 制御判定プログラムがブートモードもしくはライタモードであらかじめ書き込んであることを前提条件とします。

## 3.3 書き込み方式

- 転送元から書き換えデータを受信し，フラッシュメモリを書き換えます。
- 転送元との通信は，調歩同期式シリアル通信で行ない，SCI チャネル 0 (SCI_0) を使用します。マスタを転送元とし，スレーブを転送先とします。

## 3.4 書き換え手順のフローチャート

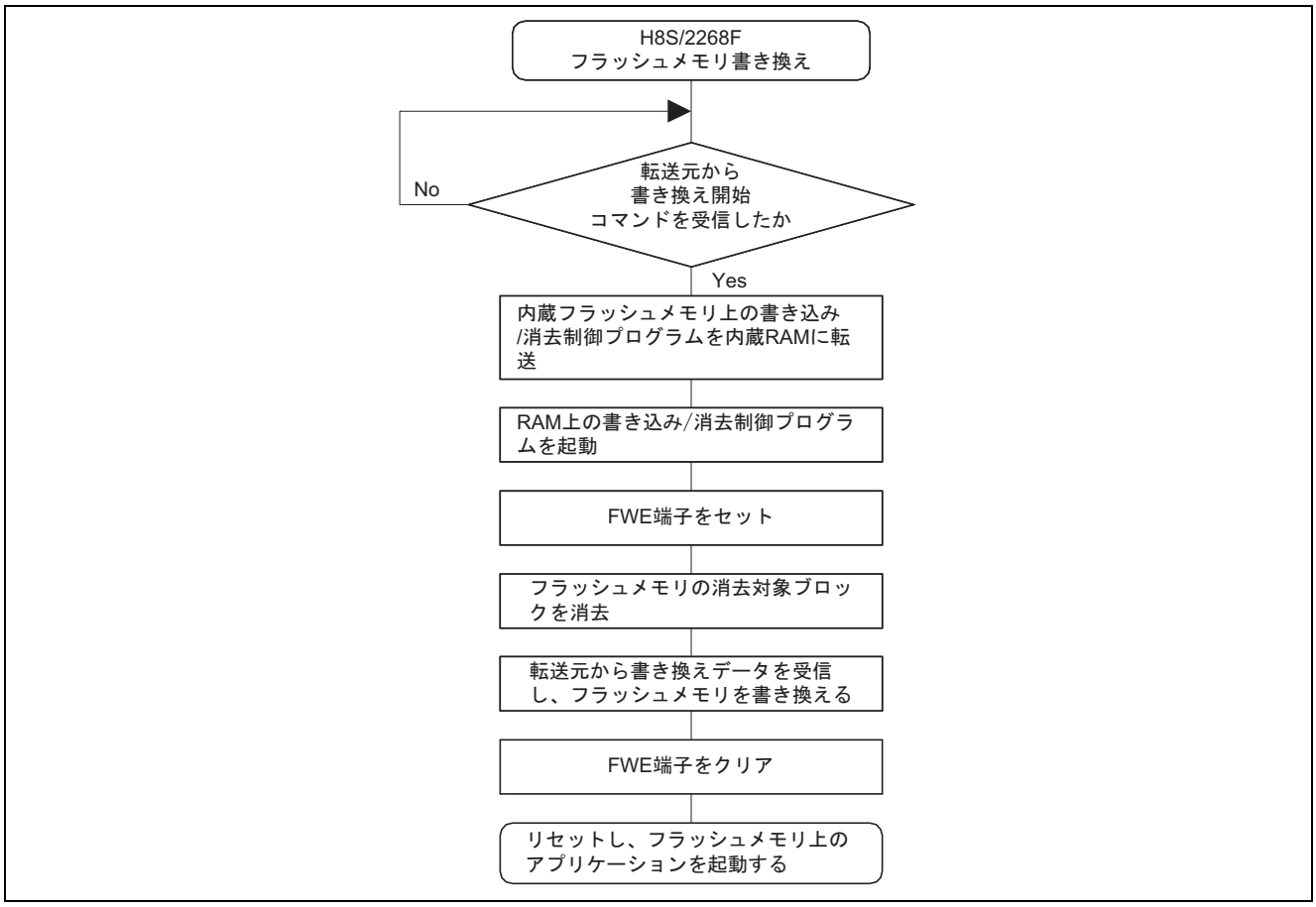



**図 2　ユーザプログラムモード書き換え手順**

## 3.5 マスタスレーブ間の接続図

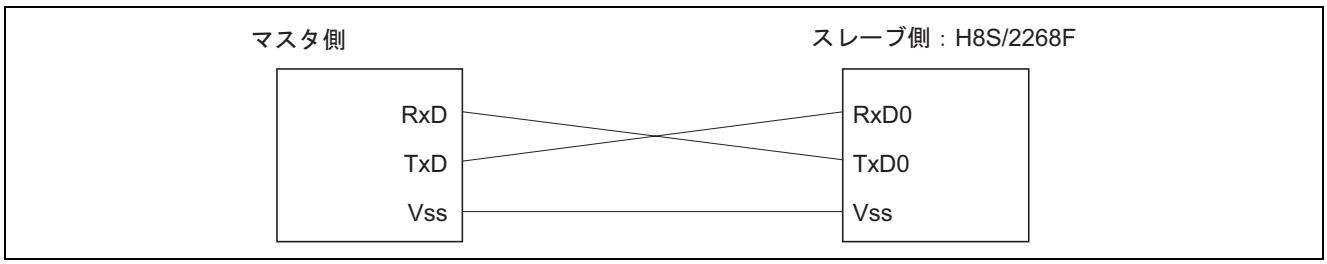



図 3　マスタスレーブ間の接続図

## 3.6 通信仕様

表 3　通信仕様

| 転送速度 | 31250bps |
|---|---|
| 通信方式 | 調歩同期式シリアル通信 |
| データビット | 8 |
| ストップビット | 1 |
| パリティ | なし |

## 3.7 通信コマンド

表 4　通信コマンド

| 通信コマンド | 説明 |
|---|---|
| H'00 | 通信正常（コマンド名: OK コマンド） |
| H'01 | 通信異常（コマンド名: NG コマンド） |
| H'11 | 送信開始要求 |
| H'55 | 書き換え開始コマンド |
| H'66 | FWE 端子設定コマンド |
| H'77 | 消去コマンド |
| H'88 | 書き込みコマンド |

## 3.8 メモリ配置

H8S/2268F のフラッシュメモリ消去ブロックを表 5 に示します。

**表 5　フラッシュメモリ消去ブロック**

| ブロック（サイズ） | アドレス |
|---|---|
| EB0 (4K バイト) | H'000000～H'000FFF |
| EB1 (4K バイト) | H'001000～H'001FFF |
| EB2 (4K バイト) | H'002000～H'002FFF |
| EB3 (4K バイト) | H'003000～H'003FFF |
| EB4 (4K バイト) | H'004000～H'004FFF |
| EB5 (4K バイト) | H'005000～H'005FFF |
| EB6 (4K バイト) | H'006000～H'006FFF |
| EB7 (4K バイト) | H'007000～H'007FFF |
| EB8 (32K バイト) | H'008000～H'00FFFF |
| EB9 (64K バイト) | H'010000～H'01FFFF |
| EB10 (64K バイト) | H'020000～H'02FFFF |
| EB11 (64K バイト) | H'030000～H'03FFFF |

H8S/2268F の通常動作時，フラッシュメモリ書き換え動作時のメモリマップを図 4 に示します。

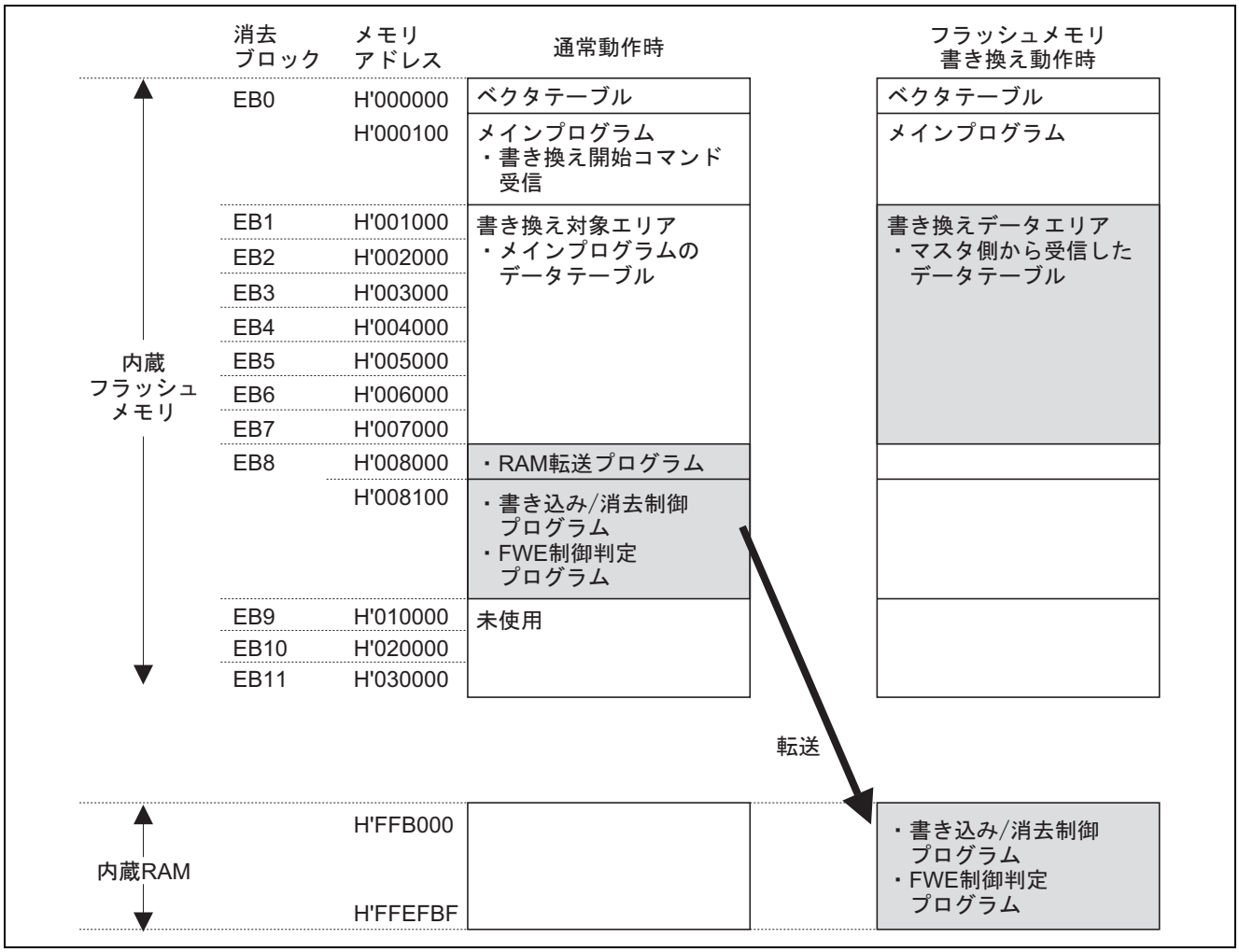



**図 4　メモリマップ (スレーブ側)**

# 4. 動作概要

## 4.1 通常動作

(1) 通常アプリケーションは，フラッシュメモリ上のデータテーブルをアクセスするものとし，データテーブルは，マスタ側から受信し，書き換えます。
(2) スレーブ側のフラッシュメモリ上に，書き込み/消去制御プログラム，書き換え開始コマンド受信プログラム，RAM 転送プログラム，FWE 制御判定プログラムをあらかじめ書き込んでおきます。
(3) マスタ，スレーブ間のデータ通信は，調歩同期式シリアル通信で行ない，SCI チャネル 0 (SCI_0) を使用します。
(4) スレーブ側は，P10 出力端子に LED1, P11 出力端子に LED2 が接続されています。P10, P11 が High レベルのとき，LED1, LED2 は消灯します。P10, P11 が Low レベルのとき，LED1, LED2 は点灯します。

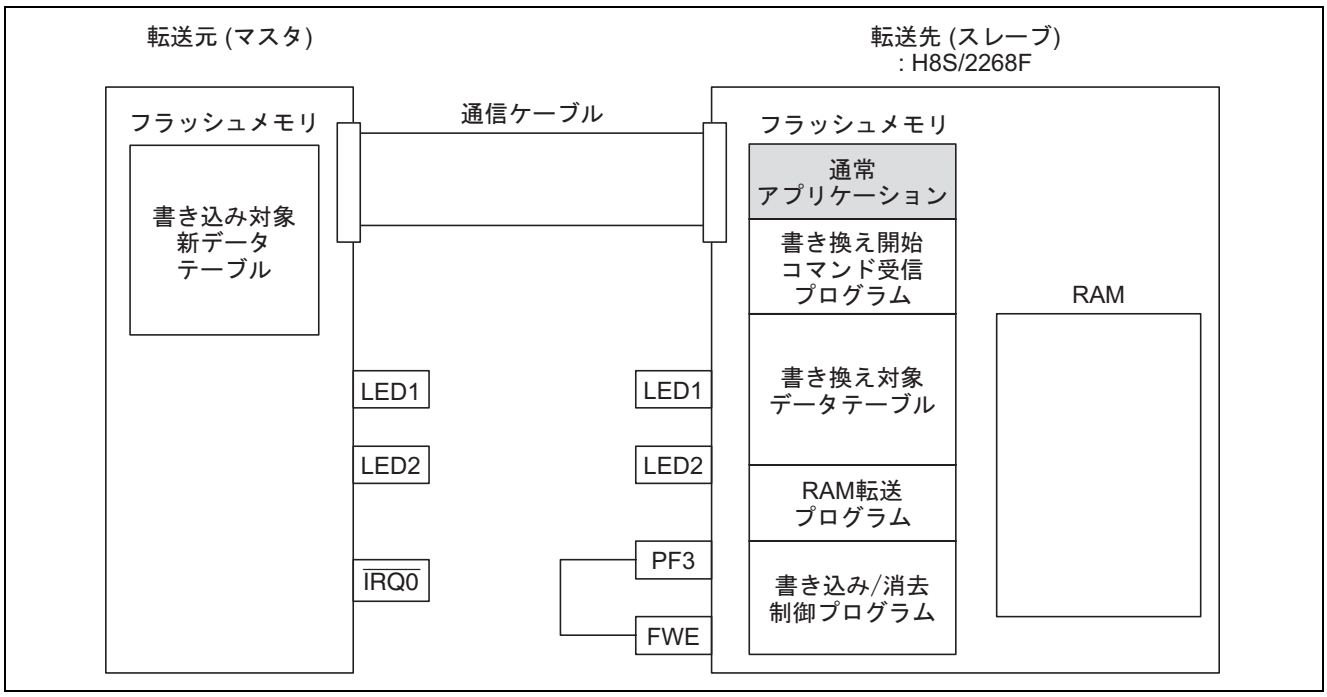



**図 5 通常動作**

## 4.2 オンボード書き換え準備

(1) マスタ側の $\overline{\text{IRQ0}}$ 端子で立ち上がりエッジが検出されると，マスタ側から書き換え開始コマンド H'55 を送信します。
(2) このとき，マスタ側の LED1 は消灯，LED2 は点灯します。

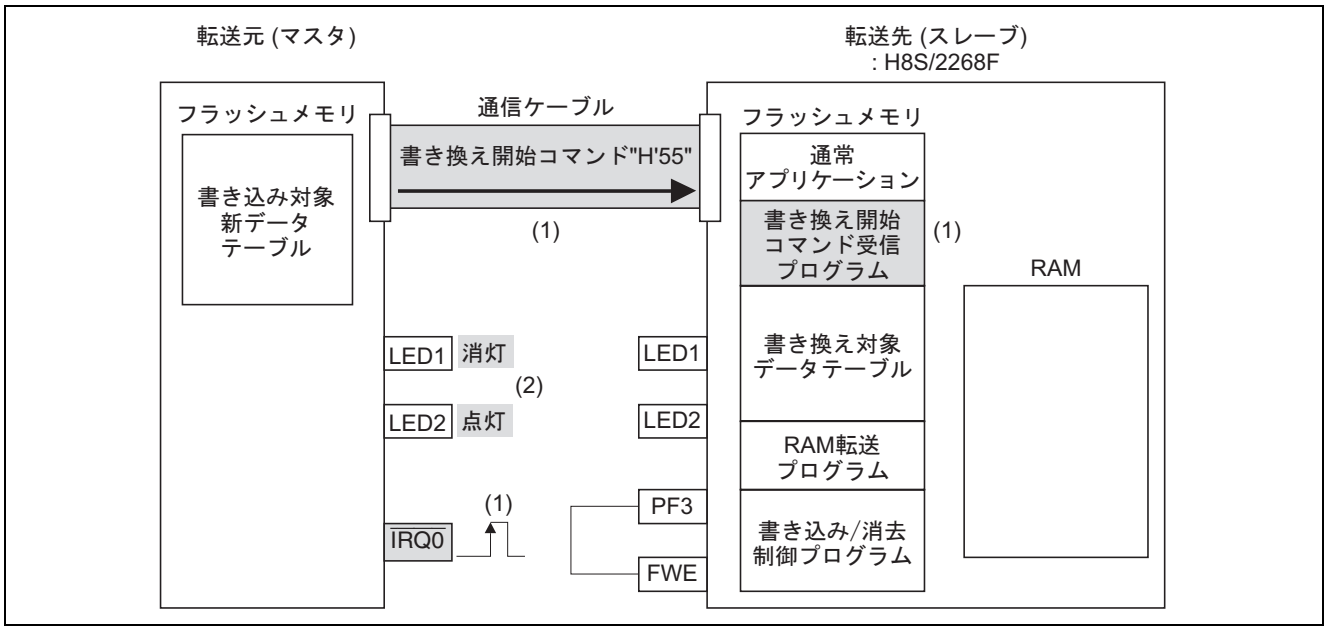



**図 6　オンボード書き換え準備**

## 4.3 オンボード書き換え開始

(1) スレーブ側は H'55 を受信すると，RAM 転送プログラムを起動し，書き込み/消去制御プログラムを内蔵 RAM に転送します。
(2) このとき，スレーブ側の LED1 は消灯，LED2 は点灯します。

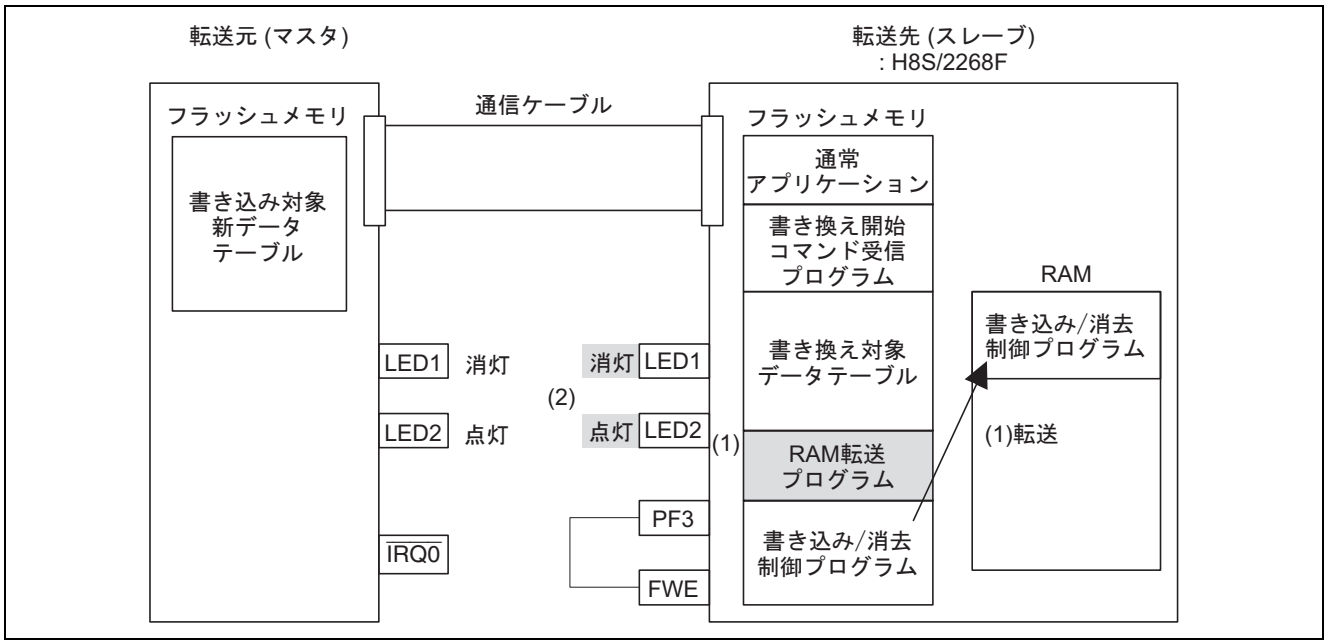



**図 7　オンボード書き換え開始**

## 4.4 書き込み/消去制御プログラム起動

(1) RAM 転送プログラムによる転送終了後，RAM 上の書き込み/消去制御プログラムへ分岐します。

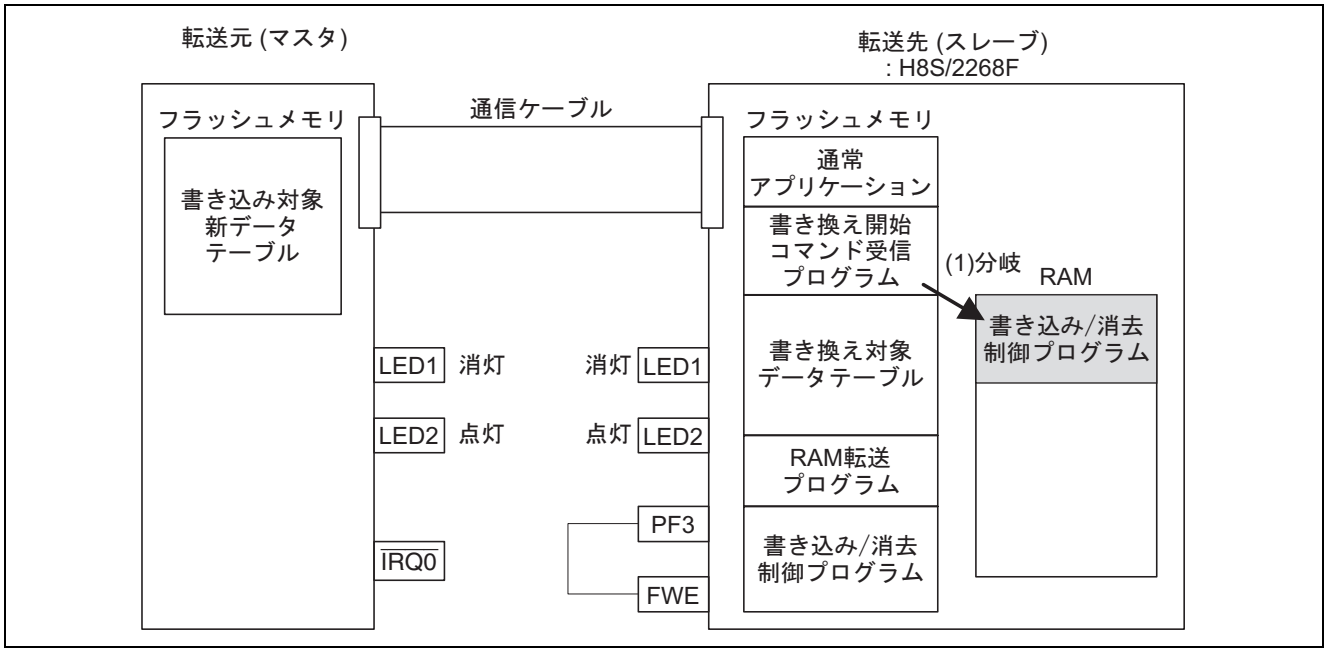



**図 8　書き込み/消去制御プログラム起動**

## 4.5 FWE 端子設定

(1) 転送元から FWE 端子設定コマンド H'66 を受信します。
(2) 書き込み/消去制御プログラムは PF3 を制御し，FWE 端子を 1 に設定します。

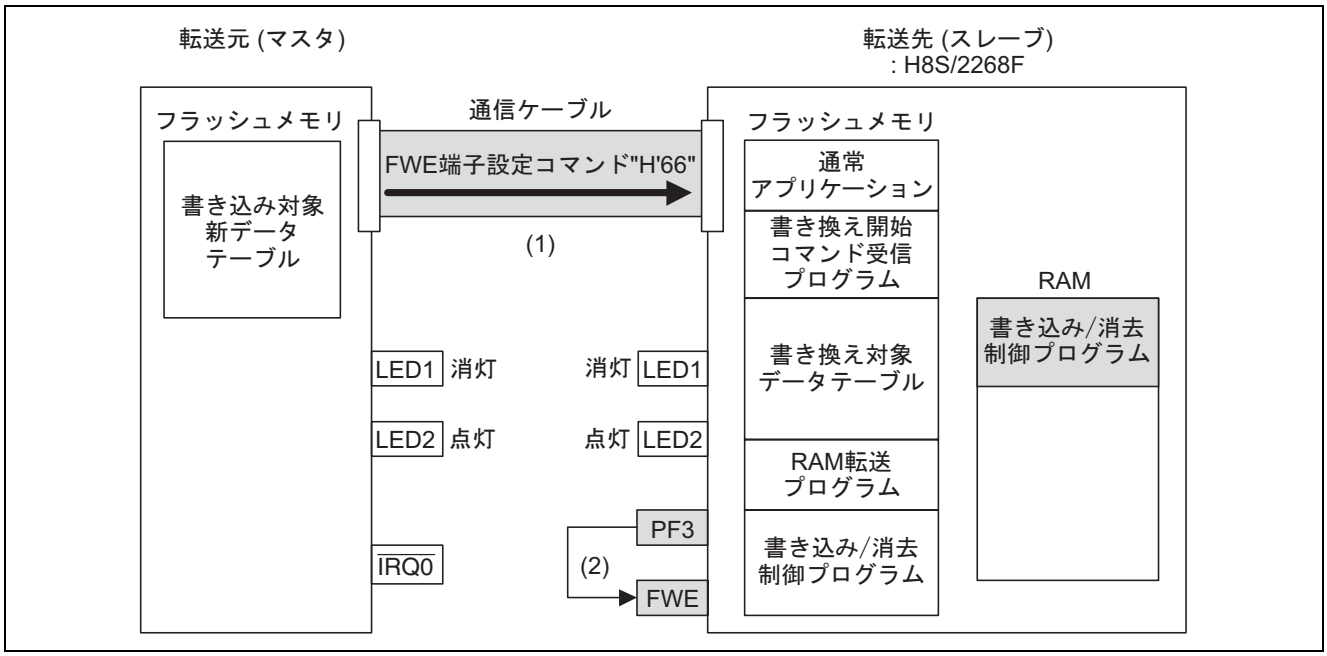



**図 9　FWE 端子設定**

## 4.6 フラッシュメモリ消去

(1) マスタ側から消去コマンド H'77 を受信します。
(2) 書き込み/消去制御プログラムは，フラッシュメモリの消去対象ブロックを消去します。

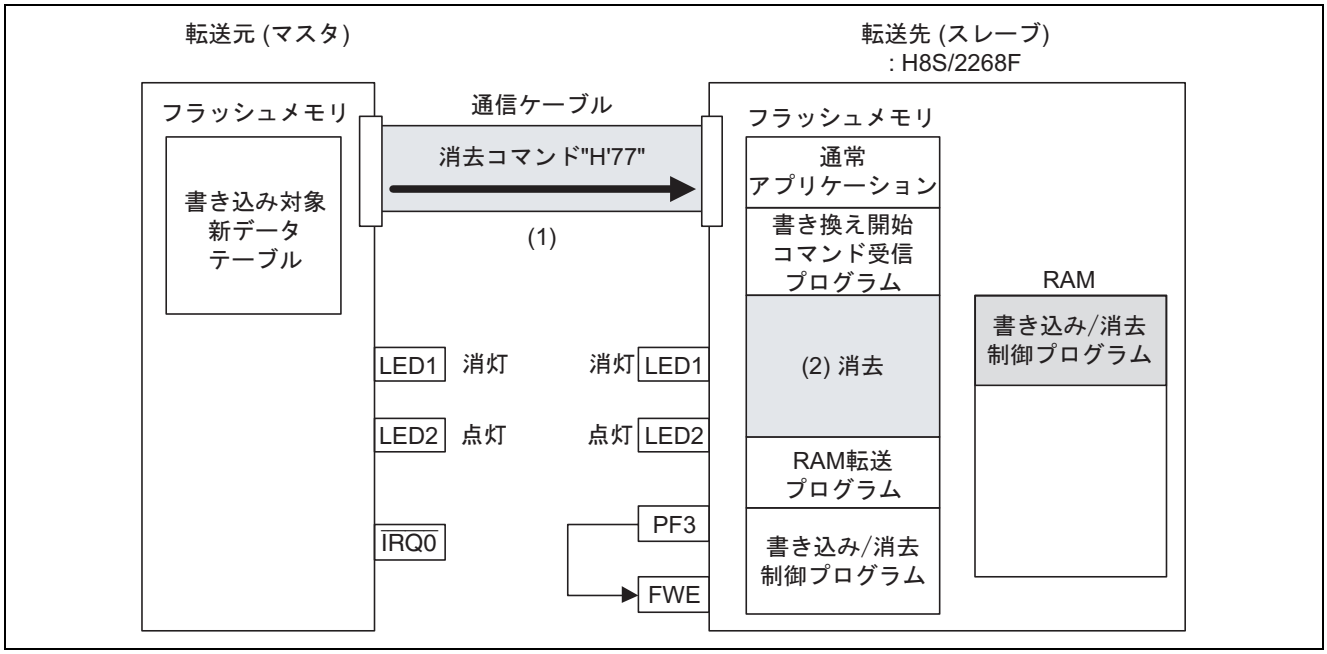



**図 10　フラッシュメモリ消去**

## 4.7 フラッシュメモリ書き込み

(1) 転送元から書き込みコマンド H'88 を受信します。
(2) 書き込み/消去制御プログラムは，転送元から新データテーブルを受信し，フラッシュメモリに書き込みます。
(3) 書き込み終了後，マスタ，スレーブ共に LED1 は点灯，LED2 は消灯します。

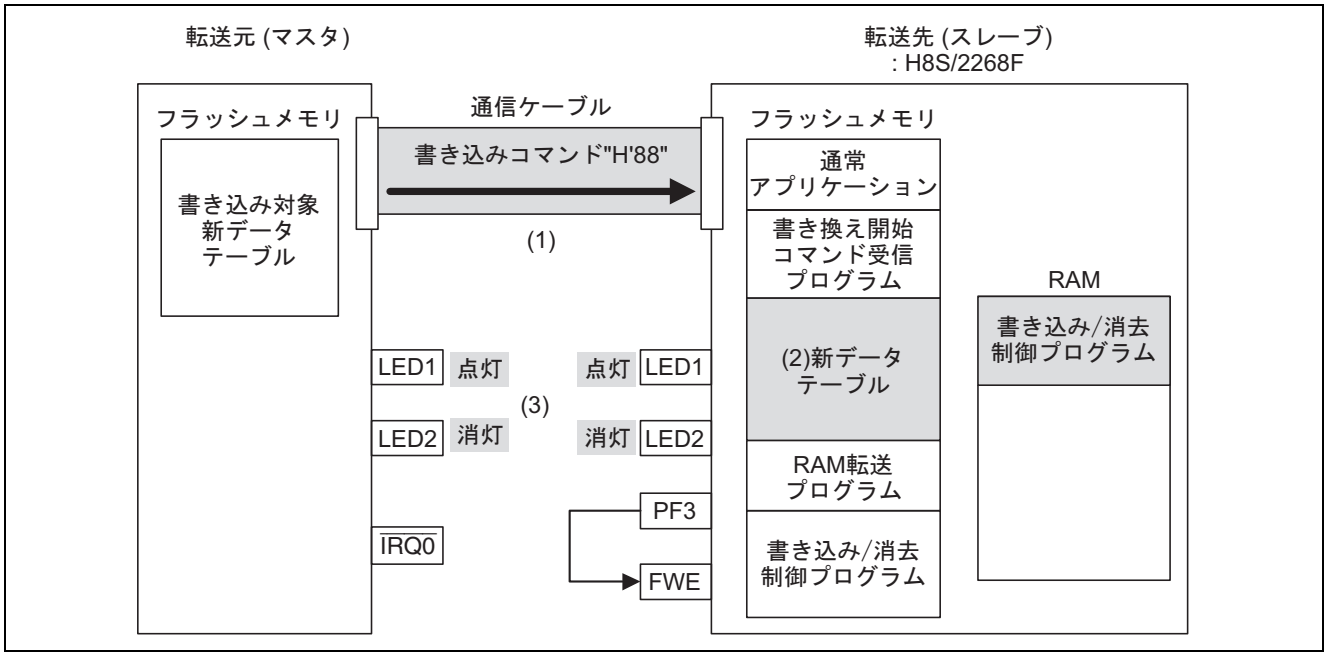



**図 11　フラッシュメモリ書き込み**

## 4.8 FWE 端子解除

(1) 書き込み/消去制御プログラムは PF3 を制御し，FWE 端子を 0 に設定します。

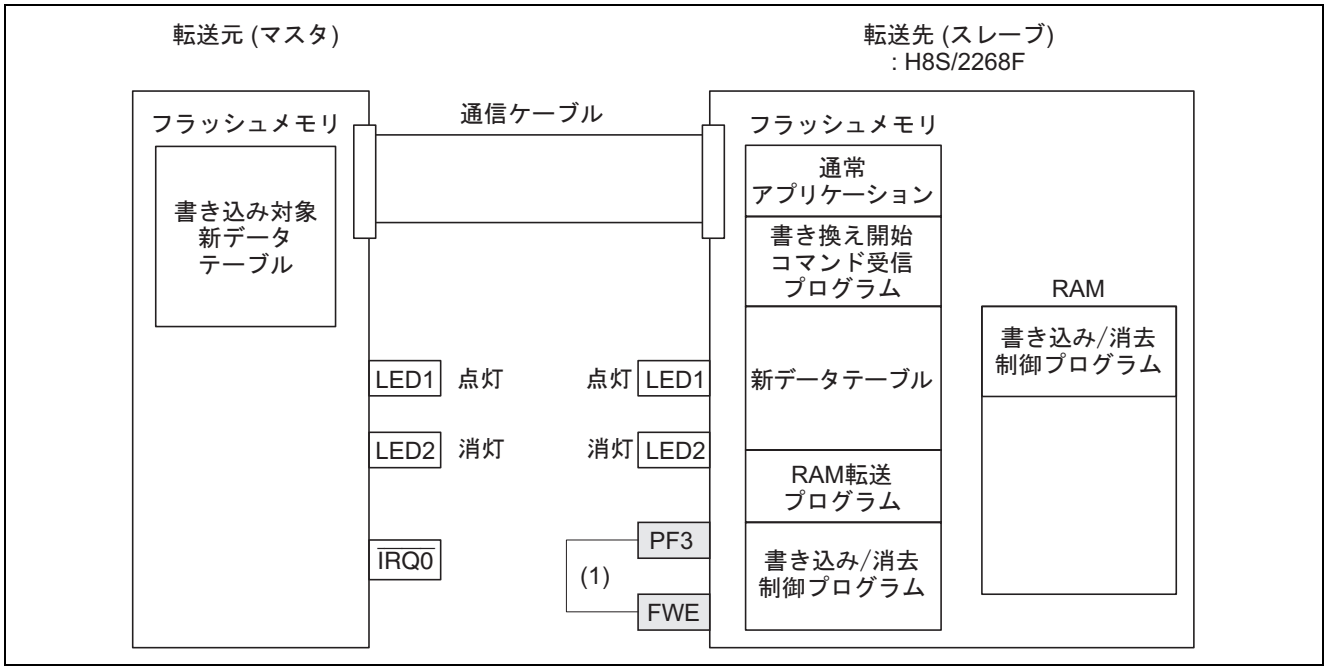



**図 12　FWE 端子解除**

## 4.9 プログラム起動

(1) リセットし，新データテーブルをアクセスする通常アプリケーションを起動します。

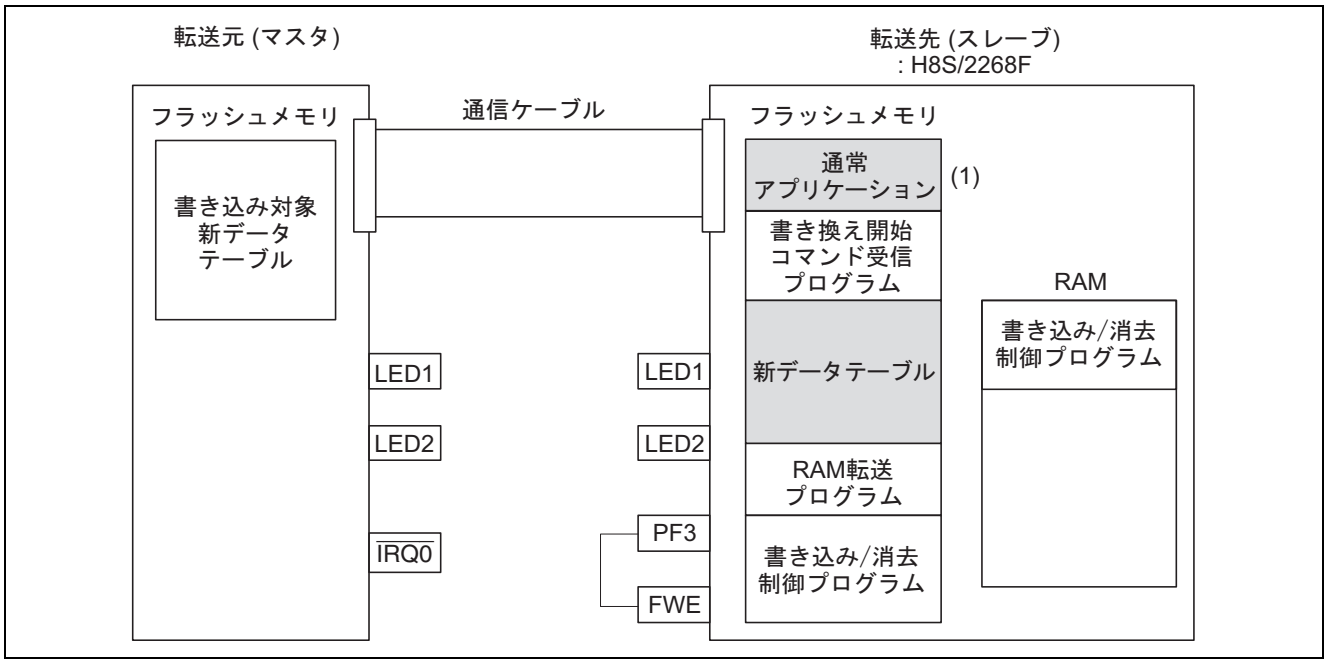



**図 13　プログラム起動**

## 5. シーケンス図

(1) 通常時

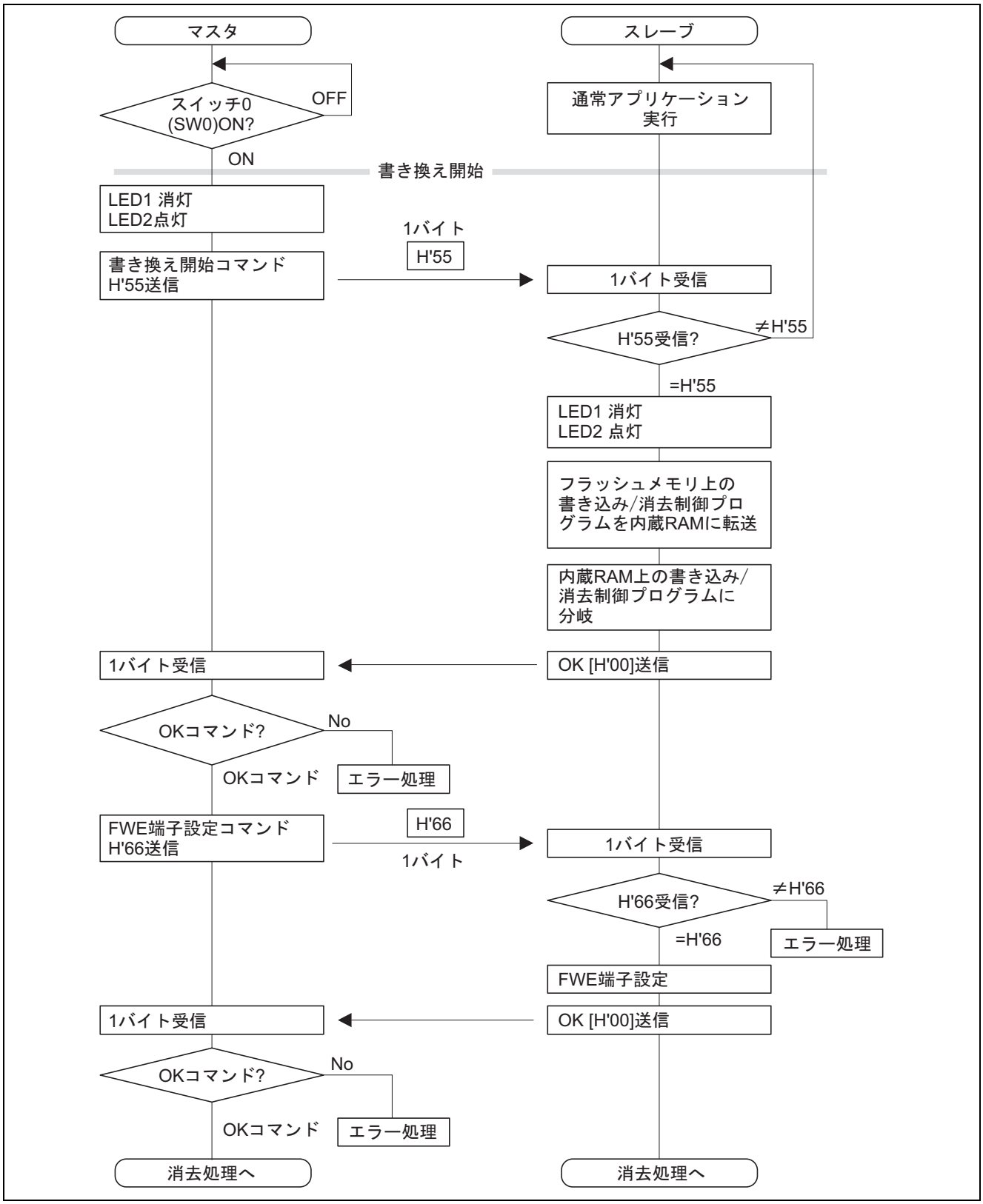



**図 14　通常動作**

(2) 消去処理

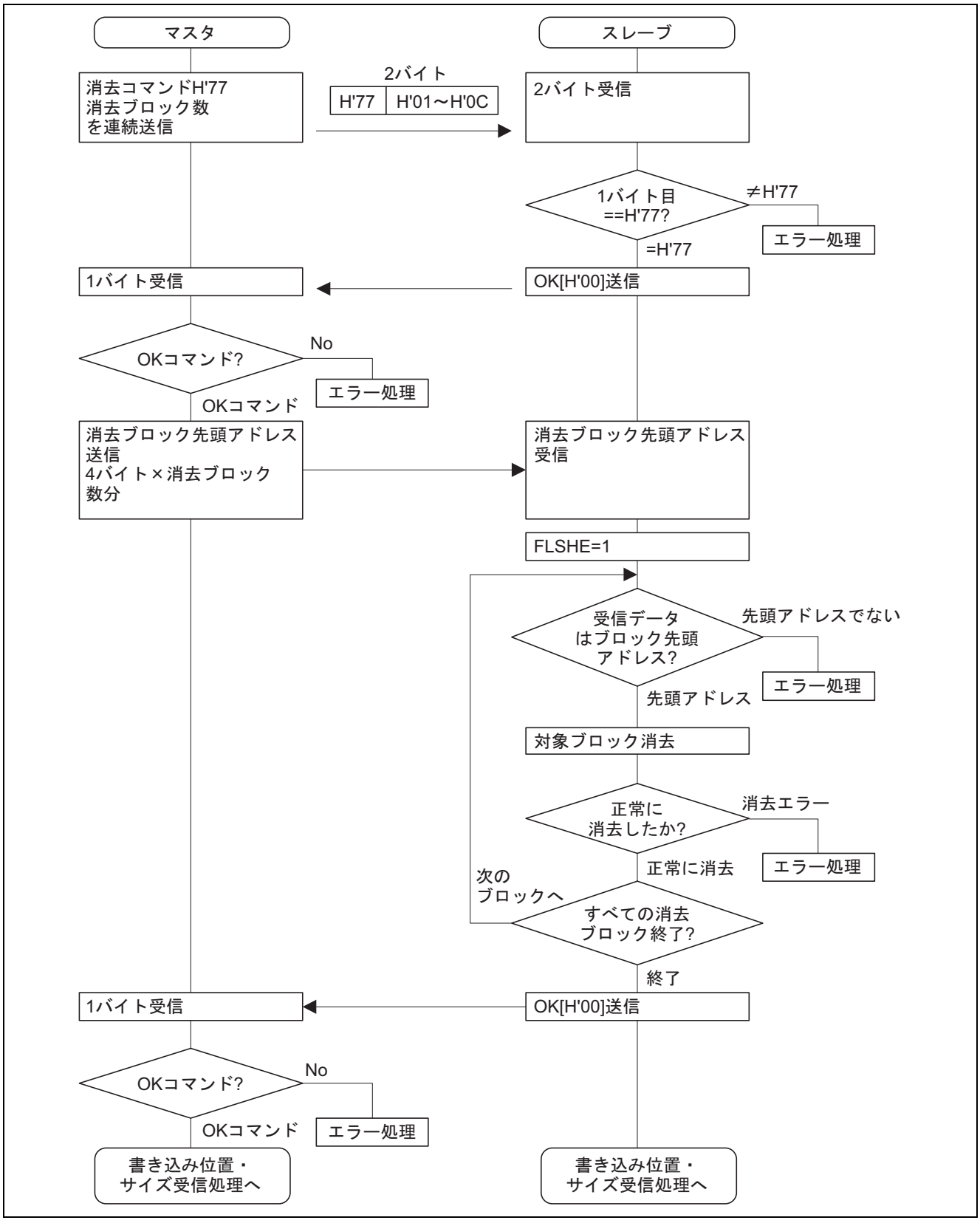



**図 15　消去処理**

(3) 書き込み位置・サイズ受信処理

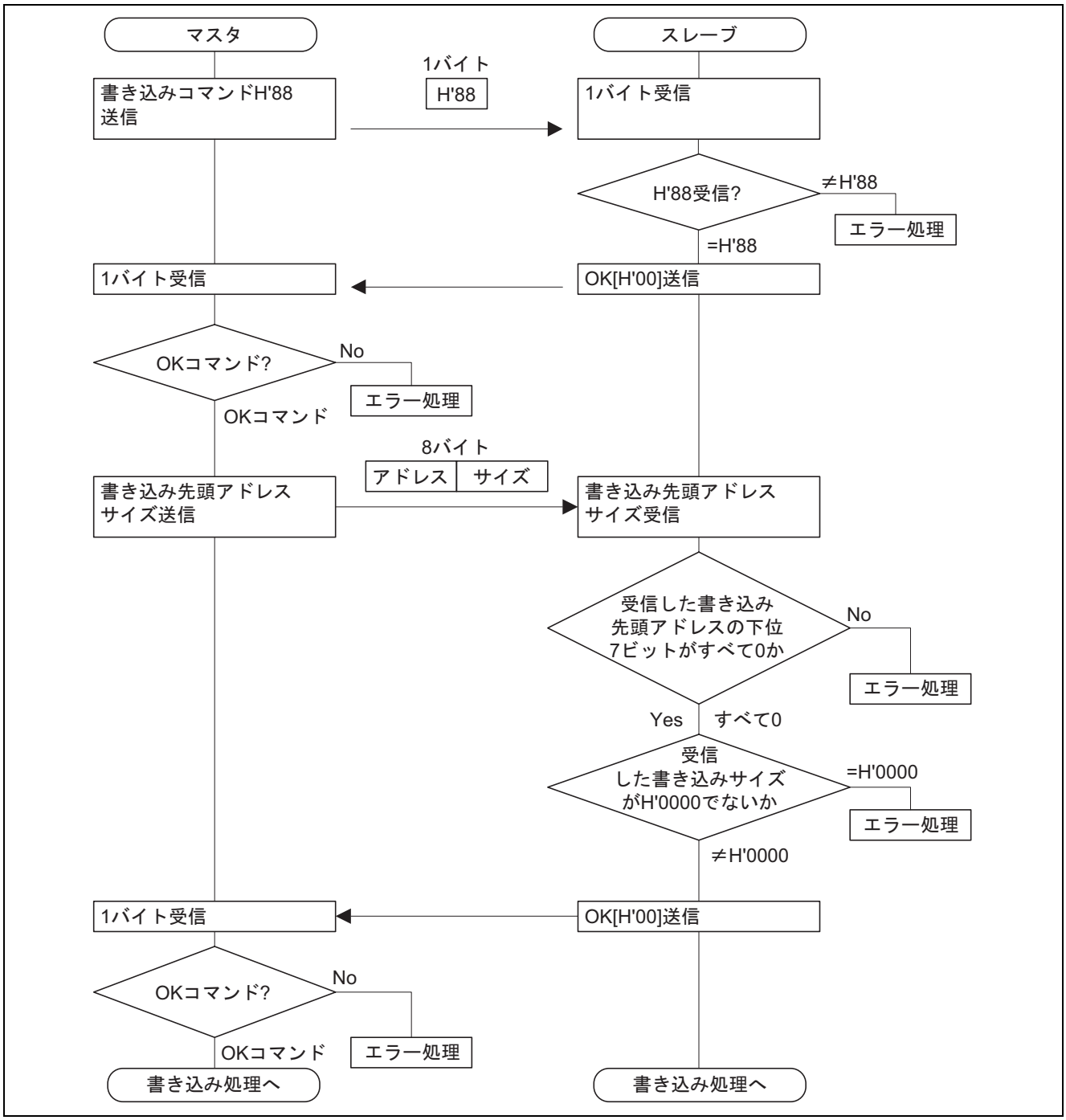



**図 16　書き込み位置・サイズ受信処理**

(4) 書き込み処理

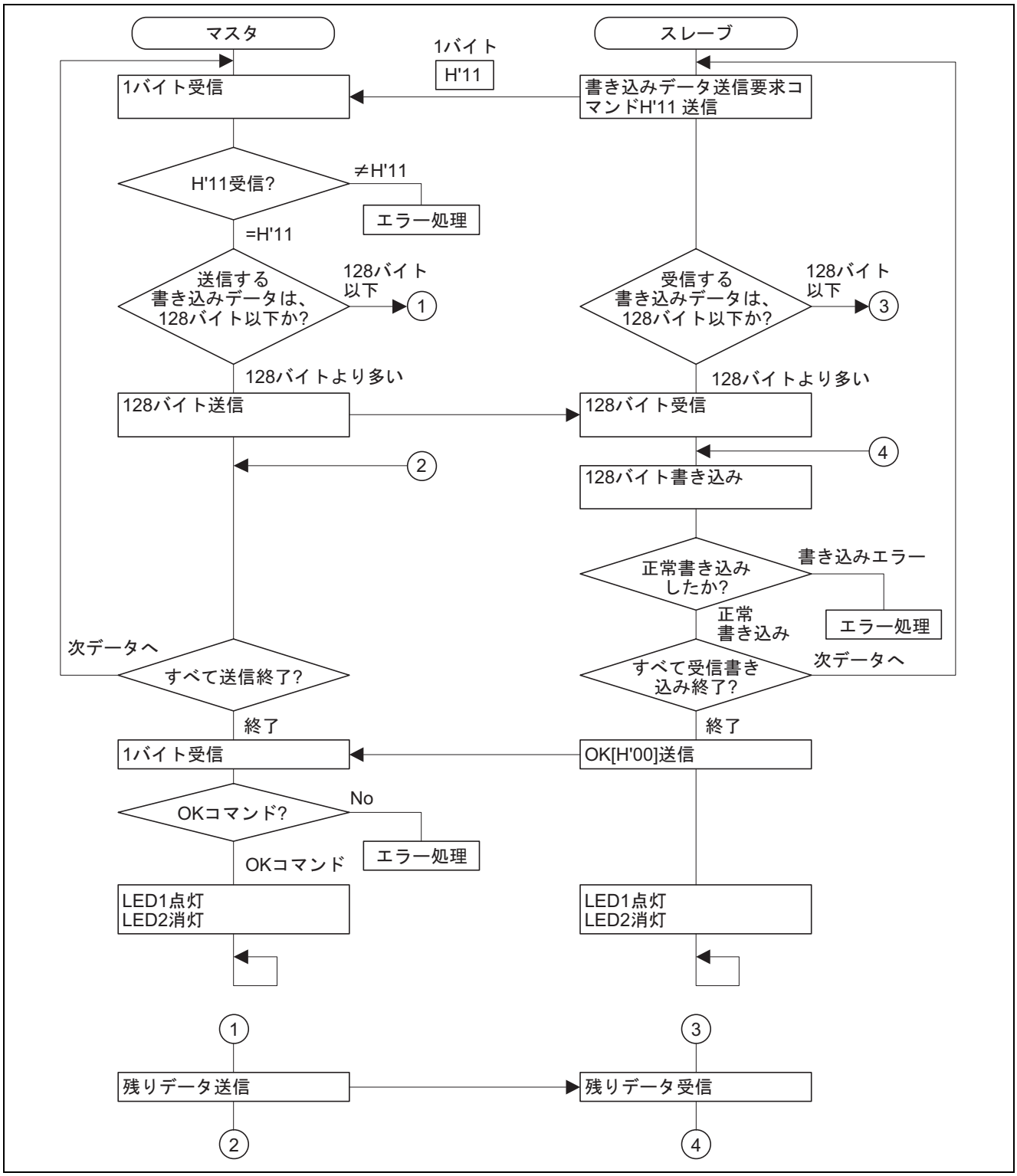



**図 17　書き込み処理**

(5) エラー処理

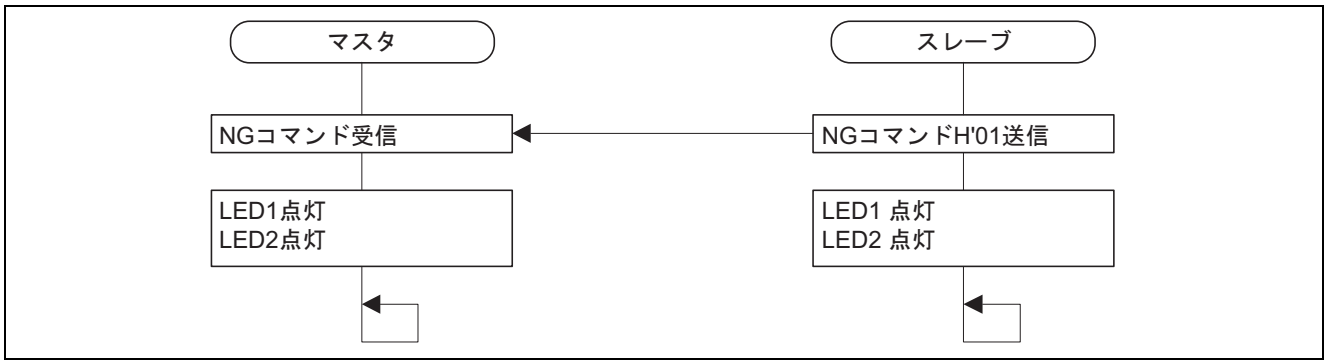



**図 18　エラー処理**

# 6. スレーブ側通常プログラム

## 6.1 階層構造

フラッシュメモリ上で実行するスレーブ側通常プログラムは，ユーザアプリケーションプログラム (通常アプリケーション) の実行，書き換え開始コマンド受信，フラッシュメモリ上の書き込み/消去制御プログラムを内蔵 RAM に転送する処理を行ないます。スレーブ側通常プログラムで使用するルーチンの階層構造を図 19 に示します。

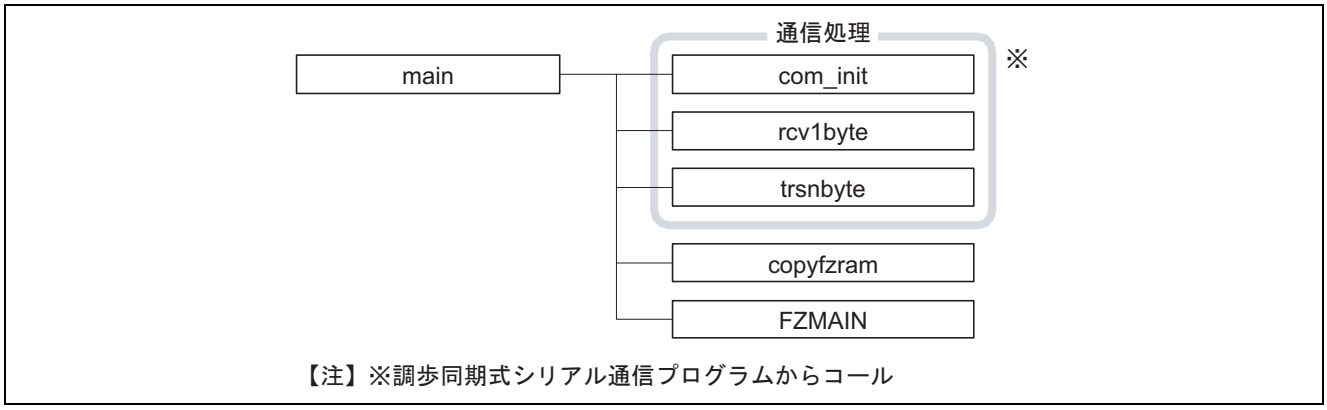



**図 19　スレーブ側通常プログラムの階層構造**

## 6.2 関数一覧

**表 6　スレーブ側通常プログラム関数一覧**

| 関数名 | 概要 |
|---|---|
| main | 通常アプリケーションの実行，書き換え開始コマンド受信，フラッシュメモリ上の書き込み/消去制御プログラムを内蔵 RAM に転送する |
| copyfzram | フラッシュメモリの書き込み/消去制御プログラムを RAM に転送する |
| FZMAIN | フラッシュメモリの書き込み/消去制御プログラム |

## 6.3 関数説明

(1) main()関数

(a) 仕様

void main (void)

(b) 動作説明

- ユーザアプリケーションプログラム (通常アプリケーション) の実行
- 書き換え開始コマンド受信処理
- 書き込み/消去制御プログラムの RAM 転送処理
- 書き込み/消去制御プログラムへの分岐

(c) 引数の説明

- 入力値：なし
- 出力値：なし

(d) グローバル変数

なし

(e) 使用サブルーチン

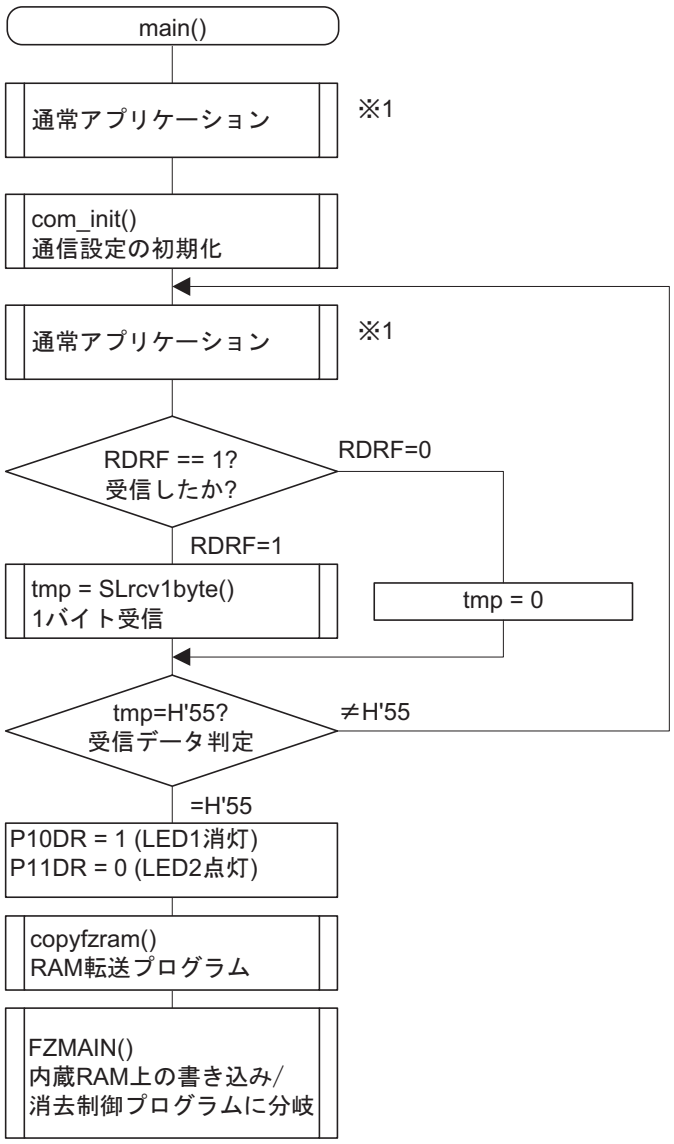
com_init() ：通信設定の初期化
SLrcv1byte() ：データを 1 バイト受信する
copyfzram() ：書き込み/消去制御プログラムを RAM に転送する
FZMAIN() ：書き込み/消去制御プログラムへの分岐

(f) 使用内部レジスタ

**表 7 main()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| MSTPCRD | | モジュールストップコントロールレジスタ D | H'FFFC60 | — |
| | MSTPD6 | サンプルの通常アプリケーションで使用する | ビット 6 | — |
| P7DDR | | ポート 7 データディレクションレジスタサンプルの通常アプリケーションで使用する | H'FFFE36 | — |
| P7DR | | ポート 7 データレジスタ<br>サンプルの通常アプリケーションで使用する | H'FFFF06 | — |
| PORT7 | | ポート 7 レジスタ | H'FFFFB6 | — |
| | P70 | ポート 70 サンプルの通常アプリケーションで使用する | ビット 0 | — |
| P1DDR | | ポート 1 データディレクションレジスタ<br>・P1DDR = H'03 のとき，P11, P10 端子が出力端子になる | H'FFFE30 | H '03 |
| P1DR | | ポート 1 データレジスタ | H'FFFF00 | — |
| | P11DR | ポート 11 データレジスタ<br>・P11DR = 0 のとき，P11 端子の出力レベルは"Low"<br>・P11DR = 1 のとき，P11 端子の出力レベルは"High" | ビット 1 | 0 |
| | P10DR | ポート 10 データレジスタ<br>・P10DR = 0 のとき，P10 端子の出力レベルは"Low"<br>・P10DR = 1 のとき，P10 端子の出力レベルは"High" | ビット 0 | 1 |
| SSR | | シリアルステータスレジスタ 0 | H'FFFF7C | — |
| | RDRF | レシーブデータレジスタフル<br>・RDRF = 0 のとき，RDR_0 に受信データが格納されていない<br>・RDRF = 1 のとき，RDR_0 に受信データが格納されている | ビット 6 | — |

(g) フローチャート




【注】※1　通常アプリケーションは、フラッシュメモリ書き込み処理とは関係のないサンプルプログラムです。本例では、上記2ヵ所に設定してあります。

(2) copyfzram()関数

(a) 仕様

void copyfzram (void)

(b) 動作説明

- フラッシュメモリ書き込み/消去制御プログラムを RAM に転送する

(c) 引数の説明

- 入力値：なし
- 出力値：なし

(d) グローバル変数

なし

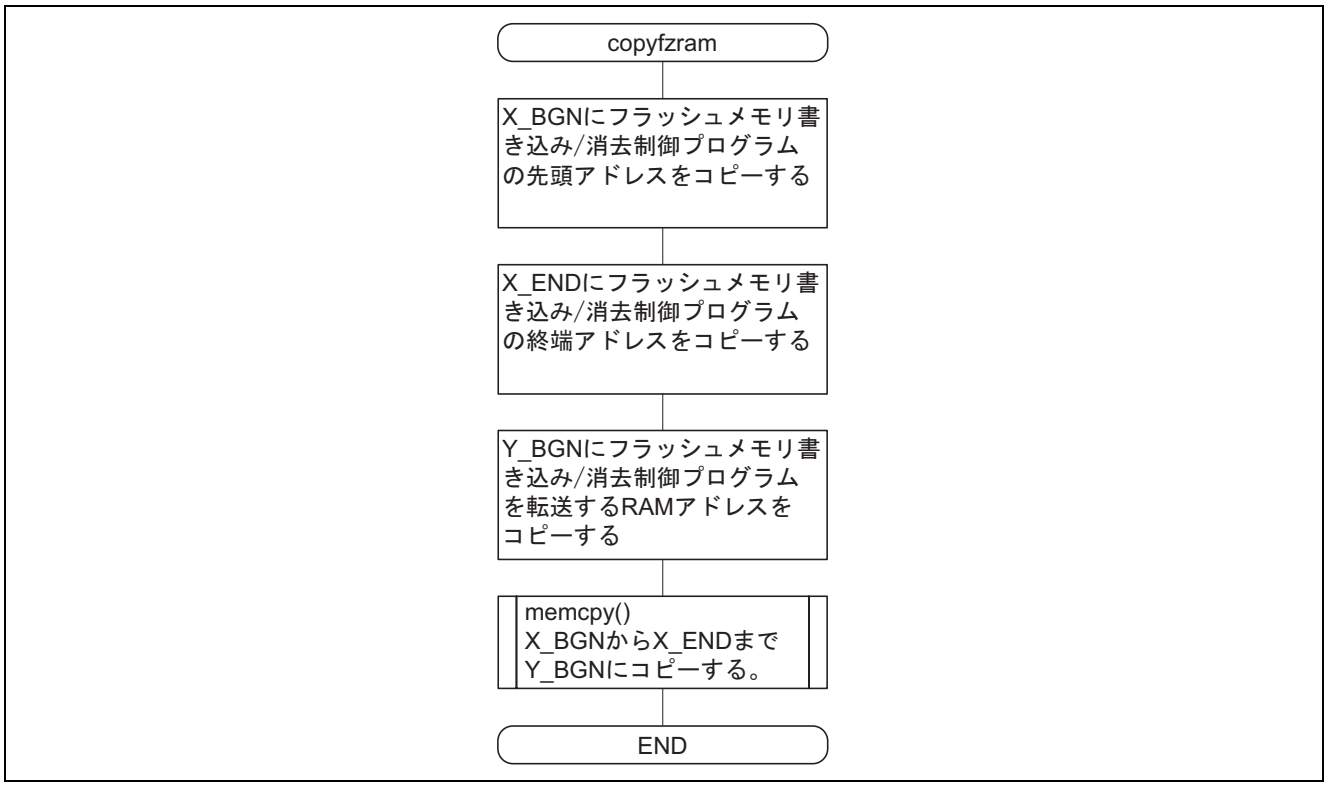
(e) 使用サブルーチン

なし

(f) 使用内部レジスタ

なし

(g) フローチャート




(3) FZMAIN()関数

書き込み/消去制御プログラムのメインルーチンを呼び出す

# 7. スレーブ側書き込み/消去制御プログラム

## 7.1 階層構造

書き込み/消去制御プログラムは，消去ブロック単位の消去，フラッシュメモリ書き込みデータの受信，フラッシュメモリへの書き込みを行ないます。書き込み/消去制御プログラムで使用するルーチンの階層構造を図 20 に示します。FZMAIN()関数を除くサブルーチンは，通信処理とフラッシュメモリ書き込み/消去処理に分けられます。

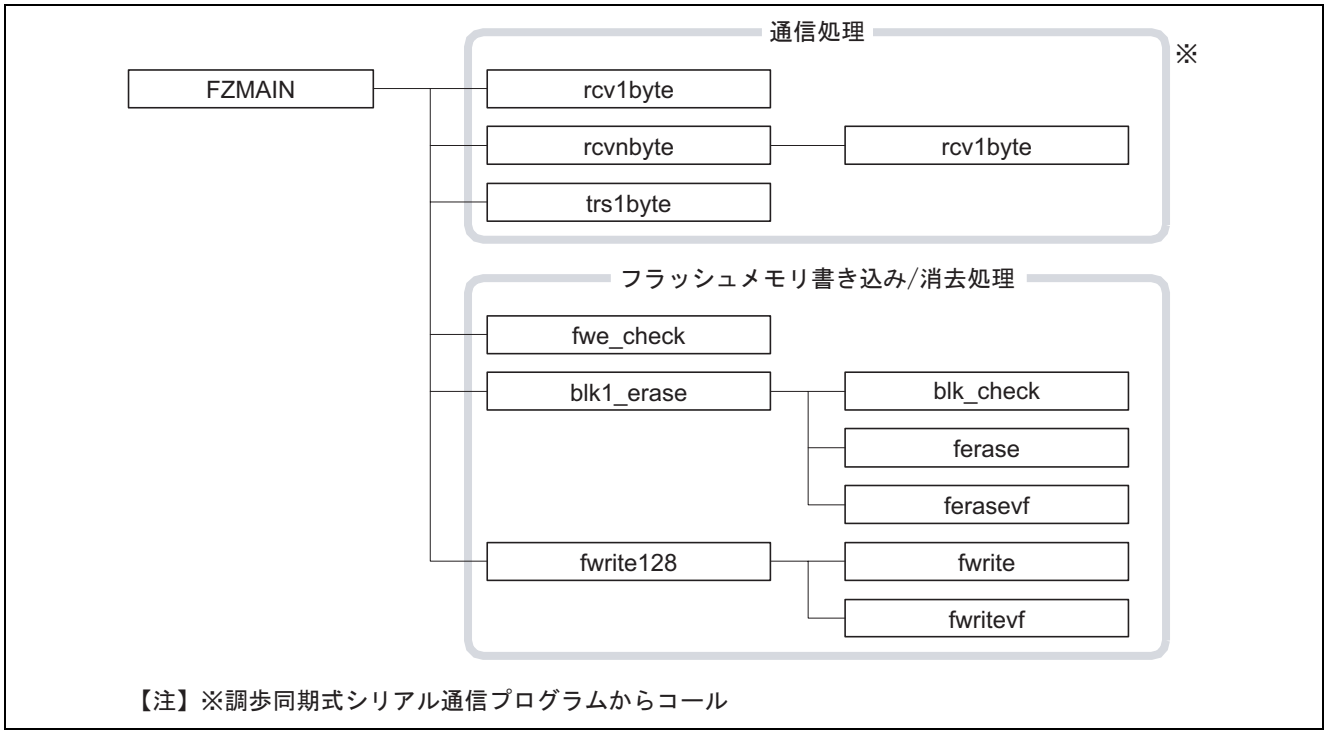



**図 20　書き込み/消去制御プログラムの階層構造**

## 7.2 関数一覧

**表 8　書き込み/消去制御プログラム関数一覧**

| 関数名 | 概要 |
|---|---|
| FZMAIN | **書き込み/消去制御プログラムのメインルーチン** |
| fwe_check | FWE **端子の制御，判定を行なう** |
| blk_check | **消去先頭アドレスから消去対象ブロックのビット番号を判定する** |
| blk1_erase | **フラッシュメモリの指定ブロックを消去する** |
| ferase | **指定ブロックの消去** |
| ferasevf | **指定ブロックの消去ベリファイ** |
| fwrite128 | 128 **バイトの書き込み，ベリファイ** |
| fwrite | **対象アドレスの書き込み** |
| fwritevf | **対象アドレスレスのベリファイ，再書き込みデータの作成** |

## 7.3 定数一覧

表 9 定数一覧

| 定数名 | 値 | 内容 |
|---|---|---|
| OK | H'00 | 正常時の戻り値 |
| NG | H'01 | 異常時の戻り値 |
| WNG | H'02 | 書き込みエラー |
| MAXBLK1 | H'0C | フラッシュメモリの全ブロック数 (12) |
| OW_COUNT | H'06 | 再書き込み回数 |
| WLOOP1 | 1 × HZ/KEISU1+1 = 4 (H'04) | WAIT 文実行回数 1μsWAIT |
| WLOOP2 | 2 × MHZ/KEISU1+1 = 7 (H'07) | WAIT 文実行回数 2μsWAIT |
| WLOOP4 | 4 × MHZ/KEISU1+1 = 14 (H'0E) | WAIT 文実行回数 4μsWAIT |
| WLOOP5 | 5 × MHZ/KEISU1+1 = 17 (H'11) | WAIT 文実行回数 5μsWAIT |
| WLOOP10 | 10 × MHZ/KEISU1+1 = 34 (H'22) | WAIT 文実行回数 10μsWAIT |
| WLOOP20 | 20 × MHZ/KEISU1+1 = 67 (H'43) | WAIT 文実行回数 20μsWAIT |
| WLOOP50 | 50 × MHZ/KEISU1+1 = 167 (H'A7) | WAIT 文実行回数 50μsWAIT |
| WLOOP100 | 100 × MHZ/KEISU1+1 = 334 (H'14E) | WAIT 文実行回数 100μsWAIT |
| TIME10 | 10 × MHZ/KEISU1+1 = 34 (H'22) | WAIT 文実行回数 10μsWAIT |
| TIME30 | 30 × MHZ/KEISU1+1 = 101 (H'65) | WAIT 文実行回数 30μsWAIT |
| TIME200 | 200 × MHZ/KEISU1+1 = 667 (H'29B) | WAIT 文実行回数 200μsWAIT |
| TIME10000 | (10000/KEISU1) × MHZ+1 = 33334 (H'8236) | WAIT 文実行回数 10msWAIT |

【注】 MHZ:10 ・・・動作周波数 10MHz
KEISU1:3 ・・・for 文 1 ループの最小ステート数

## 7.4 使用 RAM 説明

FZMAIN 関数実行中は, 表 10 のスタックメモリを使用します。この他にも, プログラム作業用としてスタックメモリを使用しますが，使用量はコンパイラのバージョンやオプション設定により変化します。

表 10 主な使用 RAM

| データ名 | スタックメモリ使用量 |
|---|---|
| 書き込みデータ | 128 バイト |
| 再書き込みデータ | 128 バイト |
| 追加書き込みデータ | 128 バイト |

## 7.5 関数説明

(1) FZMAIN()関数

(a) 仕様

void FZMAIN (void)

(b) 動作説明

- FWE 端子の制御，判定
- フラッシュメモリの消去
- フラッシュメモリ書き込みデータの受信
- フラッシュメモリへのデータ書き込み
- 書き込み終了後にリセットスタートを実施

(c) 引数の説明

- 入力値：なし
- 出力値：なし

(d) グローバル変数

なし

(e) 使用サブルーチン

fwe_check()：FWE 端子の制御，判定を行なう
rcv1byte()：データを 1 バイト受信する
rcvnbyte()：データを n バイト受信する
trs1byte()：データを 1 バイト送信する
fwrite128()：128 バイトの書き込み，ベリファイを行なう
blk_check()：消去先頭アドレスから消去対象ブロックのビット番号を判定する
blk1_erase()：フラッシュメモリの指定ブロックを消去する

(f) 使用内部レジスタ

**表 11　FZMAIN()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| P1DR | | **ポート** 1 **データレジスタ** | H'FFFF00 | — |
| | P11DR | **ポート** 11 **データレジスタ**<br>・P11DR = 0 **のとき，**P11 **端子の出力レベルは**"Low"<br>・P11DR = 1 **のとき，**P11 **端子の出力レベルは**"High" | **ビット** 1 | 1 |
| | P10DR | **ポート** 10 **データレジスタ**<br>・P10DR = 0 **のとき，**P10 **端子の出力レベルは**"Low"<br>・P10DR = 1 **のとき，**P10 **端子の出力レベルは**"High" | **ビット** 0 | 0 |
| TCSR_0 *[1] | | **タイマコントロール/ステータスレジスタ** 0 | H'FFFF74 | H'00 |
| | OVF | **オーバフローフラグ**<br>・OVF = 0 **のとき，**TCNT_0 **がオーバフローしていない**<br>・OVF = 1 **のとき，**TCNT_0 **がオーバフローしている** | **ビット** 7 | 0 |
| | WT/$\overline{IT}$ | **タイマモードセレクト**<br>・WT/$\overline{IT}$ = 0 **のとき，インターバルタイマとして使用する**<br>・WT/$\overline{IT}$ = 0 **のとき，ウォッチドッグタイマとして使用する** | **ビット** 6 | 0 |

表 11　FZMAIN()関数の使用レジスタ（続き）

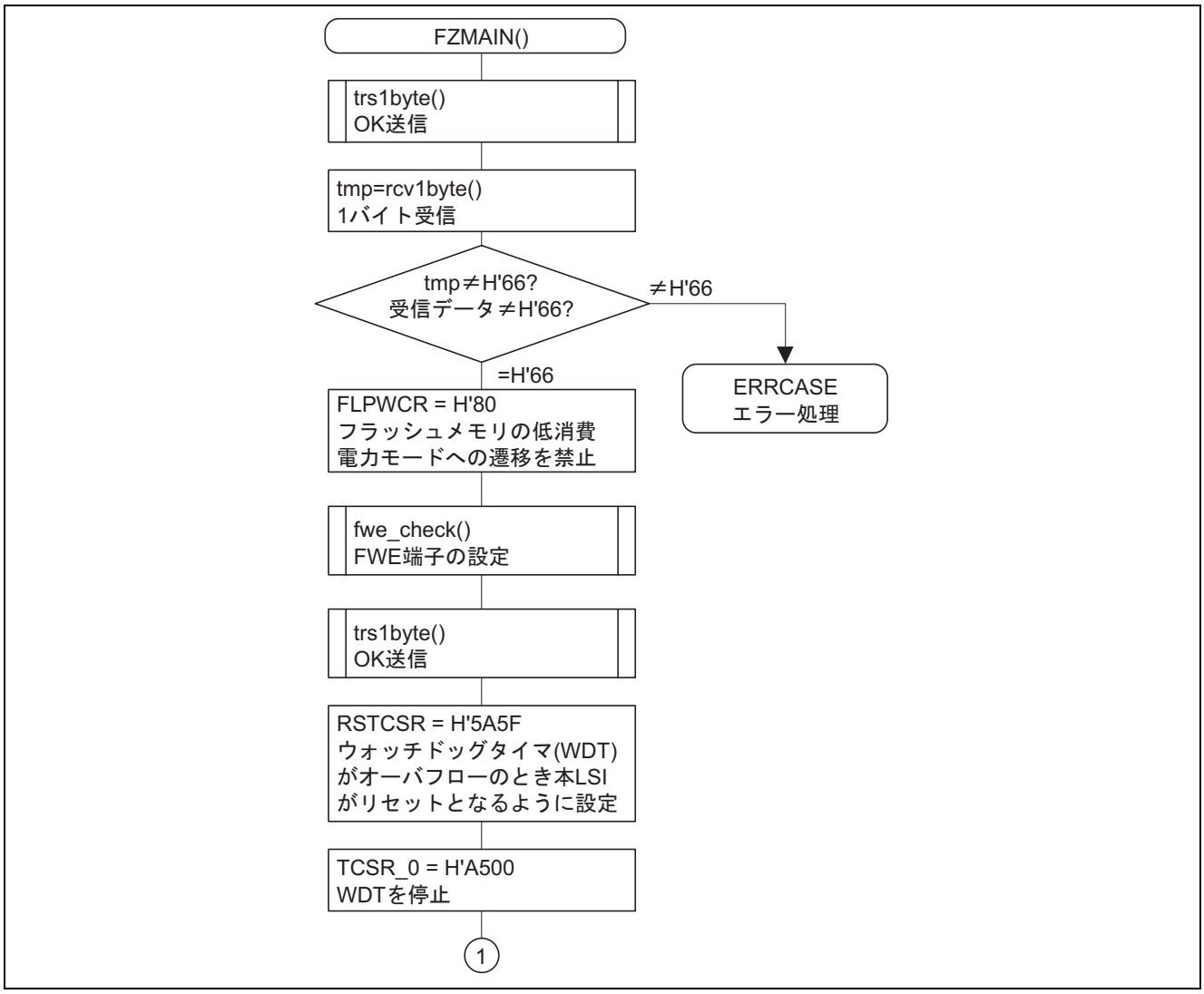
| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| TCSR_0 *1 | TME | タイマイネーブル<br>・TME = 0 のとき，TCNT_0 がカウントを開始<br>・TME = 1 のとき，TCNT_0 がカウントを停止 | ビット 5 | 0 |
| | CKS2<br>CKS1<br>CKS0 | クロックセレクト 2～0<br>・CKS2 = 0, CKS1 = 0, CKS0 = 0 のとき，TCNT_0 にφ/2 のクロックを入力する | ビット 2<br>ビット 1<br>ビット 0 | CKS2 = 0<br>CKS1 = 0<br>CKS0 = 0 |
| TCNT_0 *2 | | タイマカウンタ<br>・8 ビットのアップカウンタ。 | H'FFFF74 | H'FF |
| RSTCSR *3 | | リセットコントロール/ステータスレジスタ | H'FFFF76 | H'SF |
| | RSTE | リセットイネーブル<br>・RSTE = 0 のとき，TCNT_0 がオーバフローしても本 LSI 内部はリセットされない<br>・RSTE = 1 のとき，TCNT_0 がオーバフローすると本 LSI 内部がリセットされる | ビット 6 | 1 |
| FLPWCR | | フラッシュメモリパワーコントロールレジスタ | H'FFFFAC | H'80 |
| | PDWND | パワーダウンディスエーブル<br>・PDWND = 0 のとき，フラッシュメモリの低消費電力モードへの遷移を許可<br>・PDWND = 1 のとき，フラッシュメモリの低消費電力モードへの遷移を禁止 | ビット 7 | 1 |

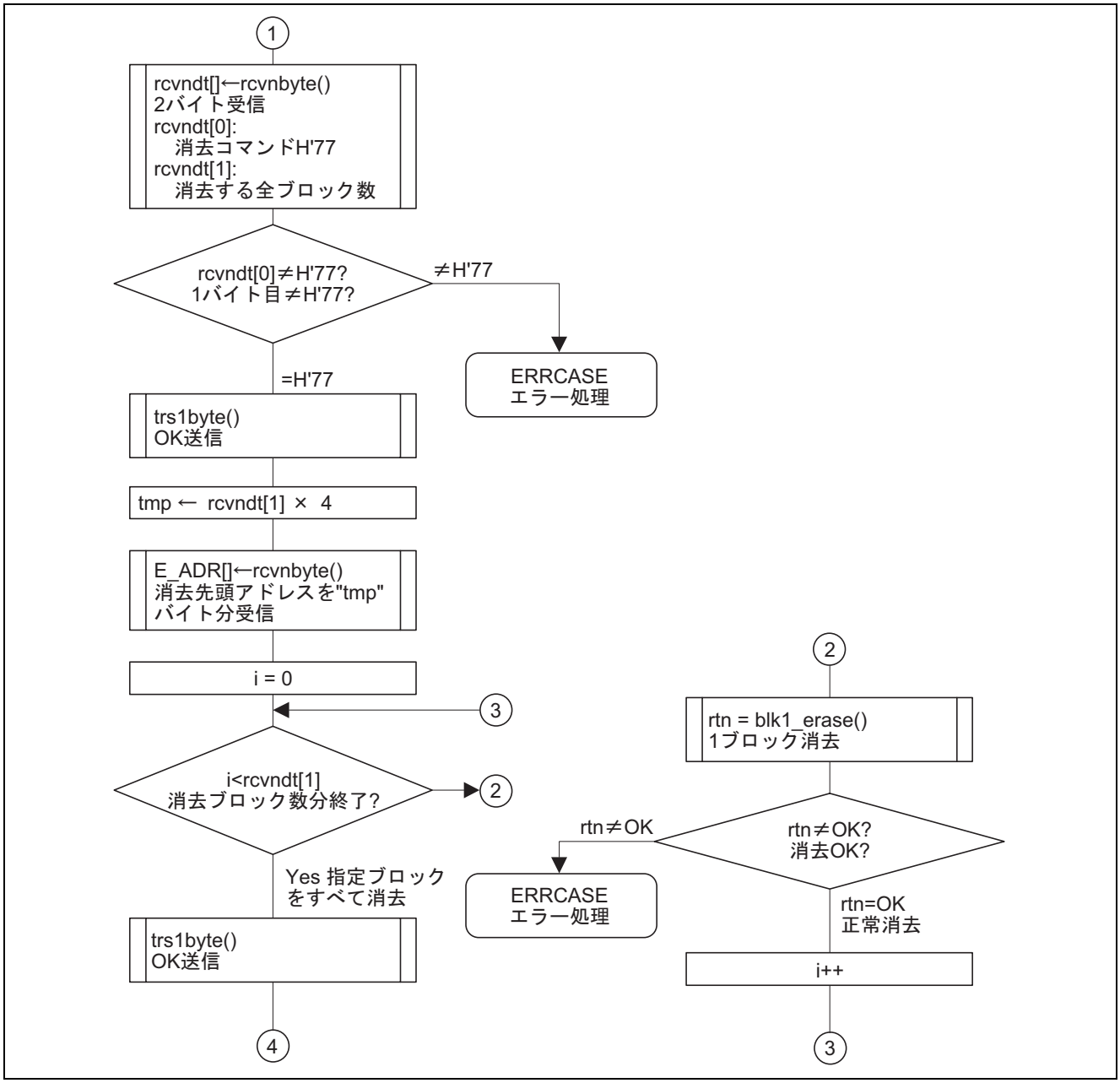
【注】 *1. TCSR_0 は，一般のレジスタとライト方法が異なります。
- ライト時は，H'FFFF74 に対してワード転送を行ないます。
- 上位バイトを H'A5 にし，下位バイトをライトデータにします。
- 本関数の場合，次のようにします。
  TCSR_0 = H'A500

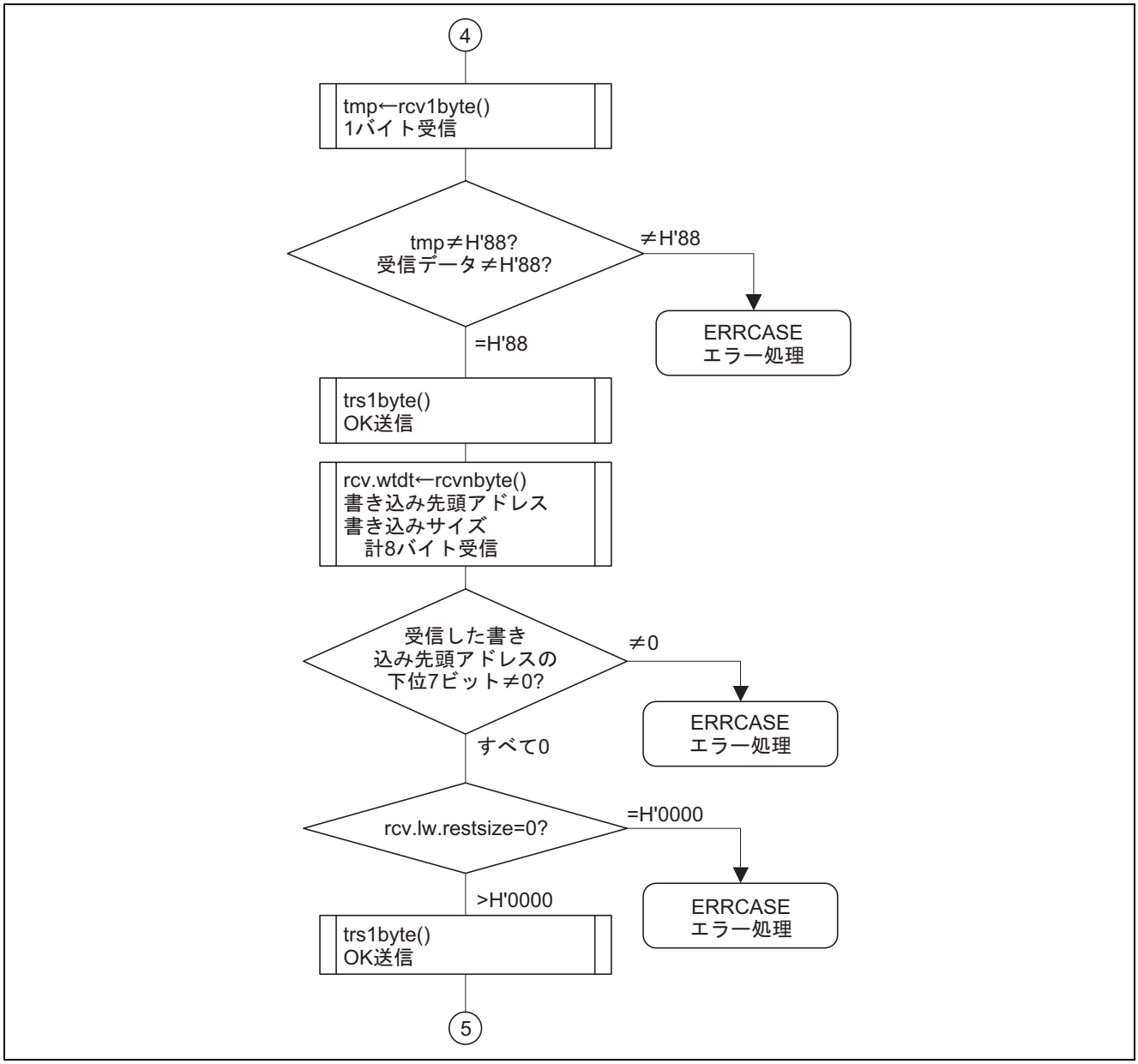
*2. TCNT_0 は，一般のレジスタとライト方法が異なります。
- ライト時は，H'FFFF74 に対してワード転送を行ないます。
- 上位バイトを H'5A にし，下位バイトをライトデータにします。
- 本関数の場合，次のようにします。
  TCNT_0 = H'5AFF

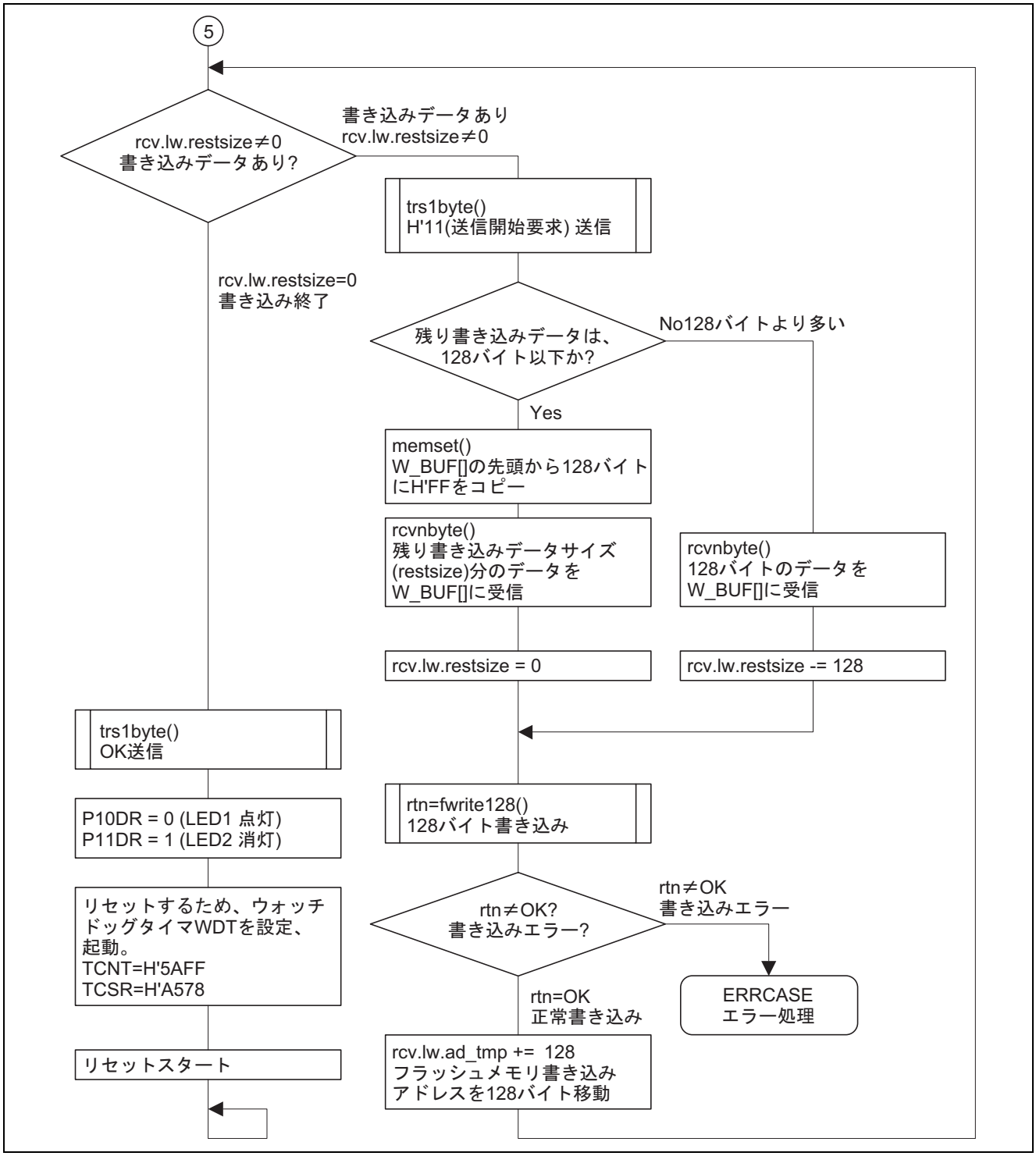
*3. RSTCSR は，一般のレジスタとライト方法が異なります。
- ライト時は，H'FFFF76 に対してワード転送を行ないます。
- RSTE ビットにライトする場合は，上位バイトを H'5A にし，下位バイトをライトデータにします。
- 本関数の場合，次のようにします。
  RSTCSR = H'5A5F

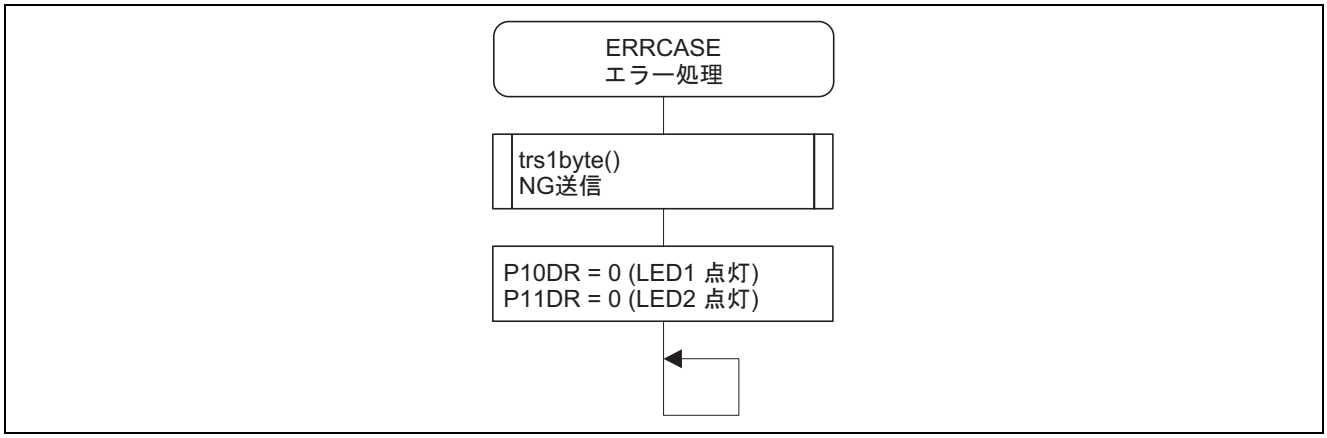
(g) フローチャート


FZMAIN()
trs1byte()
OK送信
tmp=rcv1byte()
1バイト受信
tmp≠H'66?
受信データ≠H'66?
≠H'66
ERRCASE
エラー処理
=H'66
FLPWCR = H'80
フラッシュメモリの低消費
電力モードへの遷移を禁止
fwe_check()
FWE端子の設定
trs1byte()
OK送信
RSTCSR = H'5A5F
ウォッチドッグタイマ(WDT)
がオーバフローのとき本LSI
がリセットとなるように設定
TCSR_0 = H'A500
WDTを停止
1


1
rcvndt[]←rcvnbyte()
2バイト受信
rcvndt[0]:
消去コマンドH'77
rcvndt[1]:
消去する全ブロック数
rcvndt[0]≠H'77?
1バイト目≠H'77?
≠H'77
ERRCASE
エラー処理
=H'77
trs1byte()
OK送信
tmp ← rcvndt[1] × 4
E_ADR[]←rcvnbyte()
消去先頭アドレスを"tmp"
バイト分受信
i = 0
3
i<rcvndt[1]
消去ブロック数分終了?
2
Yes 指定ブロック
をすべて消去
trs1byte()
OK送信
4
2
rtn = blk1_erase()
1ブロック消去
rtn≠OK
rtn≠OK?
消去OK?
ERRCASE
エラー処理
rtn=OK
正常消去
i++
3


4
tmp←rcv1byte()
1バイト受信
tmp≠H'88?
受信データ≠H'88?
≠H'88
ERRCASE
エラー処理
=H'88
trs1byte()
OK送信
rcv.wtdt←rcvnbyte()
書き込み先頭アドレス
書き込みサイズ
計8バイト受信
受信した書き
込み先頭アドレスの
下位7ビット≠0?
≠0
ERRCASE
エラー処理
すべて0
rcv.lw.restsize=0?
=H'0000
ERRCASE
エラー処理
>H'0000
trs1byte()
OK送信
5


5
rcv.lw.restsize≠0
書き込みデータあり?
書き込みデータあり
rcv.lw.restsize≠0
trs1byte()
H'11(送信開始要求) 送信
rcv.lw.restsize=0
書き込み終了
残り書き込みデータは、
128バイト以下か?
No128バイトより多い
Yes
memset()
W_BUF[]の先頭から128バイト
にH'FFをコピー
rcvnbyte()
残り書き込みデータサイズ
(restsize)分のデータを
W_BUF[]に受信
rcvnbyte()
128バイトのデータを
W_BUF[]に受信
rcv.lw.restsize = 0
rcv.lw.restsize -= 128
trs1byte()
OK送信
P10DR = 0 (LED1 点灯)
P11DR = 1 (LED2 消灯)
リセットするため、ウォッチ
ドッグタイマWDTを設定、
起動。
TCNT=H'5AFF
TCSR=H'A578
リセットスタート
rtn=fwrite128()
128バイト書き込み
rtn≠OK?
書き込みエラー?
rtn≠OK
書き込みエラー
ERRCASE
エラー処理
rtn=OK
正常書き込み
rcv.lw.ad_tmp += 128
フラッシュメモリ書き込み
アドレスを128バイト移動


ERRCASE
エラー処理
trs1byte()
NG送信
P10DR = 0 (LED1 点灯)
P11DR = 0 (LED2 点灯)

(2) fwe_check()関数

(a) 仕様

void fwe_check (void)

(b) 動作説明

- FWE 端子の制御，判定を行なう

(c) 引数の説明

なし

(d) グローバル変数

なし

(e) 使用サブルーチン

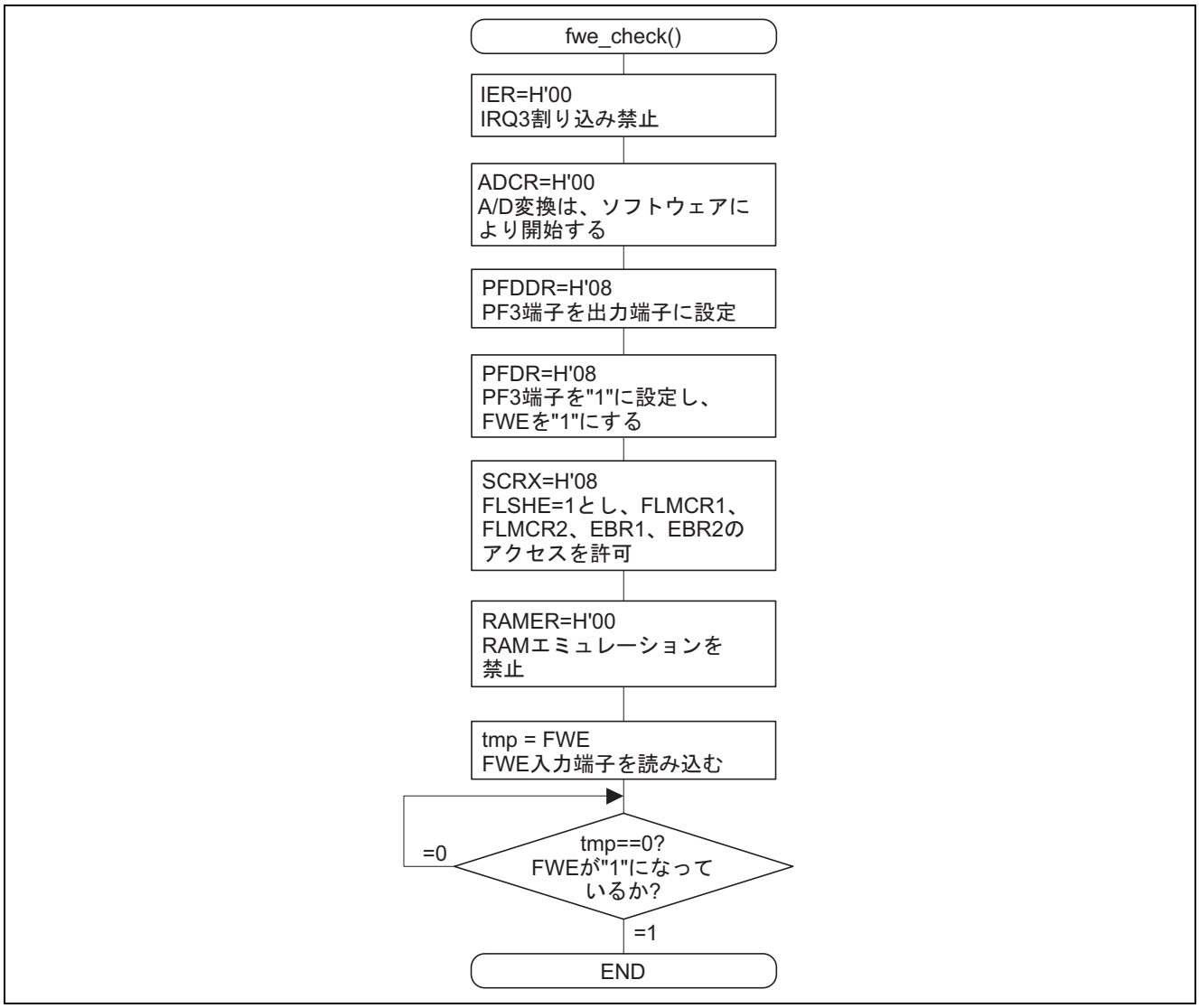
なし

(f) 使用内部レジスタ

表 12　fwe_check()関数の使用レジスタ

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| SCRX | | シリアルコントロールレジスタ X | H'FFFDB4 | H'08 |
| | FLSHE | フラッシュメモリコントロールレジスタイネーブル<br>・FLSHE = 0 のとき，フラッシュメモリの制御レジスタ (FLMCR1, FLMCR2, EBR1, EBR2) のアクセスを禁止<br>・FLSHE = 1 のとき，フラッシュメモリの制御レジスタ (FLMCR1, FLMCR2, EBR1, EBR2) のアクセスを許可 | ビット 3 | 1 |
| IER | | IRQ イネーブルレジスタ | H'FFFE14 | H'00 |
| | IRQ3E | IRQ3 イネーブル<br>・IRQ3E = 0 のとき，IRQ3 割り込み要求を禁止<br>・IRQ3E = 1 のとき，IRQ3 割り込み要求を許可 | ビット 3 | 0 |
| PFDDR | | ポート F データディレクションレジスタ | H'FFFE3E | H'08 |
| | PF3DDR | ポート F3 データディレクションレジスタ<br>・PF3DDR = 0 のとき，PF3 端子が入力端子になる<br>・PF3DDR = 1 のとき，PF3 端子が出力端子になる | ビット 3 | 1 |
| RAMER | | RAM エミュレーションレジスタ | H'FFFEDB | H'00 |
| | RAMS | RAM セレクト<br>・RAMS = 0 のとき，RAM エミュレーションを禁止<br>・RAMS = 1 のとき，RAM エミュレーションを許可 | ビット 3 | 0 |
| PFDR | | ポート F データレジスタ | H'FFFF0E | H'08 |
| | PF3DR | ポート F3 データレジスタ<br>・PF3DR = 0 のとき，PF3 端子の出力レベルは"Low"<br>・PF3DR = 1 のとき，PF3 端子の出力レベルは"High" | ビット 3 | 1 |
| ADCR | | A/D コントロールレジスタ | H'FFFF99 | H'00 |
| | TRGS1<br>TRGS0 | タイマトリガセレクト 1, 0<br>・TRGS1 = 0, TRGS0 = 0 のとき，ソフトウェアにより A/D 変換を開始する | ビット 7<br>ビット 6 | TRGS1 = 0<br>TRGS0 = 0 |
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | FWE | フラッシュライトイネーブル<br>・FWE = 0 のとき，FWE 入力端子が Low レベル<br>・FWE = 1 のとき，FWE 入力端子が High レベル | ビット 7 | — |

(g) フローチャート


fwe_check()
IER=H'00
IRQ3割り込み禁止
ADCR=H'00
A/D変換は、ソフトウェアにより開始する
PFDDR=H'08
PF3端子を出力端子に設定
PFDR=H'08
PF3端子を"1"に設定し、FWEを"1"にする
SCRX=H'08
FLSHE=1とし、FLMCR1、FLMCR2、EBR1、EBR2のアクセスを許可
RAMER=H'00
RAMエミュレーションを禁止
tmp = FWE
FWE入力端子を読み込む
tmp==0?
FWEが"1"になっているか?
=0
=1
END

(3) blk1_erase()関数

(a) 仕様

```
char blk1_erase(
unsigned long ers_ad,
unsigned char ET_COUNT
)
```

(b) 動作説明

- 消去先頭アドレスから消去対象ブロックのビット番号を判定する
- フラッシュメモリの指定ブロックを消去する

(c) 引数の説明

- 入力値：
  ers_ad：消去先頭アドレス
  ET_COUNT：消去最大回数
- 出力値：
  戻り値：結果フラグ (OK = H'00, NG = H'01)

(d) グローバル変数

なし

(e) 使用サブルーチン

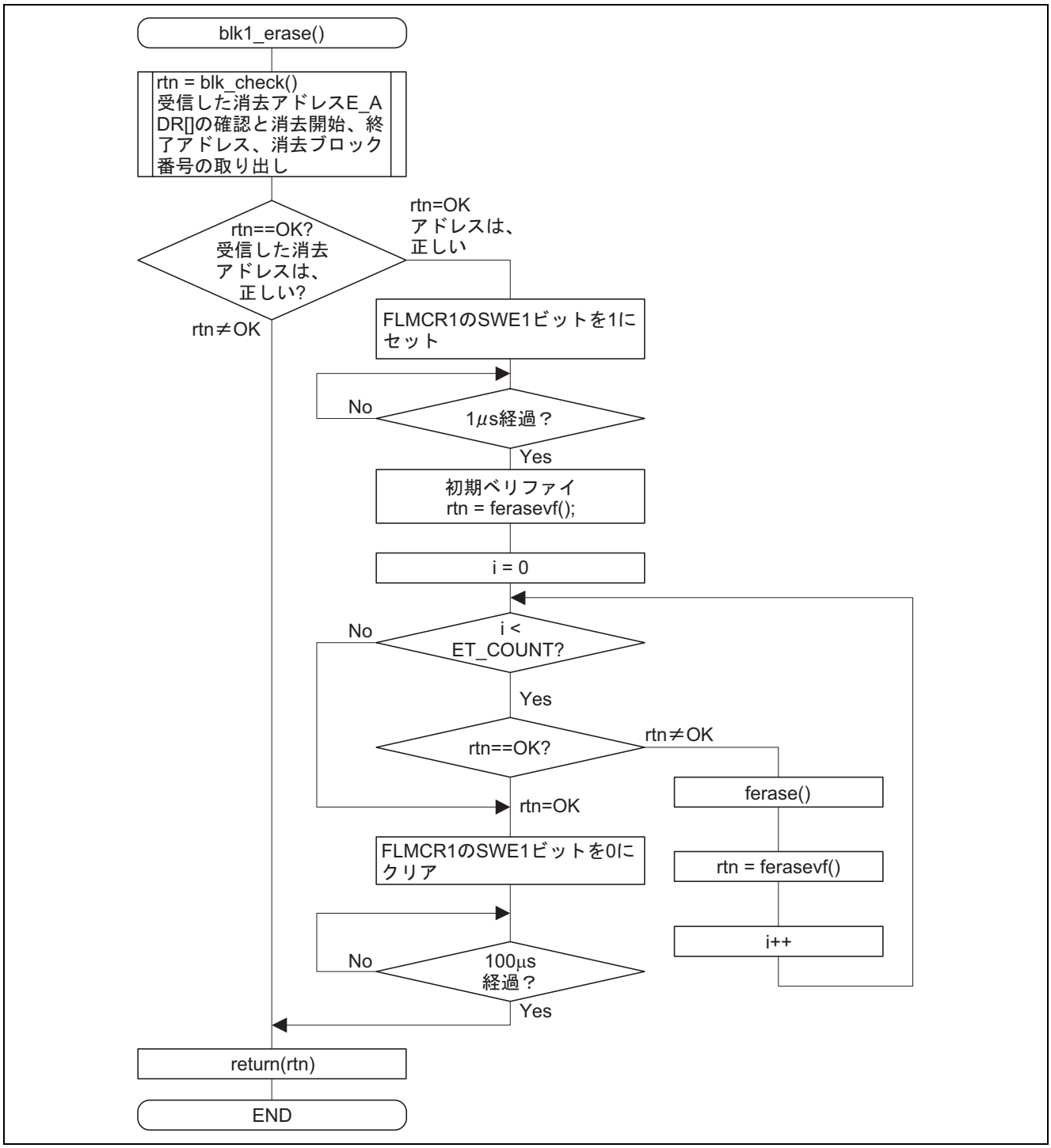
blk_check()：消去先頭アドレスから消去対象ブロックのビット番号を判定する
ferase()：指定ブロックの消去
ferasevf()：指定ブロックの消去ベリファイ

(f) 使用内部レジスタ

**表 13　blk1_erase()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | SWE1 | ソフトウェアライトイネーブル<br>・SWE1 = 0 のとき，フラッシュメモリの書き込み/消去が無効<br>・SWE1 = 1 のとき，フラッシュメモリの書き込み/消去が可能 | ビット 6 | 1 |

(g) フローチャート


blk1_erase()
rtn = blk_check()
受信した消去アドレスE_ADR[]の確認と消去開始、終了アドレス、消去ブロック番号の取り出し
rtn==OK?
受信した消去アドレスは、正しい?
rtn=OK
アドレスは、正しい
rtn≠OK
FLMCR1のSWE1ビットを1にセット
No
1μs経過？
Yes
初期ベリファイ
rtn = ferasevf();
i = 0
No
i < ET_COUNT?
Yes
rtn==OK?
rtn≠OK
ferase()
rtn=OK
FLMCR1のSWE1ビットを0にクリア
rtn = ferasevf()
i++
No
100μs経過？
Yes
return(rtn)
END

(4) blk_check()関数

(a) 仕様

char blk_check(
unsigned long eck_ad,
unsigned long *eck_st,
unsigned long *eck_ed,
unsigned char *blk_no
)

(b) 動作説明

- 消去先頭アドレスから消去対象ブロックのビット番号を判定する
- 受信した消去先頭アドレスが正しい値かどうか，BLOCKADR[]と比較判定し，結果フラグ，消去先頭アドレス，消去終了アドレス，消去対象ブロックのビット番号を返す

(c) 引数の説明

- 入力値：
  eck_ad　：消去先頭アドレス
  *eck_st　：ベリファイ後の消去先頭アドレス
  *eck_ed　：ベリファイ後の消去終了アドレス
  *blk_no　：消去対象ブロックのビット番号
- 出力値：
  戻り値　：結果フラグ (OK = H'00, NG = H'01)
  *eck_st　：ベリファイ後の消去先頭アドレス
  *eck_ed　：ベリファイ後の消去終了アドレス
  *blk_no　：消去対象ブロックのビット番号

(d) グローバル変数

BLOCKADR[]：フラッシュメモリ各ブロックの先頭アドレスを格納する

```
unsigned long BLOCKADR[13] ={   /*    Erase    Block Address  */
H'000000,                       /*    EB0      4KBYTE         */
H'001000,                       /*    EB1      4KBYTE         */
H'002000,                       /*    EB2      4KBYTE         */
H'003000,                       /*    EB3      4KBYTE         */
H'004000,                       /*    EB4      4KBYTE         */
H'005000,                       /*    EB5      4KBYTE         */
H'006000,                       /*    EB6      4KBYTE         */
H'007000,                       /*    EB7      4KBYTE         */
H'008000,                       /*    EB8      32KBYTE        */
H'010000,                       /*    EB9      64KBYTE        */
H'020000,                       /*    EB10     64KBYTE        */
H'030000,                       /*    EB11     64KBYTE        */
H'040000,                       /*    End      Block Address  */
};
```

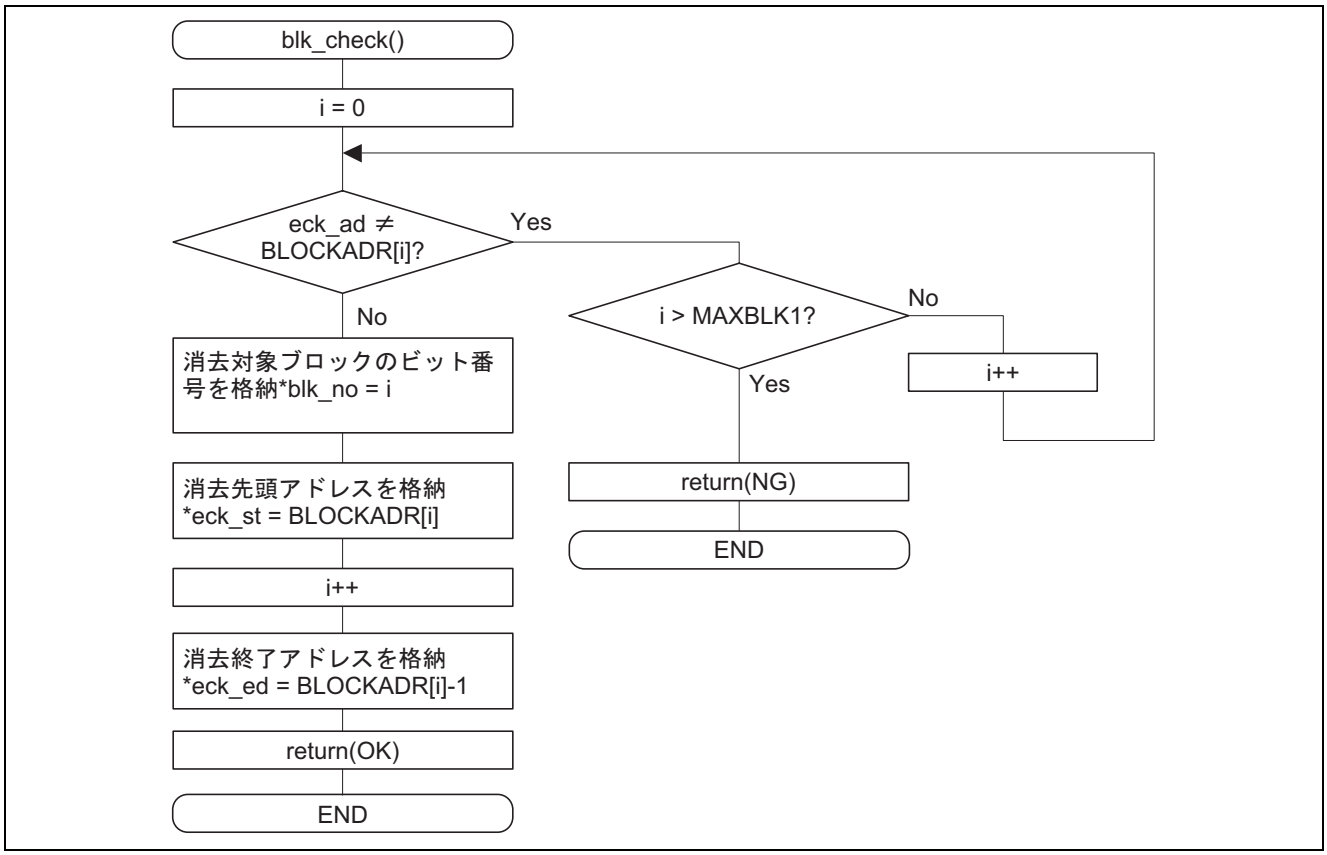
(e) 使用サブルーチン

なし

(f) 使用内部レジスタ

なし

(g) フローチャート


blk_check()
i = 0
eck_ad ≠ BLOCKADR[i]?
Yes
No
i > MAXBLK1?
No
Yes
i++
消去対象ブロックのビット番号を格納*blk_no = i
消去先頭アドレスを格納 *eck_st = BLOCKADR[i]
i++
消去終了アドレスを格納 *eck_ed = BLOCKADR[i]-1
return(OK)
END
return(NG)
END

(5) ferase()関数

(a) 仕様

void ferase (unsigned char e_blk_no)

(b) 動作説明

- フラッシュメモリの指定ブロックを消去する

(c) 引数の説明

- 入力値

  e_blk_no：消去対象ブロック番号

- 出力値

なし

(d) グローバル変数

なし

(e) 使用サブルーチン

なし

(f) 使用内部レジスタ

**表 14　ferase()関数の使用レジスタ**

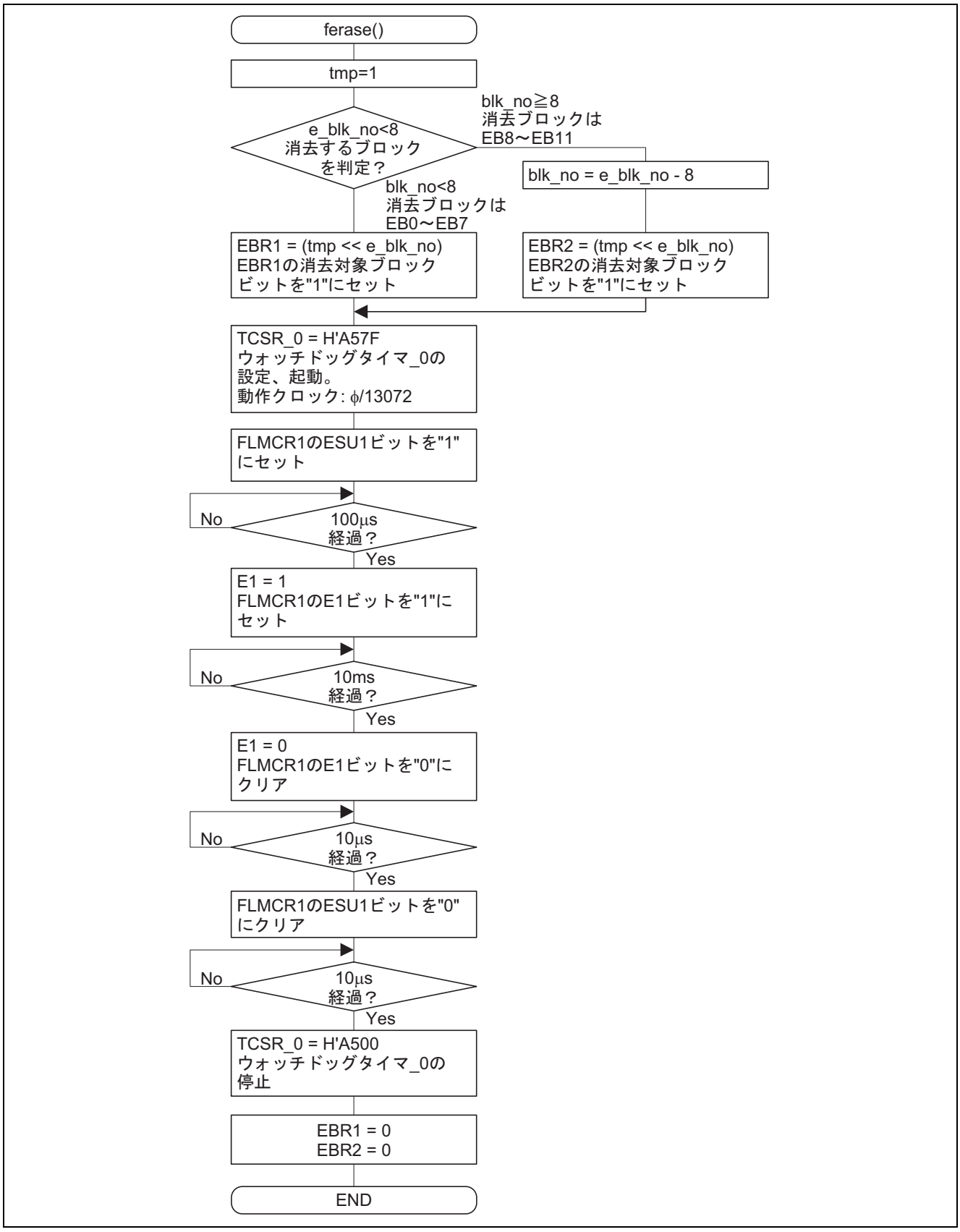
| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | ESU1 | イレースセットアップ<br>・ESU1 = 0 のとき，イレースセットアップ状態を解除<br>・FWE1 = 1, SWE1 = 1 の状態で ESU1 = 1 のとき，イレースセットアップ状態に遷移 | ビット 5 | 1 |
| | E1 | イレース<br>・E1 = 0 のとき，イレースモードを解除<br>・SWE1 = 1, ESU1 = 1 の状態で E1 = 1 のとき，イレースモードに遷移 | ビット 1 | 1 |
| EBR1 EB | EB7<br>:<br>:<br>EB0 | 消去ブロック指定レジスタ 1<br>・EB7～EB0 の各ビットを 1 に設定すると対応するフラッシュメモリのブロックが消去可能 | H'FFFFAA | — |
| EBR2 E | EB11<br>EB10<br>EB9<br>EB8 | 消去ブロック指定レジスタ 2<br>・EB11～EB8 の各ビットを 1 に設定すると対応するフラッシュメモリのブロックが消去可能 | H'FFFFAB | — |
| TCSR_0 *1 | | タイマコントロール/ステータスレジスタ 0 | H'FFFF74 | H'7F |
| | OVF | オーバフローフラグ<br>・OVF = 0 のとき，TCNT_0 がオーバフローしていない<br>・OVF = 1 のとき，TCNT_0 がオーバフローしている | ビット 7 | 0 |
| | WT/$\overline{IT}$ | タイマモードセレクト<br>・WT/$\overline{IT}$ = 0 のとき，インターバルタイマとして使用する<br>・WT/$\overline{IT}$ = 0 のとき，ウォッチドッグタイマとして使用する | ビット 6 | 1 |
| | TME | タイマイネーブル<br>・TME = 0 のとき，TCNT_0 がカウントを開始<br>・TME = 1 のとき，TCNT_0 がカウントを停止 | ビット 5 | 1 |

表 14　ferase()関数の使用レジスタ（続き）

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| TCSR_0 *1 | CKS2<br>CKS1<br>CKS0 | クロックセレクト 2～0<br>・CKS2 = 1, CKS1 = 1, CKS0 = 1 のとき，TCNT_0 に φ/131072 のクロックを入力する | ビット 2<br>ビット 1<br>ビット 0 | CKS2 = 1<br>CKS1 = 1<br>CKS0 = 1 |

【注】　*1. TCSR_0 は，一般のレジスタとライト方法が異なります。
・ライト時は，H'FFFF74 に対してワード転送を行ないます。
・上位バイトを H'A5 にし，下位バイトをライトデータにします。
・本関数の場合，次のようにします。
TCSR_0 = H'A57F

(g) フローチャート


ferase()
tmp=1
e_blk_no<8
消去するブロック
を判定？
blk_no≧8
消去ブロックは
EB8～EB11
blk_no = e_blk_no - 8
blk_no<8
消去ブロックは
EB0～EB7
EBR1 = (tmp << e_blk_no)
EBR1の消去対象ブロック
ビットを"1"にセット
EBR2 = (tmp << e_blk_no)
EBR2の消去対象ブロック
ビットを"1"にセット
TCSR_0 = H'A57F
ウォッチドッグタイマ_0の
設定、起動。
動作クロック: φ/13072
FLMCR1のESU1ビットを"1"
にセット
No
100μs
経過？
Yes
E1 = 1
FLMCR1のE1ビットを"1"に
セット
No
10ms
経過？
Yes
E1 = 0
FLMCR1のE1ビットを"0"に
クリア
No
10μs
経過？
Yes
FLMCR1のESU1ビットを"0"
にクリア
No
10μs
経過？
Yes
TCSR_0 = H'A500
ウォッチドッグタイマ_0の
停止
EBR1 = 0
EBR2 = 0
END

(6) ferasevf()関数

(a) 仕様

```
char ferasevf(
unsigned short *evf_st,
unsigned short *evf_ed
)
```

(b) 動作説明

- フラッシュメモリの指定ブロックが消去されたかベリファイする

(c) 引数の説明

- 入力値
  evf_st：消去先頭アドレス
  evf_ed：消去終了アドレス
- 出力値
  戻り値：結果フラグ (OK = H'00, NG = H'01)

(d) グローバル変数

なし

(e) 使用サブルーチン

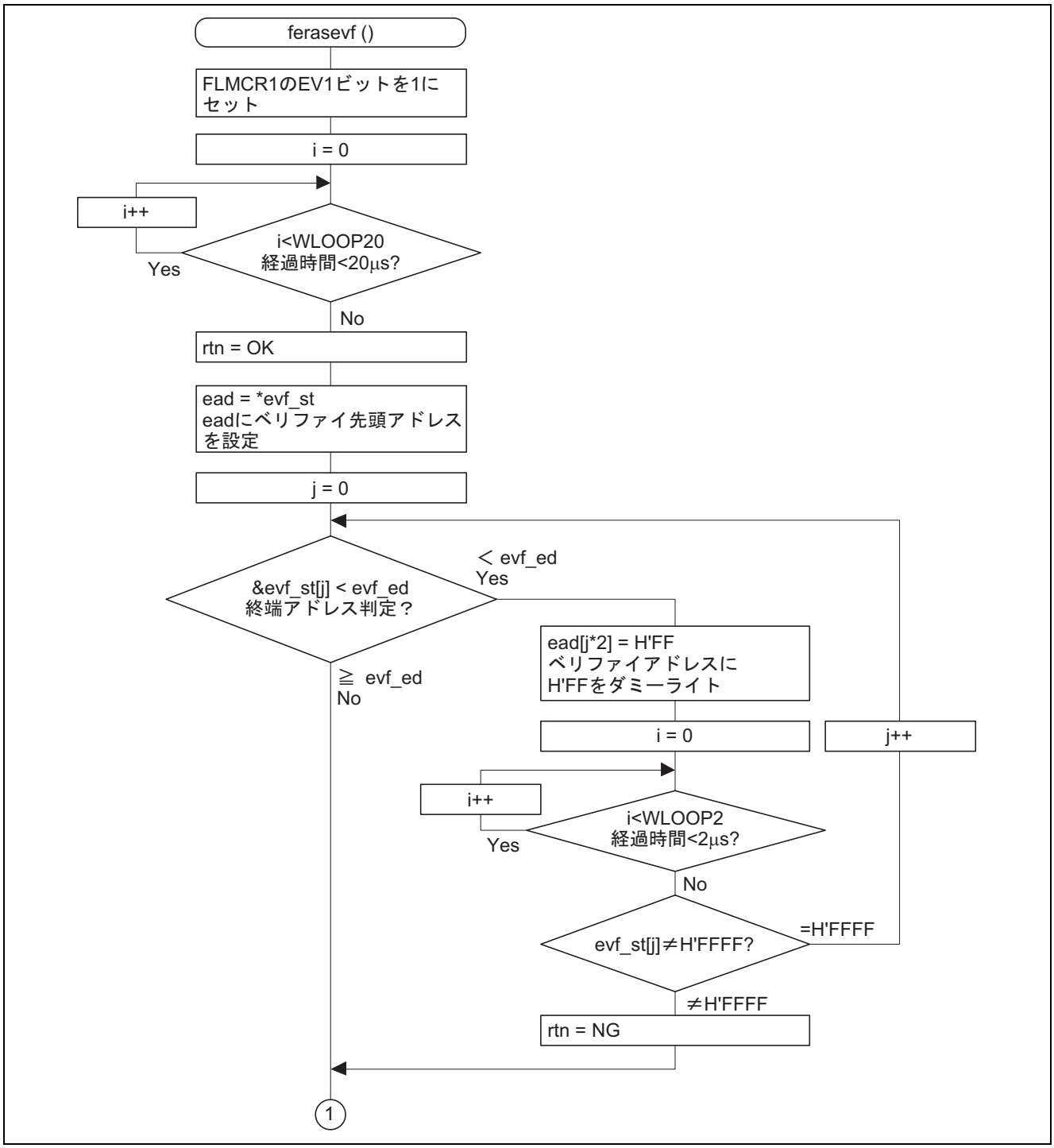
なし

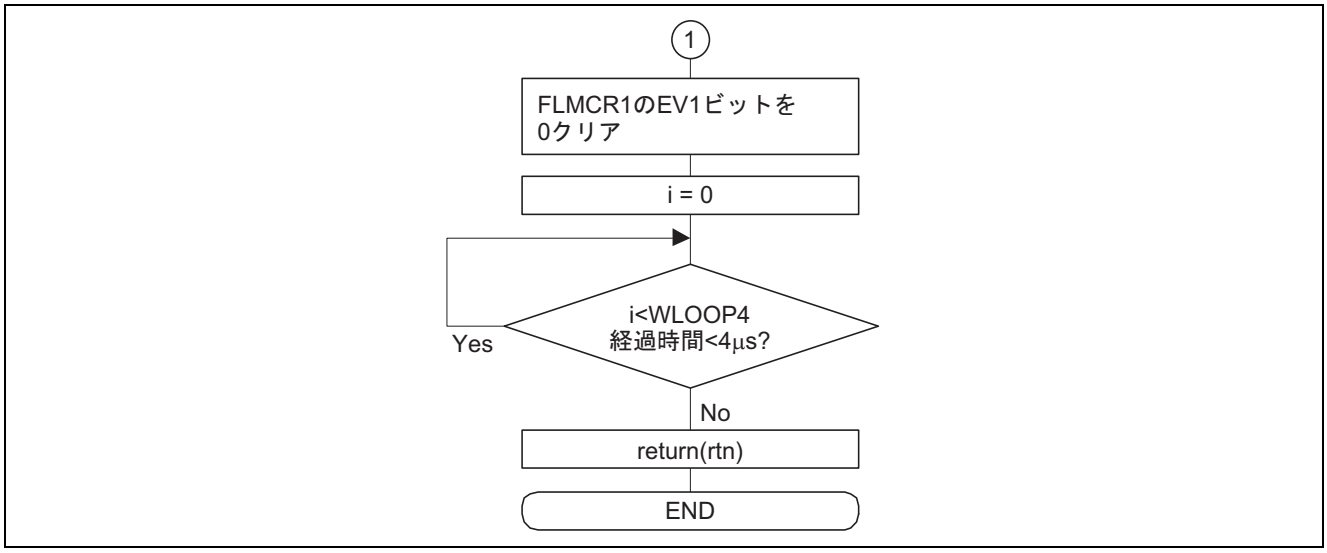
(f) 使用内部レジスタ

**表 15　ferasevf()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | EV1 | イレースベリファイ<br>・EV1 = 0 のとき，イレースベリファイモードを解除<br>・EV1 = 1 のとき，イレースベリファイモードに遷移 | ビット 3 | 1 |

(g) フローチャート


ferasevf ()
FLMCR1のEV1ビットを1にセット
i = 0
i++
i<WLOOP20
経過時間<20μs?
Yes
No
rtn = OK
ead = *evf_st
eadにベリファイ先頭アドレスを設定
j = 0
&evf_st[j] < evf_ed
終端アドレス判定？
< evf_ed
Yes
≧ evf_ed
No
ead[j*2] = H'FF
ベリファイアドレスに
H'FFをダミーライト
i = 0
j++
i++
i<WLOOP2
経過時間<2μs?
Yes
No
evf_st[j]≠H'FFFF?
=H'FFFF
≠H'FFFF
rtn = NG
1


1
FLMCR1のEV1ビットを
0クリア
i = 0
i<WLOOP4
経過時間<4μs?
Yes
No
return(rtn)
END

(7) fwrite128()関数

(a) 仕様

```
char fwrite128(
unsigned char *wt_buf,
unsigned char *wt_adr,
unsigned short WT_COUNT
)
```

(b) 動作説明

- 128 バイトの書き込み，ベリファイ

(c) 引数の説明

- 入力値
  *w_adr：書き込みアドレス
  *w_buf：書き込みデータ 128 バイト
  WT_COUNT：書き込み最大回数
- 出力値
  戻り値：結果フラグ (OK = H'00, NG = H'01)
  *w_adr：書き込みアドレス
  *w_buf：書き込みデータ 128 バイト

(d) グローバル変数

なし

(e) 使用サブルーチン

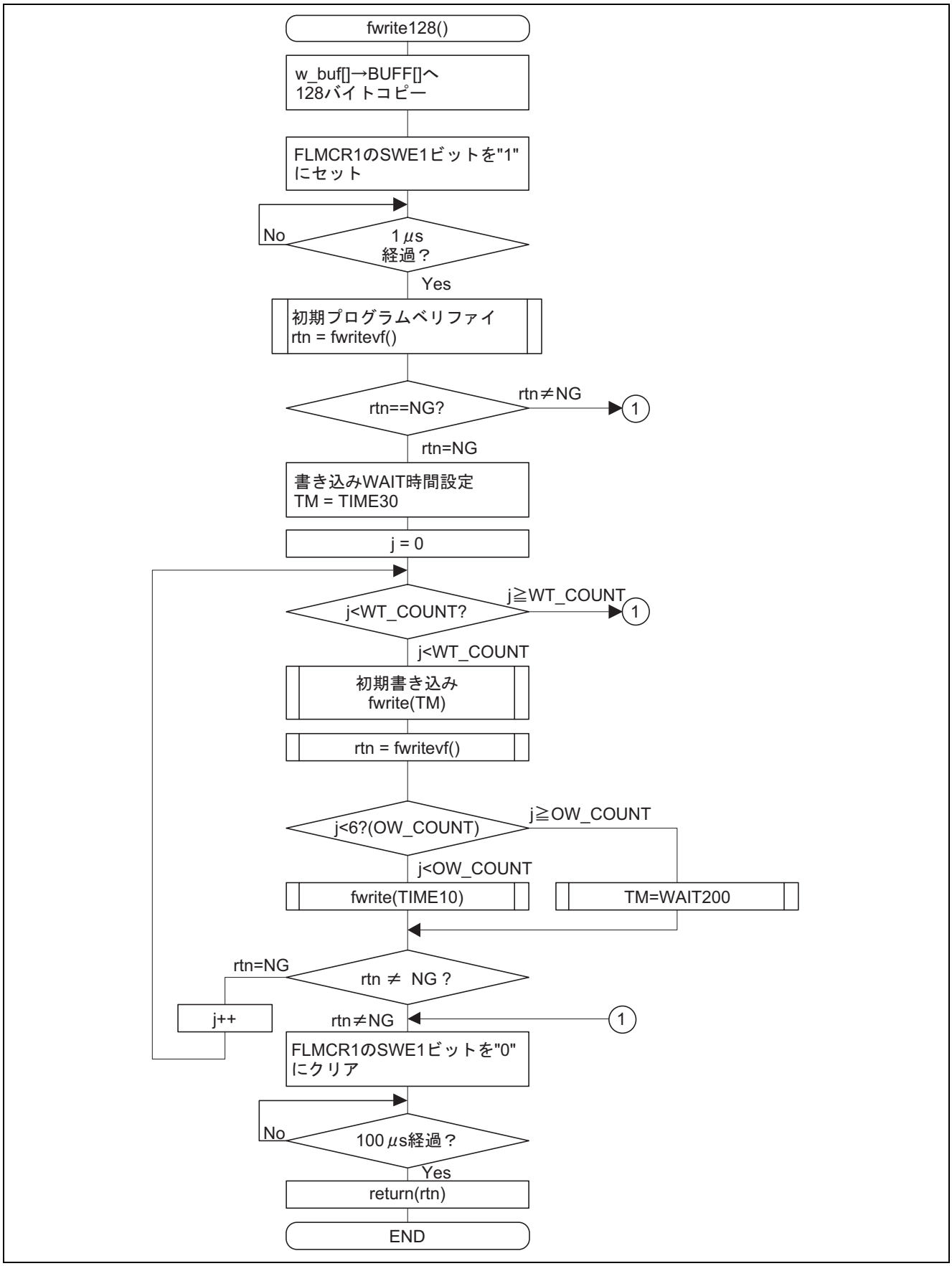
fwrite：対象アドレスの書き込み
fwritevf：対象アドレスのベリファイ，再書き込みデータの作成

(f) 使用内部レジスタ

**表 16　fwrite128()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | SWE1 | ソフトウェアライトイネーブル<br>・SWE1 = 0 のとき，フラッシュメモリの書き込み/消去が無効<br>・SWE1 = 1 のとき，フラッシュメモリの書き込み/消去が可能 | ビット 6 | 1 |

(g) フローチャート


fwrite128()
w_buf[]→BUFF[]へ
128バイトコピー
FLMCR1のSWE1ビットを"1"
にセット
No
1μs
経過？
Yes
初期プログラムベリファイ
rtn = fwritevf()
rtn==NG?
rtn≠NG
1
rtn=NG
書き込みWAIT時間設定
TM = TIME30
j = 0
j<WT_COUNT?
j≧WT_COUNT
1
j<WT_COUNT
初期書き込み
fwrite(TM)
rtn = fwritevf()
j<6?(OW_COUNT)
j≧OW_COUNT
j<OW_COUNT
fwrite(TIME10)
TM=WAIT200
rtn=NG
rtn ≠ NG ?
j++
rtn≠NG
1
FLMCR1のSWE1ビットを"0"
にクリア
No
100μs経過？
Yes
return(rtn)
END

(8) fwrite()関数

(a) 仕様

```
void fwrite(
unsigned char *buf,
unsigned char *w_adr,
unsigned char ptime
)
```

(b) 動作説明

- 対象アドレスの書き込み

(c) 引数の説明

- 入力値

  *buf　：書き込みの先頭アドレス (再書き込みデータ or 追加書き込みデータ)
  *w_adr　：書き込みアドレス
  ptime　：P1 ビットセット時間 (10μs or 30μs or 2000μs)

- 出力値

なし

(d) グローバル変数

なし

(e) 使用サブルーチン

なし

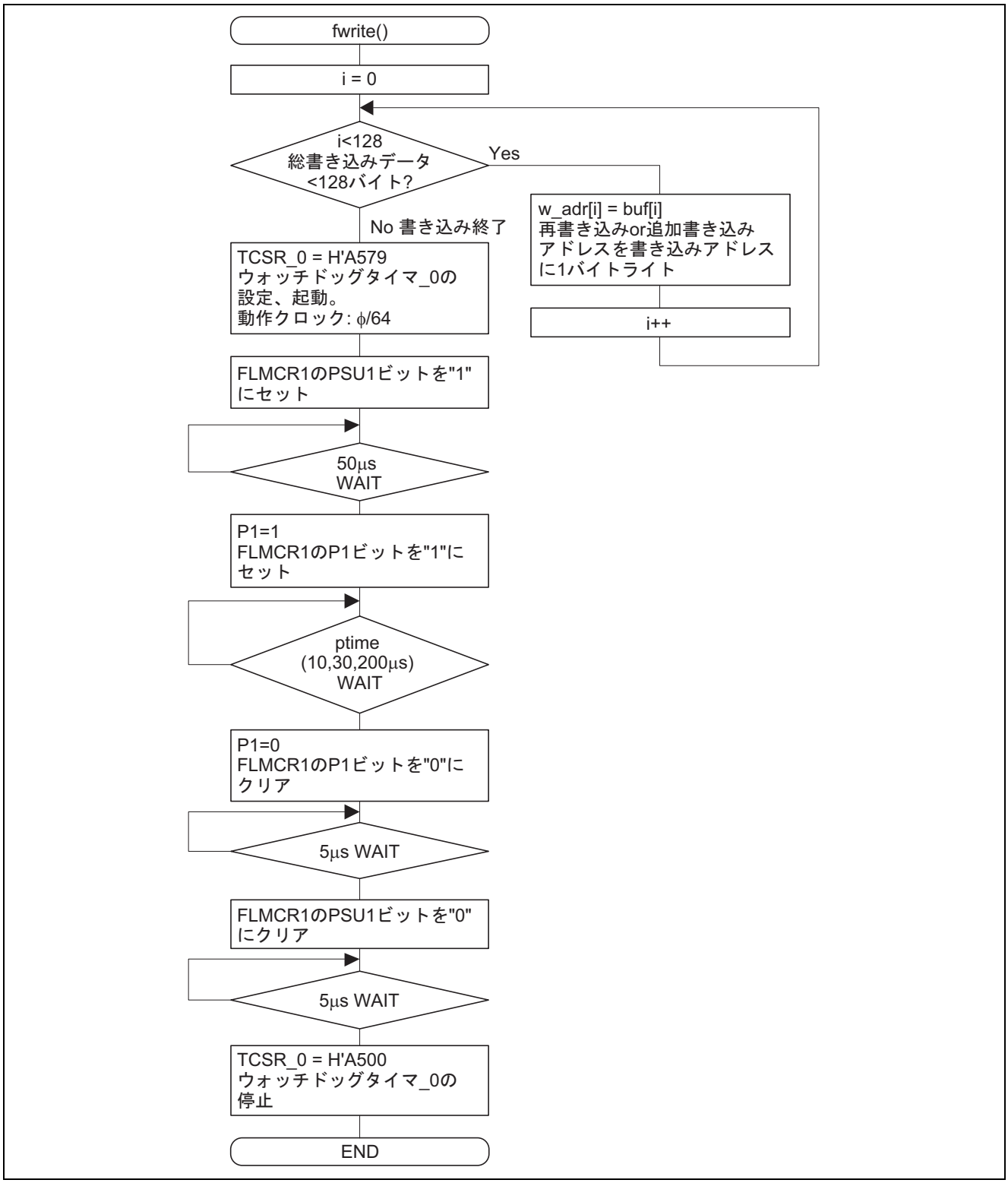
(f) 使用内部レジスタ

**表 17　fwrite()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | PSU1 | プログラムセットアップ<br>・PSU1 = 0 のとき，プログラムセットアップ状態を解除<br>・PSU1 = 1 のとき，プログラムセットアップ状態に遷移 | ビット 4 | 1 |
| | P1 | プログラム<br>・P1 = 0 のとき，プログラムモードを解除<br>・SWE1 = 1, PSU1 = 1 の状態で P1 = 1 のとき，プログラムモードに遷移 | ビット 0 | 1 |
| TCSR_0 *1 | | タイマコントロール/ステータスレジスタ 0 | H'FFFF74 | H'79 |
| | OVF | オーバフローフラグ<br>・OVF = 0 のとき，TCNT_0 がオーバフローしていない<br>・OVF = 1 のとき，TCNT_0 がオーバフローしている | ビット 7 | 0 |
| | WT/$\overline{IT}$ | タイマモードセレクト<br>・WT/$\overline{IT}$ = 0 のとき，インターバルタイマとして使用する<br>・WT/$\overline{IT}$ = 0 のとき，ウォッチドッグタイマとして使用する | ビット 6 | 1 |
| | TME | タイマイネーブル<br>・TME = 0 のとき，TCNT_0 がカウントを開始<br>・TME = 1 のとき，TCNT_0 がカウントを停止 | ビット 5 | 1 |
| | CKS2<br>CKS1<br>CKS0 | クロックセレクト 2～0<br>・CKS2 = 0, CKS1 = 0, CKS0 = 1 のとき，TCNT_0 にφ/64 のクロックを入力する | ビット 2<br>ビット 1<br>ビット 0 | CKS2 = 0<br>CKS1 = 0<br>CKS0 = 1 |

【注】 *1. TCSR_0 は，一般のレジスタとライト方法が異なります。
- ライト時は，H'FFFF74 に対してワード転送を行ないます。
- 上位バイトを H'A5 にし，下位バイトをライトデータにします。
- 本関数の場合，次のようにします。
  TCSR_0 = H'A579

(g) フローチャート


fwrite()
i = 0
i<128
総書き込みデータ
<128バイト?
Yes
No 書き込み終了
w_adr[i] = buf[i]
再書き込みor追加書き込み
アドレスを書き込みアドレス
に1バイトライト
i++
TCSR_0 = H'A579
ウォッチドッグタイマ_0の
設定、起動。
動作クロック: φ/64
FLMCR1のPSU1ビットを"1"
にセット
50μs
WAIT
P1=1
FLMCR1のP1ビットを"1"に
セット
ptime
(10,30,200μs)
WAIT
P1=0
FLMCR1のP1ビットを"0"に
クリア
5μs WAIT
FLMCR1のPSU1ビットを"0"
にクリア
5μs WAIT
TCSR_0 = H'A500
ウォッチドッグタイマ_0の
停止
END

(9) fwritevf()関数

(a) 仕様

```
char fwritevf(
unsigned short *owbuff,
unsigned short *buff,
unsigned short *wvf_buf,
unsigned short *wvf_adr
)
```

(b) 動作説明

- 対象アドレスのベリファイ，再書き込みデータの作成

(c) 引数の説明

- 入力値
  *owbuff：追加書き込みデータ 128 バイト
  *buff：再書き込みデータ 128 バイト
  *wvf_buf：書き込みデータ 128 バイト
  *wvf_adr：書き込みアドレス
- 出力値
  戻り値：結果フラグ (OK = H'00, NG = H'01, WNG = H'02)
  *owbuff：追加書き込みデータ 128 バイト
  *buff：再書き込みデータ 128 バイト
  *wvf_buf：書き込みデータ 128 バイト
  *wvf_adr：書き込みアドレス

(d) グローバル変数

なし

(e) 使用サブルーチン

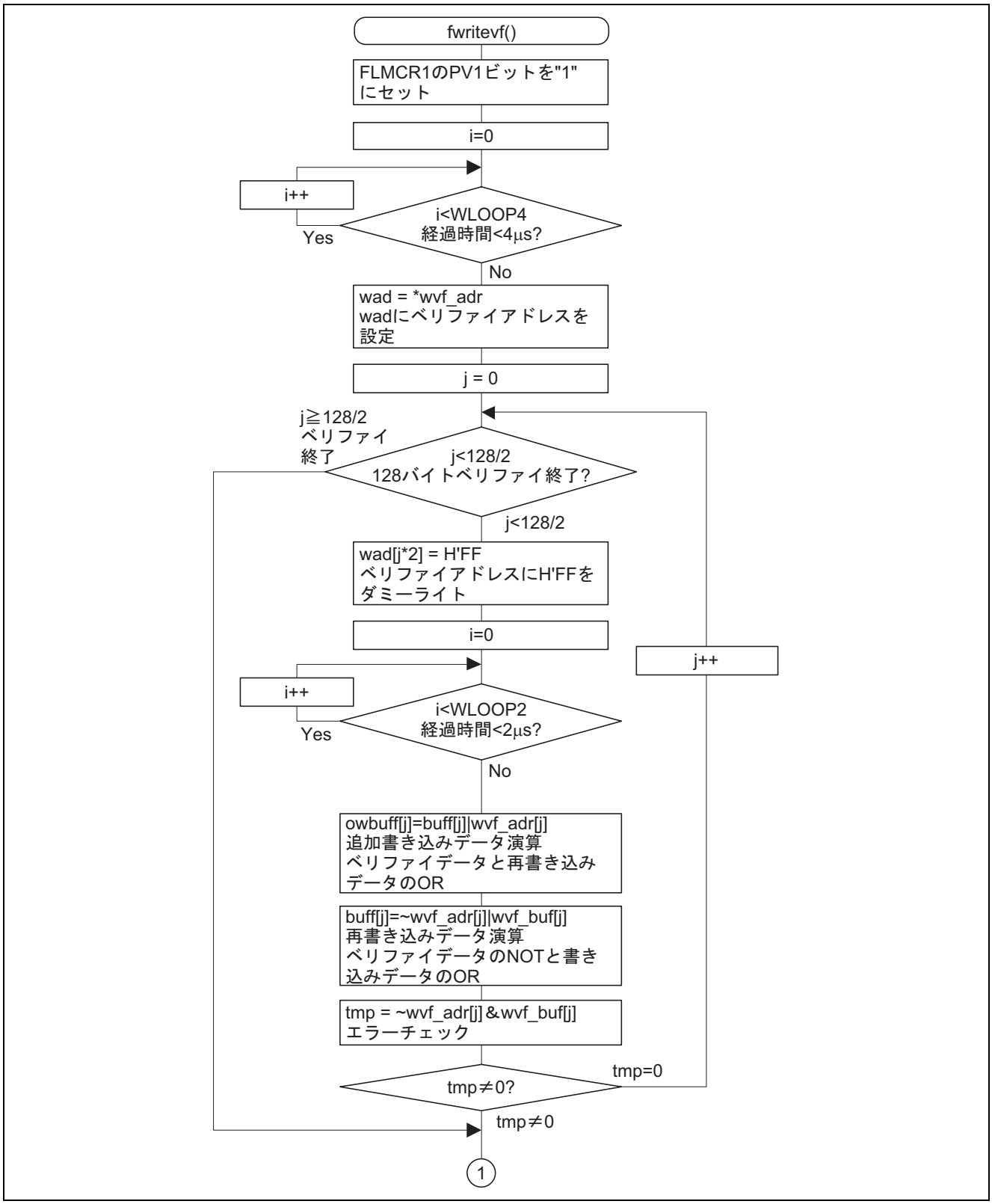
なし

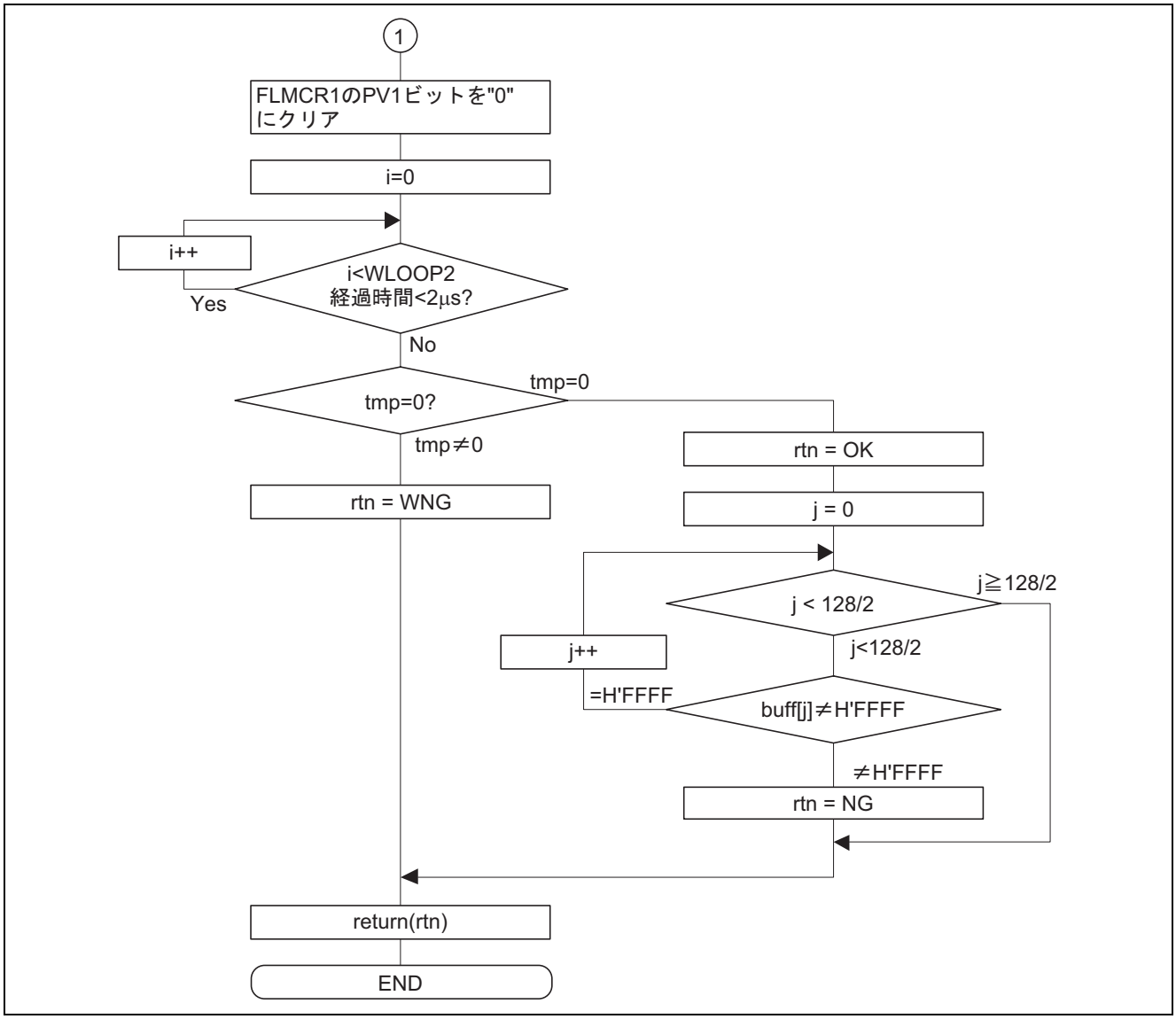
(f) 使用内部レジスタ

**表 18　fwritevf()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| FLMCR1 | | フラッシュメモリコントロールレジスタ 1 | H'FFFFA8 | — |
| | PV1 | プログラムベリファイ<br>・PV1 = 0 のとき，プログラムベリファイモードを解除<br>・PV1 = 1 のとき，プログラムベリファイモードに遷移 | ビット 2 | 1 |

(g) フローチャート


fwritevf()
FLMCR1のPV1ビットを"1"にセット
i=0
i++
i<WLOOP4
経過時間<4μs?
Yes
No
wad = *wvf_adr
wadにベリファイアドレスを設定
j = 0
j≧128/2
ベリファイ終了
j<128/2
128バイトベリファイ終了?
j<128/2
wad[j*2] = H'FF
ベリファイアドレスにH'FFをダミーライト
i=0
j++
i++
i<WLOOP2
経過時間<2μs?
Yes
No
owbuff[j]=buff[j]|wvf_adr[j]
追加書き込みデータ演算
ベリファイデータと再書き込みデータのOR
buff[j]=~wvf_adr[j]|wvf_buf[j]
再書き込みデータ演算
ベリファイデータのNOTと書き込みデータのOR
tmp = ~wvf_adr[j]&wvf_buf[j]
エラーチェック
tmp=0
tmp≠0?
tmp≠0
1


1
FLMCR1のPV1ビットを"0"
にクリア
i=0
i++
i<WLOOP2
経過時間<2μs?
Yes
No
tmp=0?
tmp=0
tmp≠0
rtn = OK
rtn = WNG
j = 0
j < 128/2
j≧128/2
j++
j<128/2
=H'FFFF
buff[j]≠H'FFFF
≠H'FFFF
rtn = NG
return(rtn)
END

# 8. 調歩同期式シリアル通信プログラム

## 8.1 階層構造

調歩同期式シリアル通信プログラムは，マスタ側との通信処理を行ないます。

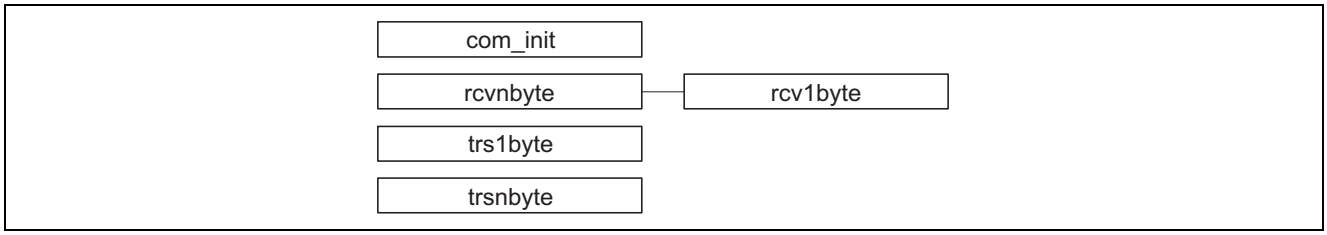



**図 21　調歩同期式シリアル通信プログラムの階層構造**

## 8.2 関数一覧

**表 19　調歩同期式シリアル通信プログラム関数一覧**

| 関数名 | 概要 |
|---|---|
| com_init | 調歩同期式シリアル通信の初期化 |
| rcv1byte | データを 1 バイト受信する |
| rcvnbyte | データを n バイト受信する |
| trs1byte | データを 1 バイト送信する |
| trsnbyte | データを n バイト送信する |

## 8.3 関数説明

(1) com_init()関数

(a) 仕様

void com_init(void)

(b) 動作説明

- 調歩同期式シリアル通信の初期化

(c) 引数の説明

- 入力値：なし
- 出力値：なし

(d) グローバル変数

なし

(e) 使用サブルーチン

なし

(f) 使用内部レジスタ

表 20　com_init()関数の使用レジスタ

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| MSTPCRB | | モジュールストップコントロールレジスタ B | H'FFFDE9 | H'7F |
| | MSTPB7 | シリアルコミュニケーションインターフェース 0<br>・MSTPB7 = 0 のとき，SCI_0 のモジュールストップモードを解除<br>・MSTPB7 = 1 のとき，SCI_0 をモジュールストップモードに設定 | ビット 7 | 0 |
| SMR_0 | | シリアルモードレジスタ | H'FFFF78 | H'00 |
| | C/$\overline{A}$ | コミュニケーションモード<br>・C/$\overline{A}$ = 0 のとき，コミュニケーションモードを調歩同期式モードに設定<br>・C/$\overline{A}$ = 1 のとき，コミュニケーションモードをクロック同期式モードに設定 | ビット 7 | 0 |
| | CHR | キャラクタレングス<br>・CHR = 0 のとき，調歩同期式モード時におけるデータ長を 8 ビットデータに設定<br>・CHR = 1 のとき，調歩同期式モード時におけるデータ長を 7 ビットデータに設定 | ビット 6 | 0 |
| | PE | パリティイネーブル<br>・PE = 0 のとき，調歩同期式モードで，送信時にパリティビットの付加およびチェックを禁止<br>・PE = 1 のとき，調歩同期式モードで，送信時にパリティビットの付加およびチェックを許可 | ビット 5 | 0 |
| | O/$\overline{E}$ | パリティモード<br>・O/$\overline{E}$ = 0 のとき，パリティの付加やチェックを偶数パリティに設定<br>・O/$\overline{E}$ = 1 のとき，パリティの付加やチェックを奇数パリティに設定 | ビット 4 | 0 |

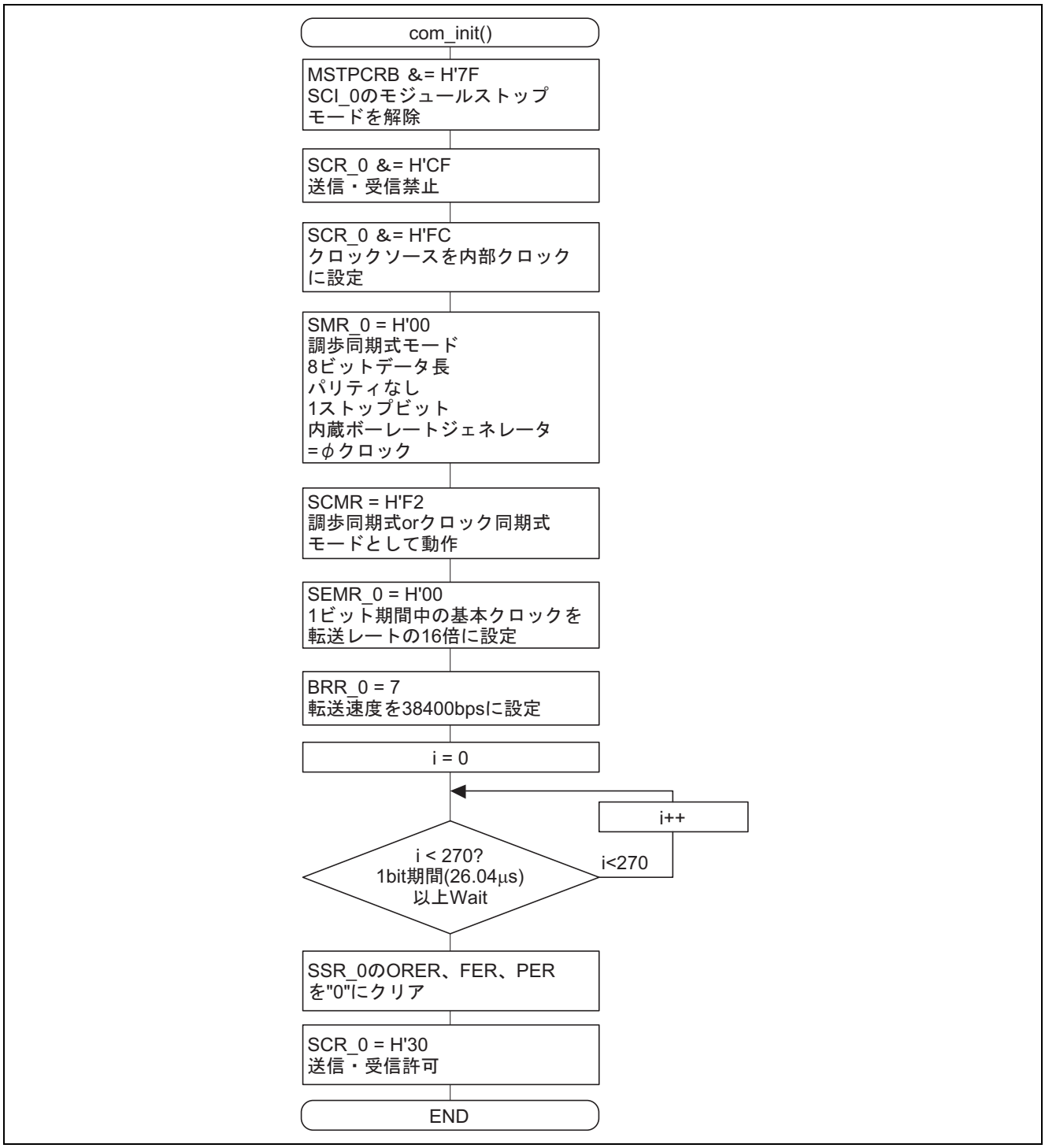
表 20　com_init()関数の使用レジスタ（続き）

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| SMR_0 | STOP | ストップビットレングス<br>・STOP = 0 のとき，調歩同期式モードで，ストップビットの長さを 1 ビットに設定<br>・STOP = 1 のとき，調歩同期式モードで，ストップビットの長さを 2 ビットに設定 | ビット 3 | 0 |
| | MP | マルチプロセッサモード<br>・MP = 0 のとき，マルチプロセッサ通信機能を禁止<br>・MP = 1 のとき，マルチプロセッサ通信機能を許可 | ビット 2 | 0 |
| | CKS1<br>CKS0 | クロックセレクト 1, 0<br>・CKS1 = 0, CKS0 = 0 のとき，内蔵ボーレートジェネレータのクロックソースをφクロックに設定 | ビット 1<br>ビット 0 | CKS1 = 0<br>CKS0 = 0 |
| BRR_0 | | ビットレートレジスタ<br>・BRR = H'09 のとき，SMR_0 の CKS1, CKS0 で選択されるボーレートジェネレータの動作クロックとあわせた送信のビットレートを 31250bps に設定 | H'FFFF79 | H'09 |
| SCR_0 | | シリアルコントロールレジスタ 3 | H'FFFF7A | — |
| | TE | トランスミットイネーブル<br>・TE = 0 のとき，送信動作を禁止<br>・TE = 1 のとき，送信動作を許可 | ビット 5 | 0 |
| | RE | レシーブイネーブル<br>・RE = 0 のとき，受信動作を禁止<br>・RE = 1 のとき，受信動作を許可 | ビット 4 | 0 |
| | CKE1<br>CKE0 | クロックイネーブル 1, 0<br>・CKE1 = 0, CKE0 = 0 のとき，調歩同期式モードにおいてクロックソースを内部クロック，SCK0 端子機能を入出力ポートに設定 | ビット 1<br>ビット 0 | CKE1 = 0<br>CKE0 = 0 |
| SSR | | シリアルステータスレジスタ | H'FFFF7C | — |
| | TDRE | トランスミットデータレジスタエンプティ<br>・TDRE = 0 のとき，TDR_0 にライトされた送信データが TSR_0 に転送されていないことを示す<br>・TDRE = 1 のとき，TDR_0 に送信データがライトされていない，または TDR_0 にライトされた送信データが TSR_0 に転送されたことを示す | ビット 7 | — |
| | RDRF | レシーブデータレジスタフル<br>・RDRF = 0 のとき，RDR_0 に受信データが格納されていない<br>・RDRF = 1 のとき，RDR_0 に受信データが格納されている | ビット 6 | — |
| | ORER | オーバランエラー<br>・ORER = 0 のとき，受信中，または受信を完了したことを示す<br>・ORER = 1 のとき，受信時にオーバランエラーが発生したことを示す | ビット 5 | 0 |
| | FER | フレーミングエラー<br>・FER = 0 のとき，受信中，または受信を完了したことを示す<br>・FER = 1 のとき，受信時にフレーミングエラーが発生したことを示す | ビット 4 | 0 |

表 20　com_init()関数の使用レジスタ (続き)

| | | | | |
|---|---|---|---|---|
| SSR_0 | PER | パリティエラー<br>・PER = 0 のとき，受信中，または受信を完了したことを示す<br>・PER = 1 のとき，受信時にパリティエラーが発生したことを示す | ビット 3 | 0 |
| | TEND | トランスミットエンド<br>・TEND = 0 のとき，送信中であることを示す<br>・TEND = 1 のとき，送信を終了したことを示す | ビット 2 | — |
| SCMR | | スマートカードモードレジスタ | H'FFFF7E | H'F2 |
| | SMIF | スマートカードインタフェースモードセレクト<br>・MIF = 0 のとき，調歩同期式またはクロック同期式モードとして動作<br>・MIF = 1 のとき，スマートカードインタフェースモードとして動作 | ビット 0 | 0 |
| SEMR_0 | | シリアル拡張モードレジスタ | H'FFFDF8 | H'00 |
| | ABCS | 調歩同期式基本クロックセレクト<br>・BCS = 0 のとき，調歩同期式モードにおいて 1 ビット期間の基本クロックを転送レートの 16 倍で動作<br>・BCS = 1 のとき，調歩同期式モードにおいて 1 ビット期間の基本クロックを転送レートの 8 倍で動作 | ビット 3 | 0 |
| | ACS2<br>ACS1<br>ACS0 | 調歩同期式クロックソースセレクト<br>・ACS2 = 0, ACS1 = 0, ACS0 = 0 のとき，調歩同期式のクロックソースを外部クロック入力に設定 | ビット 2<br>ビット 1<br>ビット 0 | ACS2 = 0<br>ACS1 = 0<br>ACS0 = 0 |

(g) フローチャート


com_init()
MSTPCRB &= H'7F
SCI_0のモジュールストップ
モードを解除
SCR_0 &= H'CF
送信・受信禁止
SCR_0 &= H'FC
クロックソースを内部クロック
に設定
SMR_0 = H'00
調歩同期式モード
8ビットデータ長
パリティなし
1ストップビット
内蔵ボーレートジェネレータ
=φクロック
SCMR = H'F2
調歩同期式orクロック同期式
モードとして動作
SEMR_0 = H'00
1ビット期間中の基本クロックを
転送レートの16倍に設定
BRR_0 = 7
転送速度を38400bpsに設定
i = 0
i++
i < 270?
1bit期間(26.04μs)
以上Wait
i<270
SSR_0のORER、FER、PER
を"0"にクリア
SCR_0 = H'30
送信・受信許可
END

(2) rcv1byte()関数

(a) 仕様

unsigned char rcv1byte (void)

(b) 動作説明

- 調歩同期式シリアルデータを 1 バイト受信

(c) 引数の説明

- 入力値：なし
- 出力値：1 バイト受信データ

(d) グローバル変数

なし

(e) 使用サブルーチン

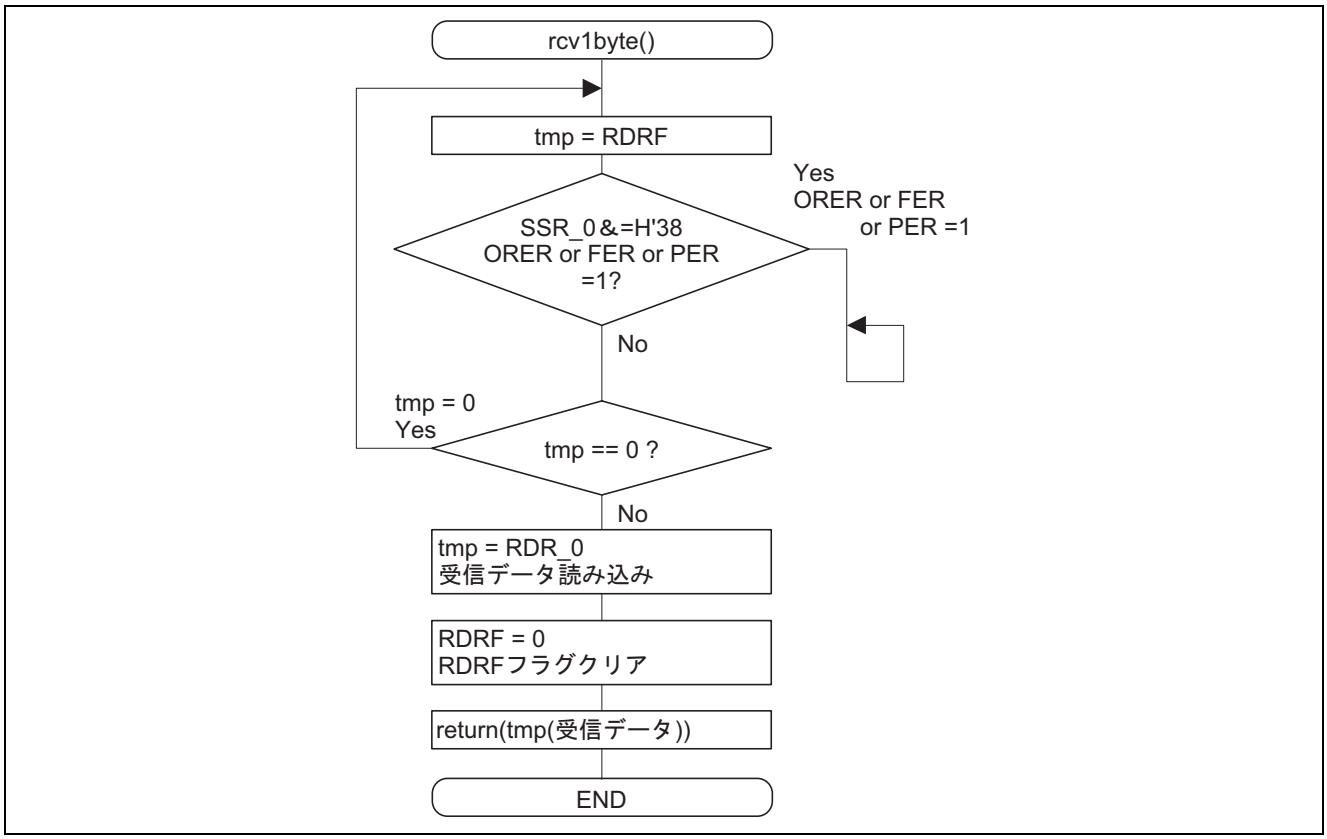
なし

(f) 使用内部レジスタ

**表 21　cv1byte()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| SSR_0 | | シリアルステータスレジスタ | H'FFFF7C | — |
| | RDRF | レシーブデータレジスタフル<br>・RDRF = 0 のとき，RDR_0 に受信データが格納されていない<br>・RDRF = 1 のとき，RDR_0 に受信データが格納されている | ビット 6 | — |
| | ORER | オーバランエラー<br>・ORER = 0 のとき，受信中，または受信を完了したことを示す<br>・ORER = 1 のとき，受信時にオーバランエラーが発生したことを示す | ビット 5 | — |
| | FER | フレーミングエラー<br>・FER = 0 のとき，受信中，または受信を完了したことを示す<br>・FER = 1 のとき，受信時にフレーミングエラーが発生したことを示す | ビット 4 | — |
| | PER | パリティエラー<br>・PER = 0 のとき，受信中，または受信を完了したことを示す<br>・PER = 1 のとき，受信時にパリティエラーが発生したことを示す | ビット 3 | — |
| RDR_0 | | レシーブデータレジスタ<br>・受信データを格納する 8 ビットのレジスタ | H'FFFF7D | — |

(g) フローチャート


rcv1byte()
tmp = RDRF
Yes
ORER or FER
or PER =1
SSR_0&=H'38
ORER or FER or PER
=1?
No
tmp = 0
Yes
tmp == 0 ?
No
tmp = RDR_0
受信データ読み込み
RDRF = 0
RDRFフラグクリア
return(tmp(受信データ))
END

(3) rcvnbyte()関数

(a) 仕様

void rcvnbyte(
unsigned char *ram
unsigned char dtno,
)

(b) 動作説明

- 調歩同期式シリアルデータを n バイト受信

(c) 引数の説明

- 入力値：
  *ram：受信データを格納する RAM 先頭アドレス
  dtno：受信バイト数
- 出力値：1 バイト受信データ
  *ram：受信データ

(d) グローバル変数

なし

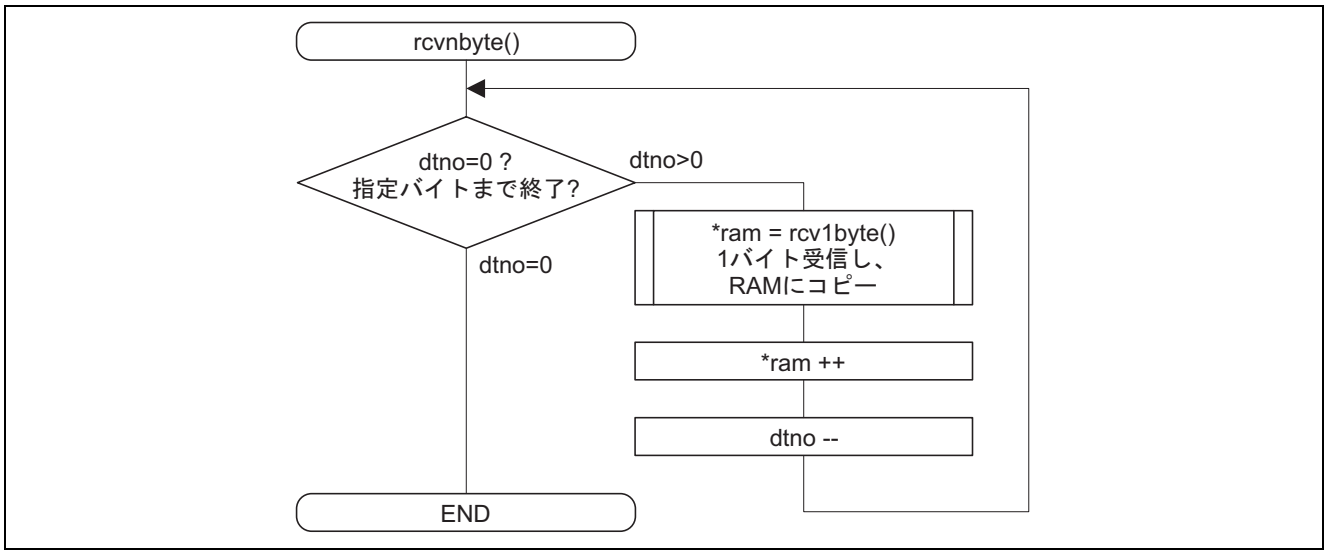
(e) 使用サブルーチン

rcv1byte：データを 1 バイト受信する

(f) 使用内部レジスタ

なし

(g) フローチャート



(4) trs1byte()関数

(a) 仕様

void trs1byte(unsigned char tdt)

(b) 動作説明

- 調歩同期式シリアルデータを 1 バイト送信

(c) 引数の説明

- 入力値：
  tdt:1 バイト送信データ
- 出力値：なし

(d) グローバル変数

なし

(e) 使用サブルーチン

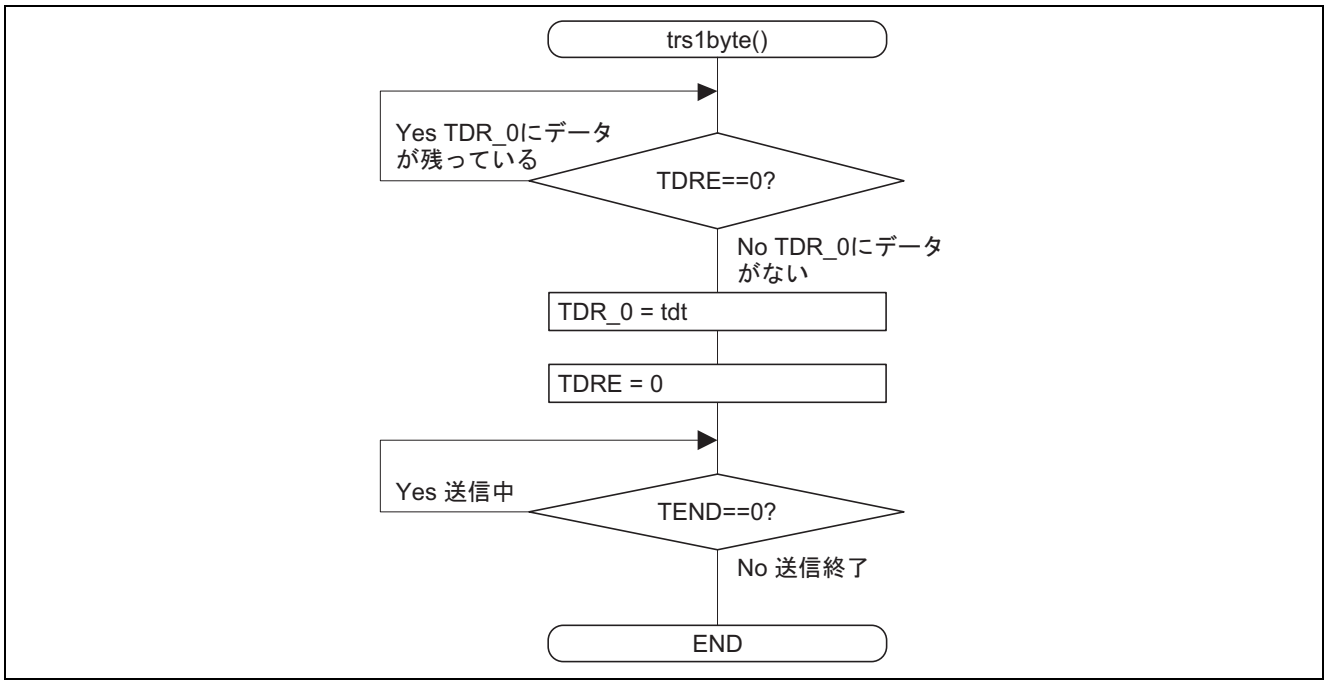
なし

(f) 使用内部レジスタ

**表 22　rs1byte()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| TDR_0 | | **トランスミットデータレジスタ**<br>**・送信データを格納する 8 ビットのレジスタ** | H'FFFF7B | — |
| SSR_0 | | **シリアルステータスレジスタ** | H'FFFF7C | — |
| | TDRE | **トランスミットデータレジスタエンプティ**<br>・TDRE = 0 **のとき，**TDR_0 **にライトされた送信データが** TSR_0 **に転送されていないことを示す**<br>・TDRE = 1 **のとき，**TDR_0 **に送信データがライトされていない，または** TDR_0 **にライトされた送信データが** TSR_0 **に転送されたことを示す** | **ビット** 7 | — |
| | TEND | **トランスミットエンド**<br>・TEND = 0 **のとき，送信中であることを示す**<br>・TEND = 1 **のとき，送信を終了したことを示す** | **ビット** 2 | — |

(g) フローチャート


trs1byte()
Yes TDR_0にデータ
が残っている
TDRE==0?
No TDR_0にデータ
がない
TDR_0 = tdt
TDRE = 0
Yes 送信中
TEND==0?
No 送信終了
END

(5) trsnbyte()関数

(a) 仕様

void trsnbyte (unsigned char *tdt, unsigned char dtno)

(b) 動作説明

- 調歩同期式シリアルデータを n バイト送信

(c) 引数の説明

- 入力値：
  *tdt:送信データの先頭アドレス
  dtno:送信サイズ
- 出力値：なし

(d) グローバル変数

なし

(e) 使用サブルーチン

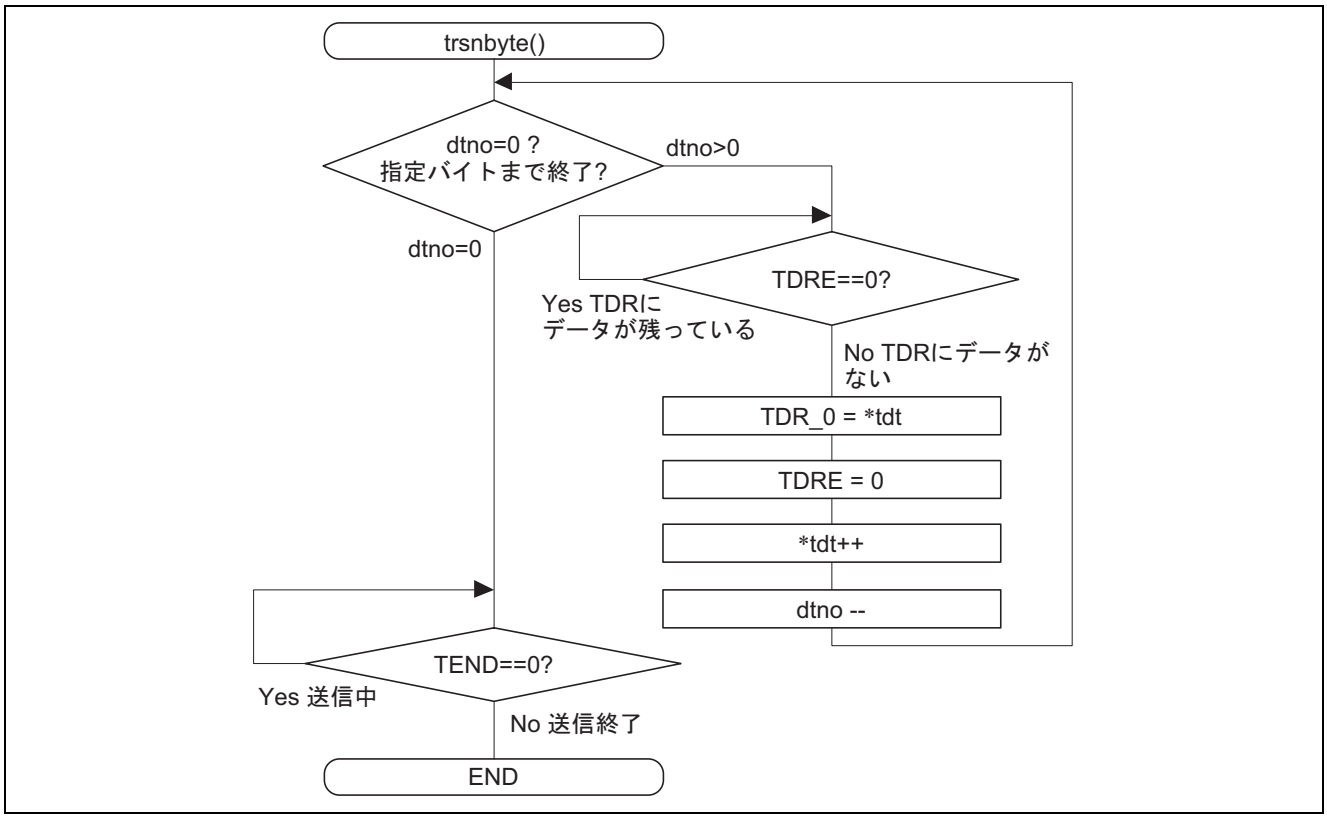
なし

(f) 使用内部レジスタ

**表 23　rs1byte()関数の使用レジスタ**

| レジスタ名 | ビット名 | 機能 | アドレス | 設定値 |
|---|---|---|---|---|
| TDR_0 | | **トランスミットデータレジスタ**<br>**・送信データを格納する 8 ビットのレジスタ** | H'FFFF7B | — |
| SSR_0 | | **シリアルステータスレジスタ** | H'FFFF7C | — |
| | TDRE | **トランスミットデータレジスタエンプティ**<br>・TDRE = 0 **のとき，**TDR_0 **にライトされた送信データが** TSR_0 **に転送されていないことを示す**<br>・TDRE = 1 **のとき，**TDR_0 **に送信データがライトされていない，または** TDR_0 **にライトされた送信データが** TSR_0 **に転送されたことを示す** | **ビット** 7 | — |
| | TEND | **トランスミットエンド**<br>・TEND = 0 **のとき，送信中であることを示す**<br>・TEND = 1 **のとき，送信を終了したことを示す** | **ビット** 2 | — |

(g) フローチャート


trsnbyte()
dtno=0 ?
指定バイトまで終了?
dtno>0
dtno=0
TDRE==0?
Yes TDRに
データが残っている
No TDRにデータが
ない
TDR_0 = *tdt
TDRE = 0
*tdt++
dtno --
TEND==0?
Yes 送信中
No 送信終了
END

# 9. プログラムリスト

## 9.1 スレーブ側通常プログラム

```
/************************************************************/
/*                                                          */
/*  H8S/2268F                                               */
/*  Flash Memory Write/Erase Application Note               */
/*                                                          */
/*  Communication Interface                                 */
/*  : Asynchronous Serial Interface                         */
/*  Function                                                */
/*  : Slave Main Program                                    */
/*                                                          */
/*  External Clock       : 10MHz                            */
/*  Internal Clock       : 10MHz                            */
/*  Sub Clock            : 32.768kHz                        */
/*                                                          */
/************************************************************/
#include                                                    <machine.h>
#include                                                    "string.h"

/************************************************************/
/*  Symbol Definition                                       */
/************************************************************/
struct BIT {
unsigned char                b7:1;               /* bit7 */
unsigned char                b6:1;               /* bit6 */
unsigned char                b5:1;               /* bit5 */
unsigned char                b4:1;               /* bit4 */
unsigned char                b3:1;               /* bit3 */
unsigned char                b2:1;               /* bit2 */
unsigned char                b1:1;               /* bit1 */
unsigned char                b0:1;               /* bit0 */
};
#define SSR_0_BIT            (*(volatile struct BIT *)0xFFFF7C)        /* Serial Status Register            */
#define RDRF_0               SSR_0_BIT.b6                              /* Receive Data Register Full        */
#define LPCR                 *(volatile unsigned char *)0xFFFC30       /* LCD Port Control Register         */
#define LCR                  *(volatile unsigned char *)0xFFFC31       /* LCD Control Register              */
#define LCR2                 *(volatile unsigned char *)0xFFFC32       /* LCD Control Register 2            */
#define LCDRAM               (volatile unsigned char *)0xFFFC4A        /* LCD RAM                           */
#define MSTPCRD              *(volatile unsigned char *)0xFFFC60       /* Module Stop Control Registers D   */
#define P1DDR                *(volatile unsigned char *)0xFFFE30       /* Port 1 Data Direction Register    */
#define P1DR                 *(volatile unsigned char *)0xFFFF00       /* Port 1 Data Register              */
#define P1DR_BIT             (*(volatile struct BIT *)0xFFFF00)        /* Port 1 Data Register              */
#define P11DR                P1DR_BIT.b1                               /* Port 11                           */
#define P10DR                P1DR_BIT.b0                               /* Port 10                           */
#define P7DDR                *(volatile unsigned char *)0xFFFE36       /* Port 7 data direction register    */
#define P7DR                 *(volatile unsigned char *)0xFFFF06       /* Port 7 data register              */
#define PORT7                *(volatile unsigned char *)0xFFFFB6       /* Port 7 register                   */
#define PORT7_BIT            (*(volatile struct BIT *)0xFFFFB6)        /* Port 7 register                   */
#define P70                  PORT7_BIT.b0                              /* Port 70                           */
```

```
/************************************************************/
/* Function define                                          */
/************************************************************/
extern void FZMAIN ( void );
void main ( void );
void copyfzram ( void );
extern void com_init ( void );
extern void trsnbyte ( unsigned char *tdt, unsigned char dtno );
extern unsigned char rcv1byte ( void );
extern unsigned char SAMPLEDT1[10];                     /* 0x001000 - 0x001005 Sample Data              */
extern unsigned char SAMPLEDT2[10];                     /* 0x005000 - 0x005005 Sample Data              */
extern unsigned char SAMPLEDT3[10];                     /* 0x02FFAA - 0x02FFFF Sample Data              */

/************************************************************/
/* Vector Address                                           */
/************************************************************/
#pragma section V1                                      /* VECTOR SECTOIN SET                           */
void (*const VEC_TBL1[])(void) = {
main                                                    /* 00 Reset                                     */
};

#pragma entry main(sp=0x00FFEFC0)
#pragma section                                         /* P                                            */
/************************************************************/
/* Main Program                                             */
/************************************************************/
void main ( void )
{
unsigned char        tmp;
unsigned char        tmp2;
unsigned char        swcnt;

set_ccr(0x80);
set_exr(0x00);

MSTPCRD = 0xBF;                                         /* module stop mode is cleared                  */

P7DDR = 0xF0;
P7DR = 0xE0;
P1DDR = 0xFF;
P1DR = 0xFF;

com_init();                                             /* Comunication Initialize                      */

swcnt = 0;                                              /* User Application Program Sample              */
do{
  if(swcnt == 1){
    trsnbyte(&SAMPLEDT1[0], 10);
  }
  else if(swcnt == 2){
    trsnbyte(&SAMPLEDT2[0], 10);
  }
  else if(swcnt == 3){
    trsnbyte(&SAMPLEDT3[0], 10);
  }

    swcnt++;
  if(swcnt > 3){
    swcnt = 1;
  }

  do{
    tmp2 = P70;
    tmp = RDRF_0;
    tmp2 = tmp2&(~tmp);
  }while(tmp2);

  if(tmp != 0)                                          /* Data Receive?                                */
    tmp = rcv1byte();
```

```
}while(tmp != 0x55);                                    /* Flash Memory Erase/Write Start?              */
/*------- Flash Memory Write Mode -------------*/
P1DDR = 0x03;
P10DR = 1;                                              /* LED1 OFF                                     */
P11DR = 0;                                              /* LED2 ON                                      */

copyfzram();

FZMAIN();                                               /* Flash Memory Write Main Program              */
}

#pragma section CPYFZRAM                                /* VECTOR SECTOIN SET                           */
/*************************************************************/
/* Copy FZTAT to RAM                                     */
/*************************************************************/
void copyfzram ( void )
{
  char *X_BGN;
  char *X_END;
  char *Y_BGN;

  X_BGN = __sectop("FZTAT");                            /* Flash , Ram Address Copy                     */
  X_END = __secend("FZEND");
  Y_BGN = __sectop("RAM");

memcpy(Y_BGN,X_BGN,X_END-X_BGN);                        /* Flash -> RAM Copy                            */
}
```

## 9.2 スレーブ側書き込み/消去制御プログラム

```
/************************************************************/
/*                                                          */
/*  H8S/2268F                                               */
/*  Flash Memory Write/Erase Application Note               */
/*                                                          */
/*  Communication Interface                                 */
/*  : Asynchronous Serial Interface                         */
/*  Function                                                */
/*  : Slave Flash Memory Write/Erase Control Program        */
/*                                                          */
/*  External Clock : 10MHz                                  */
/*  Internal Clock : 10MHz                                  */
/*  Sub Clock : 32.768kHz                                   */
/*                                                          */
/************************************************************/
#pragma section FZTAT

#include <machine.h>
#include "string.h"

/************************************************************/
/*  Symbol Definition                                       */
/************************************************************/
struct BIT {
unsigned char b7:1;                  /* bit7 */
unsigned char b6:1;                  /* bit6 */
unsigned char b5:1;                  /* bit5 */
unsigned char b4:1;                  /* bit4 */
unsigned char b3:1;                  /* bit3 */
unsigned char b2:1;                  /* bit2 */
unsigned char b1:1;                  /* bit1 */
unsigned char b0:1;                  /* bit0 */
};

#define FLMCR1          *(volatile unsigned char *)0xFFFFA8       /* Flash Memory Control Register 1       */
#define FLMCR1_BIT      (*(volatile struct BIT *)0xFFFFA8)        /* Flash Memory Control Register 1       */
#define FWE             FLMCR1_BIT.b7                             /* Flash Write Enable                    */
#define SWE1            FLMCR1_BIT.b6                             /* Software Write Enable                 */
#define ESU1            FLMCR1_BIT.b5                             /* Erase Setup                           */
#define PSU1            FLMCR1_BIT.b4                             /* Program Setup                         */
#define EV1             FLMCR1_BIT.b3                             /* Erase Verify                          */
#define PV1             FLMCR1_BIT.b2                             /* Program Verify                        */
#define E1              FLMCR1_BIT.b1                             /* Erase                                 */
#define P1              FLMCR1_BIT.b0                             /* Program                               */
#define FLMCR2          *(volatile unsigned char *)0xFFFFA9       /* Flash Memory Control Register 2       */
#define FLMCR2_BIT      (*(volatile struct BIT *)0xFFFFA9)        /* Flash Memory Control Register 2       */
#define FLER            FLMCR2_BIT.b7                             /* FLER                                  */
#define EBR1            *(volatile unsigned char *)0xFFFFAA       /* Erase Block Register 1                */
#define EBR2            *(volatile unsigned char *)0xFFFFAB       /* Erase Block Register 2                */
#define RAMER           *(volatile unsigned char *)0xFFFEDB       /* RAM Emulation Register                */
#define FLPWCR          *(volatile unsigned char *)0xFFFFAC       /* Flash Memory Power Control Register   */
#define SCRX            *(volatile unsigned char *)0xFFFDB4       /* Serial Control Register X             */
#define SCRX_BIT        (*(volatile struct BIT *)0xFFFDB4)        /* Serial Control Register X             */
#define FLSHE           SCRX_BIT.b3                               /* Flash Memory Control Register Enable  */
#define TCSRW_0         *(volatile unsigned short *)0xFFFF74      /* Timer Control/Status Register W       */
#define TCNTW_0         *(volatile unsigned short *)0xFFFF74      /* Timer Counter W                       */
#define RSTCSR          *(volatile unsigned short *)0xFFFF76      /* Timer Control/Status Register W       */
#define P1DDR           *(volatile unsigned char *)0xFFFE30       /* Port 1 Data Direction Register        */
#define P1DR            *(volatile unsigned char *)0xFFFF00       /* Port 1 Data Register                  */
#define P1DR_BIT        (*(volatile struct BIT *)0xFFFF00)        /* Port 1 Data Register                  */
#define P11DR           P1DR_BIT.b1                               /* Port 11                               */
#define P10DR           P1DR_BIT.b0                               /* Port 10                               */
#define IER             *(volatile unsigned char *)0xFFFE14       /* IRQ Enable Register                   */
#define ADCR            *(volatile unsigned char *)0xFFFF99       /* A/D Control Register                  */
#define PFDDR           *(volatile unsigned char *)0xFFFE3E       /* Port F Data Direction Register        */
#define PFDR            *(volatile unsigned char *)0xFFFF0E       /* Port F Data Register                  */
```

```
/************************************************************/
/* Function define                                          */
/************************************************************/
void FZMAIN ( void );
void fwe_check ( void );
char blk1_erase ( unsigned long ers_ad, unsigned char ET_COUNT );
char blk_check ( unsigned long eck_ad,unsigned long *eck_st, unsigned long *eck_ed, unsigned char *blk_no );
void ferase ( unsigned char e_blk_no );
char ferasevf ( unsigned short *evf_st, unsigned short *evf_ed );
char fwrite128 ( unsigned char *wt_buf, unsigned char *wt_adr, unsigned short WT_COUNT );
void fwrite ( unsigned char *buf, unsigned char *w_adr, unsigned short ptime );
char fwritevf ( unsigned short *owbu ff, unsigned short *buff , un signed s hort *wvf_buf, unsigned s hort
*wvf_adr );
extern unsigned char rcv1byte ( void );
extern void rcvnbyte ( unsigned char *ram, unsigned char dtno );
extern void trs1byte ( unsigned char tdt );

/************************************************************/
/* ROM define                                               */
/************************************************************/

/****** WAIT TIME ****************************************/
#define MHZ           10                                    /* 20MHZ                                */
#define KEISU1        3                                     /* 1Loop 3Step <-- DEC.B(1)+BNE(2)      */
#define KEISU2        5                                     /* 1Loop 5Step <-- INC.W(1)+CMP.W(2)+BCS(2)  */
#define WLOOP1        1*MHZ/KEISU1+1 /* LOOP WAIT TIME */
#define WLOOP2        2*MHZ/KEISU1+1
#define WLOOP4        4*MHZ/KEISU1+1
#define WLOOP5        5*MHZ/KEISU1+1
#define WLOOP10       10*MHZ/KEISU1+1
#define WLOOP20       20*MHZ/KEISU1+1
#define WLOOP50       50*MHZ/KEISU2+1
#define WLOOP100      100*MHZ/KEISU2+1
#define TIME10        10*MHZ/KEISU1+1                       /* WRITE WAIT TIME                      */
#define TIME30        30*MHZ/KEISU1+1                       /* WRITE WAIT TIME                      */
#define TIME200       200*MHZ/KEISU2+1                      /* WRITE WAIT TIME                      */
#define TIME10000     10000*MHZ/KEISU2+1                    /* ERASE WAIT TIME                      */

/****** Fixed number definition ***************************/
unsigned long BLOCKADR[13] ={                               /* Erase Block Address                  */
0x000000,                                                   /* EB0 4KBYTE                           */
0x001000,                                                   /* EB1 4KBYTE                           */
0x002000,                                                   /* EB2 4KBYTE                           */
0x003000,                                                   /* EB3 4KBYTE                           */
0x004000,                                                   /* EB4 4KBYTE                           */
0x005000,                                                   /* EB5 4KBYTE                           */
0x006000,                                                   /* EB6 4KBYTE                           */
0x007000,                                                   /* EB7 4KBYTE                           */
0x008000,                                                   /* EB8 32KBYTE                          */
0x010000,                                                   /* EB9 64KBYTE                          */
0x020000,                                                   /* EB10 64KBYTE                         */
0x030000,                                                   /* EB11 64KBYTE                         */
0x040000                                                    /* End Block Address                    */
};

#define MAXBLK1       12
#define OK            0
#define NG            1
#define WNG           2
#define OW_COUNT 6                                          /* Over Write Count                     */
/************************************************************/
/* Flash Memory Write Main Program                          */
/************************************************************/
void FZMAIN ( void )
{
  char rtn;
  unsigned char i,tmp;
```

```
unsigned char                rcvndt[2];
  unsigned long              E_ADR[12];
  unsigned char              W_BUF[128];                  /* Write Data Area                          */
  union{
    unsigned char wtdt[8];
    struct{
        unsigned long ad_tmp;
        unsigned long restsize;
    }lw;
  }rcv;

  trs1byte(OK);                                           /* SEND OF OK Code                          */

  tmp = rcv1byte();                                       /* Recive 1byte Data -> RAM Area            */
  if(tmp != 0x66)
    goto ERRCASE;

  FLPWCR = 0x80;                                          /* flash power-down modes disabled          */
  fwe_check();                                            /* Set FWE */

  trs1byte(OK);                                           /* SEND OF OK Code                          */
  RSTCSR = 0x5A5F;                                        /* LSI Reset if WDT overflows               */
  TCSRW_0 = 0xA500;                                       /* WDT STOP                                 */

/*------- Erase --------------------------------*/
  rcvnbyte(rcvndt, 2);                                    /* RECEIVE ERASE BLOCK NUMBER               */
  if(rcvndt[0] != 0x77)                                   /* Recive Code = 0x77?                      */
    goto ERRCASE;

  trs1byte(OK);                                           /* SEND OF OK Code                          */

  tmp = rcvndt[1] << 2;
  rcvnbyte((unsigned char*)E_ADR, tmp);                   /* Recive ERASE BLOCK Address               */

  for(i = 0; i < rcvndt[1]; i++){
    rtn = blk1_erase(E_ADR[i], 3);                        /* 1 block Erase                            */
    if(rtn != OK)
        goto ERRCASE;
  }

  trs1byte(OK);                                           /* SEND OF OK Code                          */
/*------- Write Address / Size Recive --------*/

  tmp = rcv1byte();                                       /* Recive 1byte Data -> RAM Area            */
  if(tmp != 0x88)
    goto ERRCASE;

  trs1byte(OK);                                           /* SEND OF OK Code                          */

  rcvnbyte(rcv.wtdt, 8);                                  /* Recive Write Top Address & Size          */
  if(rcv.wtdt[3] & 0x7F)
    goto ERRCASE;

  if(rcv.lw.restsize == 0x0000){
    goto ERRCASE;
  }

  trs1byte(OK);                                           /* SEND OF OK Code                          */

/*------- 128 byte Flash Memory Write ---------*/

  while(rcv.lw.restsize != 0){
    trs1byte(0x11);                                       /* SEND OF Request                          */

    if(rcv.lw.restsize <= 128){                           /* Receive WriteData from HOST              */
        memset(W_BUF,0xFF,128);                           /* INITIALIZE RECEIVE BUFFER (0xFF)         */
        rcvnbyte(W_BUF,(unsigned char)rcv.lw.restsize);   /* "restsize" byte Receive                  */
```

```
      rcv.lw.restsize = 0;
      }
      else{
          rcvnbyte(W_BUF, 128);                          /* 128byte Receive                    */
          rcv.lw.restsize -= 128;
      }

      rtn = fwrite128(W_BUF, (unsigned char*)rcv.lw.ad_tmp, 1000);
      if(rtn != OK)
          goto ERRCASE;

      rcv.lw.ad_tmp = rcv.lw.ad_tmp + 128;
      }

    trs1byte(OK);                                        /* SEND OF OK Code                    */
    P10DR = 0;                                           /* LED1 ON                            */
    P11DR = 1;                                           /* LED2 OFF                           */

    TCNTW_0 = 0x5AFF;                                    /* INITIZLIZED WDT COUNT              */
    TCSRW_0 = 0xA578;                                    /* WDT START phi/2                    */

    while(1);                                            /* OK End */

/*------- Error Case --------------------------*/
ERRCASE:                                                 /* Error Case                         */
    trs1byte(NG);
    P10DR = 0;                                           /* LED1 ON                            */
    P11DR = 0;                                           /* LED2 ON                            */
    while(1);
}

/************************************************************/
/* FWE Check                                                */
/************************************************************/
void fwe_check ( void )
{
    unsigned char tmp;

    IER = 0x00;                                          /* IRQ3 Disable                       */
    ADCR = 0x00;                                         /* ADTRG OFF                          */
    PFDDR = 0x08;                                        /* PF3 Output Setting                 */

    PFDR = 0x08;                                         /* Set PF3 / Set FEW                  */
    SCRX = 0x08;                                         /* FLSHE=1                            */
    RAMER = 0x00;                                        /* RAM Emulation Register OFF         */

    do{
      tmp = FWE;
    }while(tmp==0);                                      /* FWE Set?                           */
}

/************************************************************/
/* Flash Memory 1 block Erase                               */
/************************************************************/
char blk1_erase ( unsigned long ers_ad, unsigned char ET_COUNT )
{
    char rtn;
    unsigned char i;
    unsigned short j;
    unsigned char block_no;
    unsigned long ers_st,ers_ed;

    rtn = blk_check(ers_ad,&ers_st,&ers_ed,&block_no);   /* CHECK BLOCK START ADDRESS          */

    if(rtn == OK){
      SWE1 = 1;                                          /* Set the SWE1 bit                   */
      for(i = 0; i < WLOOP1; i++);                       /* Need to wait 1 usec                */

      rtn = ferasevf((unsigned short*)ers_st,            /* Erase Verify                       */
```

```
                (unsigned short*)ers_ed);

    for(i = 0; i < ET_COUNT; i++){                          /* Count Check (Max Erase count)                */
        if(!rtn)
            break;
        ferase(block_no);                                   /* Erase */
        rtn = ferasevf((unsigned short*)ers_st,             /* Erase Verify                                 */
                (unsigned short*)ers_ed);
    }

    SWE1 = 0; /* Clear the SWE1 bit */
    for(j = 0; j < WLOOP100; j++);                          /* Need to wait 100 usec                        */
  }

  return(rtn);
}
/************************************************************/
/* Erase Block Check Routin                                 */
/************************************************************/
char blk_check ( unsigned long eck_ad,unsigned long *eck_st, unsigned long *eck_ed, unsigned char *blk_no )
{
  unsigned char i;

  for(i = 0; eck_ad != BLOCKADR[i]; i++){                   /* COMPARE BLOCK_START_ADDRESS                  */
    if(MAXBLK1 < i)                                         /* BLOCK NUMBER MAX?                            */
        return(NG);                                         /* ERASE BLOCK ADDRESS ERROR                    */
  }

  *blk_no = i;                                              /* ERASE BLOCK NUMBER                           */
  *eck_st = BLOCKADR[i];                                    /* ERASE START ADDRESS                          */
  i++;
  *eck_ed = BLOCKADR[i]-1;                                  /* ERASE END ADDRESS                            */

  return(OK);
}

/************************************************************/
/* Erase                                                    */
/************************************************************/
void ferase ( unsigned char e_blk_no )
{
  unsigned char i;
  unsigned short j;
  unsigned char tmp;

  tmp = 1;
  if(e_blk_no < 8){
    tmp <<= e_blk_no;
    EBR1 = tmp;                                             /* Set the EBR1 Erase Block bit                 */

  }
  else{
    e_blk_no = e_blk_no - 8;
    tmp <<= e_blk_no;
    EBR2 = tmp;                                             /* Set the EBR2 Erase Block bit                 */
  }

  TCSRW_0 = 0xA57F;                                         /* WDT START phi/13072                          */

  ESU1 = 1;                                                 /* Set the ESU1 bit                             */
  for(j = 0; j < WLOOP100; j++);                            /* Need to wait 100 usec                        */

  E1 = 1;                                                   /* Set the E1 bit (ERASE)                       */
  for(j = 0; j < TIME10000; j++);                           /* Need to wait 10 msec                         */

  E1 = 0;                                                   /* Clear the E1 bit                             */
  for(i = 0; i < WLOOP10; i++);                             /* Need to wait 10 usec                         */

  ESU1 = 0;                                                 /* Clear the ESU1 bit                           */
```

```
    for(i = 0; i < WLOOP10; i++);                           /* Need to wait 10 usec                          */

    TCSRW_0 = 0xA500;                                       /* WDT STOP                                      */

    EBR1 = 0;
    EBR2 = 0;
  }

  /*********************************************************/
  /* Erase Verify                                          */
  /*********************************************************/
  char ferasevf ( unsigned short *evf_st, unsigned short *evf_ed )
  {
    char rtn;
    unsigned char i;
    unsigned short j;
    unsigned char *ead;

    EV1 = 1;                                /* Set the EV bit                                                */
    for(i = 0; i < WLOOP20; i++);                           /* Need to wait 20 usec                          */

    rtn = OK;
    ead = (unsigned char*)evf_st;
    for( j = 0; &evf_st[j] < evf_ed; j++){
      ead[j*2] = 0xFF;                      /* Perform dummy write                                           */
      for(i = 0; i < WLOOP2; i++);                          /* Need to wait 2 usec                           */
      if(evf_st[j] != 0xFFFF){                              /* Verify                                        */
          rtn = NG;                                         /* NG flag set                                   */
          break;
      }
    }

    EV1 = 0;                                /* Clear the EV bit                                              */
    for(i = 0; i < WLOOP4; i++);                            /* Need to wait 4 usec                           */

    return(rtn);                                            /* OK flag set                                   */
  }

  /*********************************************************/
  /* Flash Memory 128 byte Write                           */
  /*********************************************************/
  char fwrite128 ( unsigned char *wt_buf, unsigned char *wt_adr, unsigned short WT_COUNT )
  {
    char rtn;
    unsigned char i;
    unsigned short j;
    unsigned short TM;
    unsigned char OWBUFF[128];                              /* Over Write Data Area                          */
    unsigned char BUFF[128];                                /* Retry Write Data Area                         */

    memcpy(BUFF,wt_buf,128);                                /* W_BUF -> BUFF BLOCK COPY                      */
    SWE1 = 1;                                               /* Set the SWE1 bit                              */
    for(i = 0; i < WLOOP1; i++);                            /* Need to wait 1 usec                           */

    rtn = fwritevf((unsigned short *)OWBUFF,                /* 1st Program Verify                            */
                   (unsigned short *)BUFF,
                   (unsigned short *)wt_buf,
                   (unsigned short *)wt_adr);
    if(rtn == NG){                                          /* 1st Verify END                                */
      TM = TIME30;                                          /* Input P Palse(30 usec)                        */
      for(j = 0; j < WT_COUNT; j++){
          fwrite(BUFF,wt_adr,TM);                           /* Input P Palse(10,30,200 usec)                 */
          rtn = fwritevf((unsigned short *)OWBUFF,
                         (unsigned short *)BUFF,
                         (unsigned short *)wt_buf,
                         (unsigned short *)wt_adr);
```

```
          if(j < OW_COUNT){                                   /* Count Check(additive Write Count)          */
              fwrite(OWBUFF,wt_adr,TIME10);                   /* Input P Palse(10 usec)                     */
          }
          else{
              TM = TIME200;                                   /* Input P Palse(200 usec)                    */
          }

          if(rtn != NG){
              break;                                          /* NG Write Over Error                        */
          }
      }
    }

    SWE1 = 0; /* Clear the SWE1 bit */
    for(j = 0; j < WLOOP100; j++);                            /* Need to wait 100 usec                      */
    return(rtn);
}

/************************************************************/
/* Flash Memory Write              */
/************************************************************/
void fwrite ( unsigned char *buf, unsigned char *w_adr, unsigned short ptime )
{
    unsigned char i;
    unsigned short j;

    for(i = 0; i < 128; i++){                                 /* 128 byte repeat                            */
      w_adr[i] = buf[i];                                      /* Rewrite data dummy write                   */
    }

    TCSRW_0 = 0xA579;                                         /* WDT START phi/64                           */

    PSU1 = 1;                                                 /* Set the PSU1 bit                           */
    for(j = 0; j < WLOOP50; j++);                             /* Need to wait 50 usec                       */

    P1 = 1;                                                   /* Set the P1 bit                             */
    for(j = 0; j < ptime; j++);                               /* Writing Time 10/30/200 usec                */

    P1 = 0;                                                   /* Clear the P1 bit                           */
    for(i = 0; i < WLOOP5; i++);                              /* Need to wait 5 usec                        */

    PSU1 = 0;                                                 /* Clear the PSU1 bit                         */
    for(   i = 0; i < WLOOP5; i++);                           /* Need to wait 5 usec                        */

    TCSRW_0 = 0xA500;                                         /* WDT STOP                                   */
}

/************************************************************/
/* Flash Memory Verify                                      */
/************************************************************/
char fwritevf ( unsigned short *owbu ff, unsigned short *buff , un signed s hort *wvf_buf, unsigned s hort
*wvf_adr )
{
    char rtn;
    unsigned char i;
    unsigned char j;
    unsigned short tmp;
    unsigned char *wad;

    PV1 = 1;                                                  /* Set the PV1 bit                            */
    for(i = 0; i < WLOOP4; i++);                              /* Need to wait 4 usec                        */

    wad = (unsigned char*)wvf_adr;
    for(j = 0; j < 128/2; j++){
      wad[j*2] = 0xFF;                                        /* Dummy Write                                */
      for(i = 0; i < WLOOP2; i++);                            /* Need to wait 2 usec                        */

      owbuff[j] = buff[j] | wvf_adr[j];
```

```
    tmp = ~wvf_adr[j];
    buff[j] = tmp | wvf_buf[j];

    tmp = tmp & wvf_buf[j];                          /* Error Check                    */
    if(tmp != 0)
      break;
    }

    PV1 = 0;                                         /* PV1 bit Clear                  */
    for(i = 0; i < WLOOP2; i++);                     /* Need to wait 2 usec            */

    if(tmp == 0){
      rtn = OK;
      for(j = 0; j < 128/2; j++){                    /* 128 byte OK?                   */
          if(buff[j] != 0xFFFF){                     /* Error Check                    */
              rtn = NG;
              break;
          }
      }
    }

    else{
      rtn = WNG;                                     /* Write Error                    */
    }

    return(rtn);
}

#pragma section FZEND
```

## 9.3 調歩同期式シリアル通信プログラム

```
/************************************************************/
/*                                                          */
/* H8S/2268F                                                */
/* SCI Program                                              */
/*                                                          */
/* External Clock : 10MHz                                   */
/* Internal Clock : 10MHz                                   */
/* Sub Clock : 32.768kHz                                    */
/*                                                          */
/************************************************************/
#pragma section ASSCI

#include <machine.h>

/************************************************************/
/* Symbol Definition                                        */
/************************************************************/
struct BIT {
unsigned char    b7:1;    /* bit7 */
unsigned char    b6:1;    /* bit6 */
unsigned char    b5:1;    /* bit5 */
unsigned char    b4:1;    /* bit4 */
unsigned char    b3:1;    /* bit3 */
unsigned char    b2:1;    /* bit2 */
unsigned char    b1:1;    /* bit1 */
unsigned char    b0:1;    /* bit0 */
};

#define SMR_0         *(volatile unsigned char *)0xFFFF78    /* Serial Mode Register                        */
#define BRR_0         *(volatile unsigned char *)0xFFFF79    /* Bit Rate Register                           */
#define SCR_0         *(volatile unsigned char *)0xFFFF7A    /* Serial Control Register 3                   */
#define SCR_0_BIT     (*(volatile struct BIT *)0xFFFF7A)     /* Serial Control Register 3                   */
#define TE_0          SCR_0_BIT.b5                           /* Transmit Enable                             */
#define RE_0          SCR_0_BIT.b4                           /* Receive Enable                              */
#define CKE1_0        SCR_0_BIT.b1                           /* Clock Enable 1                              */
#define CKE0_0        SCR_0_BIT.b0                           /* Clock Enable 0                              */
#define TDR_0         *(volatile unsigned char *)0xFFFF7B    /* Transmit Data Register                      */
#define SSR_0         *(volatile unsigned char *)0xFFFF7C    /* Serial Status Register                      */
#define SSR_0_BIT     (*(volatile struct BIT *)0xFFFF7C)     /* Serial Status Register                      */
#define TDRE_0        SSR_0_BIT.b7                           /* Transmit Data Register Empty                */
#define RDRF_0        SSR_0_BIT.b6                           /* Receive Data Register Full                  */
#define ORER_0        SSR_0_BIT.b5                           /* Overrun Erorr                               */
#define FER_0         SSR_0_BIT.b4                           /* Framing Erorr                               */
#define PER_0         SSR_0_BIT.b3                           /* Parity Erorr                                */
#define TEND_0        SSR_0_BIT.b2                           /* Transmit End                                */
#define RDR_0         *(volatile unsigned char *)0xFFFF7D    /* Receive data Register                       */
#define SCMR          *(volatile unsigned char *)0xFFFF7E    /* Smart Card Mode Register                    */
#define SEMR_0        *(volatile unsigned char *)0xFFFDF8    /* Serial Expansion Mode Register              */
#define MSTPCRB       *(volatile unsigned char *)0xFFFDE9    /* Module Stop Control Registers C             */

/************************************************************/
/* Communication Initialize                                 */
/************************************************************/
void com_init ( void )
{
  unsigned short i;

  MSTPCRB &= 0x7F;                                           /* module stop mode is cleared                 */

  SCR_0 &= 0xCF;                                             /* TE,RE=0                                     */
  SCR_0 &= 0xFC;                                             /* CKE1,CKE0=0                                 */
  SMR_0 = 0x00;                                              /* Initialize Serial Mode Register             */
  SCMR = 0xF2;                                               /* Don't use Smart Card                        */
  SEMR_0 = 0x00;                                             /* 1bit-interval base clock is                 */
                                                             /* 16times the transfer rate.                  */
  BRR_0 = 7;                                                 /* 38400 bps phi=10MHz                         */
```

```
  for(i = 0; i < 270; i++);                                 /* Dummy Loop ,26.04us over Wait                */

  i = SSR_0;
  SSR_0 &= 0xC7;                                            /* ORER,FER,PER=0                               */

  SCR_0 = 0x30;                                             /* TE=1,RE=1                                    */
}

/************************************************************/
/* Receive 1 byte                                           */
/************************************************************/
unsigned char rcv1byte ( void )
{
  unsigned char tmp;

  do{
    tmp = RDRF_0;
    if(SSR_0 & 0x38)                  /* ORER/FER/PER = 1 ? */
        while(1);                                           /* Receive Error                                */
  }while(tmp == 0);                                         /* End Serial Receiving                         */

  tmp = RDR_0;                                              /* Read Receive data                            */
  RDRF_0 = 0;                                               /* Clear RDRF bit                               */

  return(tmp);
}

/************************************************************/
/* Receive N byte                                           */
/************************************************************/
void rcvnbyte ( unsigned char *ram, unsigned char dtno )
{
  while(dtno--){                                            /* dtno = 0 ?                                   */
    *ram = rcv1byte();                                      /* 1byte Receive Data -> RAM                    */
    *ram++;
  }
}

/************************************************************/
/* Transmit 1 byte                                          */
/************************************************************/
void trs1byte ( unsigned char tdt )
{
  while(TDRE_0 == 0);                                       /* End Serial Transmitting                      */
  TDR_0 = tdt;
  TDRE_0 = 0;
  while(TEND_0 == 0);                                       /* End Serial Transmitting                      */
}

/************************************************************/
/* Transmit N byte                                          */
/************************************************************/
void trsnbyte ( unsigned char *tdt, unsigned char dtno )
{
  while(dtno--){
    while(TDRE_0 == 0);                                     /* End Serial Transmitting                      */
    TDR_0 = *tdt;
    TDRE_0 = 0;

    *tdt++;
  }

  while(TEND_0 == 0);                                       /* End Serial Transmitting                      */
}
```

## ホームページとサポート窓口

ルネサステクノロジホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry
csc@renesas.com

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 1.00 | 2005.02.18 | — | 初版発行 |
| 2.00 | 2006.08.29 | 3, 23 | 新規追加および内容修正 |
| | | | |
| | | | |
| | | | |

## 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。