

V850E2/ML4

R01AN1343JJ0100

Rev.1.00

2013.03.01

CAN コントローラを使用した フラッシュ・セルフ・プログラミングによるプログラムアップデート例

要旨

本アプリケーションノートでは、V850E2/ML4の CAN 通信で転送したデータをフラッシュ・セルフ・プログラミングで書き換えるプログラムアップデート例について説明します。

本アプリケーションノートで実現するプログラムアップデート例の特長を以下に示します。

- CAN 通信により受信したインテル拡張ヘキサ・フォーマット形式のアップデートプログラムファイルを使用して、フラッシュ・メモリ領域のプログラムを書き換えます。
- 意図せず書き換え処理が中断するなど正常に書き換えできなかった場合の対策として、チェックサムによるエラー制御を実装しています。

対象デバイス

V850E2/ML4

本アプリケーションノートを他の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. 仕様	4
2. 動作確認条件	5
3. 関連アプリケーションノート	6
4. 周辺機能説明	6
4.1 フラッシュ・セルフ・プログラミングの用語	6
4.2 フラッシュ・セルフ・プログラミングの留意点	7
4.2.1 リンク・ディレクティブ・ファイルの設定方法	8
4.2.2 プロローグ／エピローグ・ライブラリ不使用の設定方法	10
4.2.3 RAM上に配置するセクションのROM化設定方法	11
4.2.4 far jump機能の設定方法	12
4.2.5 スタートアップ・ルーチンの設定方法	14
4.2.6 FSL使用中に発生する割り込み処理の注意事項	16
5. ハードウェア説明	17
5.1 使用端子一覧	17
6. ソフトウェア説明	18
6.1 動作概要	18
6.1.1 セクション配置設定	18
6.1.2 フラッシュ・メモリ書き換え動作概要	19
6.1.3 起動から通常動作まで	20
6.1.4 INTP1 割り込み入力後のフラッシュ書き換え処理	20
6.1.5 データ受信処理	20
6.1.6 データ受信／書き換え後の処理	21
6.1.7 通信制御シーケンス	22
6.2 ファイル構成	23
6.3 定数一覧	24
6.4 変数一覧	26
6.5 関数一覧	27
6.6 関数仕様	28
6.7 フローチャート	38
6.7.1 スタートアップ・ルーチンの処理	38
6.7.2 メイン処理	39
6.7.3 例外ハンドラ・アドレス切り替え処理	40
6.7.4 書き換え領域のチェックサム判定処理	41
6.7.5 INTP1 割り込み初期化処理	42
6.7.6 INTP1 割り込み処理	43
6.7.7 フラッシュ書き換え処理	44
6.7.8 フラッシュ環境の初期化処理	46
6.7.9 フラッシュ環境の開始処理	47
6.7.10 FSLによるFLMD0 端子のチェック処理	48
6.7.11 指定ブロックの消去処理	49
6.7.12 指定アドレスからの書き込み処理	50
6.7.13 指定ブロックの内部ベリファイ処理	51
6.7.14 フラッシュ環境終了処理	52
6.7.15 FLMD0 端子レベル設定処理	53
6.7.16 受信データ格納処理	54
6.7.17 テキストバイナリ変換処理	56
6.7.18 定周期LED点滅用TAUA0 初期化処理（書き込み領域、予備領域サンプル関数）	57
6.7.19 TAUA0 インターバル・タイマ割り込み処理	58

6.7.20	CANコントローラ チャンネル0 (FCN0) 初期化処理.....	59
6.7.21	FCN0 ポート初期化処理.....	60
6.7.22	FCN0 メッセージ・バッファ初期化処理.....	61
6.7.23	FCN0 メッセージ送信処理.....	63
6.7.24	FCN0 送信処理.....	64
6.7.25	FCN0 受信処理.....	65
6.7.26	FCN0 受信完了割り込み処理.....	66
6.7.27	FCN0 エラー割り込み処理.....	67
7.	操作概要.....	68
8.	サンプルコード.....	70
9.	参考ドキュメント.....	70

1. 仕様

本アプリケーションノートでは、フラッシュ・セルフ・プログラミングを使用して、内蔵のフラッシュ・メモリ書き換えによるプログラムアップデートを行います。

任意の機器とのCAN通信により、インテル拡張ヘキサ・フォーマット形式のアップデート用プログラムファイルデータを受信して、内蔵フラッシュ・メモリ領域のプログラムを書き換えます。

表 1.1に使用する周辺機能と用途を、図 1.1にシステム構成図を示します。

表1.1 使用する周辺機能と用途

周辺機能	用途
フラッシュ・メモリ（内蔵フラッシュ）	プログラム格納領域
フラッシュ・マクロ・サービス	フラッシュ・メモリ書き換え
CANコントローラ（FCN）	書き換えデータ／メッセージ通信

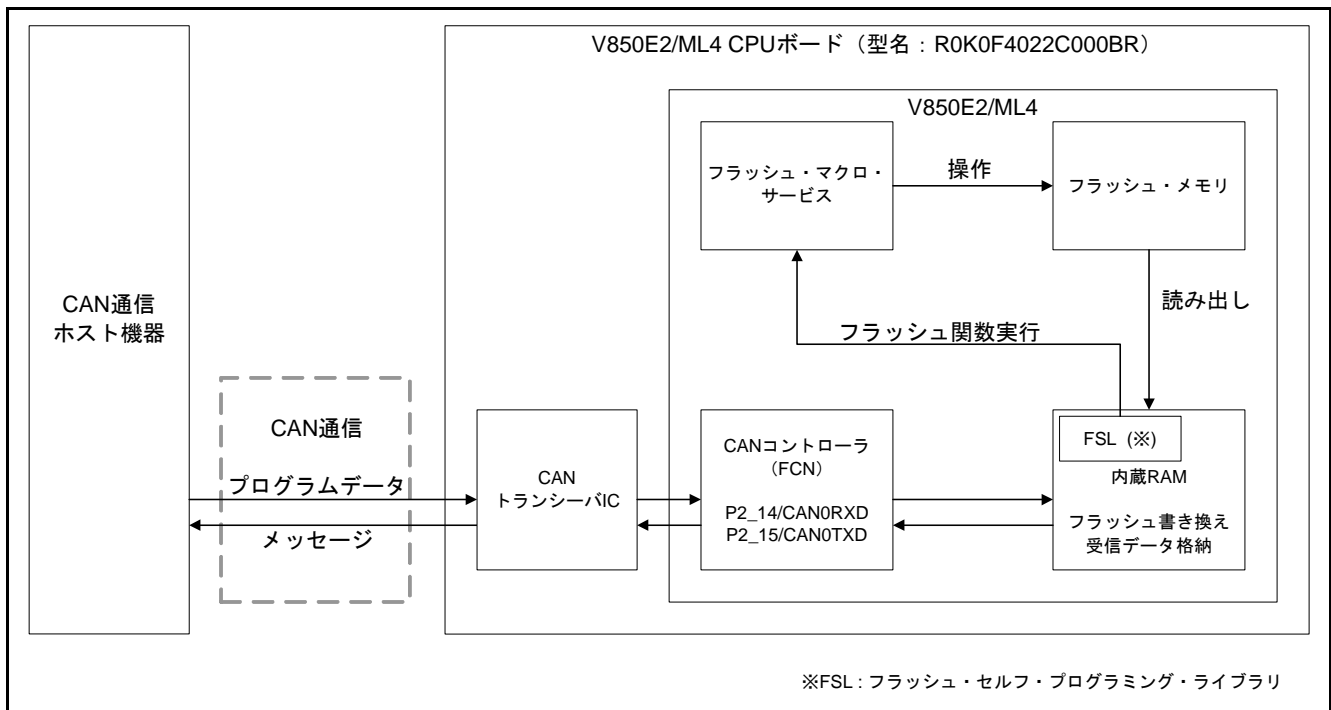


図1.1 システム構成図

2. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表2.1 動作確認条件

項目	内容
使用マイコン	V850E2/ML4
動作周波数	内部システム・クロック (f _{CLK}) : 200MHz Pバス・クロック (f _{PCLK}) : 66.667MHz
動作電圧	外部端子用正電源 (EV _{DD}) : 3.3V 内部ユニット用正電源 (IV _{DD}) : 1.2V
統合開発環境	ルネサス エレクトロニクス製 CubeSuite+ Ver.1.02.01
C コンパイラ	ルネサス エレクトロニクス製 CX コンパイラパッケージ Ver.1.21 コンパイル・オプション -Cf4022 -oDefaultBuild¥v850e2ml4_flash_update_can.lmf -Xobj_path=DefaultBuild -g -Xpro_epi_runtime=off -IC:¥Workspace¥v850e2ml4_flash_update_can¥inc -IC:¥Workspace¥v850e2ml4_flash_update_can¥FSL -Xdef_var -Xfar_jump=v850e2ml4_flash_update_can.fjp -Xlink_directive=v850e2ml4_flash_update_can.dir -Xstartup=DefaultBuild¥cstart.obj +Xide -Xmap=DefaultBuild¥v850e2ml4_flash_update_can.map -Xsymbol_dump -IFSL_T05_REC_R32 -LC:¥Workspace¥v850e2ml4_flash_update_can¥FSL¥lib -Xrompsec_text=FSL_CODE.text -Xrompsec_text=FSL_CODE_ROMRAM.text -Xrompsec_text=FSL_CODE_RAM.text -Xrompsec_text=FSL_CODE_RAM_USRINT.text -Xrompsec_text=FSL_CODE_RAM_USR.text -Xrompsec_text=FSL_CODE_RAM_EX_PROT.text -Xrompsec_text=INTP1RAM.text -Xrompsec_text=INTTAUA0I0RAM.text -Xrompsec_text=INTFCN0IERRRAM.text -Xrompsec_text=INTFCN0IRECRAM.text -Xhex=DefaultBuild¥v850e2ml4_flash_update_can.hex
動作モード	通常動作モード (書き換え時にフラッシュ・メモリ・プログラミング・モードに変更)
サンプルコードのバージョン	1.00
使用ボード	R0K0F4022C000BR
その他	CAN 通信ホスト機器

3. 関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せて参照してください。

- V850 マイクロコントローラ フラッシュ・セルフ・プログラミング・ライブラリ Type05 (R01AN0661JJ)
- V850 マイクロコントローラ フラッシュ・メモリ・セルフ・プログラミング・ライブラリ Type05 Ver.1.03 使用上の留意点 (R20AN0112JJ)

4. 周辺機能説明

フラッシュ・メモリを V850E2/ML4 上で動作するソフトウェアで書き換えるために必要となるフラッシュ・セルフ・プログラミング・ライブラリについて補足します。基本的な内容は「V850E2/ML4 ユーザーズマニュアル ハードウェア編」と「V850 マイクロコントローラ フラッシュ・セルフ・プログラミング・ライブラリ Type05」に記載しています。

4.1 フラッシュ・セルフ・プログラミングの用語

以下に、本アプリケーションノートで使用しているフラッシュ・セルフ・プログラミングの用語について説明します。

- フラッシュ・マクロ・サービス
デバイスが内蔵している、フラッシュ・メモリを操作する機能です。
- フラッシュ環境
フラッシュ・マクロ・サービスを使用し、コード・フラッシュの操作が可能である状態です。通常のプログラムの実行とは異なる制限事項があります。フラッシュ環境を終了しないと他の環境へは遷移できません。
- フラッシュ関数
セルフ・ライブラリを構成する個々の関数です。C 言語で利用できます。
- 内部ベリファイ
フラッシュ・メモリへの書き込み後、内部での信号レベルのチェックを行い、書き込み/消去状態を確認するコマンドです。

4.2 フラッシュ・セルフ・プログラミングの留意点

V850E2/ML4 は、フラッシュ・メモリを操作する機能であるフラッシュ・マクロ・サービスを内蔵しており、本サンプルコードでは、フラッシュ・マクロ・サービスを C 言語から利用できるフラッシュ・セルフ・プログラミング・ライブラリ (FSL) を使用してプログラムの書き換えを行います。以下に、FSL を使用する際の留意点を示します。

- フラッシュ環境中に実行されるプログラム (ランタイム・ライブラリ含む) の RAM 上配置。
 - 該当プログラムを RAM 上に配置するためのセクション設定：
セクション設定にはリンク・ディレクティブ・ファイルの作成と設定が必要です。詳細は「4.2.1 リンク・ディレクティブ・ファイルの設定方法」を参照してください。
 - 関数のプロローグ/エピローグ・ランタイム・ライブラリの不使用設定または RAM 上配置設定：
本サンプルコードでは、プロローグ/エピローグ・ランタイム・ライブラリの不使用設定を実施しています。詳細は「4.2.2 プロローグ/エピローグ・ライブラリ不使用の設定方法」を参照してください。
 - 割り込みを使用する場合の、例外ハンドラ・アドレス切り替え機能の設定：
例外ハンドラ・アドレス切り替え機能の設定はソフトウェアで行います。詳細は「6.7.3 例外ハンドラ・アドレス切り替え処理」を参照してください。
 - RAM 配置先プログラム領域の初期化：
V850E2/ML4 で RAM 上にプログラムを配置する際、配置先プログラム領域を含む 16 バイト境界領域 (H'xxxx_xxx0~H'xxxx_xxxF) を初期化 (0 クリア) する必要があります。本サンプルコードでは、スタートアップ・ルーチンの中でこの初期化を行っています。スタートアップ・ルーチンの変更については「4.2.5 スタートアップ・ルーチンの設定方法」を、処理内容については「6.7.1 スタートアップ・ルーチンの処理」を参照してください。
 - プログラムを RAM 上に展開するためのセクション ROM 化設定：
CubeSuite+でのROM化設定については、「4.2.3 RAM上に配置するセクションのROM化設定方法」を参照してください。
- 割り込みハンドラ内におけるフラッシュ関数の実行禁止。
- 2M バイト以上離れたアドレスに配置された関数をコールする際の、CX コンパイラへの far jump オプション指定。
本サンプルコードでは、フラッシュ・メモリ上からコールするRAM上配置関数に対しfar jumpオプションを指定しています。詳細は「4.2.4 far jump機能の設定方法」を参照してください。
- 割り込みハンドラ内で C 言語のグローバル変数へのアクセスを行う場合の gp レジスタ、ep レジスタの回避、設定および復帰。
割り込みハンドラ内でデータセクションにアクセスする際に上記の操作が必要になる場合があります。詳細は「4.2.6 FSL使用中に発生する割り込み処理の注意事項」を参照してください。

FSL の関数仕様とシステム構築については、関連アプリケーションノート「V850 マイクロコントローラ フラッシュ・セルフ・プログラミング・ライブラリ Type05」を参照してください。

FSL のビルド方法とその留意点については、関連アプリケーションノート「V850 マイクロコントローラ フラッシュ・メモリ・セルフ・プログラミング・ライブラリ Type05 使用上の留意点」を参照してください。

CubeSuite+上での CX コンパイラへのセクション指定と配置アドレス設定、ROM 化設定、far Jump オプション指定については「CubeSuite+ V1.03.00 統合開発環境ユーザーズマニュアル ビルド編 (CX コンパイラ)」を参照してください。

例外ハンドラ・アドレス切り替えの実行については「V850E2M ユーザーズマニュアル アーキテクチャ編」を参照してください。

4.2.1 リンク・ディレクティブ・ファイルの設定方法

セクション配置を変更するためには、リンク・ディレクティブ・ファイルの作成と CubeSuite+ への設定が必要です。CubeSuite+ のメニューから生成せずに、テキストエディタでリンク・ディレクティブ・ファイルを作成した場合、CubeSuite+ の設定が必要です。エクスプローラなどからリンク・ディレクトリ・ファイルをドラッグして、プロジェクト・ツリーの下部の空白部分にドロップしてください。CubeSuite+ では、拡張子が「dir」または「dr」のファイルはリンク・ディレクティブ・ファイルとみなされます。プロジェクト・ツリーの「CX (ビルド・ツール)」を選択した上で、プロパティの「リンク・オプション」タブをクリックし、「入力ファイル」を開くと「使用するリンク・ディレクティブ・ファイル」が確認できます。詳細は、CubeSuite+ に付属の「CubeSuite+ V1.03.00 統合開発環境ユーザーズマニュアル コーディング編 (CX コンパイル)」をご参照ください。

リンク・ディレクティブ・ファイルを作成する際、本サンプルコードでは、デフォルトの領域以外にフラッシュ・メモリ上に書き換え領域セクション (MasterPRG.text) と予備領域セクション (SparePRG.text) および、FSL 領域 (FSL.CONST) を作成してください。また、RAM 上に FSL 使用領域とユーザ・プログラム領域のセクション (FSL_DATA.bss, FSL_CODE.text, FSL_CODE_ROMRAM.text, FSL_CODE_RAM.text, FSL_CODE_RAM_USRINT.text, FSL_CODE_RAM_USR.text, FSL_CODE_RAM_EX_PROT.text) および、例外ハンドラ・アドレスのセクション (INTP1RAM.text, INTTAUA0I0RAM.text, INTFCN0IERRAM.text, INTFCN0IRECRAM.text) を作成してください。

本サンプルコードでは、MasterPRG.text セクションの開始アドレスは H'0000 8000 を想定しています。また、例外ハンドラ・アドレスのセクションの開始アドレスについては、転送先のベース・アドレス H'FEDF E000 にそれぞれの割り込みハンドラ・アドレスを加算したアドレスを想定しています。

図 4.1 にリンク・ディレクティブ・ファイルの登録箇所を示します。

図 4.2 にリンク・ディレクティブ・ファイルの作成・セクション設定例を示します。

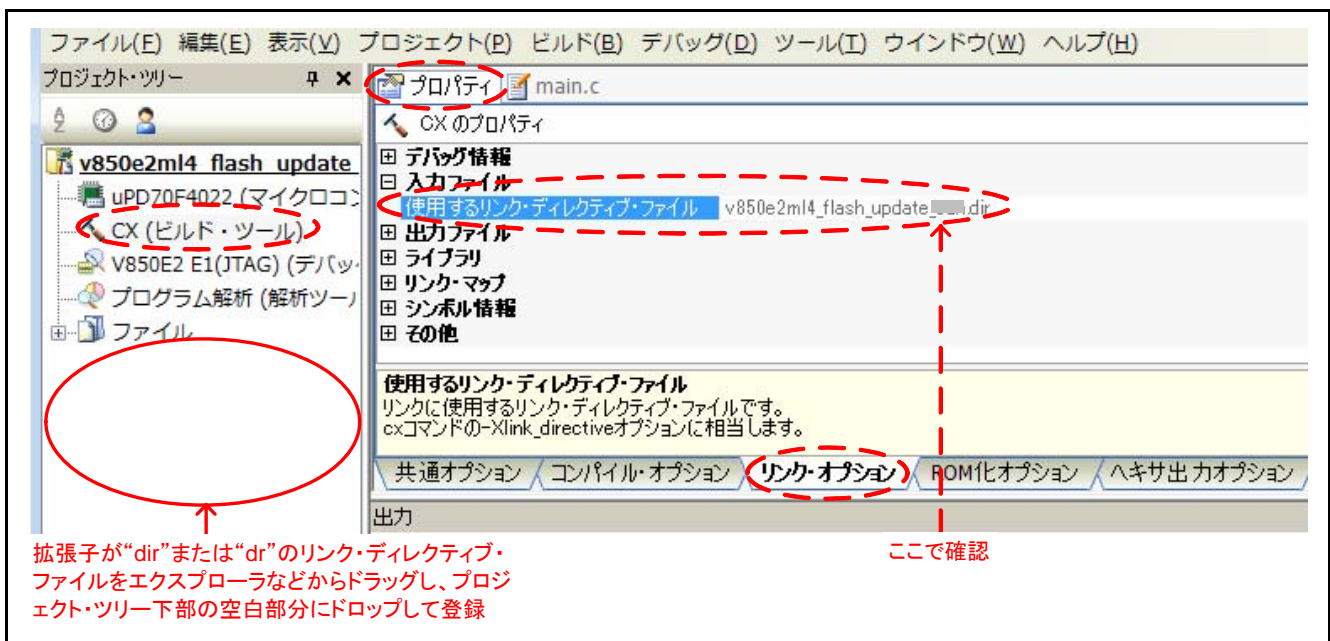


図 4.1 リンク・ディレクティブ・ファイルの登録箇所


```

SCONST: !LOAD ?R {
  .sconst = $PROGBITS ?A .sconst ;
};

CONST: !LOAD ?R V0x00001100 {
  .const = $PROGBITS ?A .const ;
  FSL_CONST.const = $PROGBITS ?A FSL_CONST.const ; # FSL使用領域
};

TEXT: !LOAD ?RX {
  .pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime ;
  .text = $PROGBITS ?AX .text ;
};

# 予備領域
SparePRG: !LOAD ?RX V0x00006000 {
  SparePRG.text = $PROGBITS ?AX V0x00006000 SparePRG.text ;
};

# 書き換え領域
MasterPRG: !LOAD ?RX V0x00008000 {
  MasterPRG.text = $PROGBITS ?AX V0x00008000 MasterPRG.text ;
};

DATA: !LOAD ?RW V0xfedf0000 {
  .data = $PROGBITS ?AW .data ;
  .sdata = $PROGBITS ?AWG .sdata ;
  .sbss = $NOBITS ?AWG .sbss ;
  FSL_DATA.bss = $NOBITS ?AW FSL_DATA.bss ; # FSL使用領域
  .bss = $NOBITS ?AW .bss ;
};

SEDATA: !LOAD ?RW {
  .sedata = $PROGBITS ?AW .sedata ;
  .sebss = $NOBITS ?AW .sebss ;
};

SIDATA: !LOAD ?RW {
  .tidata.byte = $PROGBITS ?AW .tidata.byte ;
  .tibss.byte = $NOBITS ?AW .tibss.byte ;
  .tidata.word = $PROGBITS ?AW .tidata.word ;
  .tibss.word = $NOBITS ?AW .tibss.word ;
  .tidata = $PROGBITS ?AW .tidata ;
  .tibss = $NOBITS ?AW .tibss ;
  .sidata = $PROGBITS ?AW .sidata ;
  .sibss = $NOBITS ?AW .sibss ;
};

# RAM上配置プログラム領域
RAM_PROG: !LOAD ?RX V0xfedfc000 {
  FSL_CODE.text = $PROGBITS ?AX FSL_CODE.text ;
  FSL_CODE_ROMRAM.text = $PROGBITS ?AX FSL_CODE_ROMRAM.text ;
  FSL_CODE_RAM.text = $PROGBITS ?AX FSL_CODE_RAM.text ;
  FSL_CODE_RAM_USRINT.text = $PROGBITS ?AX FSL_CODE_RAM_USRINT.text ;
  FSL_CODE_RAM_USR.text = $PROGBITS ?AX FSL_CODE_RAM_USR.text ;
  FSL_CODE_RAM_EX_PROT.text = $PROGBITS ?AX FSL_CODE_RAM_EX_PROT.text ;
};

# RAM上配置例外ハンドラ領域
INTRAM: !LOAD ?RX V0xfedfe000 L0x00001080 {
  INTPIRAM.text = $PROGBITS ?AX V0xfedfe170 H0x0000000a INTPIRAM.text ;
  INTTAUA0IORAM.text = $PROGBITS ?AX V0xfedfe3b0 H0x0000000a INTTAUA0IORAM.text ;
  INTFCN0IERRRAM.text = $PROGBITS ?AX V0xfedfeb30 H0x0000000a INTFCN0IERRRAM.text ;
  INTFCN0IRECRAM.text = $PROGBITS ?AX V0xfedfeb40 H0x0000000a INTFCN0IRECRAM.text ;
};

__tp_TEXT@ %TP_SYMBOL ;
__gp_DATA@ %GP_SYMBOL & __tp_TEXT { DATA } ;
__ep_DATA@ %EP_SYMBOL ;

```

ROM上にFSL領域用のセクションを作成

ROM上に予備領域用のセグメント、セクションを作成

ROM上に書き換え領域用のセグメント、セクションを作成

RAM上にFSL使用領域用のセクションを作成

RAM上にFSL領域と、ユーザプログラム領域用のセグメント、セクションを作成

RAM上に例外ハンドラ用のセグメント、セクションを作成

図4.2 リンク・ディレクティブ・ファイルの作成・セクション設定例

4.2.2 プロローグ/エピローグ・ライブラリ不使用の設定方法

プロローグ/エピローグ・ライブラリ不使用の設定は、CubeSuite+で行います。プロジェクト・ツリーの「CX (ビルド・ツール)」を選択した上で、プロパティの「コンパイル・オプション」のタブをクリックし、「最適化 (詳細)」の「プロローグ/エピローグ・ライブラリを使用する」を「いいえ (-Xpro_epi_runtime=off)」に設定します。

図 4.3 に プロローグ/エピローグ・ライブラリ不使用の設定箇所を示します。

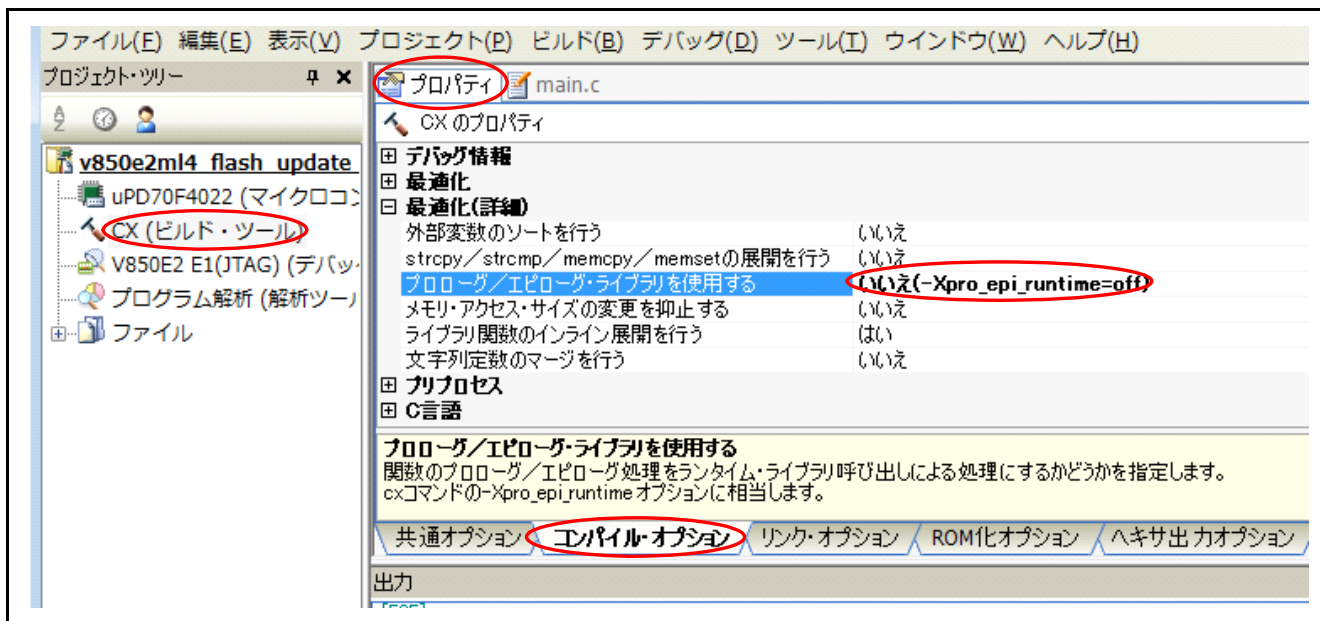


図4.3 プロローグ/エピローグ・ライブラリ不使用の設定箇所

4.2.3 RAM 上に配置するセクションの ROM 化設定方法

セクションを RAM 上に展開するための ROM 化には、CubeSuite+ の設定が必要です。プロジェクト・ツリーの「CX (ビルド・ツール)」を選択した上で、「プロパティ」の「ROM 化オプション」のタブをクリックし、「rompsec セクションに含めるテキスト・セクション」にて、RAM 上に配置するセクションの中で ROM 化が必要なセクションを指定します。右端の「...」ボタンをクリックすることで表示される「テキスト編集」ウィンドウに、対象のセクション名を (1 行に 1 セクション名で) 記述します。

図 4.4 に RAM 上に配置するセクションの ROM 化設定登録箇所を示します。

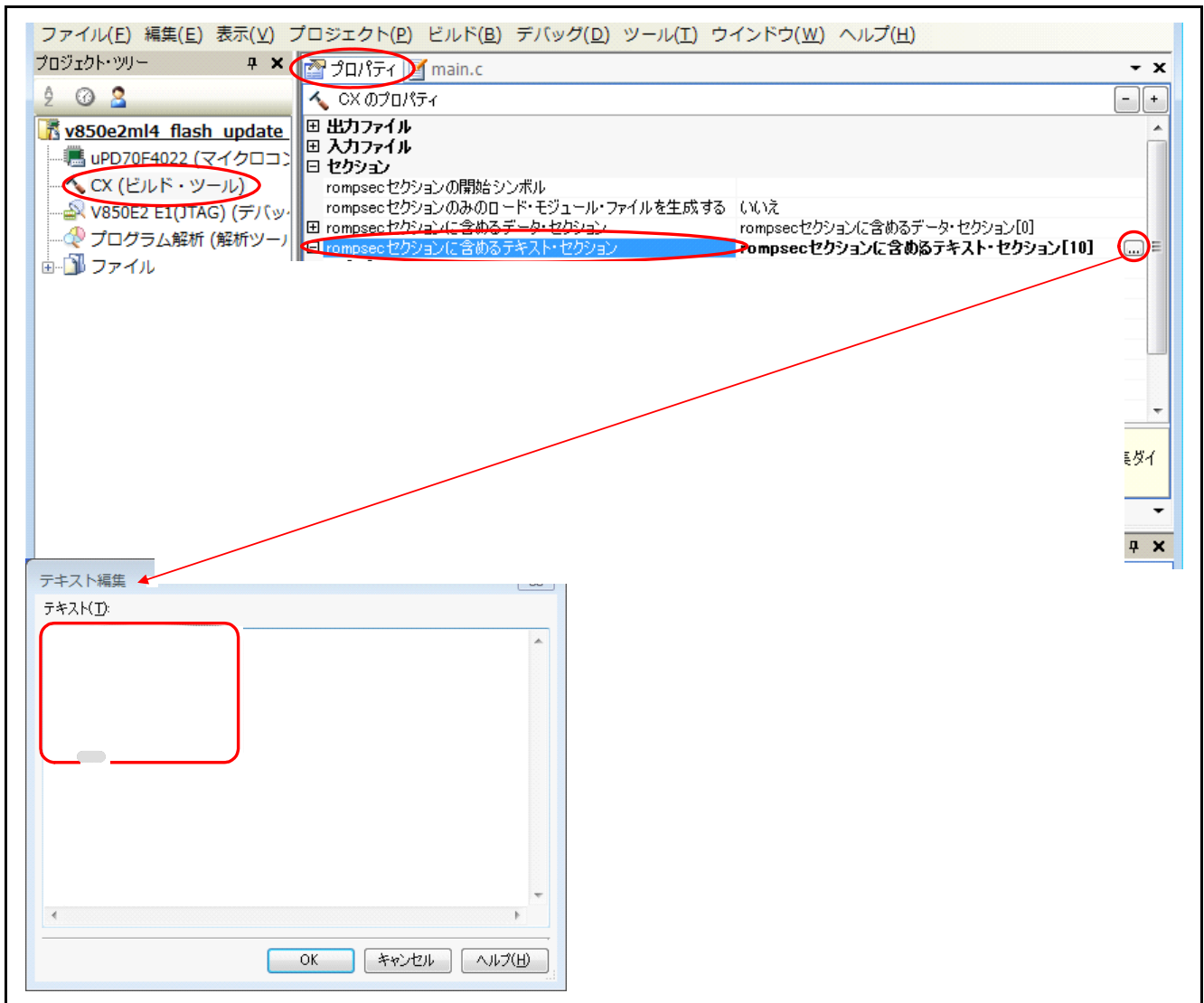


図4.4 RAM 上に配置するセクションの ROM 化設定登録箇所

4.2.4 far jump 機能の設定方法

V850E2/ML4 では、フラッシュ・メモリの終端アドレスと内蔵 RAM の先頭アドレスとの間が 2M バイト以上離れています。CX コンパイラでは、関数コール時に±2M バイト以上離れた領域に分岐する場合、コール先の関数に対し **far jump** オプションを指定する必要があります。本サンプルコードでは、内蔵 RAM に配置する関数のうちフラッシュ・メモリ上の関数からコールされるもの、および使用する全ての割り込みハンドラに対し、**far jump** オプションを指定しています。

far jump オプションを指定するためには、指定する関数を列挙したファイル(**far jump** 呼び出し関数一覧ファイル)を作成し、コンパイル・オプション「**-Xfar_jump**」にて当該ファイル名を指定します。CubeSuite+上で設定するには、プロジェクト・ツリーの「**CX (ビルド・ツール)**」を選択した上で、プロパティの「コンパイル・オプション」タブをクリックし、「出力コード」の「**far jump** ファイル名」の項目右端にある「...」ボタンをクリックして、作成した **far jump** 呼び出し関数一覧ファイルのパスを記述してください。(なお、**far jump** 呼び出し関数一覧ファイル名の拡張子は任意ですが、「.fjp」にすることを推奨します。)

far jump 呼び出し関数一覧ファイルでは、1 行に 1 関数名を記述し、このときの関数名は C 言語の関数名の先頭に「_ (アンダスコア)」を付記します。また、「**{all_interrupt}**」と記述すると、全ての割り込みハンドラ関数が対象になります。**far jump** 呼び出し関数一覧ファイルの作成方法の詳細は「CubeSuite+ V1.03.00 統合開発環境ユーザズマニュアル コーディング編 (CX コンパイラ)」の「3.3.3 **far jump** 機能」を参照してください。

図 4.5に **far jump**呼び出し関数一覧ファイルの登録箇所を示します。

図 4.6に **far jump**呼び出し関数一覧ファイル作成例を示します。

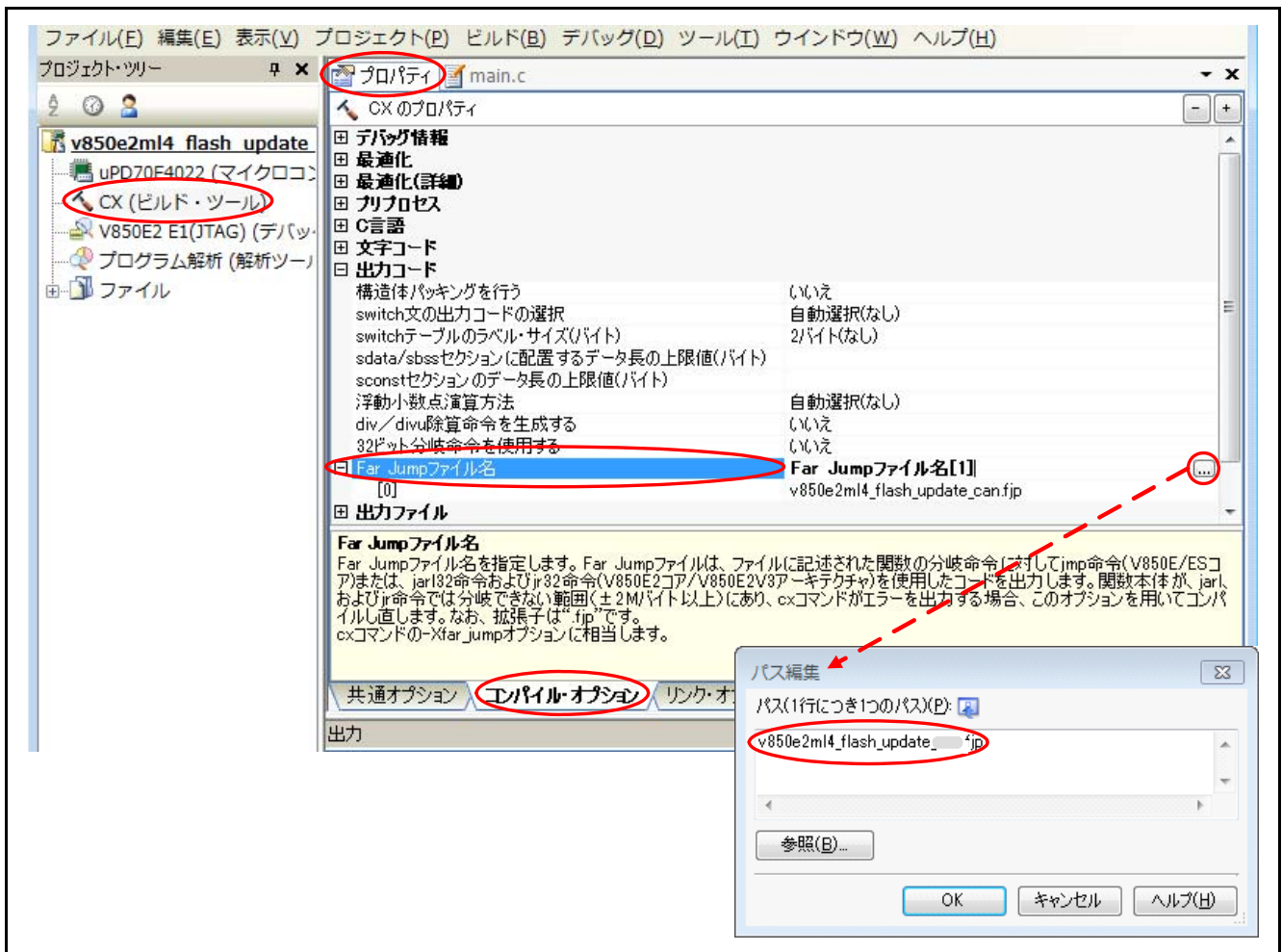


図4.5 far jump 呼び出し関数一覧ファイルの登録箇所

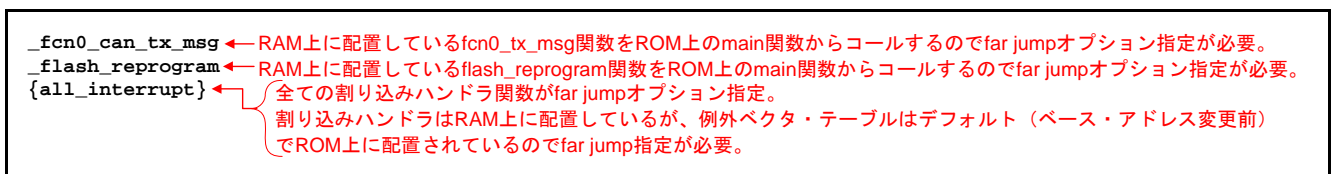


図4.6 far jump 呼び出し関数一覧ファイル作成例

4.2.5 スタートアップ・ルーチンの設定方法

本サンプルコードで使用するスタックは、標準のスタートアップ・ルーチンで設定されているスタック・サイズ（512 バイト）よりも大きな領域を必要とします。また、標準のスタートアップ・ルーチンでは、初期値ありデータおよび RAM 配置プログラムを展開するための関数「_rcopy」（ROM 化処理）を実行していますが、プログラム領域に対し ROM 化処理を行う場合は、「_rcopy」実行前に、プログラム配置先の 16 バイト境界領域に対し初期化（0 クリア）する必要があります。本サンプルコードでは、標準のスタートアップ・ルーチンが記述されているアセンブラ・ソース・ファイル「cstart.asm」に対し、スタック・サイズ変更、およびプログラム配置先の 16 バイト境界領域に対する初期化処理を追加しています。

標準のスタートアップ・ルーチンを切り替える場合、スタートアップ・ルーチンを記述したユーザ作成のアセンブラ・ソース・ファイルを準備して、CubeSuite+ のプロジェクトに登録する必要があります。CubeSuite+ のプロジェクト・ツリーの「ファイル」の中にある項目「スタートアップ」を右クリックすると、スタートアップ・ルーチンのソース・ファイルを追加するためのメニューが表示されます。

図 4.7 のスタートアップ・ルーチンの登録箇所を示します。

図 4.8 のスタートアップ・ルーチンの作成例（cstart.asm の一部）を示します。

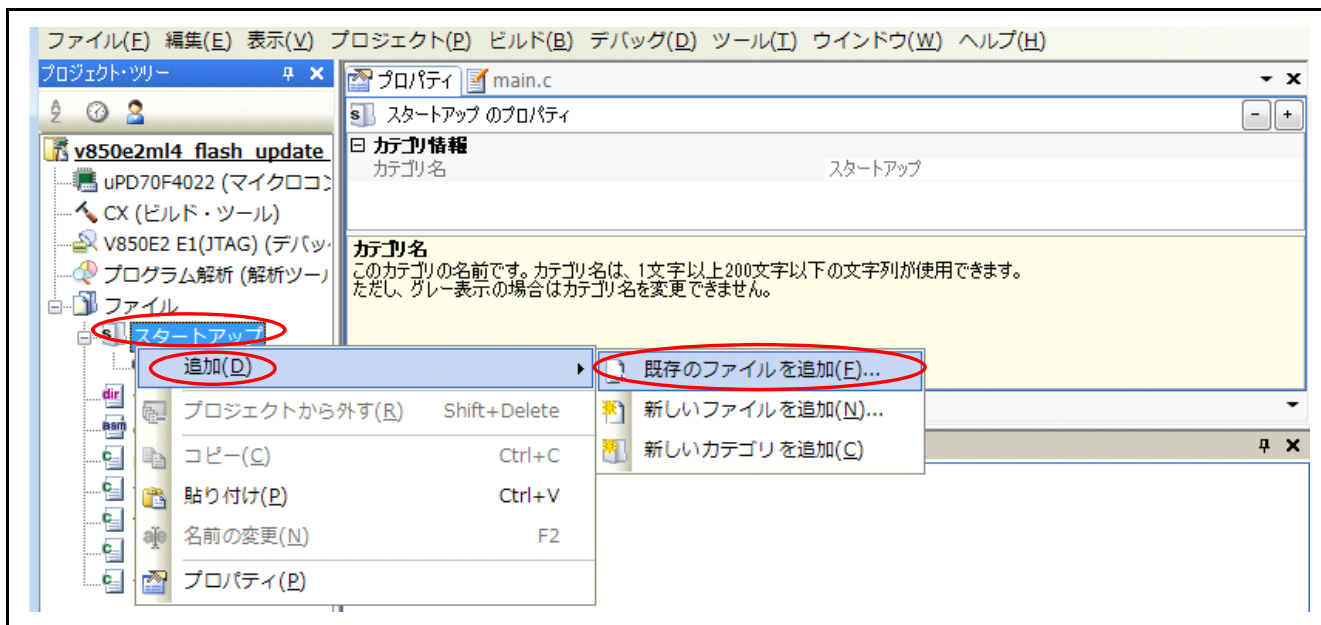


図4.7 スタートアップ・ルーチンの登録箇所

```

:
: (cstart.asm の途中から抜粋)
:
#-----
#      system stack
#-----
STACKSIZE      .set      0x500
                .dseg    bss
                .align   4
__stack:       .ds      (STACKSIZE)

#-----
#      RESET vector
#-----

RESET .cseg text
      jr      __start

      .cseg text
      .align 4
__start:
      mov32  #__tp_TEXT, tp      ; set tp register
      mov32  #__gp_DATA, gp      ; set gp register offset
      add    tp, gp              ; set gp register
      mov32  #__stack+STACKSIZE, sp ; set sp register
      mov32  #__ep_DATA, ep      ; set ep register

      mov32  #__PROLOG_TABLE, r12 ; for prologue/epilogue runtime
      ldsr   r12, 20             ; set CTBP (CALLT base pointer)

      jarl   __hdwinit, lp       ; initialize hardware

      mov32  #__ssbss, r6        ; clear sbss section
      mov32  #__esbss, r7
      jarl   __zeroclr, lp

      mov32  #__sbss, r6         ; clear bss section
      mov32  #__ebss, r7
      jarl   __zeroclr, lp

      mov32  0xfedfc000, r6      ; clear ram_prog section for e2core prefetch processing
      mov32  0xfedfffff, r7
      jarl   __zeroclr, lp

:
: (続く)
:

```

STACKSIZE .set 0x500 ← スタックサイズをFSLとユーザプログラムの実行に必要なサイズに変更

_rcopyを実行するまえに
RAM上のプログラムとして使用する
領域周辺を0クリア

図4.8 スタートアップ・ルーチンの作成例 (cstart.asm の一部)

4.2.6 FSL 使用中に発生する割り込み処理の注意事項

FSL 使用中に発生する割り込み処理内で gp レジスタ、ep レジスタを用いたデータ・アクセスを行う場合、データ・アクセスを行う前に、gp レジスタ、ep レジスタに適切な値を設定してください。このとき、gp レジスタ、ep レジスタに適切な値を設定する前に、gp レジスタ、ep レジスタの退避処理が必要です。また、割り込み処理からの復帰を行う前に gp レジスタ、ep レジスタの復帰処理が必要となります。上記の措置を行わない場合、gp レジスタ、ep レジスタを用いたデータ・アクセスは正常に動作できません。

- gp レジスタをベース・アドレスとしてアクセスを行うセクション：
(セクション指定せずに作成したグローバル変数は、.sdata, .sbss に配置されます。)
 - .data
 - .bss
 - .sdata
 - .sbss

- ep レジスタをベース・アドレスとしてアクセスを行うセクション：
 - .sdata
 - .sebss
 - .sidata
 - .sibss
 - .tidata.byte
 - .tibss.byte
 - .tidata.word
 - .tibss.word

本サンプルコードでは、ep レジスタをベース・アドレスとしてアクセスするセクションを使用していないため、ep レジスタの退避／設定／復帰の処理を割り込み処理で行っておりません。また、V850E2/ML4 では、FSL を使用する際の gp レジスタの退避／設定／復帰は不要です。

使用マイコンを変更する場合や上記セクションを使用する場合は、割り込み処理内で gp レジスタや ep レジスタの退避／設定／復帰が必要になる場合がありますので、応用する際は注意してください。

5. ハードウェア説明

5.1 使用端子一覧

表 5.1に 使用端子と機能を示します。

表5.1 使用端子と機能

端子名	入出力	内容
P2_14/CAN0RXD	入力	CAN0 受信データ入力
P2_15/CAN0TXD	出力	CAN0 送信データ出力
P2_3/INTP1	入力	INTP1 割り込み

6. ソフトウェア説明

6.1 動作概要

本サンプルコードでは、CAN 通信でインテル拡張ヘキサ・フォーマット形式のアップデート用プログラムファイルデータを受信し、フラッシュ・メモリ領域のプログラムを書き換えます。ここでは、その動作概要について説明します。

6.1.1 セクション配置設定

フラッシュ・メモリ書き換え中はフラッシュ・メモリへのアクセスが禁止されているため、フラッシュ・メモリ書き換え中に使用するプログラムは全てフラッシュ・メモリ以外の領域に転送する必要があります。本サンプルコードでは、フラッシュ・メモリ書き換え中に使用するプログラムは全て内蔵 RAM に転送するように、セクション配置を設定しています。

表 6.1 にフラッシュ・メモリ書き換え中に使用するセクションを示します。

表 6.1 フラッシュ・メモリ書き換え中に使用するセクション

セクション名	プログラム内容	関数名
FSL_CODE_ROMRAM.text、 FSL_CODE_RAM.text、 FSL_CODE_RAM_EX_PROT.text	FSL の領域	フラッシュ関数
FSL_CODE_RAM_USRINT.text	RAM 実行用ユーザ・プログラム・ 割り込み・セクション	fcn0_can_rx_isr、 fcn0_can_error_isr、 flash_store_can_data、hex2bin、 intp1_isr、 taua0_ch0_interval_timer_isr
FSL_CODE_RAM_USR.text	RAM 実行用ユーザ・プログラム・ セクション	fcn0_can_tx_msg、fcn0_can_tx、 fcn0_can_rx、flash_reprogram、 flash_init、flash_activate、 flash_modecheck、flash_erase、 flash_write、flash_iverify、 flash_end、flash_set_flmd0
INTP1RAM.text INTTAUA0I0RAM.text INTFCN0ITRXRAM.text INTFCN0IRECRAM.text	割り込みハンドラ関数へのジャン プ命令	なし

本サンプルコードでは、意図せずフラッシュ・メモリの書き換え処理が中断するなど正常に書き換え（アップデート）できなかった場合の対策として、予備プログラム格納用のセクション領域を別途割り当てています。また、データ受信前（初期）の書き換え領域と予備領域には、それぞれ同じ処理内容のプログラムを格納しています。表 6.2 にその対応を示します。

表 6.2 フラッシュ・メモリ上にアドレスを指定して配置する関数とセクション

項目	先頭番地（ブロック番号）	格納関数名	ROM セクション名
書き換え領域	H'0000 8000 (8)	taua0_led_sample	MasterPRG.text
予備領域	H'0000 6000 (6)	taua0_led_spare	SparePRG.text

6.1.2 フラッシュ・メモリ書き換え動作概要

図 6.1にフラッシュ・メモリ書き換え動作概要図を示します。

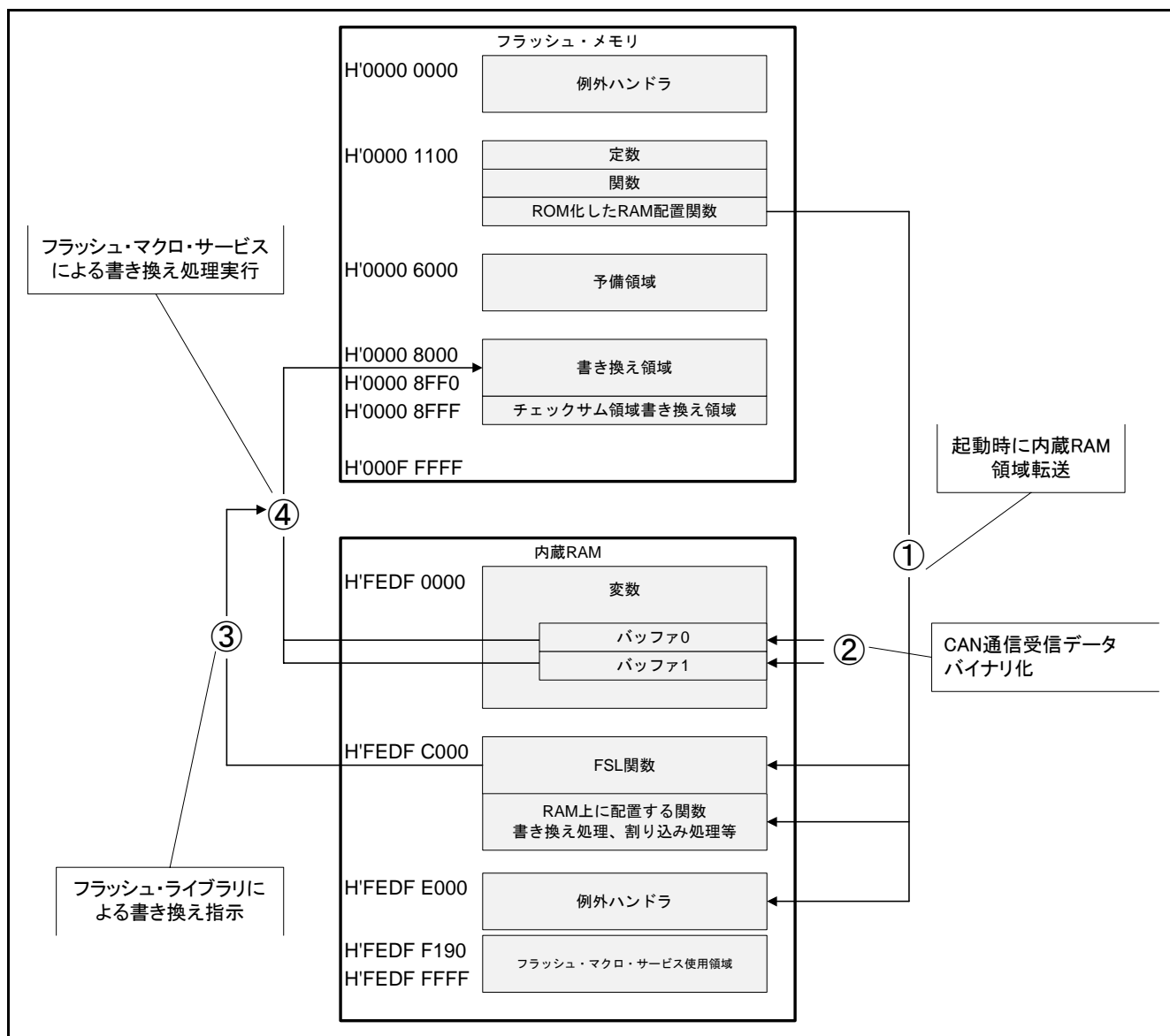


図6.1 フラッシュ・メモリ書き換え動作概要図

- ① リセット解除後、main 関数開始前に cstart.asm の処理の内部で __S_romp (ROM 化されたセクション群) は内蔵 RAM にコピーされます。
- ② CAN 通信で受信したインテル拡張ヘキサ・フォーマットのデータは、書き込みを行うバイナリデータの状態の内蔵 RAM に格納されます。
- ③ 内蔵 RAM 上に配置されたフラッシュ・ライブラリの関数からフラッシュ・マクロ・サービスの操作を行います。
- ④ フラッシュ・マクロ・サービスにより、内蔵フラッシュの書き込み処理が実行されます。

6.1.3 起動から通常動作まで

システム起動後、メイン処理にて各種初期化処理を行い、ホストに対し「Generate INTP1 interrupt for transition to flash programming event.」というメッセージを送信します。その後、チェックサム判定処理関数をコールして、書き換え領域のプログラムコードに問題がないか、チェックサムにより判定します。

本サンプルコードのチェックサムは、「プログラムコードサイズ」およびプログラムを1バイトずつ加算した「チェックサムデータ」の2つを使用します。チェックサム判定処理関数では、書き換え領域の先頭番地 (H'0000 8000) から1バイトずつ、プログラムサイズの回数分加算します。その加算結果を、データ受信時に算出したチェックサム判定データ (MasterPRG.text の最後尾 16 バイト領域に格納/詳細は 6.1.6節参照) と比較し、一致していれば書き換え領域のプログラムを、一致していなければ予備領域のプログラムを実行します。

6.1.4 INTP1 割り込み入力後のフラッシュ書き換え処理

INTP1 割り込み (立ち下がりエッジ検出/ボード上の INTP1 スイッチ押下) が発生すると、フラッシュ書き換え処理に移行します。

フラッシュ書き換え処理では、最初にホストに対して「-> INTP1 detected!」というメッセージを送信し、書き換え領域を消去します。その後、ホストに「Send subroutine code to update program in Intel expanded hex format.」というメッセージを送信し、ホストからのデータ受信待ち状態となります。

データ受信待ち状態では、フラグ変数を使用して、フラッシュ書き込みの可否をポーリングで検出します。ホストからインテル拡張ヘキサ・フォーマット形式のアップデート用プログラムファイルデータを受信すると、後述のデータ受信処理を行い、書き込みデータ格納バッファ (書き込みバッファ) にデータを格納します。書き込みバッファが満杯になって書き込み可能であることを検出すると、そのバッファデータをフラッシュ・メモリに書き込みます。

本サンプルコードでは、書き込みバッファを二重構造にしており、「データ受信処理による書き込みデータ格納」および「フラッシュ・メモリへの書き込み」において、使用する書き込みバッファを切り替えてそれぞれの処理を行います。

6.1.5 データ受信処理

データ受信待ち状態に入った後、ホストから通信データを受信するごとに CAN コントローラ チャンネル 0 (FCN0) 受信処理完了割り込みが発生します。FCN0 受信処理完了割り込みが発生すると、受信したデータを CAN 受信データ格納バッファ (受信バッファ) に順次格納します。改行コードを受信すると、それまで受信バッファに格納したデータを1行分のレコードデータと判断し、以下に説明するデータ受信処理を行って、アップデートに必要な書き込みデータを抽出します。

以下、図 6.2 に示すインテル拡張ヘキサ・フォーマット形式データ例を参考にデータ受信処理について説明します。(図 6.2 に示すデータは、機能によって色分けしています。)

```

:04000005000013C81C
:020000040000FA
:20800000E0570584CA5EEFFF605F0484E0670583CC6EEFFF606F0483407640FF2E7F054609
:20802000CF86EFFF408E40FF71870546E0970580929E1000609F0480405681FF6A070082E5
:208040002B06FAFF0000406681FF6C5F4082206EFF3F606F00C44076FFFF0E7F66608F86C8
:1A806000F00408EFFFF518766604096FFFD2BF6660019A609FC4C57F00C0
:00000001FF

```

図6.2 インテル拡張ヘキサ・フォーマット形式データ例

- 各行の処理において、受信バッファ内の 1 文字目のデータが「:」であるかを判定し、「:」ならばインテル拡張ヘキサ・フォーマット形式として 8 文字目、9 文字目 (赤) の判定を行います。1 文字目が「:」でなければ、そのレコードデータは無効となり、再び受信データ待ち状態に戻ります。また 8 文字目が「0」でないときも、そのレコードデータは無効となり、再び受信データ待ち状態に戻ります。
- 1 行目は 8 文字目と 9 文字目 (赤) が「05」になっています。この「05」はスタート・リニア・アドレス・レコードを示します。スタート・リニア・アドレス・レコードにはプログラムデータはありませんので、スタート・リニア・アドレス・レコードを受信した場合は次のレコード (行データ) が揃うまで、再び受信データ待ち状態に戻ります。
- 2 行目は 8 文字目と 9 文字目 (赤) が「04」になっています。この「04」は拡張リニア・アドレス・レコードを示します。拡張リニア・アドレス・レコードにはプログラムデータはありませんので、拡張リニア・アドレス・レコードを受信した場合は次のレコードが揃うまで、再び受信データ待ち状態に戻ります。
- 3 行目のレコードデータが揃うと、8、9 文字目 (赤) が「00」になっているので 3 行目は「データ・レコード」と判定します。このようにインテル拡張・フォーマット形式では、各レコードの先頭から 9 文字目の数値によりレコードの種類が判別できるようになっています。
- レコードの 2 文字目と 3 文字目 (青) はレコードサイズを示す 1 バイト分の 16 進数を示し、4 文字目から 8 文字目 (緑) までの 4 文字は、レコードの先頭データ格納アドレスの下位 2 バイトを示します。
- レコードの 10 文字目 (橙) 以降が、それぞれ 2 文字単位で 1 バイトを示すデータ部になっています。データ受信処理では、この 10 文字目 (橙) 以降を 2 文字単位でバイナリデータに変換 (「テキストバイナリ変換処理」関数をコール) し、変換後の 1 バイトデータを書き込みバッファに順次格納します。また、このとき、書き換え後のチェックサム判定用にその 1 バイトデータを加算 (チェックサムデータ) し、さらにそのデータ数をプログラムコードサイズとしてカウントします。これらの処理をレコードの最後の 2 文字 (黒) 手前まで繰り返したら、次のレコードが揃うまで再び受信データ待ち状態に戻ります。
- レコードデータの 8 文字目と 9 文字目 (赤) が「01」の場合は「エンド・レコード」を示します。(図 6.2 では一番下の行に相当します。) 「エンド・レコード」を判定したら、受信データの格納処理は行わずにデータ受信処理を終了します。ただし、この時点で書き込みバッファ内のデータサイズが (フラッシュ書き込み単位の) 16 バイトに満たない場合、バッファサイズが 16 バイトになるように H'FF を付加します。

本サンプルコードでは、書き込みバッファを 16 バイトサイズの二重構造としており、書き込みバッファ内の格納データが 16 バイトで満杯になると、データ受信処理の中で格納先をもう一方の書き込みバッファに切り替えます。一方の書き込みバッファが満杯になるとフラッシュ書き込み可能状態となり、フラッシュ書き換えイベント処理にてそのバッファデータをフラッシュ・メモリに書き込みます。フラッシュ・メモリへの書き込みは割り込み処理ではなく受信データ待ちのポーリング処理で行うので、満杯でバッファを切り替えるときには、書き込み可能を表すフラグ変数をセットしておきます。

6.1.6 データ受信／書き換え後の処理

データ受信処理にてエンド・レコードを判定し、受信データのフラッシュ・メモリ書き込みが終了すると、フラッシュ書き換えイベント処理内のデータ受信待ち状態を抜けて、データ受信時に算出したチェックサム判定用データ (プログラムコードサイズおよびチェックサムデータ/それぞれ 2 バイト分) をフラッシュ・メモリに書き込みます。本サンプルコードでは、チェックサム判定用データを、書き換え領域の最後尾部分 4 バイト H'0000 8FF0~H'0000 8FF3 (H'0000 8FF0~H'0000 8FF1 : プログラムコードサイズ、H'0000 8FF2~H'0000 8FF3 : チェックサムデータ) に格納します。

チェックサム判定用のデータ書き込み後はホストにメッセージを送信し、リセット待ち状態となります。

6.1.7 通信制御シーケンス

図 6.3に本サンプルコードの 通信制御シーケンスを示します。

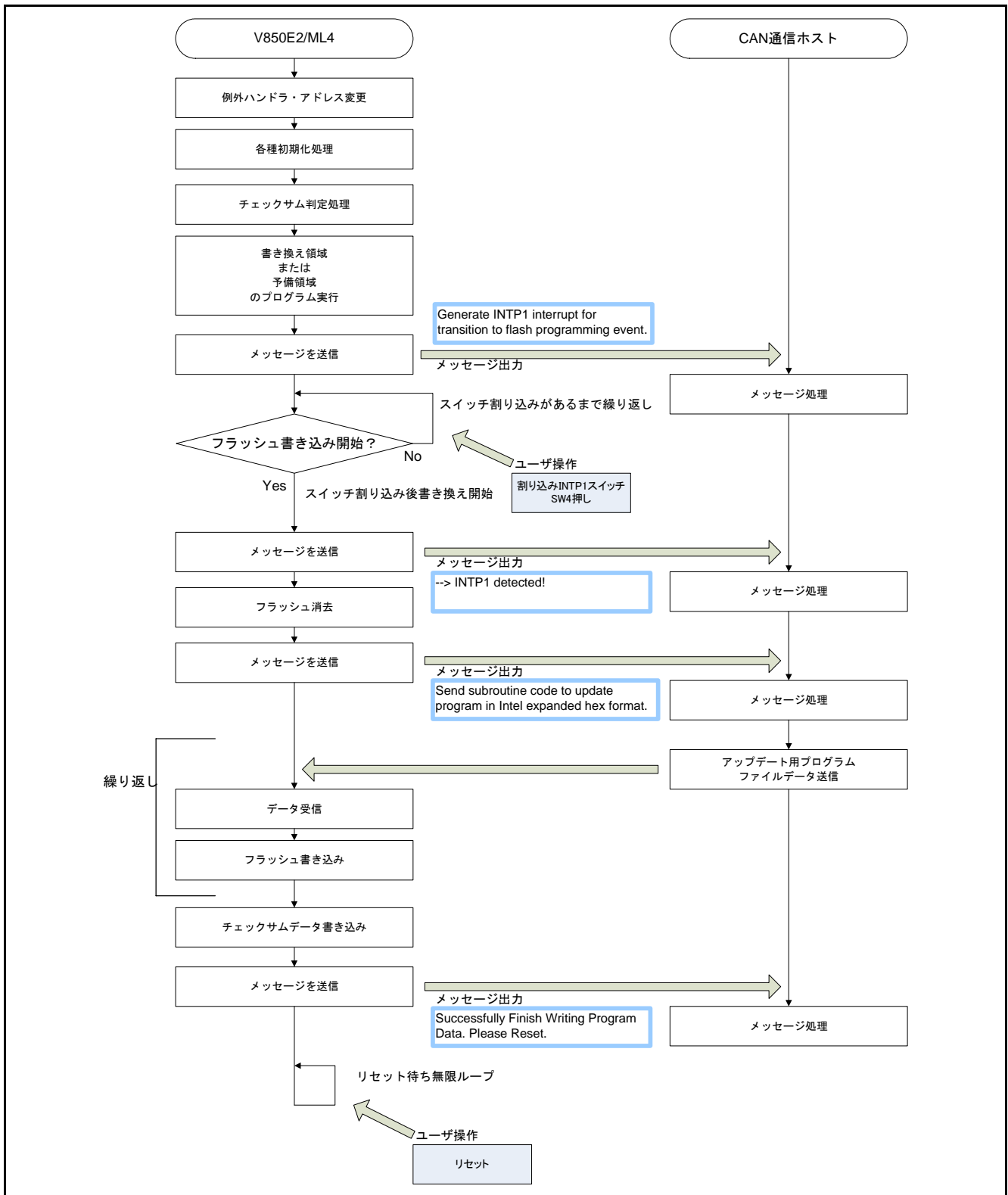


図6.3 通信制御シーケンス

6.2 ファイル構成

表 6.3に サンプルコードで使用するファイルを示します。なお、統合開発環境で自動生成されるファイルは除きます。

表6.3 サンプルコードで使用するファイル

ファイル名	概要	備考
main.c	メイン処理	
intp1.c	INTP1 割り込み処理	
flash.c	フラッシュ書き換え関連処理	
fcn0_can.c	CAN コントローラ関連処理	
taua0_led_sample.c	アップデート用サンプルプログラム、 LED 点滅ポート処理	
flash.h	フラッシュ・メモリ書き換え処理向け共通 ヘッダ	
r_typedefs.h	固定長整数型定義ヘッダ	
FSL.h	FSL ヘッダファイル	
except_handler_ram.asm	RAM 上例外ハンドラ*	RAM 上から割り込み処理 関数へジャンプ
cstart.asm	スタートアップ・ルーチン	標準のスタートアップ・ ルーチンからスタック・ サイズを変更し RAM の プログラム領域の初期化 を追加
libFSL_T05_REC_R32.lib	FSL ライブラリ (32 レジスタモード)	
v850e2ml4_flash_update_can.dir	リンク・ディレクティブ設定ファイル	
v850e2ml4_flash_update_can.fjp	far jump 呼び出し関数一覧ファイル	

【注】* 例外ハンドラ上に配置する、割り込みのハンドラ・アドレスから割り込みハンドラ関数へのジャンプ命令を記述しています。

6.3 定数一覧

表 6.4、表 6.5 に サンプルコードで使用する定数を示します。

表6.4 サンプルコードで使用する定数

定数名	設定値	内容
RET_OK	0	正常終了
RET_ERR	-1	異常終了
RET_ERR_FLASH_ACTIVATE	-1	フラッシュ環境の開始失敗
RET_ERR_FLASH_MODECHECK	-2	FLMD0 端子のチェック NG
RET_ERR_FLASH_ERASE	-3	消去処理失敗
RET_ERR_FLASH_WRITE	-4	書き込み失敗
RET_ERR_FLASH_IVERIFY	-5	内部ベリファイが失敗
RET_ERR_FLASH_DEACTIVATE	-6	フラッシュ環境の終了失敗
RET_ERR_FLASH_FLMD0_HIGH	-7	FLMD0 端子 High レベル設定失敗
RET_ERR_FLASH_FLMD0_LOW	-8	FLMD0 端子 Low レベル設定失敗
RET_ERR_FLASH_CAN_RX_NUM	-9	CAN 通信の受信データサイズ異常
RET_ERR_FLASH_HEX_LINESIZE	-10	ヘキサ・ファイルの行データ数が異常
RET_ERR_FLASH_HEX_DATA	-11	ヘキサ・ファイルのプログラムデータが異常
BLOCK_MASTER_PRG	8	書き換え領域ブロック番号
TOP_ADDR_MASTER_PRG	H'00008000	書き換え領域先頭アドレス
SIZE_MASTER_PRG	H'1000	書き換え領域サイズ (4K バイト)
SIZE_WRITE	16	書き込み指定サイズ
TOP_ADDR_MASTER_PRG_CHKSUM	TOP_ADDR_MASTER_PRG + SIZE_MASTER_PRG - SIZE_WRITE	チェックサム領域先頭アドレス (H'00008FF0)
TOP_ADDR_EXT_HANDLER	H'FEDF E000	転送先例外ハンドラ・アドレス先頭

表6.5 サンプルコードで使用する定数

定数名	設定値	内容
FLASH_STATUS_FLMD0_HIGH	H'01	FLMD0 の High 設定完了状態(プルアップ有効)
FLASH_STATUS_FSL_ACTIVATE	H'02	FSL の開始状態
HEXDATA_POS_RECMARK	0	ヘキサ・データのレコード・マークの位置
HEXDATA_POS_BYTE_NUM	1	ヘキサ・データのバイト数の位置
HEXDATA_POS_RECTYPE_UPPER	7	ヘキサ・データのレコード・タイプ上位桁の位置
HEXDATA_POS_RECTYPE_LOWER	8	ヘキサ・データのレコード・タイプ下位桁の位置
HEXDATA_POS_CODE_TOP	9	ヘキサ・データのコードの先頭位置
SIZE_BUF_RX_DATA	525	受信データ格納バッファサイズ (下記の合計) レコード・マーク : 1 文字、 バイト数 : 2 文字、 ロケーション・アドレス : 4 文字、 レコード・タイプ : 2 文字、 コード : (最大) 512 文字、 チェックサム : 2 文字、 リターン (¥r) + ニューライン (¥n) : 2 文字
FCN0_CAN_SIZE_DATA_MAX	8	最大送受信データ長
FCN0_CAN_NUM_MB	64	メッセージ・バッファの数
FCN0_CAN_GAP_MB_ADDR	H'40	メッセージ・バッファのアドレス間隔
FCN0_CAN_ADDR_M0_STRB	H'FF481024	FCN0M0STRB レジスタのアドレス
FCN0_CAN_ADDR_M0_CTL	H'FF489038	FCN0M0CTL レジスタのアドレス
FCN0_CAN_ADDR_M0_DAT	H'FF481000	FCN0M0DAT レジスタのアドレス
FCN0_CAN_ID_SEND_MSG	0	fcn0_can_tx_msg 関数で使用するメッセージ ID
PORT_BIT_P1_4	H'0010	ポート機能設定 P1_4 のビット位置
PORT_BIT_P2_3	H'0008	ポート機能設定 P2_3 のビット位置
PORT_BIT_P2_14	H'4000	ポート機能設定 P2_14 のビット位置
PORT_BIT_P2_15	H'8000	ポート機能設定 P2_15 のビット位置

6.4 変数一覧

表 6.6に グローバル変数を示します。

表6.6 グローバル変数

型	変数名	内容	使用関数
uint8_t	g_flag_start_flash_reprog	フラッシュ書き込み開始フラグ	main, intp1_isr
fsl_status_t	g_error_fsl_status	FSL のエラー保存	main, flash_activate, flash_modecheck, flash_erase, flash_write, flash_iverify
uint32_t	g_addr_write_error	書き込みエラーアドレス	main, flash_write
uint8_t	g_flag_w_data_buf0_full	書き込みバッファ 0 フルフラグ	flash_reprogram, flash_store_can_data
uint8_t	g_flag_w_data_buf1_full	書き込みバッファ 1 フルフラグ	flash_reprogram, flash_store_can_data
uint8_t	g_status_end_record	終了レコード受信フラグ	flash_reprogram, flash_store_can_data
uint16_t	g_chksm_size	書き込みデータのプログラムコードサイズ	flash_reprogram, flash_store_can_data
uint16_t	g_chksm_data	書き込みデータのチェックサムデータ	flash_reprogram, flash_store_can_data
uint8_t	g_buf_write_data0 [SIZE_WRITE]	書き込みデータ格納バッファ 0	flash_reprogram, flash_store_can_data
uint32_t	g_cnt_store_buf_w_data0	書き込みデータ格納バッファ 0 データ数	flash_reprogram, flash_store_can_data
uint8_t	g_buf_write_data1 [SIZE_WRITE]	書き込みデータ格納バッファ 1	flash_reprogram, flash_store_can_data
uint32_t	g_cnt_store_buf_w_data1	書き込みデータ格納バッファ 1 データ数	flash_reprogram, flash_store_can_data
uint32_t	g_index_rx_data	受信データ格納場所インデックス	flash_reprogram, flash_store_can_data
uint8_t	g_buf_rx_data [SIZE_BUF_RX_DATA]	受信データ格納バッファ	flash_store_can_data
int8_t	g_status_store_error	エラーフラグ	flash_reprogram, flash_store_can_data
uint8_t	g_flag_flash_status	フラッシュ環境ステータス	flash_init, flash_activate, flash_end
char	g_msg_sendcode[]	プログラム送信要求メッセージ	flash_reprogram

6.5 関数一覧

表 6.7に 関数を示します。

表6.7 関数

関数名	概要
main	メイン処理
except_handler_addr_set	例外ハンドラ・ベース・アドレスの切り替え処理
check_sum_check	書き換え領域のチェックサム判定処理
intp1_init	INTP1 割り込み初期化処理
intp1_isr	INTP1 割り込み処理
flash_reprogram	フラッシュ書き換え処理
flash_init	フラッシュ環境の初期化処理
flash_activate	フラッシュ環境の開始処理
flash_modecheck	FSL による FLMD0 端子のチェック処理
flash_erase	指定ブロックの消去処理
flash_write	指定アドレスからの書き込み処理
flash_iverify	指定ブロックの内部ベリファイ処理
flash_end	フラッシュ環境終了処理
flash_set_flmd0	FLMD0 端子レベル設定処理
flash_store_can_data	受信データ変換格納処理
hex2bin	テキストバイナリ変換処理
taua0_led_sample	定周期 LED 点滅用 TAU0 初期化処理（書き換え領域サンプル関数）
taua0_led_spare	定周期 LED 点滅用 TAU0 初期化処理（予備領域サンプル関数）
taua0_i0_interval_timer_isr *	TAU0 インターバル・タイマ割り込み処理
fcn0_can_init	CAN コントローラ チャンネル 0 (FCN0) 初期化処理
fcn0_can_port_init	FCN0 ポート初期化処理
fcn0_can_mb_init	FCN0 メッセージ・バッファ初期化処理
fcn0_can_tx_msg	FCN0 メッセージ送信処理
fcn0_can_tx	FCN0 送信処理
fcn0_can_rx	FCN0 受信処理
fcn0_can_rx_isr	FCN0 受信完了割り込み処理
fcn0_can_error_isr	FCN0 エラー割り込み例外処理

【注】 * LED の点滅処理よりも CAN 通信により受信したプログラムデータの格納処理を優先するために、割り込みハンドラ関数 taua0_ch0_interval_timer_isr は多重割り込みを許可しています。また、TAU0 インターバル・タイマ割り込みは FCN0 受信完了割り込みよりも優先順位を低く設定しています。

6.6 関数仕様

サンプルコードの関数仕様を示します。

main	
概要	メイン処理
ヘッダ	
宣言	void main (void)
説明	変数、例外ハンドラ・アドレス、INTP1 割り込み、CAN コントローラの初期化後に、チェックサム判定結果に従って、書き換え領域または予備領域に配置したプログラムを実行します。その後、割り込み許可と INTP1 割り込み要求メッセージ出力を行った後、INTP1 割り込みが発生した場合はフラッシュ書き換え処理を行い、書き換えに成功した場合はリセット要求メッセージ、失敗した場合はエラーメッセージを出力して、無限ループに入ります。
引数	なし
リターン値	なし
except_handler_addr_set	
概要	例外ハンドラ・ベース・アドレスの切り替え処理
ヘッダ	
宣言	int32_t except_handler_addr_set (uint32_t base_addr)
説明	引数に指定された値を SW_BASE レジスタに設定した後、SW_CTL レジスタの SET ビットに 1 を設定し、SW_BASE レジスタの内容を例外ハンドラ・ベース・アドレス・レジスタ(EH_BASE)へ転送します。
引数	uint32_t base_addr : 例外ハンドラ・ベース・アドレス設定値 (下位 12 ビットは 0 であること)
リターン値	0 (RET_OK) : 正常終了 -1 (RET_ERR) : 引数エラー(下位 12 ビットが 0 でない)
check_sum_check	
概要	書き換え領域のチェックサム判定処理
ヘッダ	
宣言	int32_t check_sum_check (void)
説明	書き換え領域の最後尾部分 4 バイト (H'0000 8FF0~H'0000 8FF3) に格納したプログラムコードサイズおよびチェックサムデータをもとに、書き換え領域の先頭番地 (H'0000 8000) からサム値を算出し、チェックサムデータとの一致判定を行います。
引数	なし
リターン値	0 (RET_OK) : チェックサム一致 -1 (RET_ERR) : チェックサム不一致

intp1_init

概要	INTP1 割り込み初期化処理
ヘッダ	
宣言	void intp1_init (void)
説明	INTP1 の初期化処理を行います。P2_3 の端子機能を INTP1 入力に設定した後、割り込みコントローラにて割り込み要求を入力立ち下がりエッジで検出するように設定します。その後、INTP1 の割り込み優先レベルを設定します。
引数	なし
リターン値	なし

intp1_isr

概要	INTP1 割り込み処理
ヘッダ	
宣言	void intp1_isr (void)
説明	INTP1 割り込みがあったことを表すフラグをセットします。
引数	なし
リターン値	なし

flash_reprogram

概要	フラッシュ書き換え処理	
ヘッダ	flash.h	
宣言	int32_t flash_reprogram (void)	
説明	最初にフラッシュ環境の初期化処理、フラッシュ環境の開始処理、FLMD0 端子のチェック処理、書き換えブロック消去処理を実行します。次に、プログラム送信要求メッセージを送信して、プログラム受信待ちとフラッシュ書き込みを行うループに入ります。プログラムを最後まで受信して書き込みが終了すると、最後にチェックサムデータを書き込んでフラッシュ書き換え終了処理を行います。	
引数	なし	
リターン値	0 (RET_OK)	: 正常終了
	-1 (RET_ERR_FLASH_ACTIVATE)	: フラッシュ環境の開始失敗
	-2 (RET_ERR_FLASH_MODECHECK)	: FLMD0 端子のチェック NG
	-3 (RET_ERR_FLASH_ERASE)	: 消去処理失敗
	-4 (RET_ERR_FLASH_WRITE)	: 書き込み失敗
	-5 (RET_ERR_FLASH_IVERIFY)	: 内部ペリファイが失敗
	-6 (RET_ERR_FLASH_DEACTIVATE)	: フラッシュ環境の終了失敗
	-7 (RET_ERR_FLASH_FLMD0_HIGH)	: FLMD0 端子 High レベル設定失敗
	-8 (RET_ERR_FLASH_FLMD0_LOW)	: FLMD0 端子 Low レベル設定失敗
	-9 (RET_ERR_FLASH_CAN_RX_NUM)	: CAN 通信の受信データサイズ異常
	-10 (RET_ERR_FLASH_HEX_LINESIZE)	: ヘキサ・ファイルの行データ数が異常
	-11 (RET_ERR_FLASH_HEX_DATA)	: ヘキサ・ファイルのプログラムデータ異常

flash_init

概要	フラッシュ環境の初期化処理	
ヘッダ		
宣言	int32_t flash_init (void)	
説明	FLMD0 端子レベル設定処理関数を実行して FLMD0 端子を High レベルに設定した後、FSL_Init 関数を実行してセルフ・ライブラリの初期化を行います。flash_set_flmd0 関数がエラーになった場合は RET_ERR_FLASH_FLMD0_HIGH を返します。	
引数	なし	
リターン値	0 (RET_OK)	: 正常終了
	-7 (RET_ERR_FLASH_FLMD0_HIGH)	: FLMD0 端子 High レベル設定失敗

flash_activate	
概要	フラッシュ環境の開始処理
ヘッダ	
宣言	int32_t flash_activate (void)
説明	FSL_FlashEnv_Activate 関数をコールして、フラッシュ環境を開始します。正常終了の場合は、グローバル変数 g_flag_flash_status にフラッシュ環境が開始したことを表すビットを立てた後、RET_OK を戻して終了します。FSL_FlashEnv_Activate 関数が FSL_OK 以外を戻した場合は、その戻り値をグローバル変数 g_error_fsl_status に保存して、RET_ERR_FLASH_ACTIVATE を戻して終了します。
引数	なし
リターン値	0 (RET_OK) : 正常終了 -1 (RET_ERR_FLASH_ACTIVATE) : フラッシュ環境の開始失敗
flash_modecheck	
概要	FSL による FLMD0 端子のチェック処理
ヘッダ	
宣言	int32_t flash_modecheck (void)
説明	FSL_ModeCheck 関数をコールして、FLMD0 端子のチェックを行います。正常終了の場合は、RET_OK を戻して終了します。FSL_ModeCheck 関数が FSL_OK 以外を戻した場合は、その戻り値をグローバル変数 g_error_fsl_status に保存して、RET_ERR_FLASH_MODECHECK を戻して終了します。
引数	なし
リターン値	0 (RET_OK) : 正常終了 -2 (RET_ERR_FLASH_MODECHECK) : FLMD0 端子のチェック NG
flash_erase	
概要	指定ブロックの消去処理
ヘッダ	
宣言	int32_t flash_erase (uint32_t start_block, uint32_t end_block)
説明	指定された引数に従って FSL_Erase 関数をコールすることで、ブロックの消去を行います。また、FSL_Erase 関数実行後に FSL_StatusCheck 関数をコールして、消去処理の完了が確認できるまで待ちます。FSL_Erase 関数または FSL_StatusCheck 関数がエラー値を戻した場合は、その戻り値をグローバル変数 g_error_fsl_status に保存して、RET_ERR_FLASH_ERASE を戻して終了します。
引数	uint32_t start_block : 消去を行う範囲の先頭ブロック番号 uint32_t end_block : 消去を行う範囲の最終ブロック番号
リターン値	0 (RET_OK) : 正常終了 -3 (RET_ERR_FLASH_ERASE) : 消去処理失敗

flash_write

概要	指定アドレスからの書き込み処理
ヘッダ	
宣言	int32_t flash_write (uint8_t * src_data_addr, uint32_t dst_write_addr, uint32_t length)
説明	指定された引数に従って FSL_Write 関数をコールすることで、フラッシュへの書き込みを行います。また、FSL_Write 関数実行後は、FSL_StatusCheck 関数をコールして、書き込み処理の完了が確認できるまで待ちます。FSL_Write 関数または FSL_StatusCheck 関数がエラー値を戻した場合は、その戻り値をグローバル変数 g_error_fsl_status に保存して、RET_ERR_FLASH_WRITE を戻して終了します。
引数	uint8_t * src_data_addr : 書き込みデータ先頭アドレス (内蔵 ROM 領域外) uint32_t dst_write_addr : 書き込みデータの格納先アドレス (4 ワード境界) uint32_t length : 書き込みデータ長 (ワード単位、4 ワード境界、MAX: 内蔵 ROM サイズ)
リターン値	0 (RET_OK) : 正常終了 -4 (RET_ERR_FLASH_WRITE) : 書き込み失敗

flash_iverify

概要	指定ブロックの内部ベリファイ処理
ヘッダ	
宣言	int32_t flash_iverify (uint32_t start_block, uint32_t end_block)
説明	引数に従って FSL_IVerify 関数をコールして、指定ブロックの内部ベリファイ処理を行います。また、FSL_IVerify 関数実行後は、FSL_StatusCheck 関数をコールして、内部ベリファイ処理の完了が確認できるまで待ちます。FSL_IVerify 関数または FSL_StatusCheck 関数がエラー値を戻した場合は、その戻り値をグローバル変数 g_error_fsl_status に保存して、RET_ERR_FLASH_IVERIFY を戻して終了します。
引数	uint32_t start_block : ベリファイチェックを行う範囲の先頭ブロック番号 uint32_t end_block : ベリファイチェックを行う範囲の最終ブロック番号
リターン値	0 (RET_OK) : 正常終了 -5 (RET_ERR_FLASH_IVERIFY) : 内部ベリファイが失敗

flash_end	
概要	フラッシュ書き換え終了処理
ヘッダ	
宣言	int32_t flash_end (void)
説明	FSL_FlashEnv_Deactivate 関数をコールしてフラッシュ環境を終了した後で flash_set_flmd0 関数をコールして FLMD0 端子を Low レベルに設定します。FSL_FlashEnv_Deactivate 関数がエラー値を戻した場合は、RET_ERR_FLASH_DEACTIVATE を戻します。flash_set_flmd0 関数が 0 以外を戻した場合は、RET_ERR_FLASH_FLMD0_LOW を戻して終了します。
引数	なし
リターン値	0 (RET_OK) : 正常終了 -6 (RET_ERR_FLASH_DEACTIVATE) : フラッシュ環境の終了失敗 -8 (RET_ERR_FLASH_FLMD0_LOW) : FLMD0 端子 High レベル設定失敗
flash_set_flmd0	
概要	FLMD0 端子レベル設定処理
ヘッダ	
宣言	int32_t flash_set_flmd0 (uint8_t level)
説明	FLMD 制御レジスタを設定して、FLMD0 のプルアップ/プルダウン制御を切り替えます。保護レジスタへの書き換えシーケンスに従い、FLMD 保護コマンド・レジスタに H'A5 を代入した後、FLMD 制御レジスタに引数で指定した値を代入し、反転した値を代入し、再び引数で指定した値を代入します。最後にレジスタの値が変更できていることを確認して終了します。
引数	uint8_t level : 0x00 : FLMD0 端子を Low レベルに設定 0x01 : FLMD0 端子を High レベルに設定
リターン値	0 (RET_OK) : 正常終了 -1 (RET_ERR) : FLMDCNT レジスタへの書き込み動作エラー
flash_store_can_data	
概要	受信データ変換格納処理
ヘッダ	flash.h
宣言	void flash_store_can_data (void)
説明	ヘキサ・データを 1 行毎に変換バイナリ変換してバッファ格納します。1 行分のヘキサ・データがデータ・レコードの場合、データをバイナリ形式に変換して、バッファがフルになるまで保存します。1 行分のヘキサ・データがエンド・レコードの場合、バッファの残りを H'FF で埋めて受信が終了したことを表すフラグを立てます。
引数	なし
リターン値	なし

hex2bin	
概要	テキストバイナリ変換処理
ヘッダ	
宣言	int32_t hex2bin(uint8_t upper, uint8_t lower)
説明	テキストデータ (2 文字) を 1 バイトのバイナリデータに変換します。 引数に与えたデータが「0」～「9」または「A」～「F」のテキストデータであれば有効データとみなし、H'0～H'F のバイナリデータに変換します。 第一引数 (upper) の変換結果を 4 ビット左シフトし、第二引数 (lower) の変換結果との論理和をとった後、1 バイトのバイナリデータとして返却します。
引数	uint8_t upper : 上位 4 ビット用のテキストデータ uint8_t lower : 下位 4 ビット用のテキストデータ
リターン値	0～255 : 1 バイトのバイナリデータ -1 (RET_ERR) : 入力データ不正

taua0_led_sample	
概要	定周期 LED 点滅用 TAU0 初期化処理 (書き換え領域サンプル関数)
ヘッダ	
宣言	void taua0_led_sample (void)
説明	LED を点滅させるために、LED に接続されたポートを出力に設定し、TAU0 を一定周期で割り込みを発生させるインターバル・タイマに設定します。
引数	なし
リターン値	なし

taua0_led_spare	
概要	定周期 LED 点滅用 TAU0 初期化処理 (予備領域サンプル関数)
ヘッダ	
宣言	void taua0_led_spare (void)
説明	LED を点滅させるために、LED に接続されたポートを出力に設定し、TAU0 を一定周期で割り込みを発生させるインターバル・タイマに設定します。
引数	なし
リターン値	なし

taua0_i0_interval_timer_isr

概要	TAUA0 インターバル・タイマ割り込み処理
ヘッダ	
宣言	void taua0_i0_interval_timer_isr (void)
説明	LED 点滅のためのポート P1_4 の出力を反転します。
引数	なし
リターン値	なし

fcn0_init

概要	FCN0 初期化処理
ヘッダ	
宣言	int32_t fcn0_init (void)
説明	FCN0 ポート初期化処理 (fcn0_port_init 関数) を行った後、FCN0 のシステム・クロックと通信ポーレートを設定して、モジュールの動作を許可し、関連する割り込みの許可を行います。そしてメッセージ・バッファの初期化 (fcn0_mb_init 関数) を行った後、FCN0 を通常動作モードに移行させます。
引数	なし
リターン値	0 (RET_OK) : 正常終了 -1 (RET_ERR) : レジスタ状態異常

fcn0_port_init

概要	FCN0 ポート初期化処理
ヘッダ	
宣言	void fcn0_port_init (void)
説明	P2_14 端子を CAN 通信の受信、P2_15 端子を CAN 通信の送信で使用するためのポートの設定を行います。
引数	なし
リターン値	なし

fcn0_mb_init

概要	FCN0 メッセージ・バッファ初期化処理
ヘッダ	
宣言	void fcn0_mb_init (void)
説明	全てのメッセージ・バッファに対し、最小限の初期化を行った後、メッセージ・バッファ 0 を受信用、メッセージ・バッファ 1 を送信用に設定します。
引数	なし
リターン値	なし

fcn0_can_tx_msg

概要	FCN0 メッセージ送信処理
ヘッダ	
宣言	void fcn0_can_tx_msg (char * msg)
説明	FCN0 送信処理 (fcn0_can_tx 関数) を必要な回数実行して、引数で指定されたメッセージを送信します。メッセージは文字列終端文字「¥0」があるまで、終端文字を含めて送信します。送信処理を実行する際の送信データの ID は 0 を設定します。
引数	char * msg : 送信メッセージ文字列
リターン値	なし

fcn0_can_tx

概要	FCN0 送信処理
ヘッダ	
宣言	int32_t fcn0_can_tx (int8_t length, uint8_t * send_data, uint32_t id)
説明	FCN0 のメッセージ・バッファ 1 を使用して、引数 send_data で指定されたデータを引数 length で指定されたバイト数 CAN 通信データとして送信します。FCN0M1TRQF と FCN0M1RDYF が送信可能な状態になるまで待ち、送信データを指定サイズ分メッセージ・バッファに設定し、引数 id で指定された ID を設定して送信します。
引数	int8_t length : 送信データ長 uint8_t * send_data : 送信データの先頭アドレス uint32_t id : 送信データに設定する ID
リターン値	0 (RET_OK) : 正常終了 -1 (RET_ERR) : 引数エラー

fcn0_can_receive

概要	FCN0 受信処理
ヘッダ	
宣言	int32_t fcn0_can_receive (int8_t * length, uint8_t * recv_data, uint32_t * id)
説明	FCN0 のメッセージ・バッファ 0 にデータを受信しているとき、引数 recv_data で指定したアドレスの領域に受信データを格納します。メッセージ・バッファにあるデータを格納する際に、受信データ長を引数 length に、ID を引数 id に格納します。 上記のデータ格納後に、メッセージ・バッファの保存・更新ビットが立っていたときはデータ格納処理を最初からやり直します。
引数	int8_t * length : 受信データ長 (受信データ長異常発生時は実際に保存したデータ数 8 になります。) uint8_t * recv_data : 受信データ格納領域の先頭アドレス uint32_t * id : 受信データに設定されていた ID
リターン値	0~8 (0~FCN_MAX_DATA_LENGTH) : 受信データ長、正常値 9 以上 (>FCN_MAX_DATA_LENGTH) : 受信データ長異常

fcn0_rx_isr

概要	FCN0 受信完了割り込み
ヘッダ	
宣言	void fcn0_rx_isr (void)
説明	受信データ変換格納処理(flash_store_can_data 関数)を実行します。
引数	なし
リターン値	なし

fcn0_error_isr

概要	FCN0 エラー割り込み処理
ヘッダ	
宣言	void fcn0_error_isr (void)
説明	バスオフ状態の場合、強制復帰を行います。
引数	なし
リターン値	なし

6.7 フローチャート

6.7.1 スタートアップ・ルーチンの処理

図 6.4に スタートアップ・ルーチンの処理のフローチャートを示します。

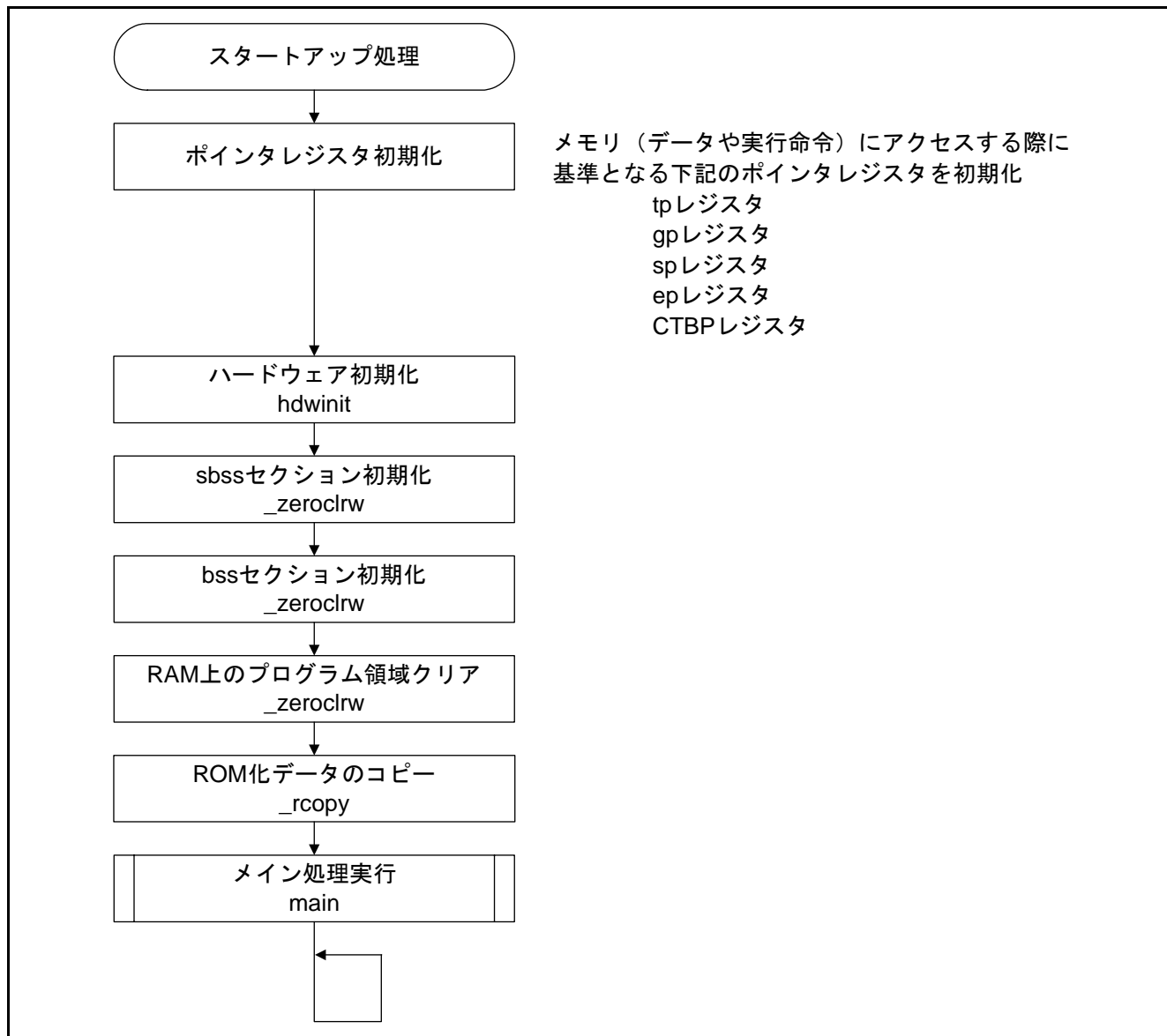


図6.4 スタートアップ・ルーチンの処理

6.7.2 メイン処理

図 6.5にメイン処理のフローチャートを示します。

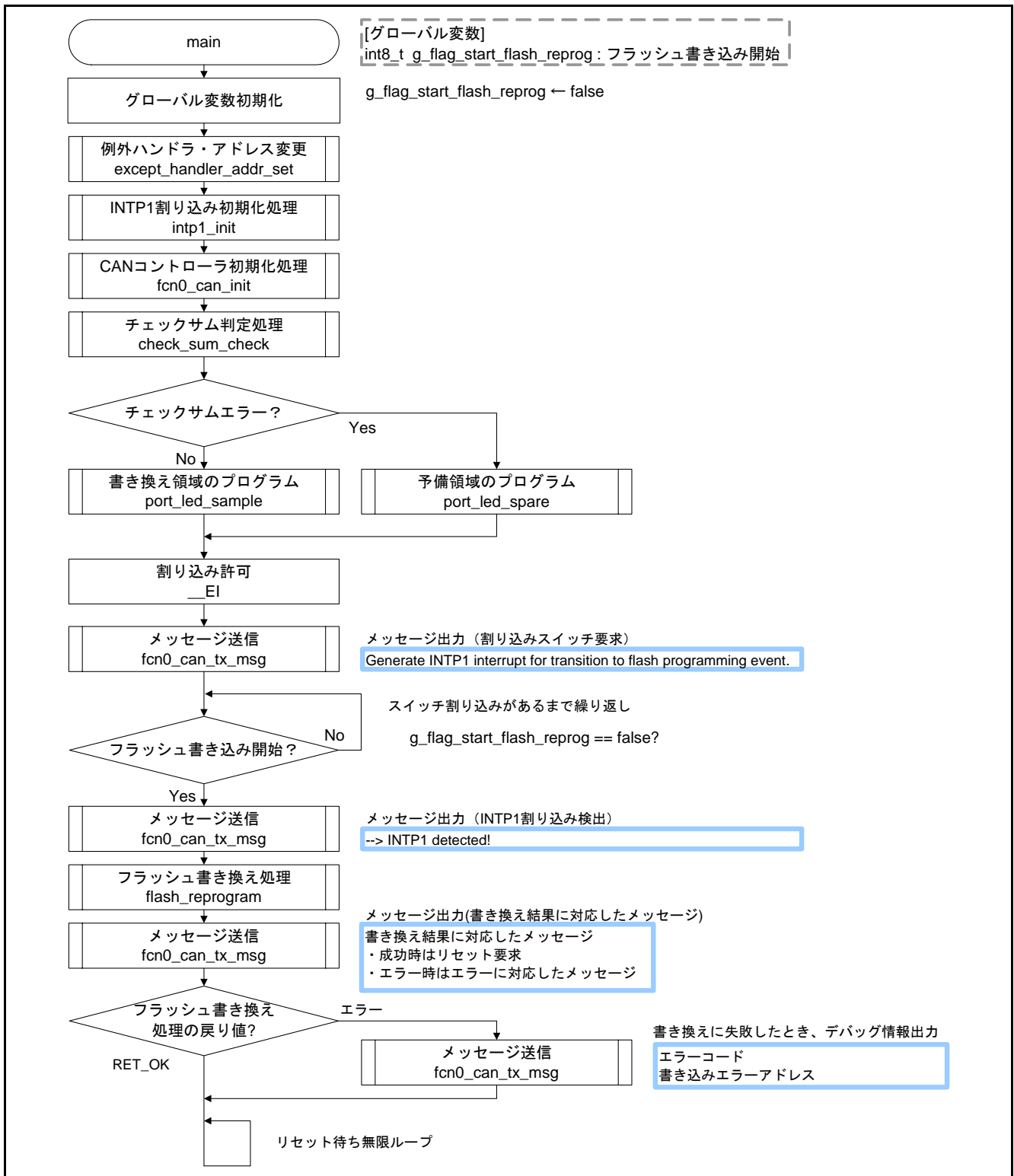


図6.5 メイン処理

6.7.3 例外ハンドラ・アドレス切り替え処理

図 6.6に 例外ハンドラ・アドレス切り替え処理のフローチャートを示します。

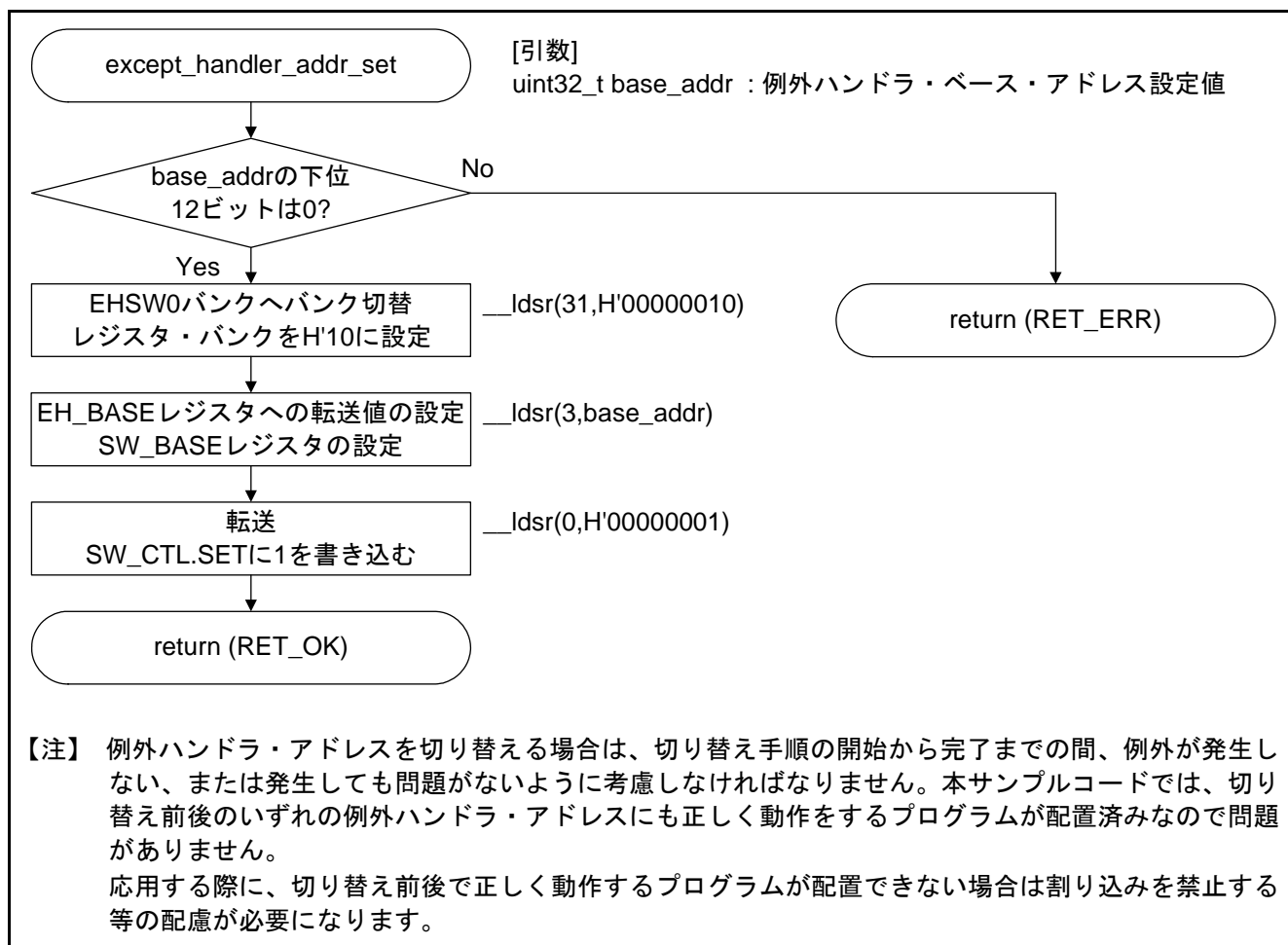


図6.6 例外ハンドラ・アドレス切り替え処理

6.7.4 書き換え領域のチェックサム判定処理

図 6.7 に 書き換え領域のチェックサム判定処理のフローチャートを示します。

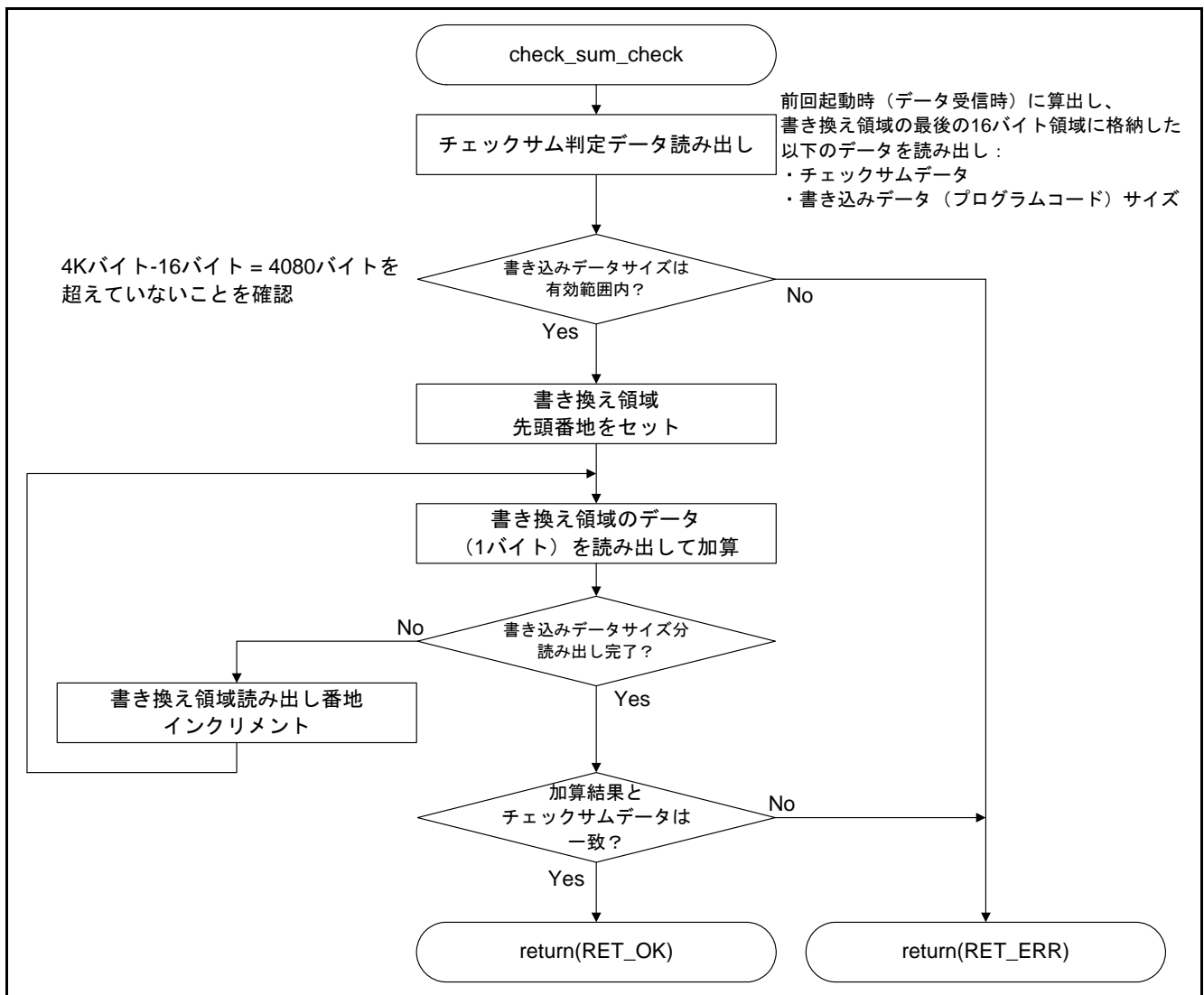


図6.7 書き換え領域のチェックサム判定処理

6.7.5 INTP1 割り込み初期化処理

図 6.8に INTP1 割り込み初期化処理のフローチャートを示します。

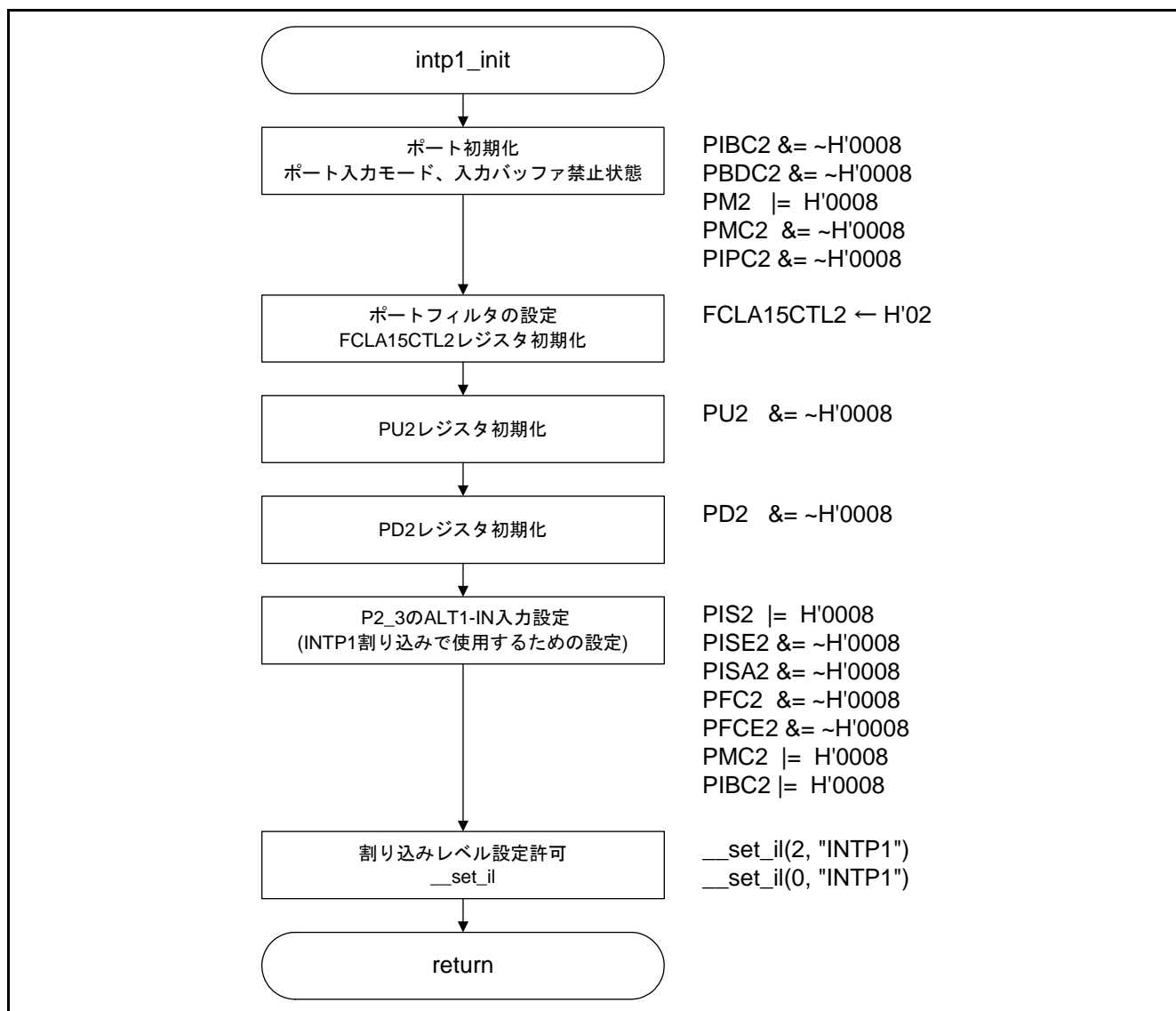


図6.8 INTP1 割り込み初期化処理

6.7.6 INTP1 割り込み処理

図 6.9に INTP1 割り込み処理のフローチャートを示します。

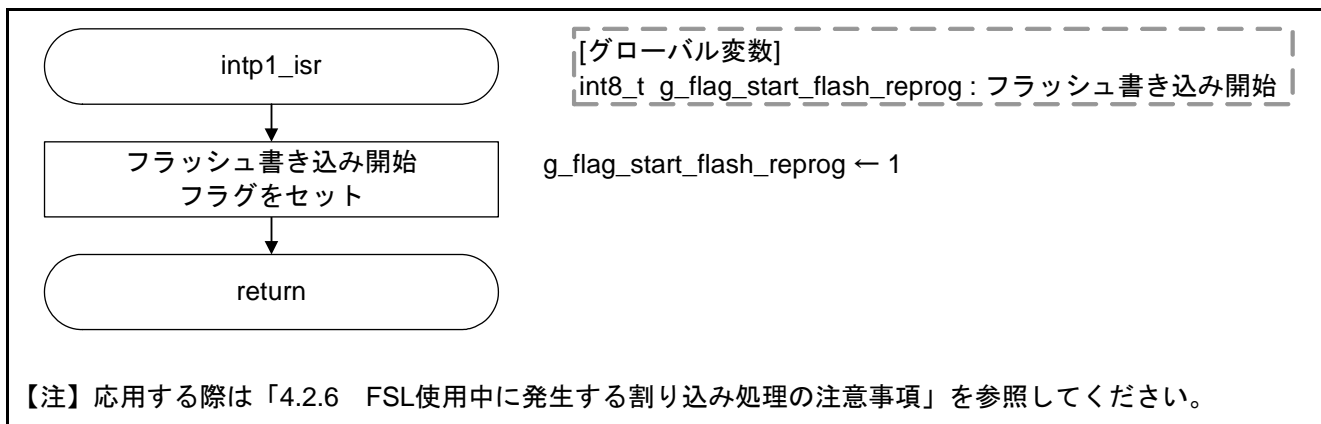


図6.9 INTP1 割り込み処理

6.7.7 フラッシュ書き換え処理

図 6.10、図 6.11にフラッシュ書き換えのフローチャートを示します。

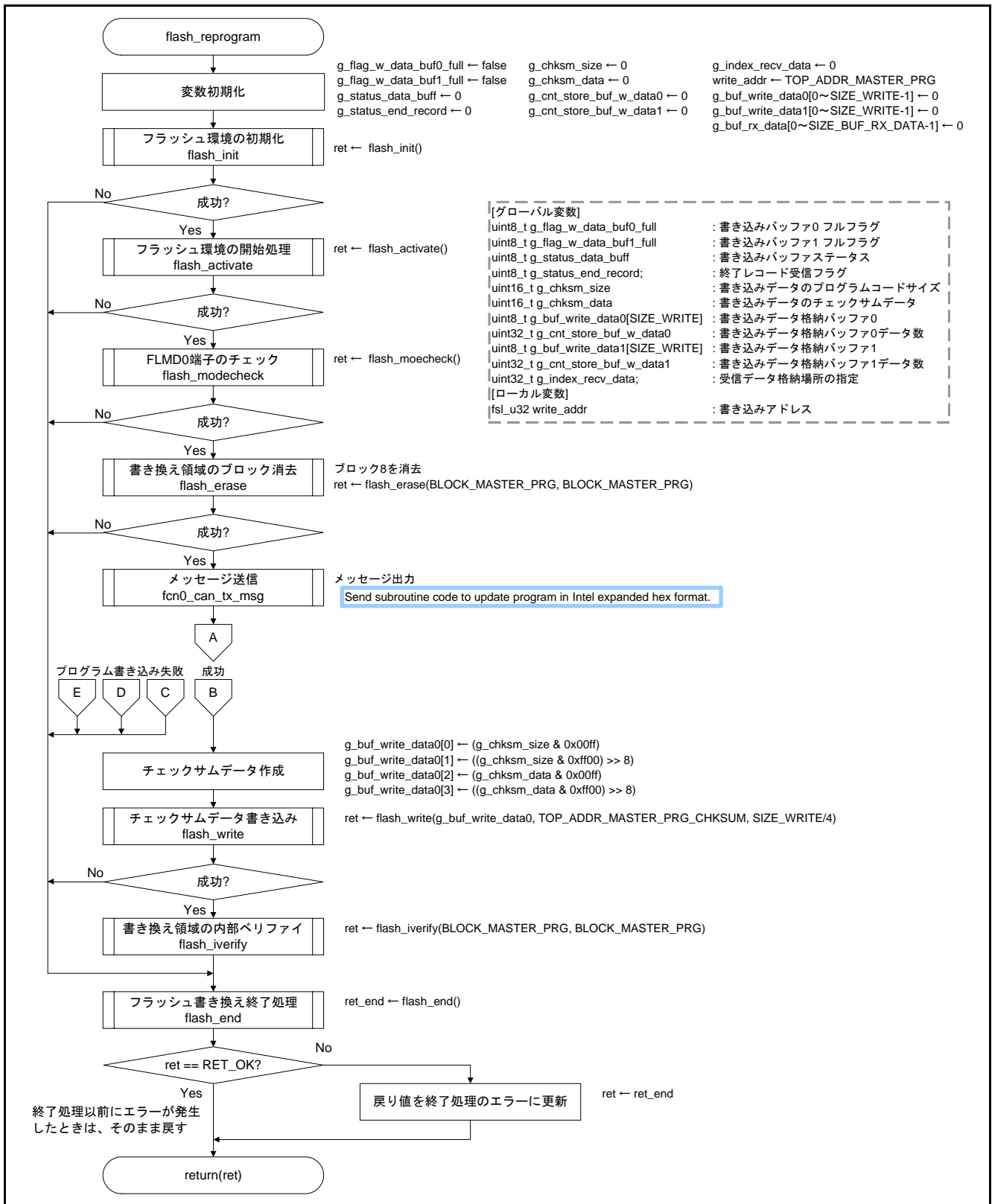


図6.10 フラッシュ書き換え処理 (1/2)

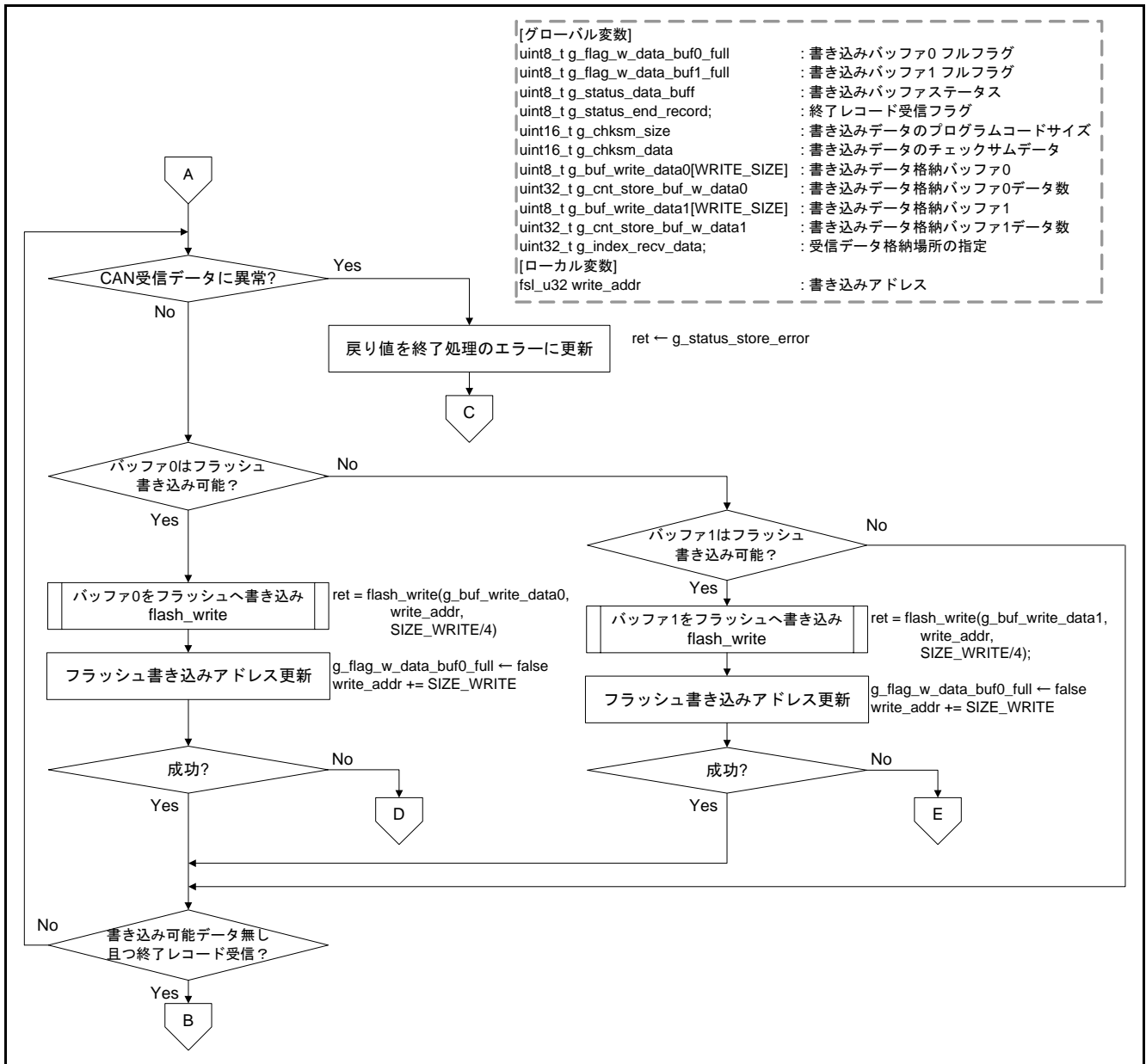


図6.11 フラッシュ書き換え処理 (2/2)

6.7.8 フラッシュ環境の初期化処理

図 6.12 にフラッシュ環境の初期化処理のフローチャートを示します。

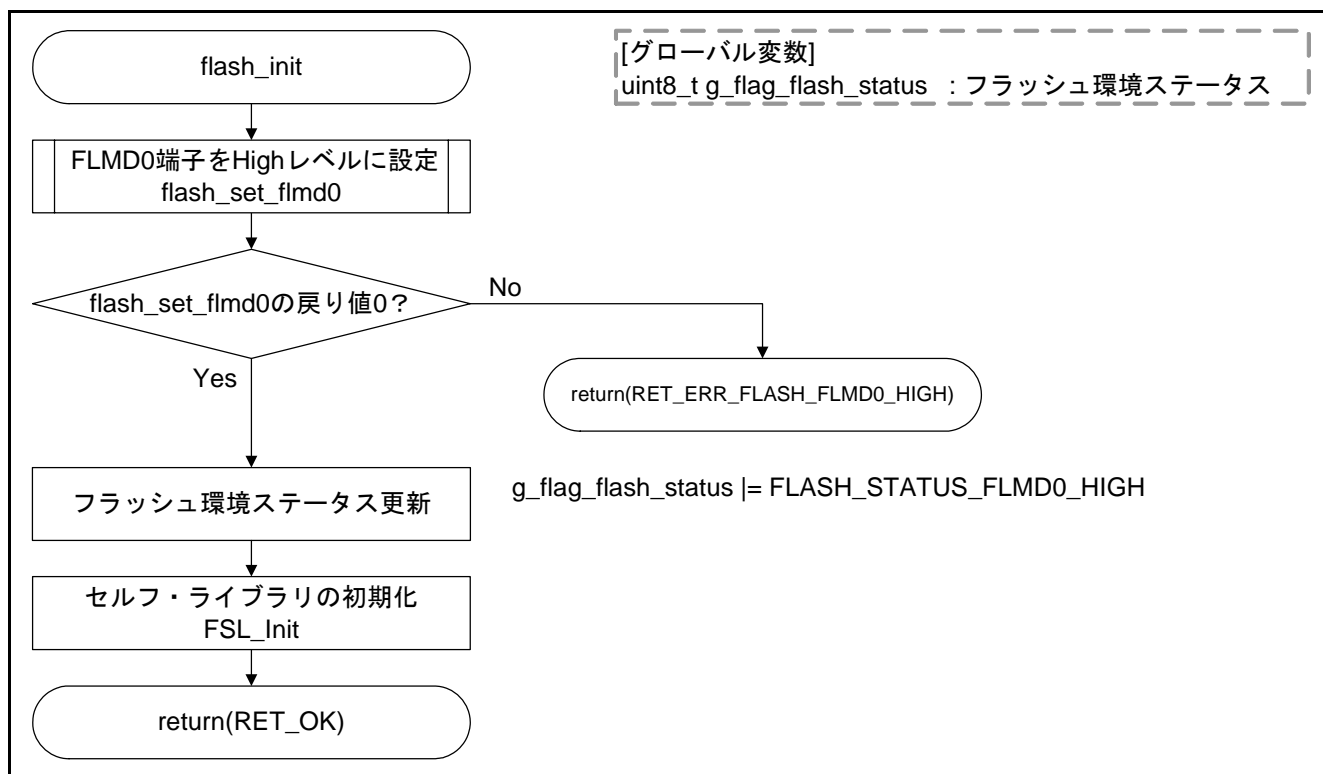


図6.12 フラッシュ環境の初期化処理

6.7.9 フラッシュ環境の開始処理

図 6.13にフラッシュ環境の開始処理のフローチャートを示します。

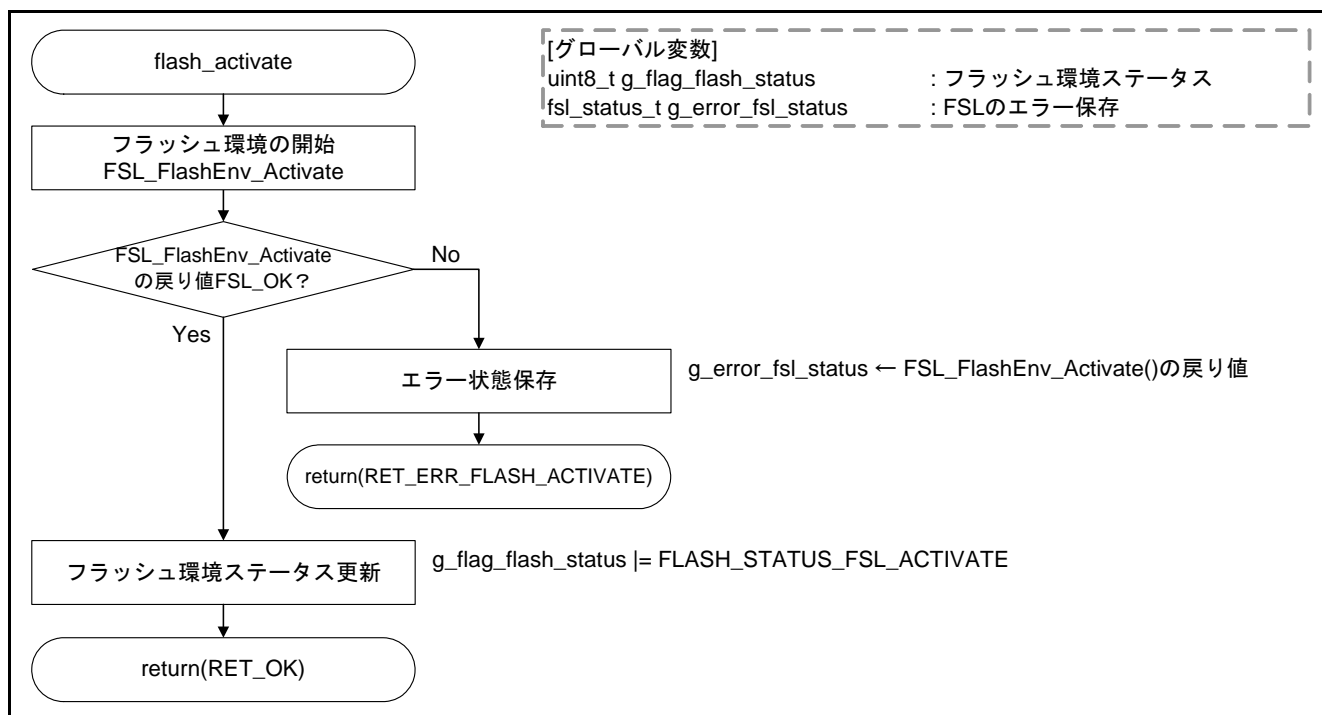


図6.13 フラッシュ環境の開始処理

6.7.10 FSL による FLMD0 端子のチェック処理

図 6.14に FSLによるFLMD0 端子のチェック処理のフローチャートを示します。

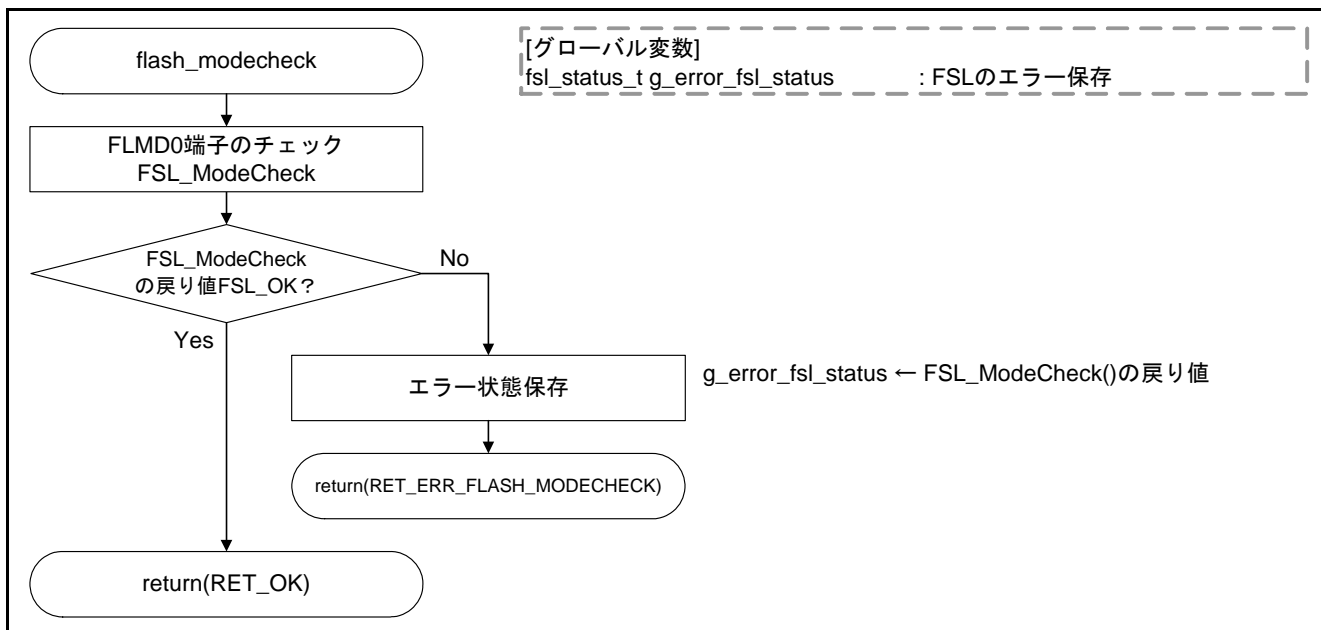


図6.14 FSL による FLMD0 端子のチェック処理

6.7.11 指定ブロックの消去処理

図 6.15に 指定ブロックの消去処理のフローチャートを示します。

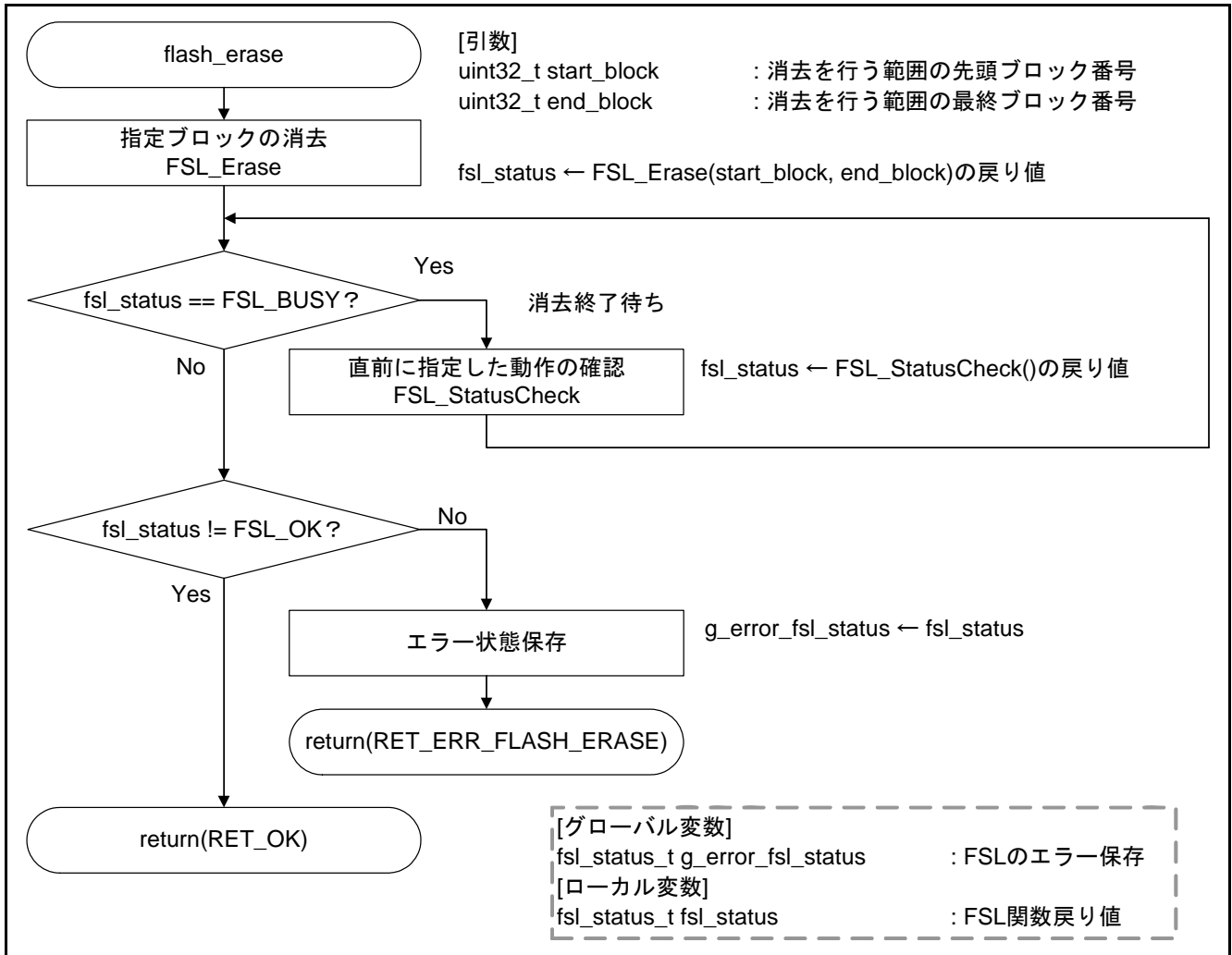


図6.15 指定ブロックの消去処理

6.7.12 指定アドレスからの書き込み処理

図 6.16に 指定アドレスからの書き込み処理のフローチャートを示します。

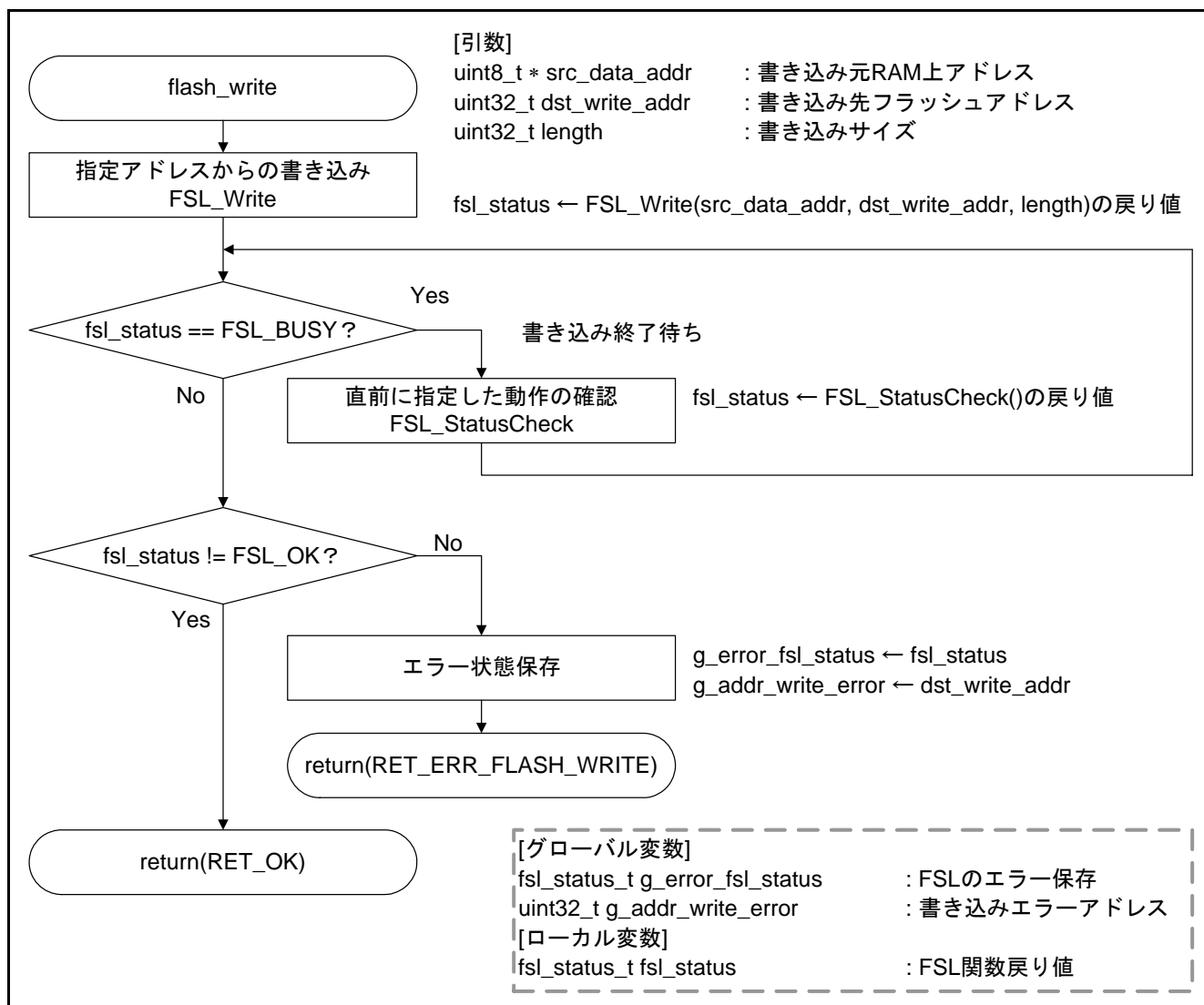


図6.16 指定アドレスからの書き込み処理

6.7.13 指定ブロックの内部ペリファイ処理

図 6.17に 指定ブロックの内部ペリファイ処理のフローチャートを示します。

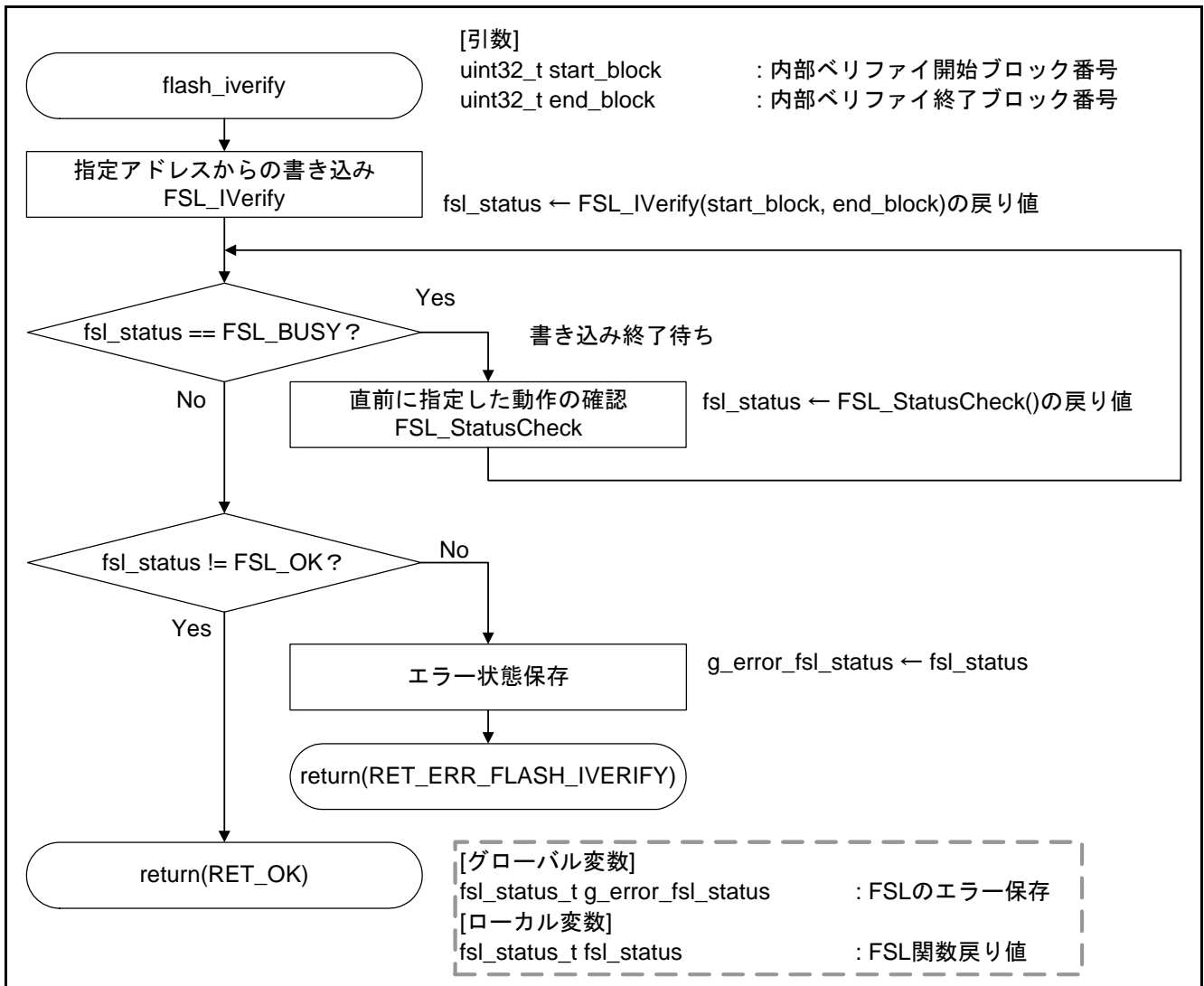


図6.17 指定ブロックの内部ペリファイ処理

6.7.14 フラッシュ環境終了処理

図 6.18にフラッシュ環境終了処理のフローチャートを示します。

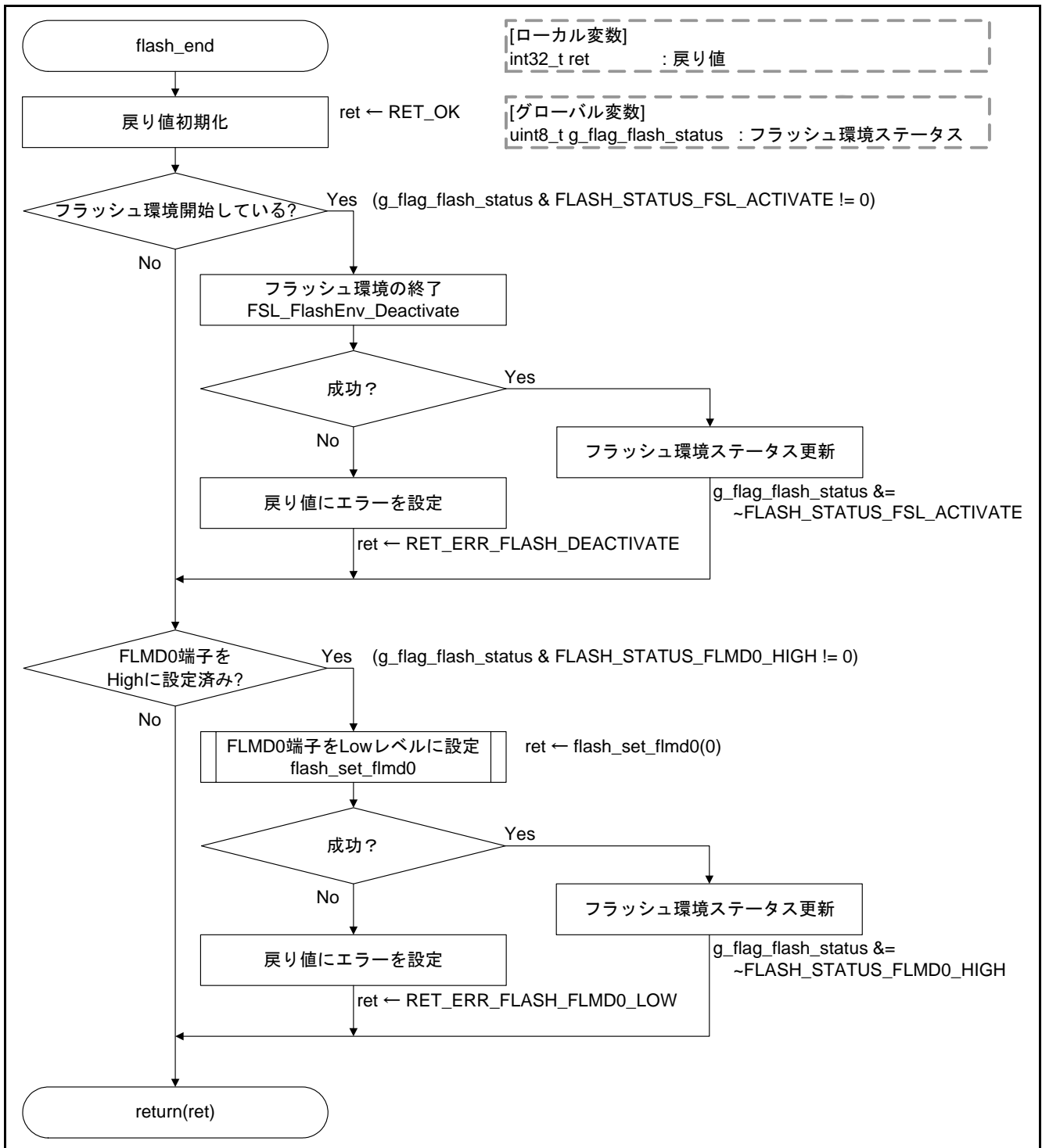


図6.18 フラッシュ環境終了処理

6.7.15 FLMD0 端子レベル設定処理

図 6.19に FLMD0 端子レベル設定処理のフローチャートを示します。

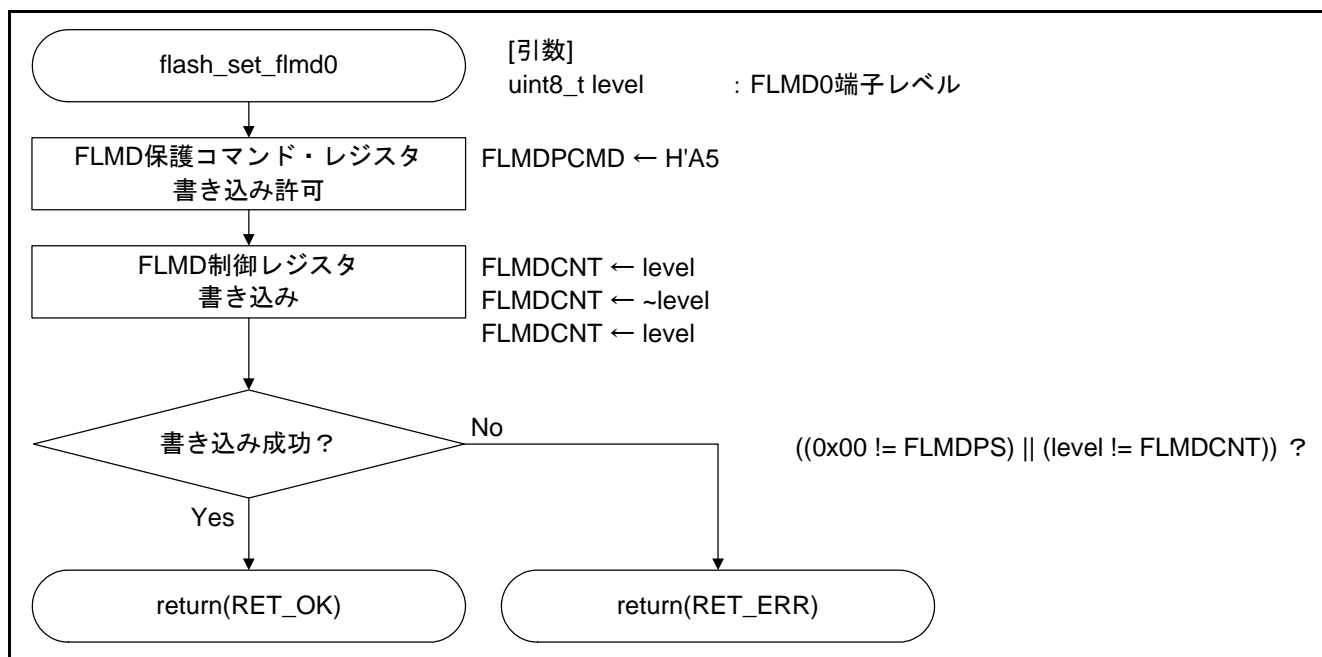


図6.19 FLMD0 端子レベル設定処理

6.7.16 受信データ格納処理

図 6.20、図 6.21 に受信データ格納処理のフローチャートを示します。

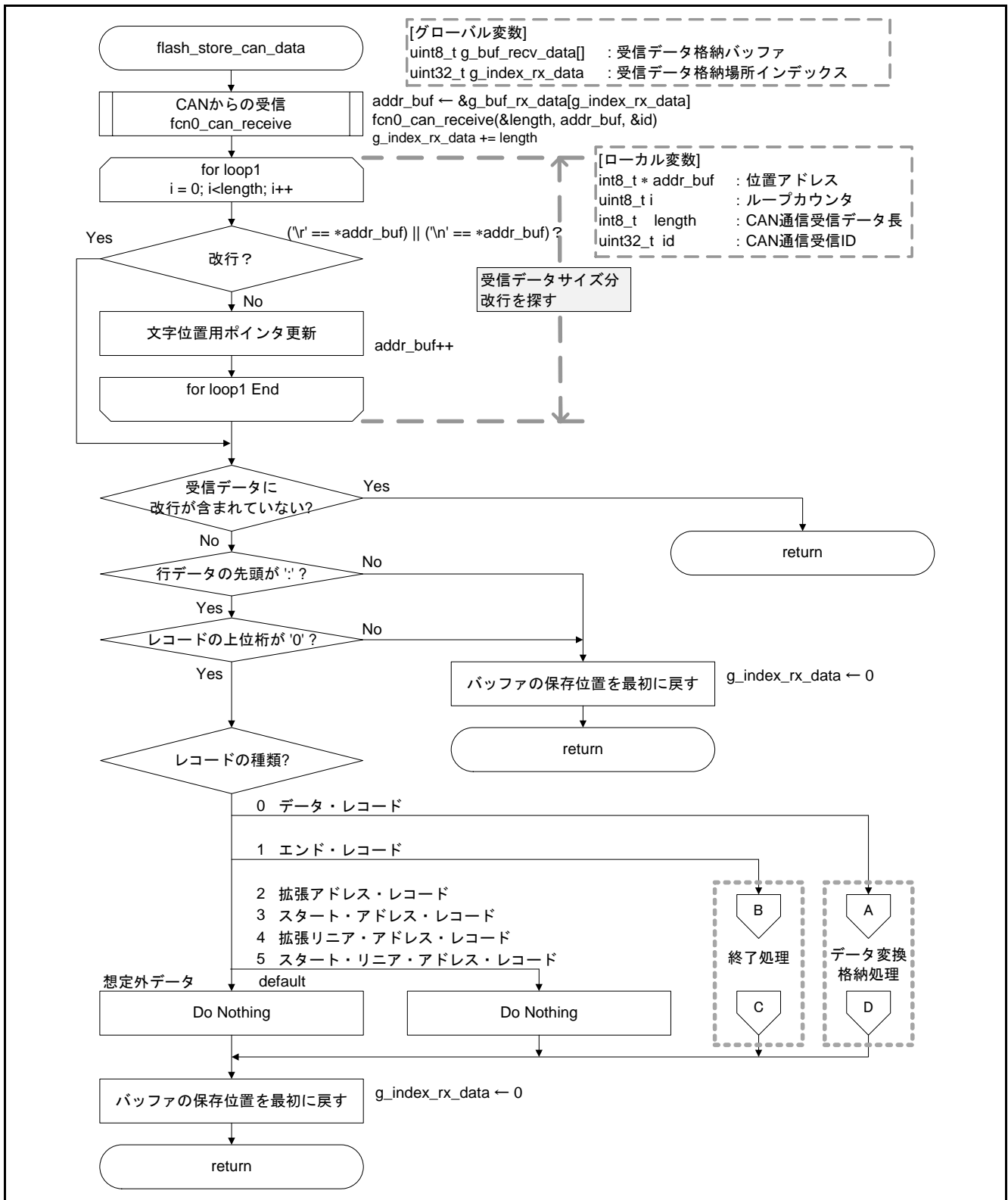


図6.20 受信データ格納処理 (1/2)

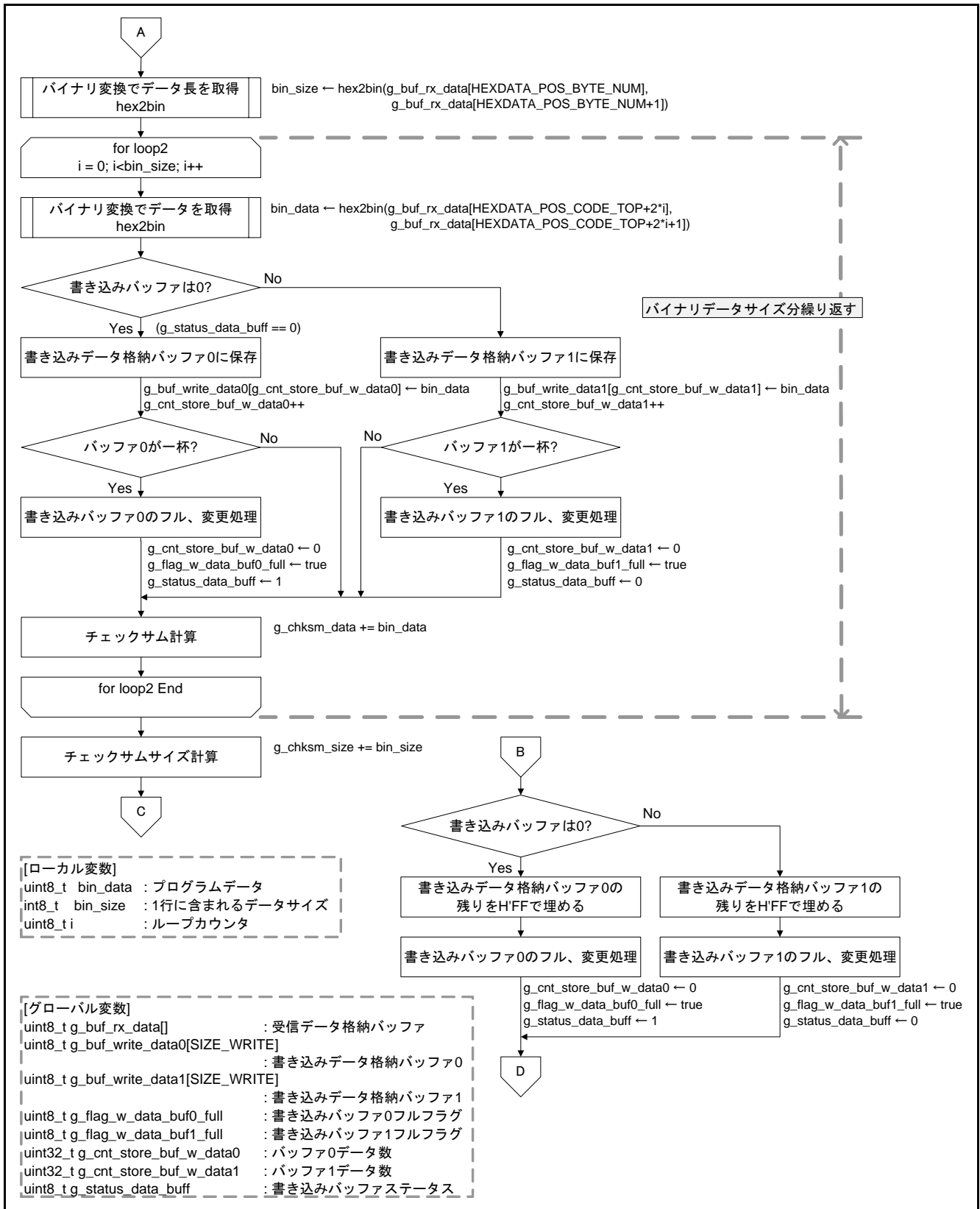


図6.21 受信データ格納処理 (2/2)

6.7.17 テキストバイナリ変換処理

図 6.22に テキストバイナリ変換処理のフローチャートを示します。

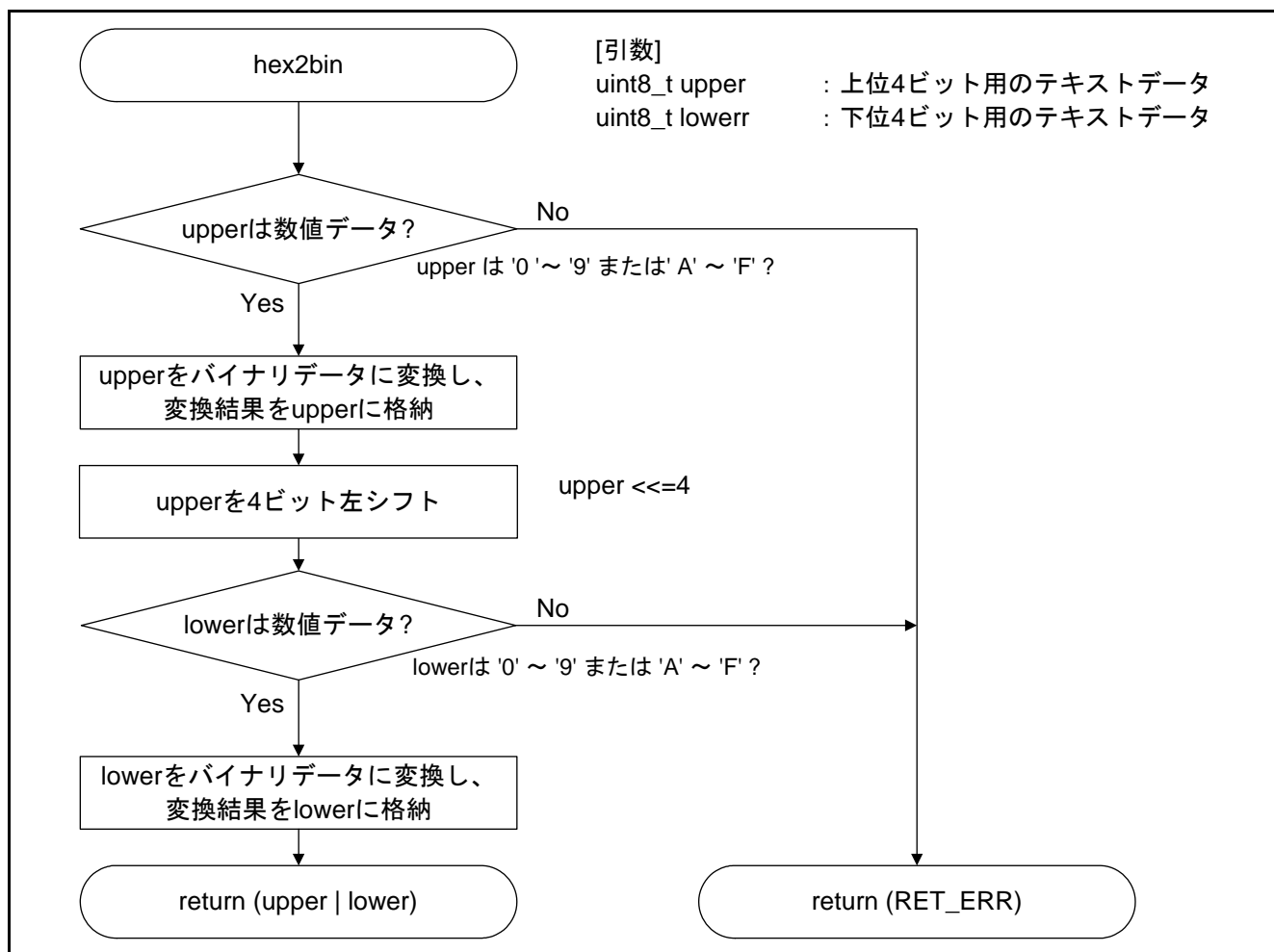


図6.22 テキストバイナリ変換処理

6.7.18 定周期 LED 点滅用 TAU0 初期化処理（書き込み領域、予備領域サンプル関数）

図 6.23に 定周期LED点滅用TAU0 初期化処理のフローチャートを示します。

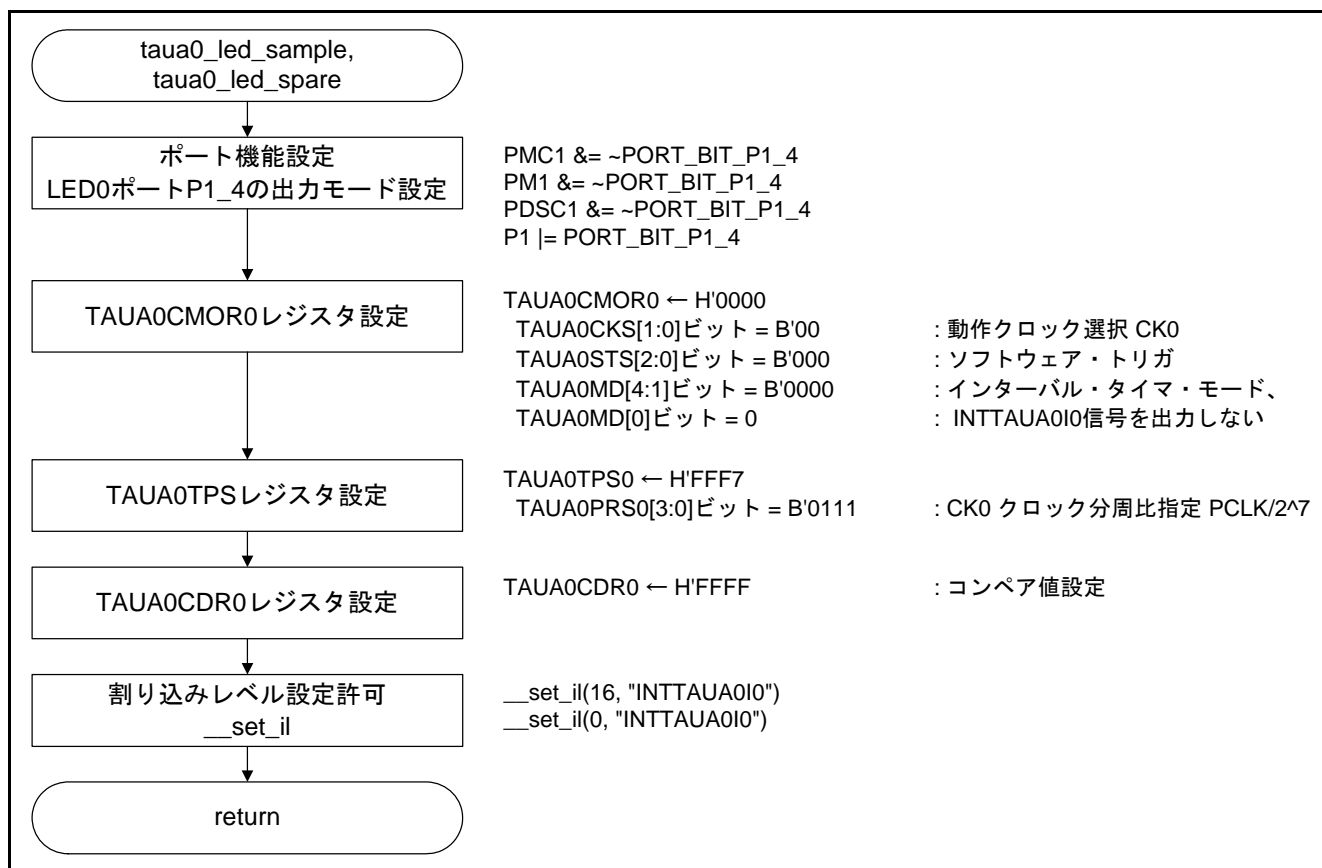


図6.23 定周期 LED 点滅用 TAU0 初期化処理

6.7.19 TAU0 インターバル・タイマ割り込み処理

図 6.24に TAU0 割り込み処理のフローチャートを示します。

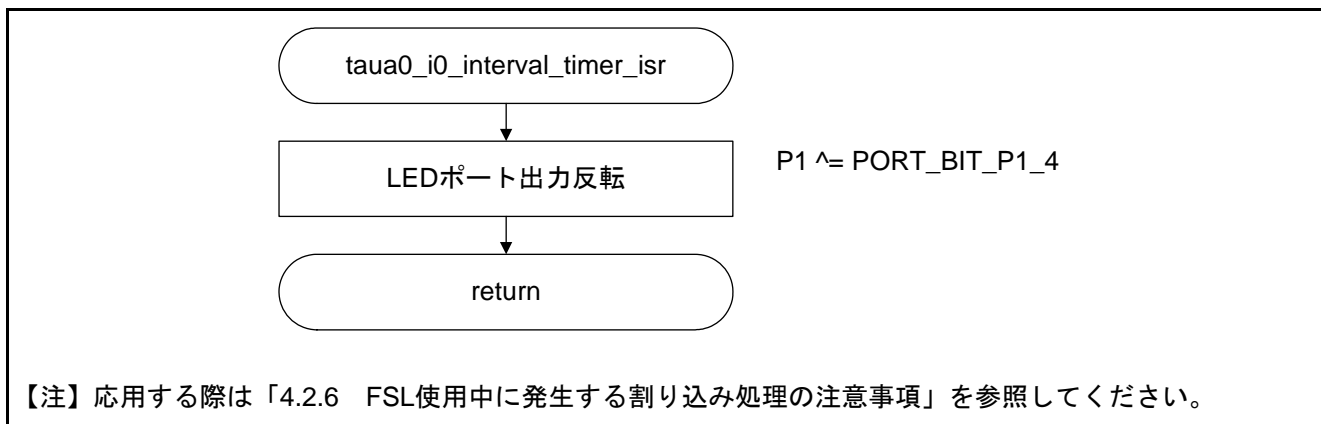


図6.24 TAU0 割り込み処理

6.7.20 CANコントローラ チャンネル0 (FCN0) 初期化処理

図 6.25に CANコントローラ チャンネル0 (FCN0) 初期化処理のフローチャートを示します。

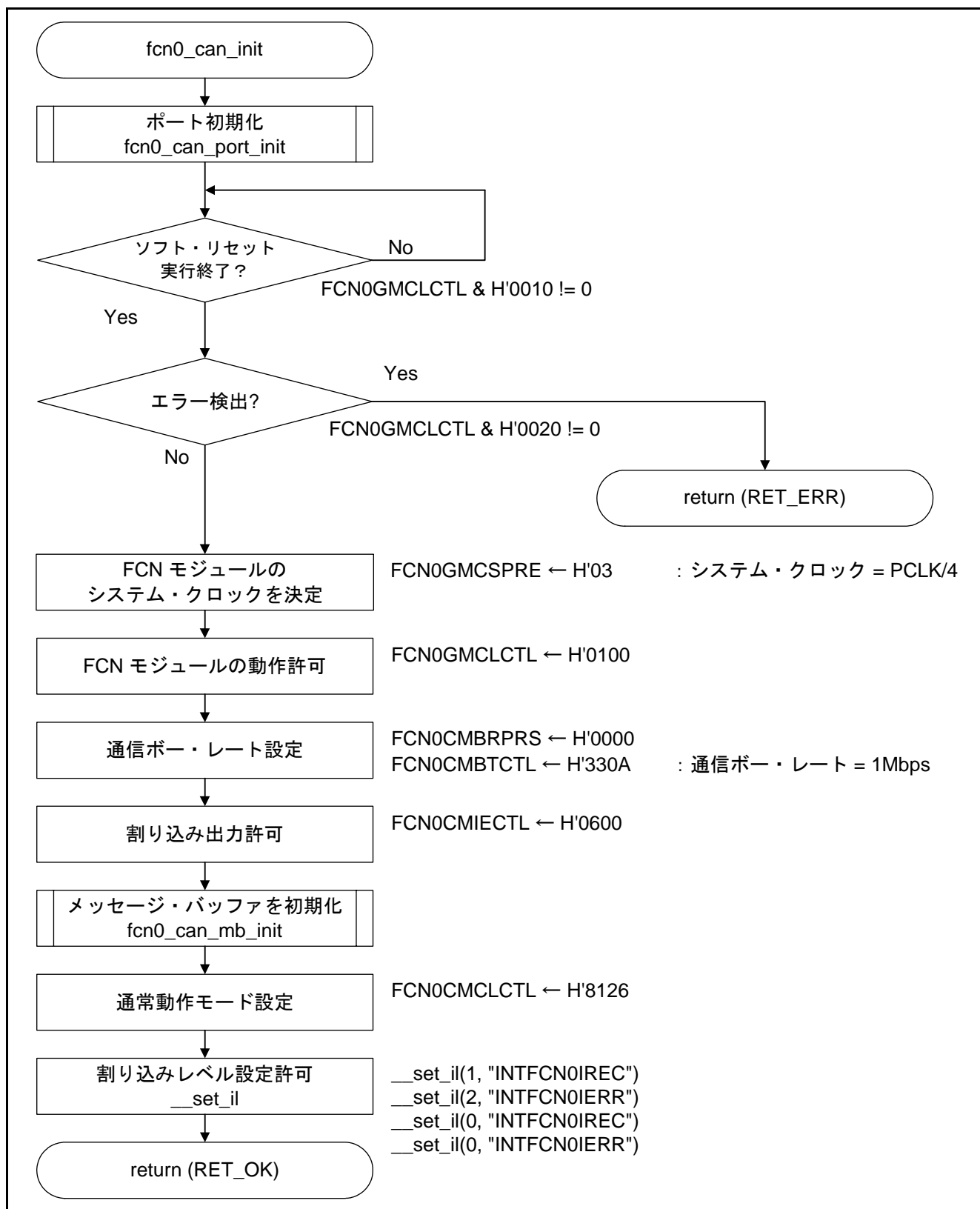


図6.25 CANコントローラ チャンネル0 (FCN0) 初期化処理

6.7.21 FCN0 ポート初期化処理

図 6.26に FCN0 ポート初期化処理のフローチャートを示します。

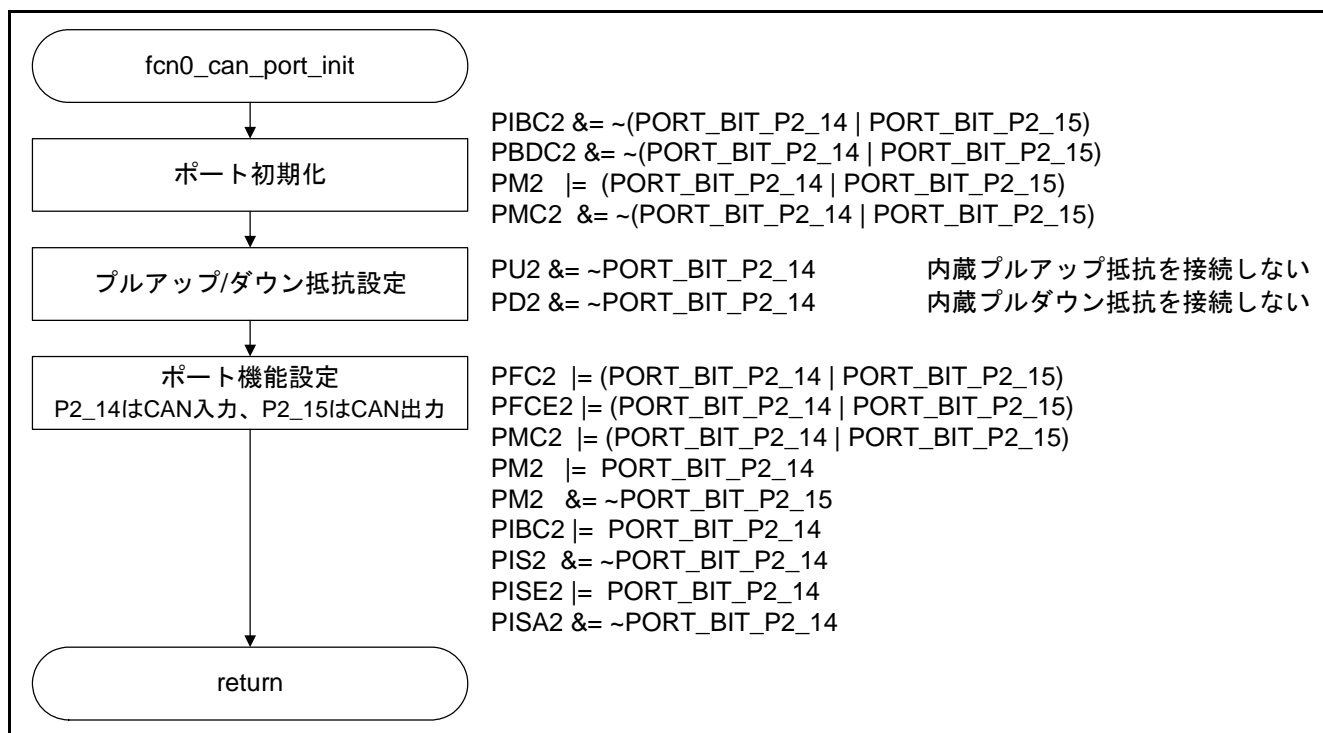


図6.26 FCN0 ポート初期化処理

6.7.22 FCN0 メッセージ・バッファ初期化処理

図 6.27、図 6.28に FCN0 メッセージ・バッファ初期化処理のフローチャートを示します。

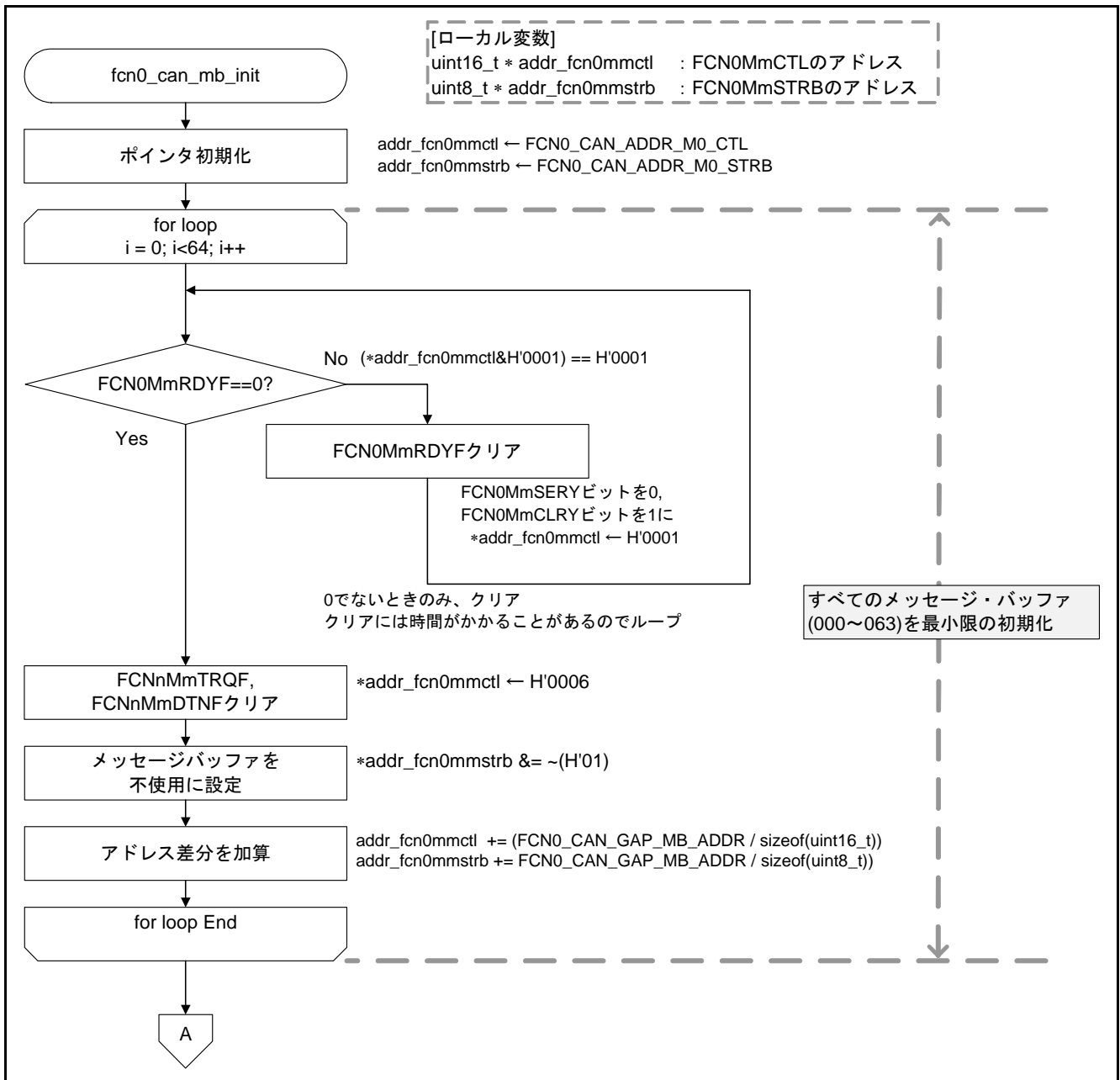


図6.27 FCN0 メッセージ・バッファ初期化処理 (1/2)

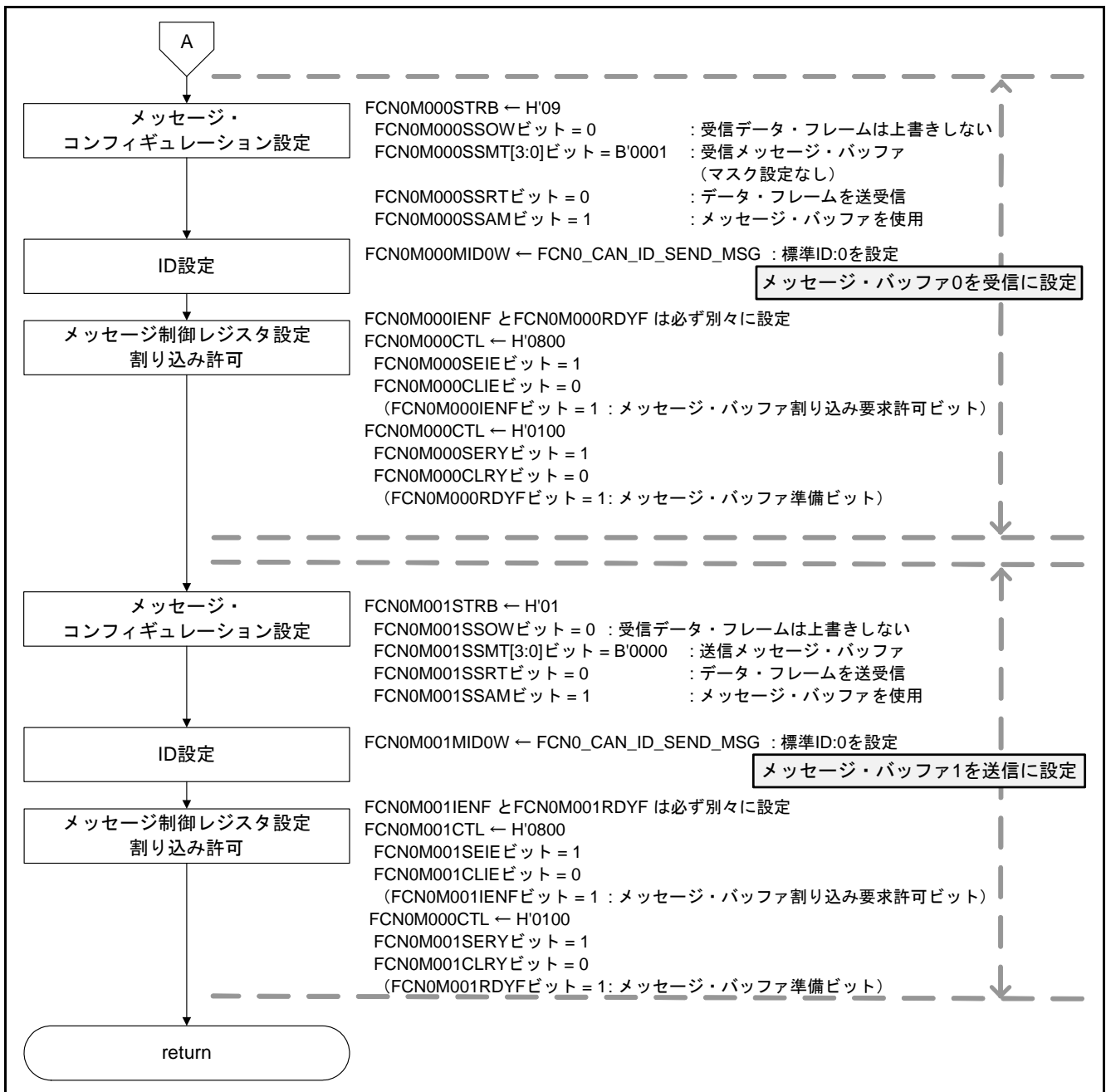


図6.28 FCN0 メッセージ・バッファ初期化処理 (2/2)

6.7.23 FCN0 メッセージ送信処理

図 6.29に FCN0 メッセージ送信処理のフローチャートを示します。

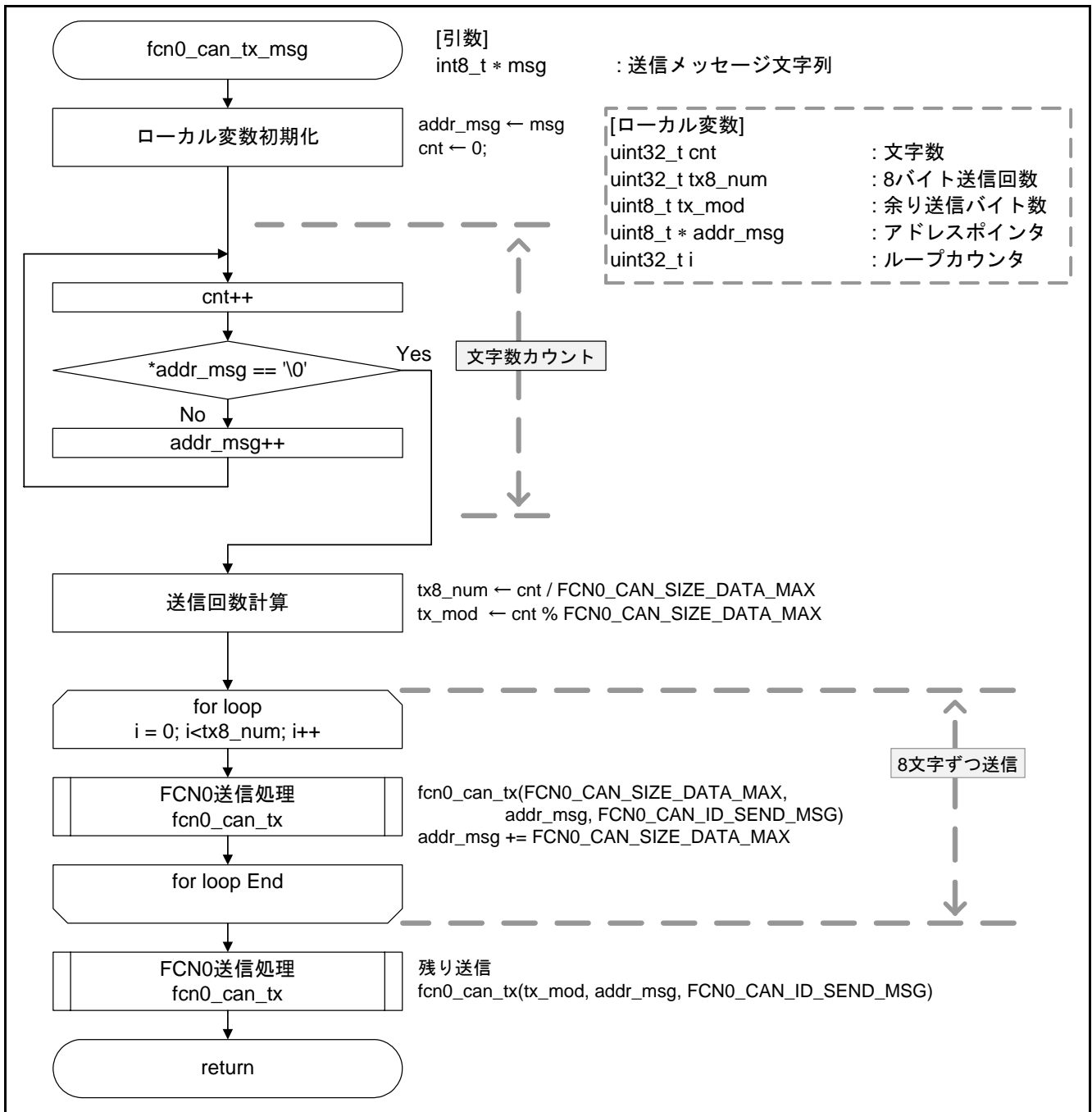


図6.29 FCN0 メッセージ送信処理

6.7.24 FCN0 送信処理

図 6.30に FCN0 送信処理のフローチャートを示します。

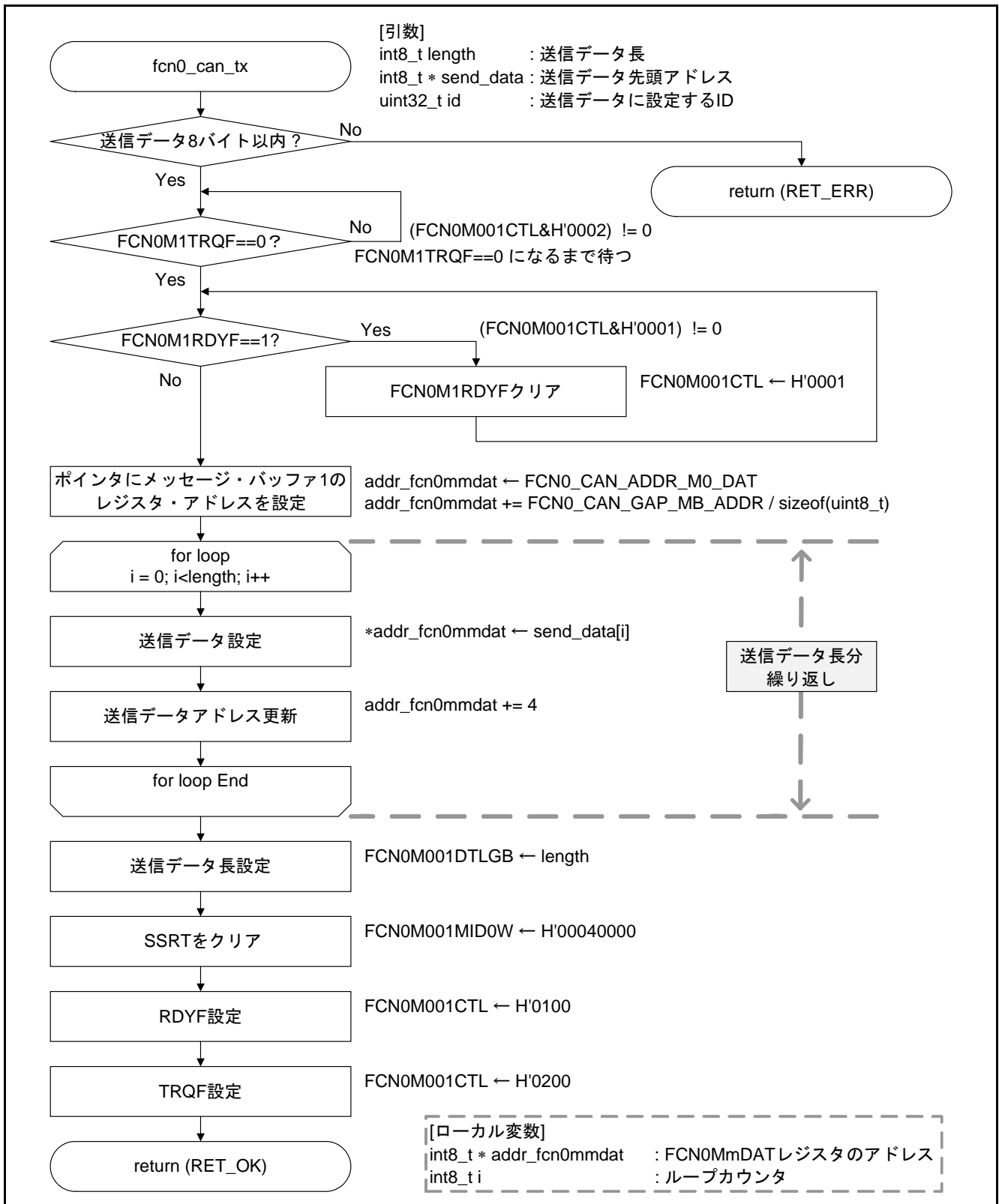


図6.30 FCN0 送信処理

6.7.25 FCN0 受信処理

図 6.31に FCN0 受信処理のフローチャートを示します。

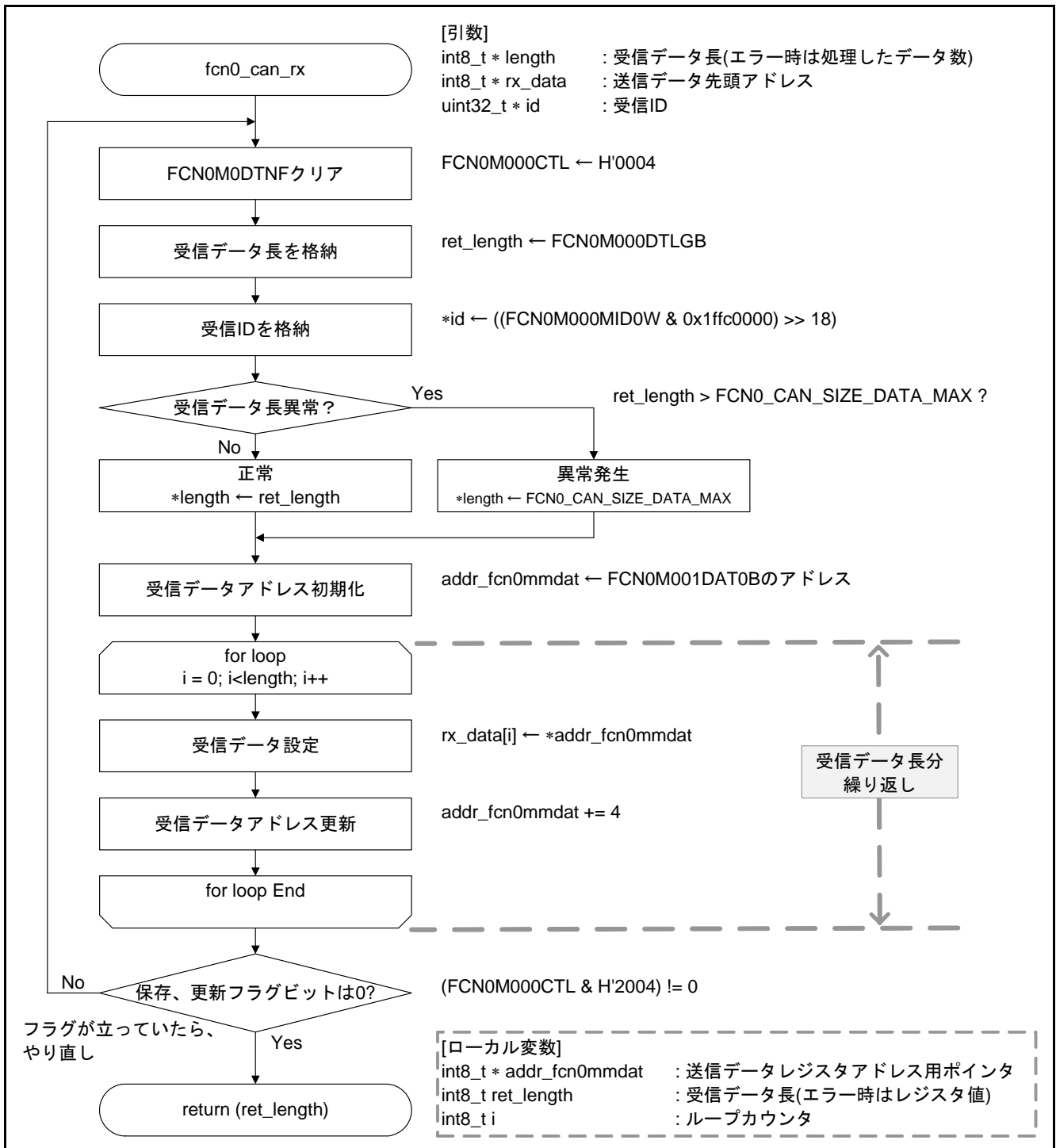


図6.31 FCN0 受信処理

6.7.26 FCN0 受信完了割り込み処理

図 6.32に FCN0 受信処理完了割り込み処理のフローチャートを示します。

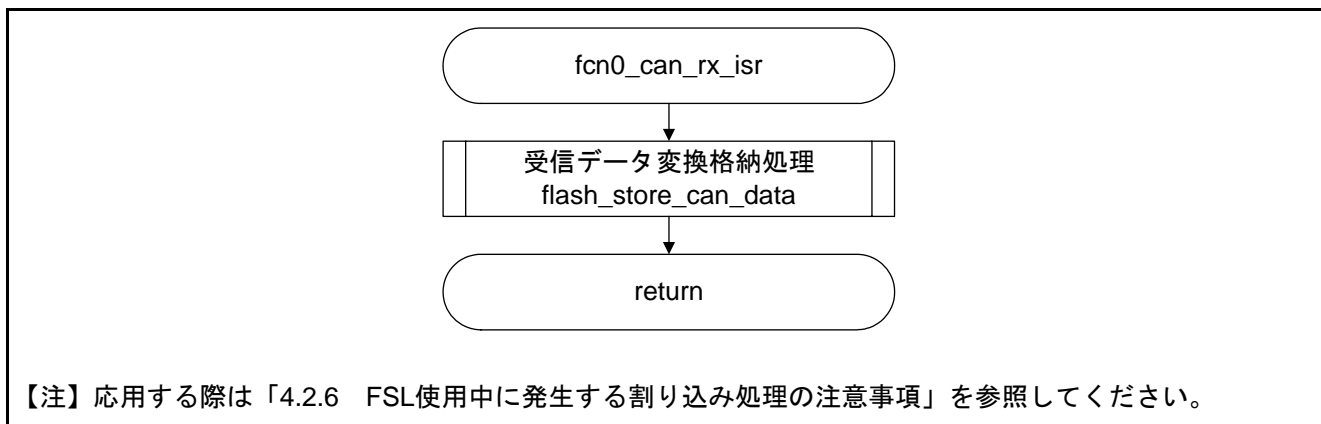


図6.32 FCN0 受信処理完了割り込み処理

6.7.27 FCN0 エラー割り込み処理

図 6.33に FCN0 エラー割り込み処理のフローチャートを示します。

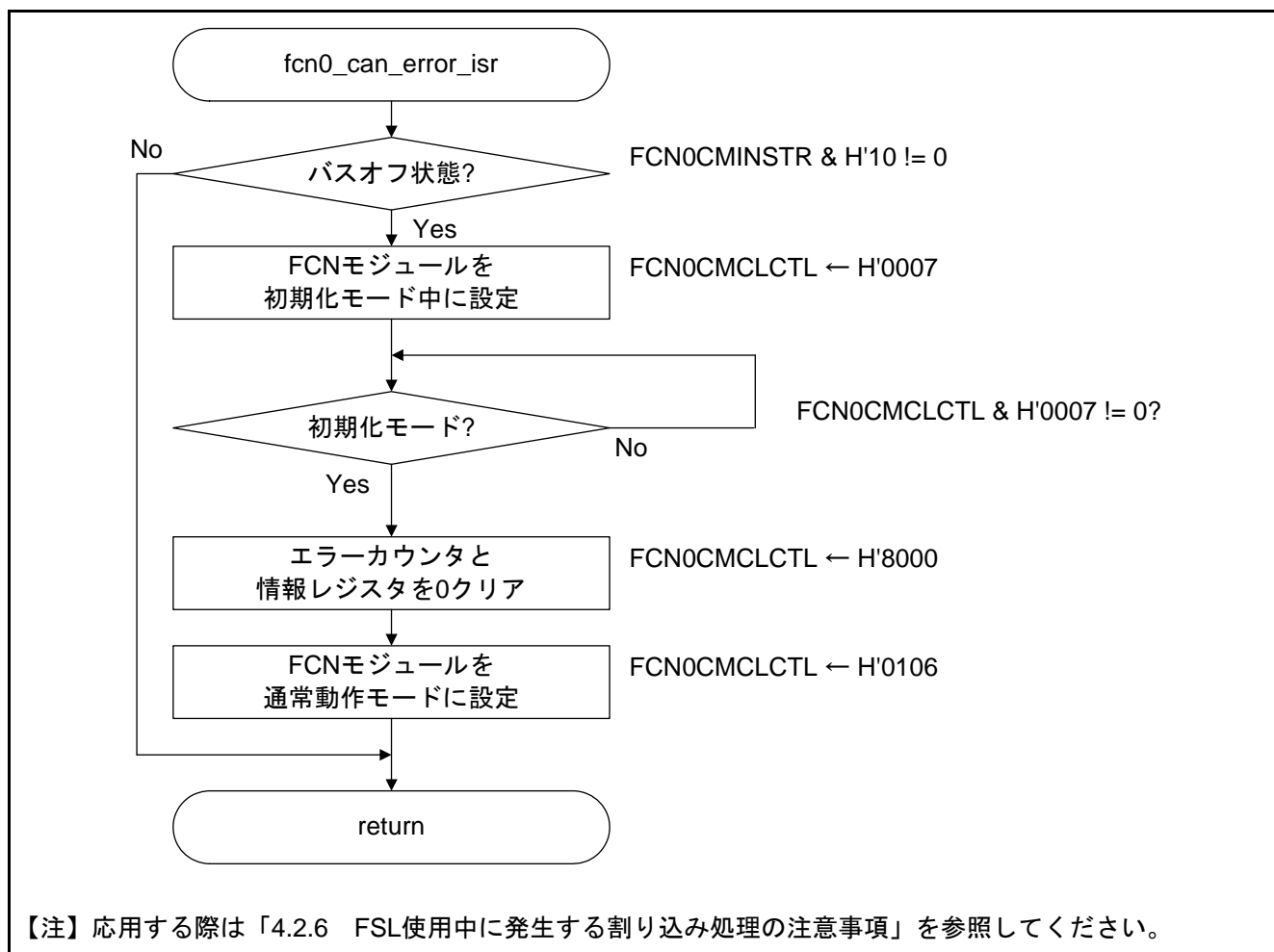


図6.33 FCN0 エラー割り込み処理

7. 操作概要

本サンプルコードでは、CAN 通信ホスト機器を使用してアップデート用のプログラムを送信します。ここでは CAN 通信ホスト機器として V850E2/ML4 CPU ボード R0K0F4022C000BR を使用し、ホスト PC で制御する例を示します。

図 7.1 に サンプルコード使用のためのハードウェア構成例を示します。本サンプルコードで動作する V850E2/ML4 と CAN 通信を行うために、もう 1 台 CPU ボードを使い、シリアル通信と CAN 通信間でデータをスルーさせます。（データをスルーさせるプログラムのロード・モジュール・ファイル (v850e2ml4_serial_can_through.lmf) は、本サンプルコードに付属しています。）

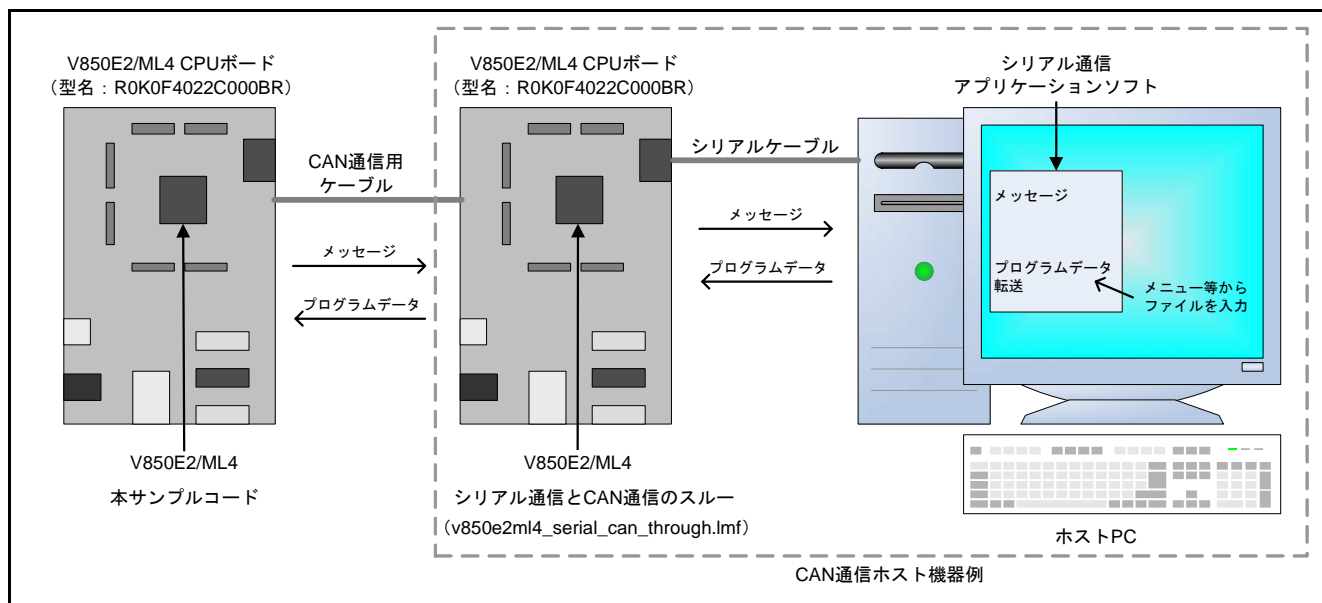


図7.1 サンプルコード使用のためのハードウェア構成例

2 台の CPU ボードは、表 7.1 に示すように、CAN を使用するために、信号選択用ジャンパ JP8 と JP10 を 2-3 側に切り替えて使用してください。また、本サンプルコードを動作させる CPU ボードは、INTP1 を使用するために、JP1 を 2-3 側へ切り替えて使用してください。

また、2 台の CPU ボード間は、CAN コネクタ (J4) 同士を CAN 通信用ケーブルで接続し、スルー用の CPU ボード (シリアルポートコネクタ (J5)) とホスト PC 間は、シリアルケーブルで接続してください。

CPU ボードのジャンパ設定とコネクタの詳細は、「V850E2/ML4 CPU ボード R0K0F4022C000BR ユーザーズマニュアル」を参照してください。

表7.1 V850E2/ML4 CPU ボードのジャンパ設定

ジャンパ名	1-2 (デフォルト)	2-3 (今回使用)
JP1*	VBUS	P2_3
JP8 (P2_14)	SDL1	CAN0RXD
JP10 (P2_15)	SDA1	CAN0TXD

【注】* 本サンプルコードを動作させる CPU ボードのみ。

上記構成例の VT100 互換端末エミュレータを使った操作例を、以下に示します。最初に、端末エミュレータを起動し、シリアルポートの接続設定を行います。端末エミュレータのシリアルポート番号はスルー用のボードと接続している番号を選択します。表 7.2 にシリアルポートの設定値を示します。

表7.2 シリアルポートの設定

項目	設定値
ビット/秒	9600bps
データビット	8bit
パリティ	なし
ストップビット	1bit
フロー制御	なし

上記の設定完了後、スルー用のボードを先に起動してから、本サンプルコードの対象ボードを起動してください。

本サンプルコードの対象ボードを起動すると、V850E2/ML4 はホストに対し「Generate INTP1 interrupt for transition to flash programming event.」というメッセージを送信します。

V850E2/ML4 はその後、書き換え領域に格納したプログラムを実行し、本サンプルコードの対象ボード上の LED を一定周期で点滅させます。

この状態で本サンプルコードの対象ボード上の INTP1 スイッチ (SW4) を押すと、V850E2/ML4 はホストに対し「-> INTP1 detected!」というメッセージを送信します。INTP1 割り込みが発生すると、V850E2/ML4 はフラッシュ書き換え処理に入り、最初に書き換え領域を消去します。消去が完了すると、V850E2/ML4 はホストに対し「Send subroutine code to update program in Intel expanded hex format.」というメッセージを送信し、ホストからのデータ受信待ち状態に入ります。

ホストからプログラムデータとしてインテル拡張ヘキサ・フォーマット形式のファイルを送信するときは、端末エミュレータの送信機能等を利用してください。該当ファイル (v850e2ml4_sample_host_send.hex 等) を選択して送信すると、スルー用のボードにシリアル通信でファイルデータが送信され、続いてスルー用のボードから本サンプルコード対象の V850E2/ML4 に CAN 通信でファイルデータが送信されます。V850E2/ML4 は受信したファイルデータをプログラムデータに変換してフラッシュ・メモリに書き込みます。

書き込み完了後、V850E2/ML4 はホストに対し「Successfully Finish Writing Program Data. Please Reset.」というメッセージを送信してリセット待ち状態に入ります。端末エミュレータ上でのメッセージ出力確認後、本サンプルコードの対象ボードを再起動してください。

本サンプルコードの対象ボードを再起動すると、ボード上の LED は前回と異なる周期で点滅します。再起動前のデータ受信/フラッシュ書き換え (アップデート) が正常に実施できなかった場合、V850E2/ML4 は再起動後にチェックサムエラーと判定します。この場合、V850E2/ML4 は予備領域のプログラムを実行します。

8. サンプルコード

サンプルコードは、ルネサス エレクトロニクスホームページから入手してください。

9. 参考ドキュメント

- ユーザーズマニュアル：ハードウェア
V850E2/ML4 ユーザーズマニュアル ハードウェア編 Rev.2.00 (R01UH0262JJ)
(最新版をルネサス エレクトロニクスホームページから入手してください。)
- テクニカルアップデート／テクニカルニュース
(最新の情報をルネサス エレクトロニクスホームページから入手してください。)
- ユーザーズマニュアル：開発環境
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル コーディング編 (CX コンパイラ) Rev.1.00 (R20UT2139JJ)
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル ビルド編 (CX コンパイラ) Rev.1.00 (R20UT2142JJ)
V850E2/ML4 CPU ボード R0K0F4022C000BR ユーザーズマニュアル Rev.1.00 (R20UT0778JJ)
(最新版をルネサス エレクトロニクスホームページから入手してください。)
- ユーザーズマニュアル：ソフトウェア
V850E2M ユーザーズマニュアル アーキテクチャ編 Rev.1.00 (R01US0001JJ)
(最新版をルネサス エレクトロニクスホームページから入手してください。)

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>
- お問い合わせ先
<http://japan.renesas.com/contact/>

改訂記録	V850E2/ML4 アプリケーションノート CANコントローラを使用したフラッシュ・セルフ・プログラミングによるプログラムアップデート例
------	---

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2013.03.01	—	初版発行

すべての商標および登録商標は、それぞれの所有者に帰属します。

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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

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. リザーブアドレスのアクセス禁止

【注意】リザーブアドレス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、事前に問題ないことをご確認下さい。

同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

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、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情報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と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 エレクトロニクス販売株式会社 〒100-0004 千代田区大手町 2-6-2（日本ビル）

■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/>