V850E2/ML4

イーサネット・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

R01AN1480JJ0100
Rev.1.00
2013.08.06

要旨

本アプリケーションノートでは、V850E2/ML4のイーサネット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。

- イーサネット通信により受信したインテル拡張ヘキサ・フォーマット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

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。

ホスト PC とのイーサネット通信により、インテル拡張ヘキサ・フォーマット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
イーサネット・コントローラ	書き換えデータ／メッセージ通信

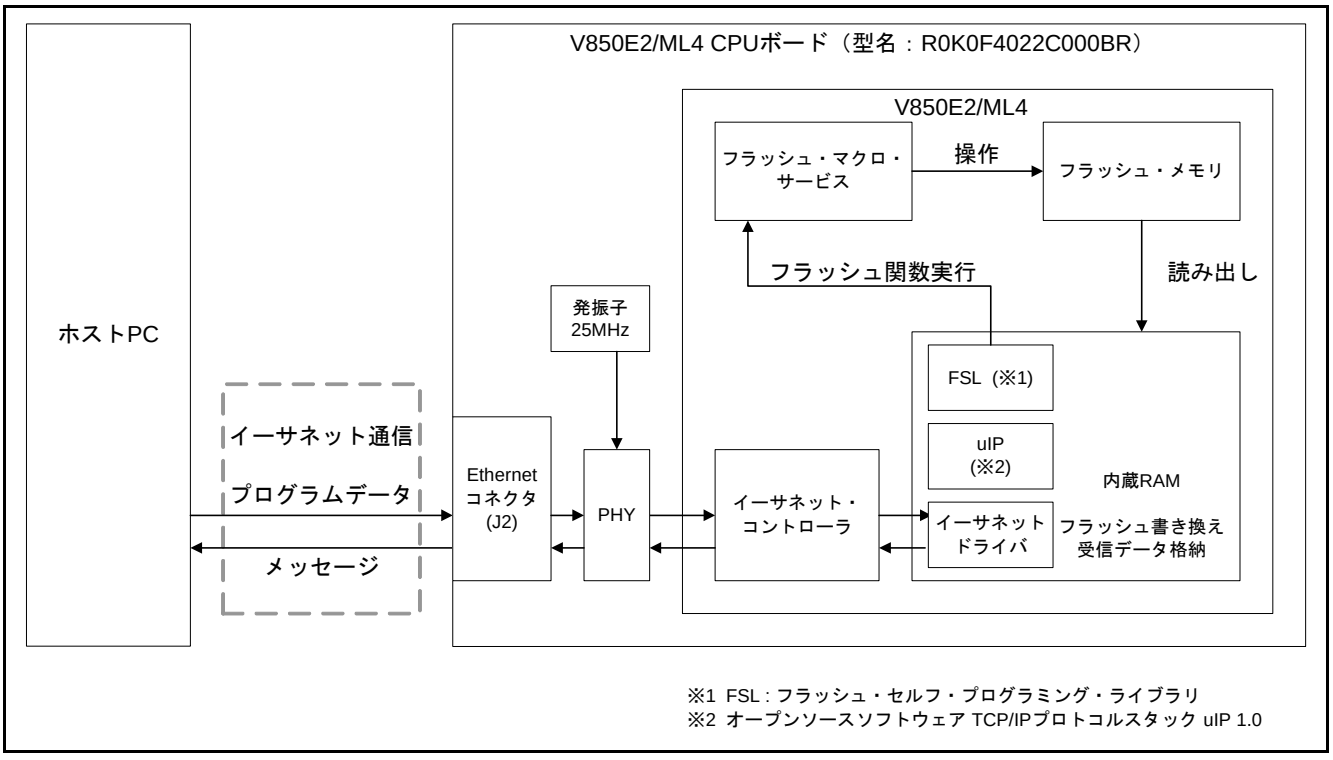


図1.1　システム構成図

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 -Xobj_path=DefaultBuild -g -Xpro_epi_runtime=off -IC:¥WorkSpace¥v850e2ml4_flash_update_ether¥inc -IC:¥WorkSpace¥v850e2ml4_flash_update_ether¥src -IC:¥WorkSpace¥v850e2ml4_flash_update_ether¥src¥FSL -IC:¥WorkSpace¥v850e2ml4_flash_update_ether¥src¥uip-1.0¥uip -Xdef_var -Xfar_jump=v850e2ml4_flash_update_ether.fjp -Xlink_directive=v850e2ml4_flash_update_ether.dir -Xstartup=DefaultBuild¥cstart.obj +Xide -Xmap=DefaultBuild¥v850e2ml4_flash_update_ether.map -lFSL_T05_REC_R32 -LC:¥WorkSpace¥v850e2ml4_flash_update_ether¥src¥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=INTOSTM0RAM.text -Xrompsec_text=INTP2RAM.text -Xrompsec_text=INTTAUA0I0RAM.text -Xrompsec_text=INTETHA0SRXRAM.text -Xrompsec_text=INTETHA0SCRXRAM.text -Xrompsec_text=INTETHA0SCTXRAM.text -Xrompsec_text=INTETHA0RSRAM.text -Xrompsec_text=INTETHA0TSRAM.text -Xrompsec_text=INTETHA0FSRAM.text -Xrompsec_text=INTETHA0MACRAM.text -Xhex=DefaultBuild¥v850e2ml4_flash_update_ether.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
使用ツール	VT100 互換端末エミュレータ（端末エミュレータソフト）
その他	ホスト PC

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）
- V850E2/ML4　イーサネット送受信設定例（R01AN1018JJ）

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 レジスタの退避、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 コンパイラ）」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（INTP2RAM.text、INTTAUA0I0RAM.text、INTTAUA0I0RAM.text、INTETHA0SRXRAM.text、INTETHA0SCRXRAM.text、INTETHA0SCTXRAM.text、INTETHA0RSRAM.text、INTETHA0TSRAM.text、INTETHA0FSRAM.text、INTETHA0MACRAM.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 B000 に設定しています。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に設定しています。

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。

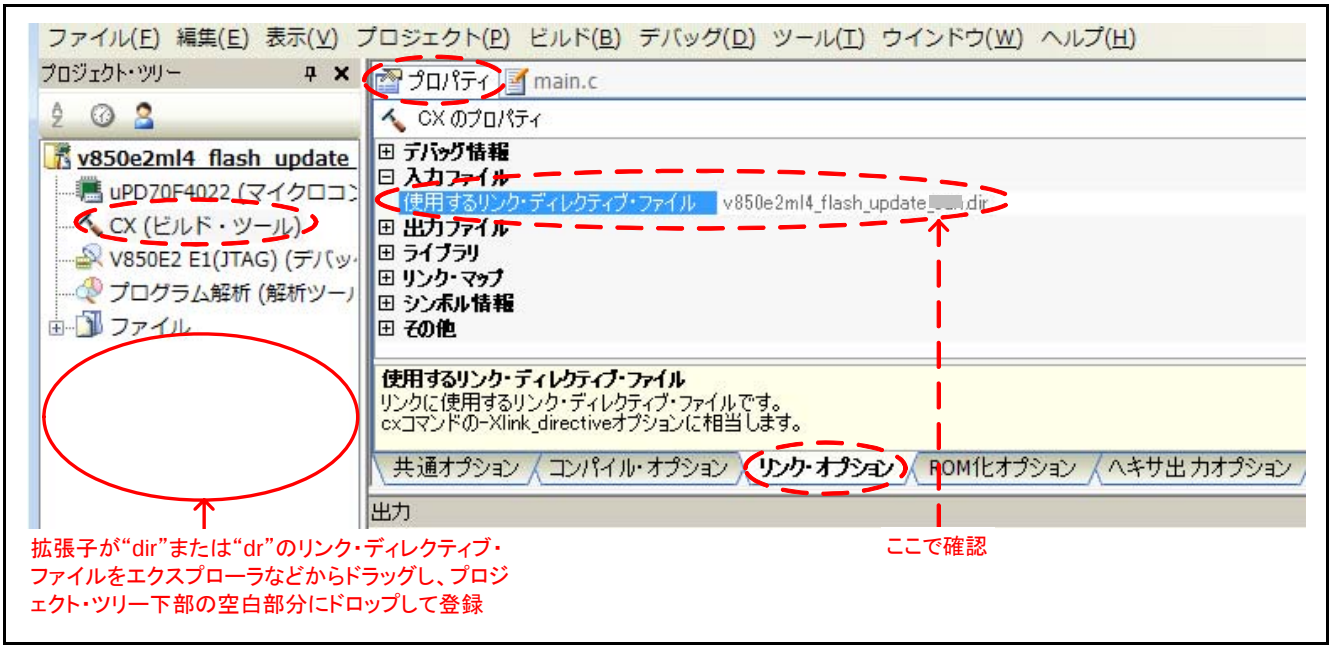


図4.1　リンク・ディレクティブ・ファイルの登録箇所

```
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 V0x0000a000 {
SparePRG.text = $PROGBITS ?AX V0x0000a000 SparePRG.text ;

};

# 書き換え領域
MasterPRG:!LOAD ?RX V0x0000b000 {
MasterPRG.text = $PROGBITS ?AX V0x0000b000 MasterPRG.text ;

};

ETH_MEMORY:!LOAD ?RW V0xf9800000 L0x00010000 {
eth_memory.data = $PROGBITS ?AW eth_memory.data ;
eth_memory.bss = $NOBITS ?AW eth_memory.bss ;

};
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 V0xfedf8000 {
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 {
INTOSTM0RAM.text = $PROGBITS ?AX V0xfedfe140 H0x0000000a INTOSTM0RAM.text ;
INTP2RAM.text = $PROGBITS ?AX V0xfedfe180 H0x0000000a INTP2RAM.text ;
INTTAUA0I0RAM.text = $PROGBITS ?AX V0xfedfe3b0 H0x0000000a INTTAUA0I0RAM.text ;
INTETHA0SRXRAM.text = $PROGBITS ?AX V0xfedfeba0 H0x0000000a INTETHA0SRXRAM.text ;
INTETHA0SCRXRAM.text = $PROGBITS ?AX V0xfedfebb0 H0x0000000a INTETHA0SCRXRAM.text ;
INTETHA0SCTXRAM.text = $PROGBITS ?AX V0xfedfebc0 H0x0000000a INTETHA0SCTXRAM.text ;
INTETHA0RSRAM.text = $PROGBITS ?AX V0xfedfebd0 H0x0000000a INTETHA0RSRAM.text ;
INTETHA0TSRAM.text = $PROGBITS ?AX V0xfedfebe0 H0x0000000a INTETHA0TSRAM.text ;
INTETHA0FSRAM.text = $PROGBITS ?AX V0xfedfebf0 H0x0000000a INTETHA0FSRAM.text ;
INTETHA0MACRAM.text = $PROGBITS ?AX V0xfedfec00 H0x0000000a INTETHA0MACRAM.text ;

};

__tp_TEXT@ %TP_SYMBOL ;
__gp_DATA@ %GP_SYMBOL &__tp_TEXT { ETH_MEMORY 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。

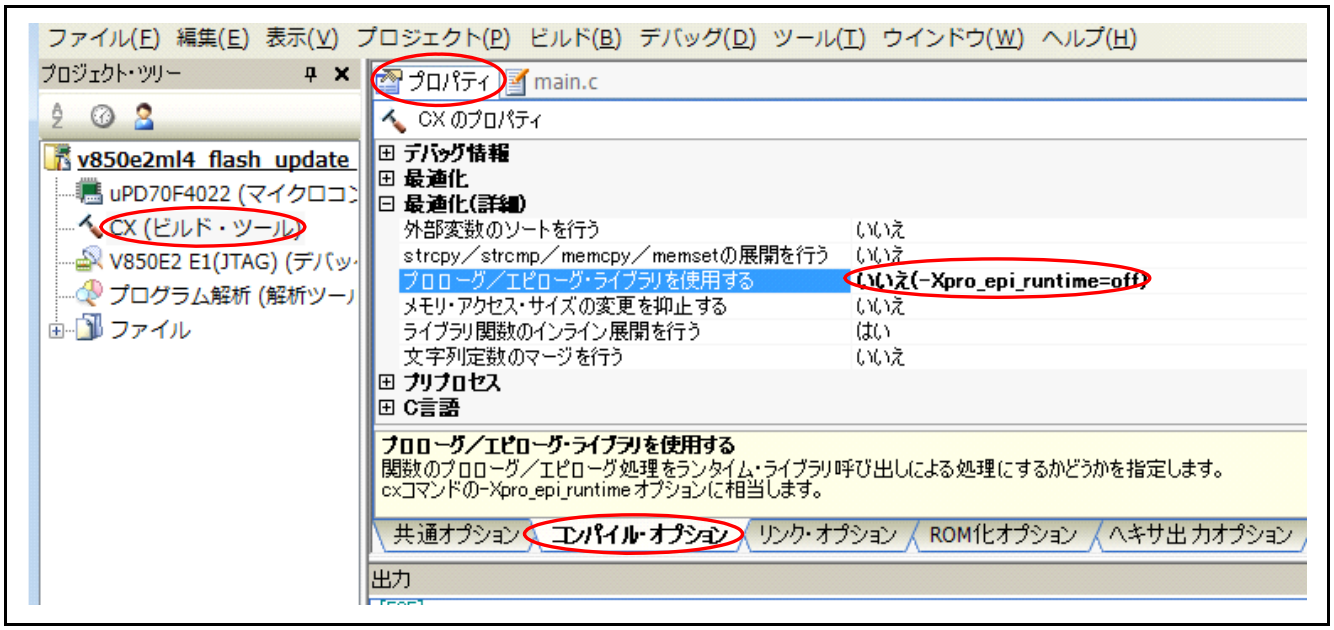


図4.3　プロローグ／エピローグ・ライブラリ不使用の設定箇所

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。

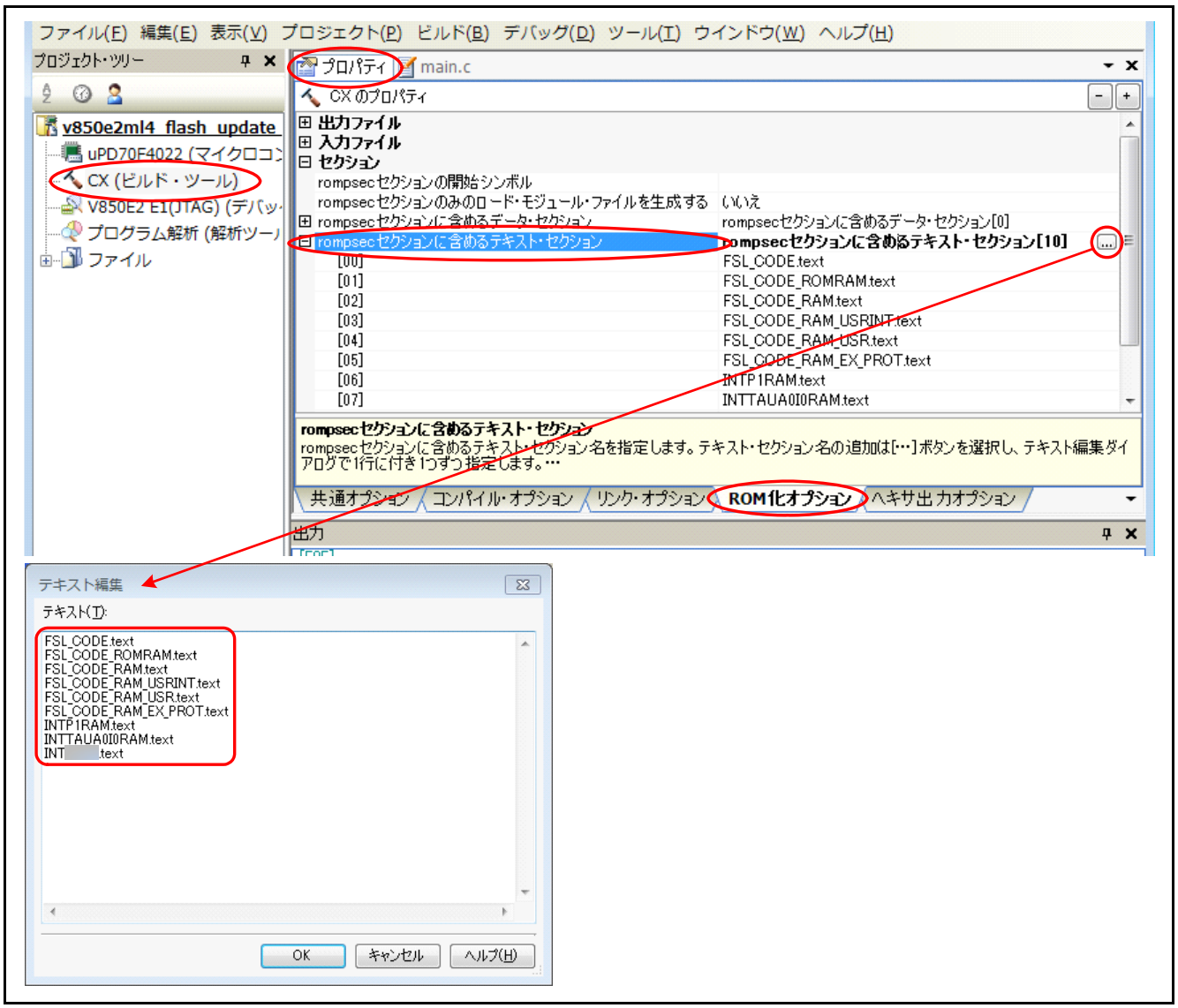


図4.4　RAM 上に配置するセクションの ROM 化設定登録箇所

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ーズマニュアル 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。

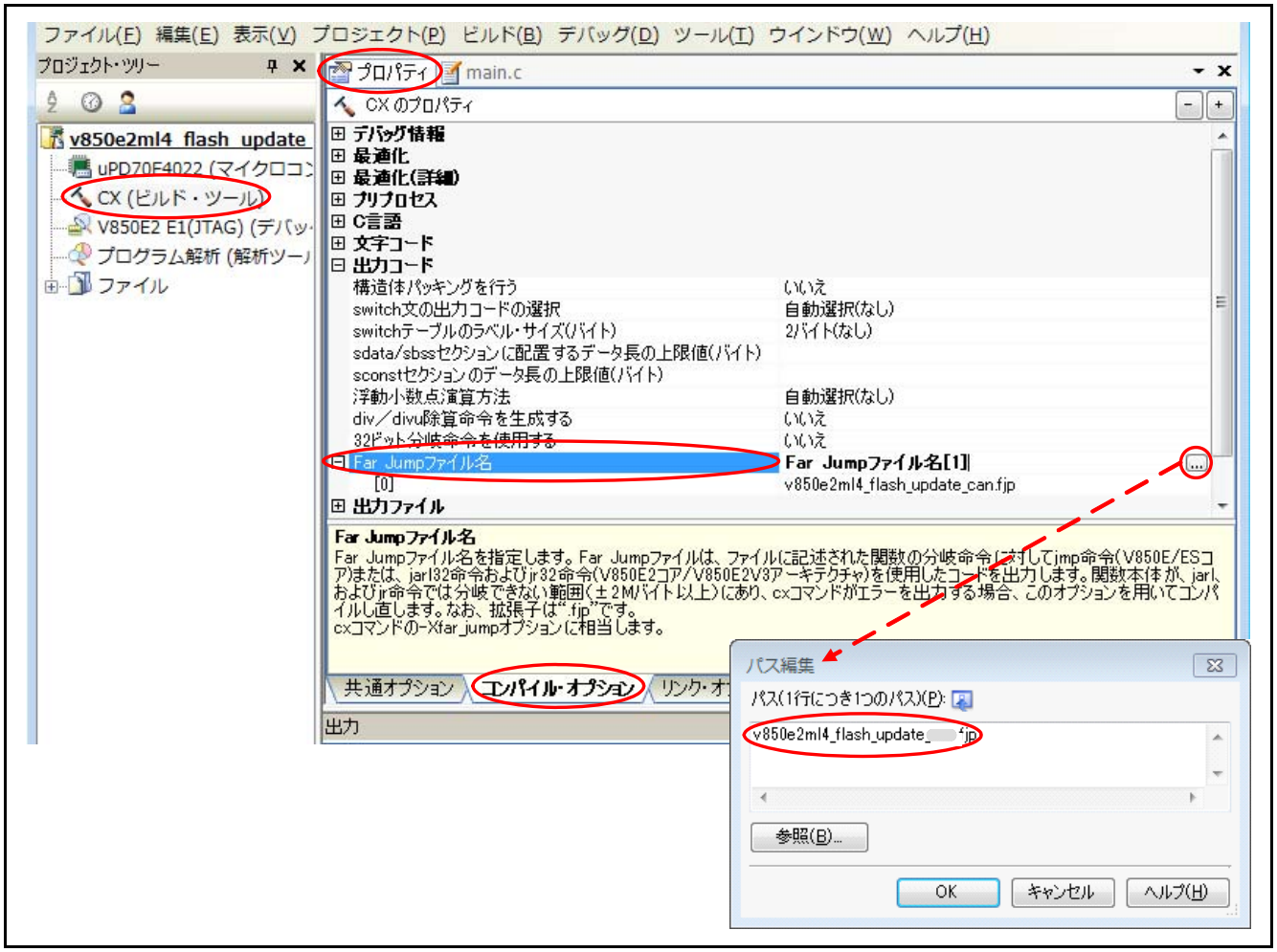


図4.5　far jump 呼び出し関数一覧ファイルの登録箇所

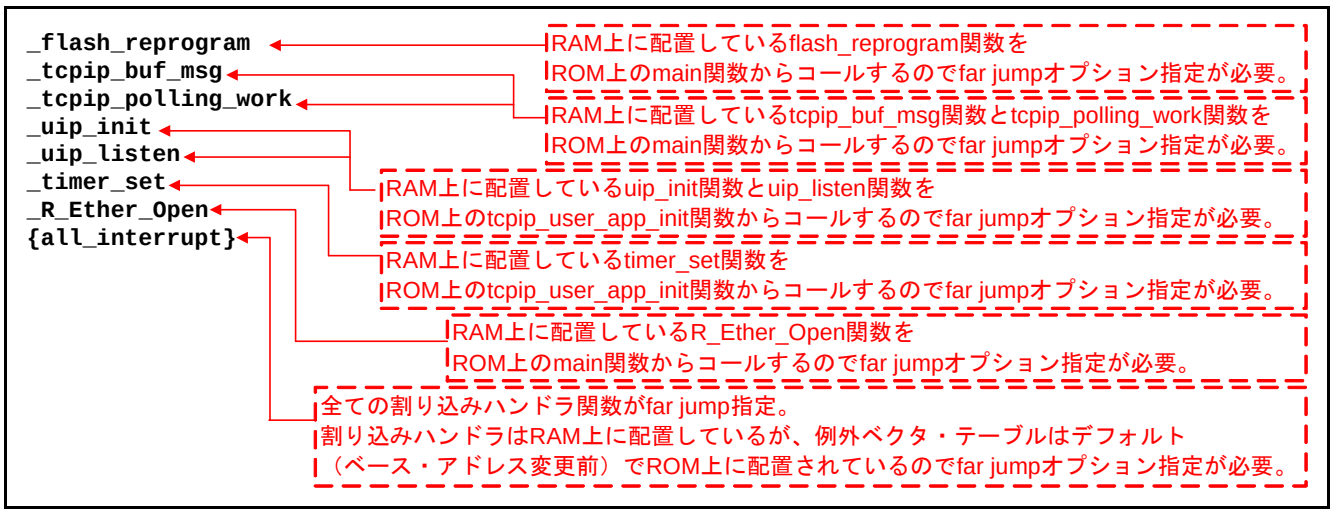


図4.6　far jump 呼び出し関数一覧ファイル作成例

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。

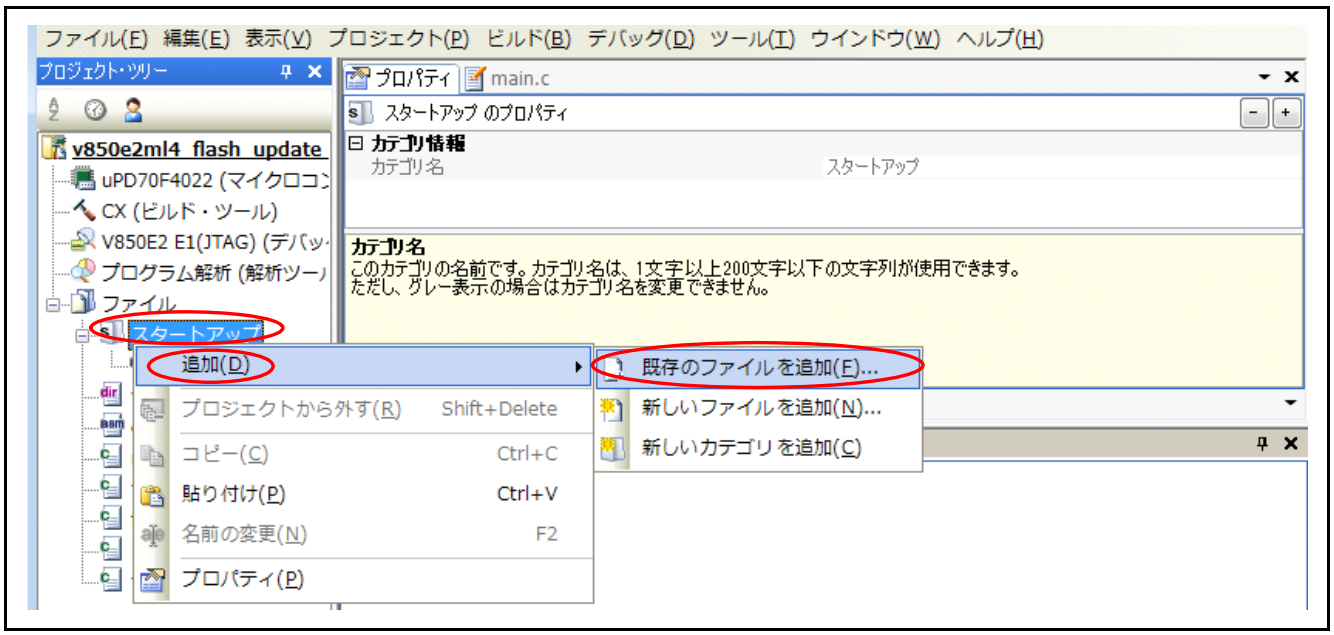


図4.7　スタートアップ・ルーチンの登録箇所

```
                                        :
                                        : (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    __zeroclrw, lp

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

        mov32   0xfedf8000, r6          ;  clear ram_prog section for e2core prefetch processing
        mov32   0xfedfffff, r7
        jarl    __zeroclrw, lp

                                        :
                                        : (続く)
                                        :
```

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：
  - .sedata
  - .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	
P7_1	入出力	MII の PHY 制御用データに使用	PHY の MDIO 端子に接続*
P7_0	出力	MII の PHY 制御用 データクロックに使用	PHY の MDC 端子に接続*
P6_15/ETH_RXD3	入力	MII の受信データに使用	PHY の P0RXD3/GPIO7 端子に接続*
P6_14/ETH_RXD2	入力	MII の受信データに使用	PHY の P0RXD2/GPIO8 端子に接続*
P6_13/ETH_RXD1	入力	MII の受信データに使用	PHY の P0RXD1 端子に接続*
P6_12/ETH_RXD0	入力	MII の受信データに使用	PHY の P0RXD0 端子に接続*
P6_11/ETH_RXDV	入力	MII の受信データ有効に使用	PHY の P0RXDV 端子に接続*
P6_10/ETH_RXCLK	入力	MII の受信クロックに使用	PHY の P0RXCLK/GPIO3 端子に接続*
P6_9/ETH_RXER	入力	MII の受信エラーに使用	PHY の P0RXERR 端子に接続*
P6_8/ETH_TXER	出力	MII の送信エラーに使用	PHY の P0TXERR/GPIO9 端子に接続*
P6_7/ETH_TXCLK	入力	MII の送信クロックに使用	PHY の P0TXCLK 端子に接続*
P6_6/ETH_TXEN	出力	MII の送信イネーブルに使用	PHY の P0TXEN 端子に接続*
P6_5/ETH_TXD0	出力	MII の送信データに使用	PHY の P0TXD0 端子に接続*
P6_4/ETH_TXD1	出力	MII の送信データに使用	PHY の P0TXD1 端子に接続*
P6_3/ETH_TXD2	出力	MII の送信データに使用	PHY の P0TXD2/GPIO11 端子に接続*
P6_2/ETH_TXD3	出力	MII の送信データに使用	PHY の P0TXD3/GPIO10 端子に接続*
P6_1/ETH_COL	入力	MII の衝突検出に使用	PHY の P0COLSD/GPIO19 端子に接続*
P6_0/ETH_CRS	入力	MII のキャリア検出に使用	PHY の P0CRS/GPIO6 端子に接続*
P2_7/INTP2	入力	INTP2 割り込み	

【注】* 本サンプルコードでは、ルネサスイーサネットデバイスの標準ルネサス API（RAPI）を使用してイーサネット・コントローラの制御を行っています。イーサネット・コントローラと PHY の制御については、アプリケーションノート「V850E2/ML4　イーサネット送受信設定例」を参照してください。

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、イーサネット通信でインテル拡張ヘキサ・フォーマット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、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	intp2_isr、taua0_ch0_interval_timer_isr、ostm0_isr
FSL_CODE_RAM_USR.text	RAM 実行用ユーザ・プログラム・セクション	flash_reprogram、flash_init、flash_activate、flash_modecheck、flash_erase、flash_write、flash_iverify、flash_end、flash_set_flmd0、flash_store_ether_data、flash_store_line_data、hex2bin、tcpip_user_appcall、tcpip_buf_msg、tcpip_tx_bufdata、tcpip__polling_work、onram_memcpy、onram_memcmp、onram_memset、clock_time、イーサネット・サンプルドライバの関数*、uIP の関数
INTOSTM0RAM.text、 INTP2RAM.text、 INTTAUA0I0RAM.text、 INTETHA0SRXRAM.text、 INTETHA0SCRXRAM.text、 INTETHA0SCTXRAM.text、 INTETHA0RSRAM.text、 INTETHA0TSRAM.text、 INTETHA0FSRAM.text、 INTETHA0MACRAM.text	割り込みハンドラ関数へのジャンプ命令	なし

【注】*ただし、イーサネット・サンプルドライバ使用のイーサネット・コントローラ割り込み関数を含みます。

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 B000（11）	taua0_led_sample	MasterPRG.text
予備領域	H'0000 A000（10）	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。

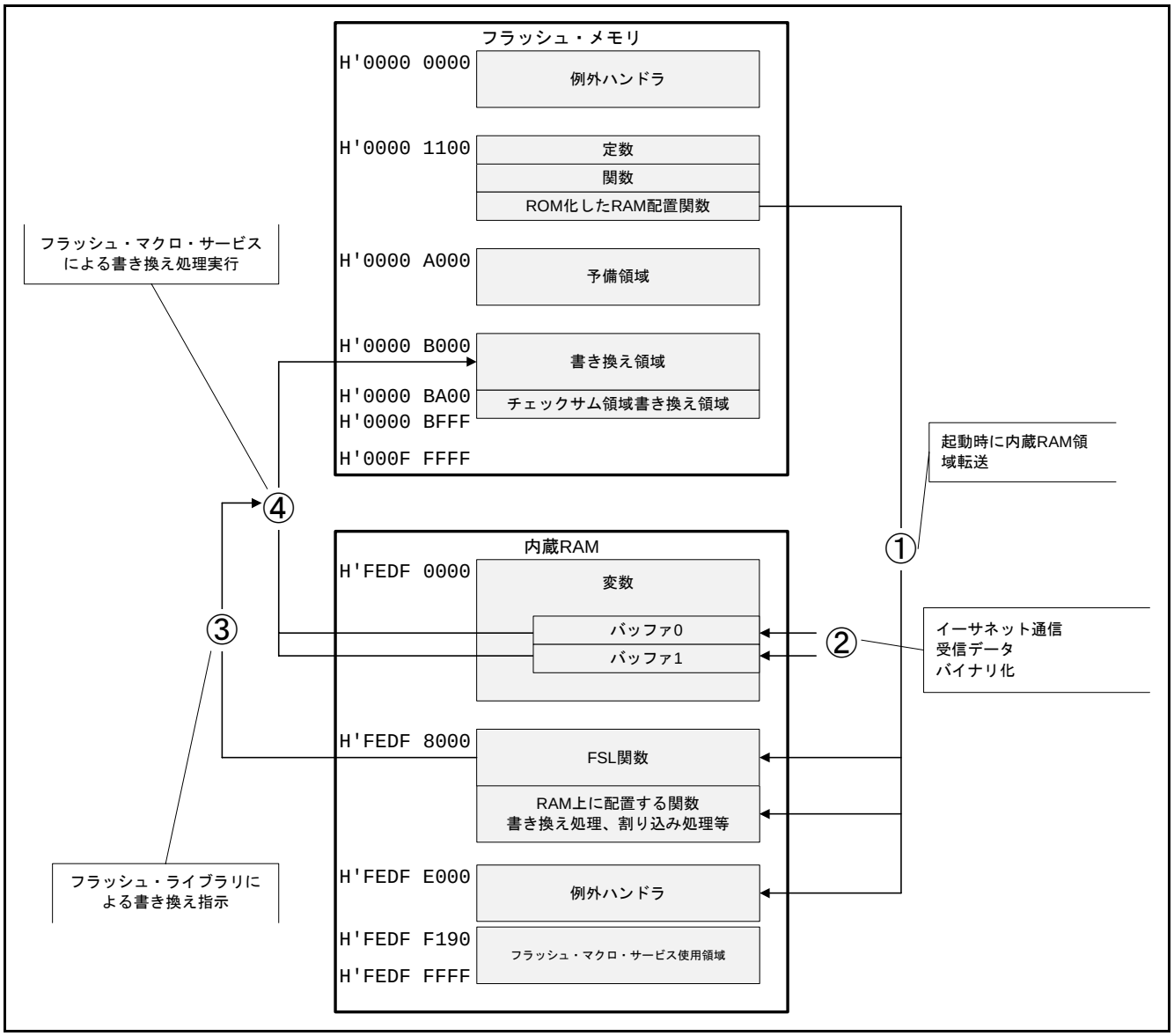


図6.1　フラッシュ・メモリ書き換え動作概要図

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 イーサネット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で内蔵 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換え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、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 B000）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 1536 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。

また、ホスト PC 上で端末エミュレータソフトを起動してボードとの接続が完了した際には、ホスト PC に対し「Generate INTP2 interrupt for transition to flash programming event.」というメッセージを送信します。メッセージを送信した後は、INTP2 割り込み待ちを行います。

6.1.4　INTP2 割り込み入力後のフラッシュ書き換え処理

INTP2 割り込み（立ち下がりエッジ検出／CPU ボード上の INTP2 外部割り込み用スイッチ（SW5）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 PC に対して「--> INTP2 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 PC 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 PC 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 PC からインテル拡張ヘキサ・フォーマット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 PC からデータが送られていないかポーリングで確認し続けます。データが送られていた場合は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・フォーマットのファイル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。）

```
:040000050000013C81C
:0200000400000FA
:20800000E0570584CA5EEFFF605F0484E0670583CC6EEFFF606F0483407640FF2E7F054609
:20802000CF86EFFF408E40FF71870546E0970580929E1000609F0480405681FF6A070082E5
:208040002B06FAFF0000406681FF6C5F4082206EFF3F606F00C44076FFFF0E7F66608F86C8
:1A8060000F00408EFFFF518766604096FFFFD2BF6660019A609FC4C57F00C0
:00000001FF
```

図6.2　インテル拡張ヘキサ・フォーマットのファイルデータ例

- 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ヘキサ・フォーマットでは、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）1536 バイトに満たない場合、バッファサイズが 1536 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 1536 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 1536 バイトで満杯になるたびに、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処理のため、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 BA00～H'0000 BA03（H'0000 BA00～H'0000 BA01：プログラムコードサイズ、H'0000 BA02～H'0000 BA03：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 PC 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 通信制御シーケンスを示します。

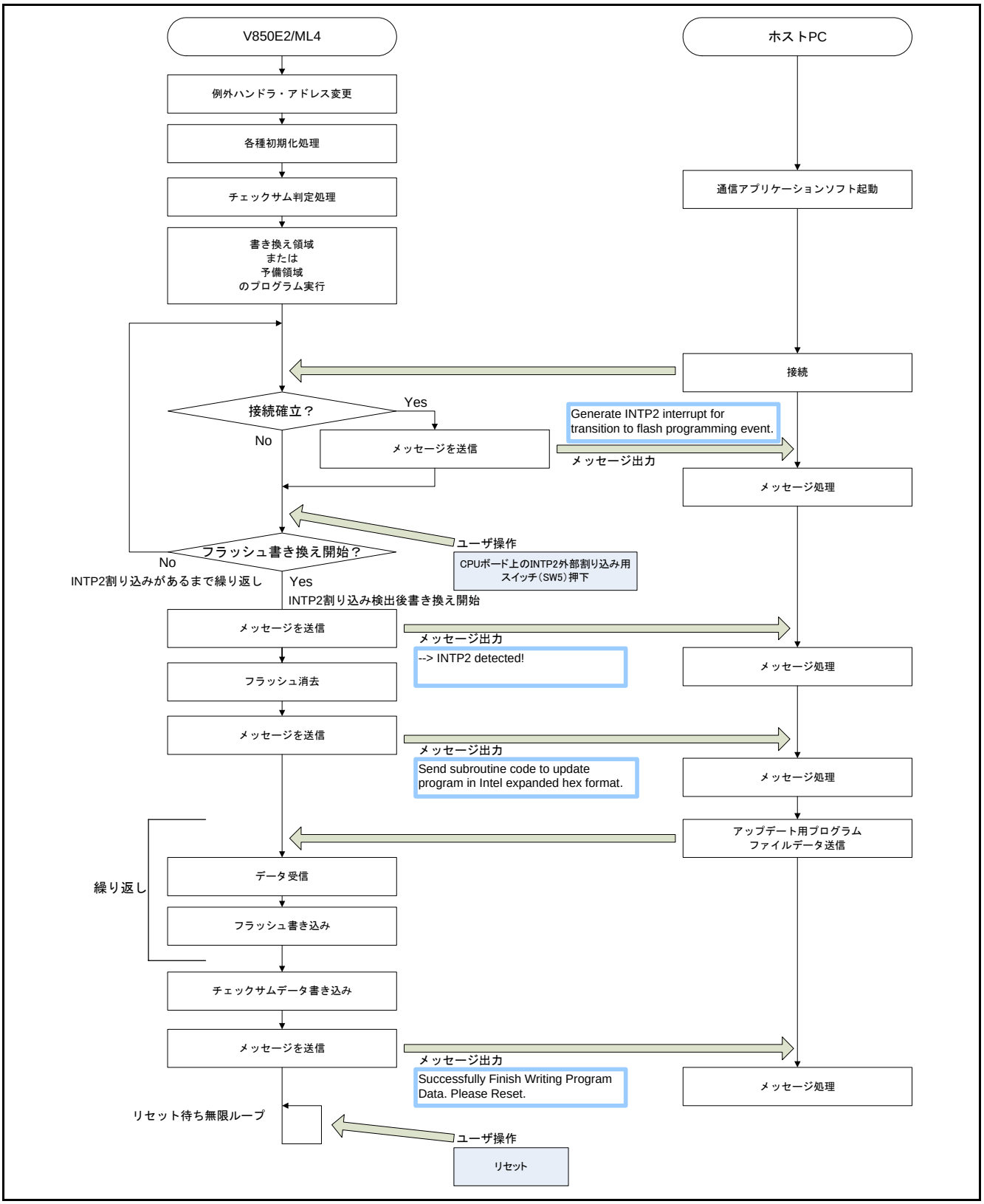


図6.3　通信制御シーケンス

6.1.8　Ethernet インタフェースの制御ソフトウェア

本サンプルコードでは、フラッシュ書き換えのためルネサス製 FSL の使用する以外に、イーサネット通信、TCP/IP 通信を行うためにルネサス製イーサネットドライバ API（イーサネット・サンプルドライバ）とオープンソースソフトウェアである uIP（uip-1.0）を使用しています。。

図 6.4に ソフトウェアとハードウェアの階層図を示します。

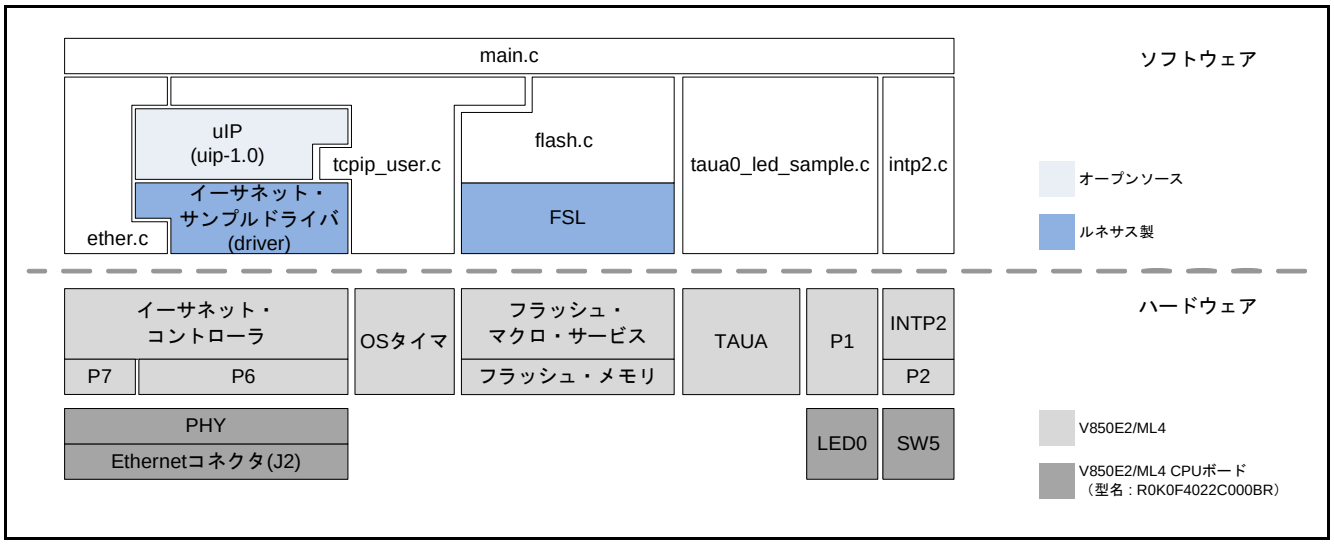


図6.4　ソフトウェアとハードウェアの階層図

本サンプルコードで使用するボード R0K0F4022C000BR は、Ethernet コネクタ（J2）を実装しており、外付け PHY 経由でイーサネット通信を行うことができます。イーサネット・コントローラおよび PHY の制御は、標準ルネサス API（RAPI）として公開されているドライバインタフェース（イーサネット・サンプルドライバ）を使用して行います。本サンプルコードのプロジェクトでは、イーサネット・サンプルドライバを RAM 上に配置して動作させるために、使用するファイルのみをビルドの対象にして、最小限の変更を加えて使用しています。

- イーサネット・サンプルドライバ使用ファイル：
  - ether_driver.c（本サンプルコードでは FSL_CODE_RAM_USER.text セクションへの配置指定「#pragma text “FSL_CODE_RAM_USER”」を追記して使用）
  - ether_phy.c（本サンプルコードでは FSL_CODE_RAM_USER.text セクションへの配置指定「#pragma text “FSL_CODE_RAM_USER”」を追記して使用）
  - ether_mem_for_cubesuite.s（本サンプルコードでは.text セクションへの配置指定「 .cseg text」を FSL_CODE_RAM_USER.text セクションへの配置指定「FSL_CODE_RAM_USER.test .cseg test」に変更して使用）
  - その他、変更なしで使用するヘッダ・ファイル：
    ether_register.h、ether_driver.h、debug.h、r_stdint.h
- イーサネットドライバ API の本サンプルコード内で使用する関数：
  - R_Ether_Open
  - R_Ether_Read
  - R_Ether_Write

本サンプルコードでは、イーサネット・サンプルドライバを使用して TCP/IP の通信を行うために TCP/IP のプロトコルスタック uIP を使用します。uIP については「uip-1.0 リファレンスマニュアル」を参照してください。本サンプルコードのプロジェクトでは、uIP を RAM 上に配置して動作させるために、使用するファイルのみをビルドの対象にして、最小限の変更を加えて使用しています。

- uIP 使用ファイル：
  - uip.c（FSL_CODE_RAM_USER.text セクションへの配置指定
    「#pragma text “FSL_CODE_RAM_USER”」を追記、memcpy 関数を onram_memcpy 関数に変更、memset 関数を onram_memset 関数に変更、memcmp 関数を onram_memcmp 関数に変更）
  - uip_arp.c（FSL_CODE_RAM_USER.text セクションへの配置指定
    「#pragma text “FSL_CODE_RAM_USER”」を追記、memcpy 関数を onram_memcpy 関数に変更、memset 関数を onram_memset 関数に変更、memcmp 関数を onram_memcmp 関数に変更）
  - timer.c（FSL_CODE_RAM_USER.text セクションへの配置指定
    「#pragma text “FSL_CODE_RAM_USER”」を追記、memcpy 関数を onram_memcpy 関数に変更、memset 関数を onram_memset 関数に変更、memcmp 関数を onram_memcmp 関数に変更）
  - uip.h（memcpy 関数を onram_memcpy 関数に変更、memset 関数を onram_memset 関数に変更、memcmp 関数を onram_memcmp 関数に変更）
  - その他、変更なしで使用するヘッダ・ファイル：
    uipopt.h、uip_arch.h、uip_arp.h、timer.h、clock.h
- uIP の本サンプルコードで使用する関数、マクロ定義、変数：
  - uip_init、uip_ipaddr、uip_sethostaddr、uip_setnetmask、uip_listen、uip_connected、uip_newdata、uip_send、uip_datalen、uip_arp_ipin、uip_input、uip_arp_out、uip_arp_arpin、uip_periodic、uip_arp_timer、timer_set、timer_reset、int timer_expired、HTONS、uip_len、uip_buf、uip_appdata、UIP_ETHTYPE_IP、UIP_ETHTYPE_ARP、IP_PORT_NUM、

本サンプルコードでは、イーサネット・サンプルドライバを使用するために必要な関数、処理を ether.c で実装しています。

- ether.c 実装内容：
  - msleep 関数（R_Ether_Open 関数内からコールされるため実装が必要）
    - 機能実現のため uIP の timer_set 関数と timer_expired 関数を使用
  - イーサネット・コントローラ用ポート機能初期化処理（本サンプルコードでは ether_port_init 関数）
  - H バス初期化処理（本サンプルコードでは hbus_init 関数）

本サンプルコードでは、uIP を動作させるために必要な関数、処理、マクロ定義、型定義を clock-arch.h、tcpip_user.h、tcpip_user.c、で実装しています。また、uIP の環境設定ヘッダ・ファイル uip-conf.h は uIP サンプルから設定値を変更して使用しています。

- uip-conf.h の変更内容：
  - ビルドオプションは、本サンプルコードの環境に合わせ下記の通り変更
    - UIP_CONF_MAX_CONNECTIONS 1
    - UIP_CONF_MAX_LISTENPORTS 1
    - UIP_CONF_BUFFER_SIZE 1514
    - UIP_CONF_BYTE_ORDER UIP_LITTLE_ENDIAN
  - アプリケーションに対するインクルードヘッダは tcpip_user.h に指定

- clock-arch.h 実装内容（uIP からインクルードされるため、ファイル名 clock-arch.h で実装が必要）：
  - マクロ定義 CLOCK_CONF_SECOND
  - 型定義 clock_timer_t

- tcpip_user.h 実装内容：
  - マクロ定義 UIP_APPCALL（uIP 内部からコールされるイベント処理指定のため必要、本サンプルコードでは tcpip_user_appcall 関数を指定）
  - 型定義 uip_tcp_appstate_t（本サンプルコードでは未使用、uIP ビルドのため必要）

- tcpip_user.c 実装内容：
  - イベント処理（本サンプルコードでは tcpip_user_appcall 関数として実装）
  - 主制御ループで行うパケット受信処理と周期タイマ関数実行処理（本サンプルコードでは tcip_ip_polling_work 関数として実装して、割り込み待ちなどのループ内から実行）
  - clock_time 関数（uIP 内部からコールされるため実装が必要）
    - 機能実現のため V850E2/ML4 の OS タイマモジュールを使用

6.2　ファイル構成

表 6.3、表 6.4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（1／2）

ファイル名	概要	備考
main.c	メイン処理	
intp2.c	INTP2 割り込み処理	
flash.c	フラッシュ書き換え関連処理	
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	
tcpip_user.c	uIP を使用した TCP/IP 関連処理	
ether.c	イーサネット・サンプルドライバ関連処理	
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	
tcpip_user.h	uIP 使用 TCP/IP 機能関連ヘッダ	
clock-arch.h	クロックタイム処理ヘッダ	
uip-conf.h	uIP 環境設定オプション	
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）・ファイル	
v850e2ml4_flash_update_ether.dir	リンク・ディレクティブ設定ファイル	
v850e2ml4_flash_update_ether.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。

表6.4　サンプルコードで使用するファイル（2／2）

ファイル名	概要	備考
timer.c	タイマライブラリ（uIP のファイル*）	FSL_CODE_RAM_USR.text 上セクション指定を追記
uip.c	uIP TCP/IP プロトコルスタック（uIP のファイル*）	• FSL_CODE_RAM_USR.text 上セクション指定を追記 • memset を onram_memset に差し替え • memcpy を onram_memcpy に差し替え
uip_arp.c	uIP ARP（uIP のファイル*）	• FSL_CODE_RAM_USR.text 上セクション指定を追記。 • memset を onram_memset に差し替え • memcpy を onram_memcpy に差し替え
ether_driver.c	イーサネットドライバ（イーサネット･サンプルドライバのファイル*）	FSL_CODE_RAM_USR.text 上セクション指定を追記
ether_phy.c	PHY コントロール（イーサネット・サンプルドライバのファイル*）	FSL_CODE_RAM_USR.text 上セクション指定を追記
uip.h	uIP TCP/IP プロトコルスタックヘッダ（uIP のファイル*）	
uipopt.h	uIP に対する環境設定オプション（uIP のファイル*）	memcpy を onram_memcpy に差し替え
uip_arch.h	uIP アーキテクチャ特有関数（uIP のファイル*）	
uip_arp.h	uIP ARP マクロ（uIP のファイル*）	
timer.h	タイマライブラリ（uIP のファイル*）	
clock.h	クロックインタフェース（uIP のファイル*）	
r_stdint.h	整数型定義ヘッダ（イーサネット・サンプルドライバのファイル*）	
ether_register.h	イーサネットレジスタアドレス定義ヘッダ（イーサネット･サンプルドライバのファイル*）	
ether_driver.h	イーサネット定義ヘッダ（イーサネット･サンプルドライバのファイル*）	
debug.h	デバッグ用 DBG_PRINT（イーサネット･サンプルドライバのファイル*）	
eth_mem_for_cubesuite.s	H バス用 memcpy（イーサネット・サンプルドライバのファイル*）	.text セクション指定を FSL_CODE_RAM_USR.text セクション指定に変更

【注】* 本サンプルコードでは、イーサネット･サンプルドライバと uIP のソースコードを組み込んでいます。これらのソースコードに含まれる API の仕様については、それぞれのマニュアルを参照してください。なお、これらのソースコードは RAM 上で動作させるために、セクション指定と使用メモリ関数の差替えを行っています。

6.3　定数一覧

表 6.5、表 6.6に サンプルコードで使用する定数を示します。

表6.5　サンプルコードで使用する定数（1／2）

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	書き込み処理失敗
RET_ERR_FLASH_IVERIFY	-5	内部ベリファイ失敗
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_HEX_LINESIZE	-9	ヘキサ・ファイルの行データ数異常
RET_ERR_FLASH_HEX_DATA	-10	ヘキサ・ファイルのプログラムデータ異常
BLOCK_MASTER_PRG	H'B	書き換え領域ブロック番号
TOP_ADDR_MASTER_PRG	H'0000B000	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	1536	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'0000BA00）
TOP_ADDR_EXT_HANDLER	H'FEDF E000	転送先例外ハンドラ・アドレス先頭
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_ACTIVE	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

表6.6　サンプルコードで使用する定数（2／2）

定数名	設定値	内容
SIZE_MAX_LINE_NUM_HEX	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
SIZE_BUF_RX_DATA	1536 + SIZE_MAX_LINE_NUM_HEX	受信データ格納バッファサイズ
FLASH_WRITE_LOOP_NOP	0	書き込み待ち時に、関数実行なし
CLOCK_CONF_SECOND*	1000	1 秒のクロック刻み回数
UIP_APPCALL*	user_appcall	uIP からのイベント処理コール関数
IP_PORT_NUM	1024	使用するポート番号
SIZE_BUF_TX	256	送信バッファのサイズ
PORT_BIT_P1_4	H'0010	ポート機能設定 P1_4 のビット位置
PORT_BIT_P2_7	H'0080	ポート機能設定 P2_7 のビット位置
PORT_BIT_P6_0	H'0001	ポート機能設定 P6_0 のビット位置
PORT_BIT_P6_1	H'0002	ポート機能設定 P6_1 のビット位置
PORT_BIT_P6_2	H'0004	ポート機能設定 P6_2 のビット位置
PORT_BIT_P6_3	H'0008	ポート機能設定 P6_3 のビット位置
PORT_BIT_P6_4	H'0010	ポート機能設定 P6_4 のビット位置
PORT_BIT_P6_5	H'0020	ポート機能設定 P6_5 のビット位置
PORT_BIT_P6_6	H'0040	ポート機能設定 P6_6 のビット位置
PORT_BIT_P6_7	H'0080	ポート機能設定 P6_7 のビット位置
PORT_BIT_P6_8	H'0100	ポート機能設定 P6_8 のビット位置
PORT_BIT_P6_9	H'0200	ポート機能設定 P6_9 のビット位置
PORT_BIT_P6_10	H'0400	ポート機能設定 P6_10 のビット位置
PORT_BIT_P6_11	H'0800	ポート機能設定 P6_11 のビット位置
PORT_BIT_P6_12	H'1000	ポート機能設定 P6_12 のビット位置
PORT_BIT_P6_13	H'2000	ポート機能設定 P6_13 のビット位置
PORT_BIT_P6_14	H'4000	ポート機能設定 P6_14 のビット位置
PORT_BIT_P6_15	H'8000	ポート機能設定 P6_15 のビット位置
PORT_BIT_P7_0	H'0001	ポート機能設定 P7_0 のビット位置
PORT_BIT_P7_1	H'0002	ポート機能設定 P7_1 のビット位置

【注】* CLOCK_CONF_SECOND と UIP_APPCALL は uIP を組み込む際にユーザが実装しなければならない定義です。uIP のリファレンスマニュアルを参照してください。

6.4　変数一覧

表 6.7、表 6.8に グローバル変数を示します。

表6.7　グローバル変数（1／2）

型	変数名	内容	使用関数
uint8_t	g_flag_start_flash_reprog	フラッシュ書き換え開始フラグ	main, intp2_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_line_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_line_data
uint8_t	g_status_cur_w_buf_store	書き込みバッファ格納ステータス	flash_reprogram, flash_store_line_data
uint8_t	g_flag_end_record	終了レコード受信フラグ	flash_reprogram, flash_store_line_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_line_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_line_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_line_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_line_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_line_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_line_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_ether_data
uint8_t	g_buf_rx_data [SIZE_BUF_RX_DATA]	受信データ格納バッファ	flash_store_ether_data
int8_t	g_status_store_error	受信データエラー	flash_reprogram, flash_store_line_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
uint8_t *	g_addr_line_head	受信データの未処理行先頭アドレス	flash_reprogram, flash_store_ether_data
uint8_t	g_status_cur_w_buf_write	書き込みバッファ書き込みステータス	flash_reprogram, flash_store_ether_data
char	g_msg_sendcode[]	プログラム送信要求メッセージ	flash_reprogram

表6.8　グローバル変数（2／2）

型	変数名	内容	使用関数
uint8_t	g_flag_connected	接続確立フラグ	tcpip_user_app_init, tcpip_user_appcall, tcpip_tx_bufdata
uint8_t	g_buf_tx [SIZE_BUF_TX]	送信バッファ	tcpip_user_app_init, tcpip_buf_msg, tcpip_tx_bufdata
uint32_t	g_tx_s	送信バッファデータ先頭	tcpip_user_app_init, tcpip_tx_bufdata
uint32_t	g_tx_e	送信バッファデータ末尾	tcpip_user_app_init, tcpip_buf_msg, tcpip_tx_bufdata
struct timer	g_periodic_timer	周期タイマ	tcpip_user_app_init, tcpip_polling_work
struct timer	g_arp_timer	ARP タイマ	tcpip_user_app_init, tcpip_polling_work
clock_time_t	g_sys_clock	システムクロックタイム	clock_init, clock_time, ostm0_isr
uint8_t	g_buf_ip_addr[]	CPU ボードの IP アドレス* 初期値：{192, 168, 1, 76}	tcpip_user_app_init
uint8_t	g_buf_subnet_mask[]	サブネットマスク* 初期値：{255, 255, 255, 0}	tcpip_user_app_init
uint8_t	g_buf_mac_addr[]	CPU ボードの MAC アドレス* 初期値：{H'00, H'01, H'02, H'03, H'04, H'05}	main

【注】* 設定値は環境に合わせて適宜変更してください。

6.5　関数一覧

表 6.9に 関数を示します。

表6.9　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
intp2_init	INTP2 割り込み初期化処理
intp2_isr	INTP2 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_ether_data	受信データ変換格納処理
flash_store_line_data	行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 TAUA0 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 TAUA0 初期化処理（予備領域サンプル関数）
taua0_i0_interval_timer_isr*	TAUA0 インターバル・タイマ割り込み処理
tcpip_user_app_init	TCP/IP 使用 uIP 設定初期化処理
tcpip_user_appcall	TCP/IP イベント処理
tcpip_buf_msg	TCP/IP 送信メッセージ格納処理
tcpip_tx_bufdata	TCP/IP バッファデータ送信処理
tcpip_polling_work	TCP/IP ポーリング処理
onram_memcpy	RAM 上配置 memcpy 関数処理
onram_memcmp	RAM 上配置 memcmp 関数処理
onram_memset	RAM 上配置 memset 関数処理
clock_init	システムクロックタイム初期化処理
clock_time	システムクロックタイム取得処理
ostm0_isr	OS タイマ割り込み処理
ether_port_init	イーサネット・コントローラ用ポート機能初期化処理
hbus_init	H バス初期化処理
msleep	ミリ秒単位スリープ処理

【注】* LED の点滅処理よりも Ethernet 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、TAUA0 インターバル・タイマ割り込みは他の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、INTP2 割り込み、uIP 設定、H バス、Ethernet ポートの初期化後に、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、割り込みを許可し、イーサネット・サンプルドライバをオープンして、INTP2 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ます。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 BA00～H'0000 BA03）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 B000）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

intp2_init	
概　要	INTP2 割り込み初期化処理
ヘッダ	
宣　言	void intp2_init (void)
説　明	INTP2 の初期化処理を行います。P2_7 の端子機能を INTP2 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の立ち下がりエッジで検出するように設定します。その後、INTP2 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

intp2_isr	
概　要	INTP2 割り込み処理
ヘッダ	
宣　言	void intp2_isr (void)
説　明	INTP2 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) ：内部ベリファイが失敗 -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_HEX_LINESIZE) ：ヘキサ・ファイルの行データ数が異常 -10 (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, func_callback_t func_callback)
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。待ち期間中は引数 loop_func で指定された関数を実行します。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） func_callback_t func_callback ：待ち期間中実行関数のアドレス、0(FLASH_WRITE_LOOP_NOP)が指定された場合は実行しない func_callback_t func_callback ：待ち期間中実行関数のアドレス、0(FLASH_WRITE_LOOP_NOP)が指定された場合は実行しない
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	フラッシュ環境終了処理
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_ether_data	
概　要	受信データ変換格納処理
ヘッダ	flash.h
宣　言	void flash_store_ether_data (void)
説　明	イーサネット受信データを確認して、受信データがあるときは受信データ格納バッファに格納します。受信データ格納バッファに格納したデータに改行があるときは、行データ変換格納処理を実行します。最後に受信データ格納バッファのデータを先頭に移動して終了します。
引　数	なし
リターン値	なし

flash_store_line_data	
概　要	行データ変換格納処理
ヘッダ	flash.h
宣　言	void flash_store_line_data (uint8_t * addr_line_head)
説　明	ヘキサ･データ 1 行をバイナリ変換して 2 面の書き込みデータ格納バッファに格納します。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。バッファがフルになった場合は、もう 1 面のバッファに切り替えます。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セットします。
引　数	uint8_t * addr_line_head　：行データ先頭アドレス
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 TAUA0 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、TAUA0 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 TAUA0 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、TAUA0 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

tcpip_user_app_init	
概　要	TCP/IP 使用 uIP 設定初期化処理
ヘッダ	
宣　言	void tcpip_user_app_init (void)
説　明	送信バッファと接続確立フラグのグローバル変数の初期化を行います。 そして、システムクロックタイム初期化処理を実行した後で、周期タイマの間隔を 0.5 秒に、ARP タイマの間隔を 10 秒に設定します。 また、uIP の初期化を行い、IP アドレスとサブネットマスク、使用ポートの設定 uIP に対してを行います。
引　数	なし
リターン値	なし

tcpip_user_appcall	
概　要	TCP/IP イベント処理
ヘッダ	
宣　言	void tcpip_user_appcall (void)
説　明	接続イベントが発生したか確認して、発生していた場合は接続確立フラグをセットします。 次に、受信データがあるか確認をして、あった場合は受信データ格納処理を実行します。 最後に送信バッファのデータ送信を行います。
引　数	なし
リターン値	なし

tcpip_buf_msg	
概　要	TCP/IP 送信メッセージ格納処理
ヘッダ	
宣　言	void tcpip_buf_msg (char * msg)
説　明	引数で指定されたメッセージを送信バッファに格納します。メッセージは文字列終端文字'¥0'があるまで送信します。
引　数	char * msg　　：メッセージ文字列
リターン値	なし

tcpip_tx_bufdata	
概　要	TCP/IP バッファデータ送信処理
ヘッダ	
宣　言	void tcpip_tx_bufdata (void)
説　明	接続確立フラグを確認して接続が終わっている場合は、送信バッファにあるデータをすべて送信します。
引　数	なし
リターン値	なし

tcpip_polling_work	
概　要	TCP/IP ポーリング処理
ヘッダ	
宣　言	void tcpip_polling_work (void)
説　明	uIP 動作に必要なパケット到着、周期タイマ監視処理を行います。 パケットが到着しているときは、対応する関数を実行します。 パケットが到着していないとき、周期タイマがタイムアウトしてれば周期タイマをリセットして、周期処理関数を実行します。また、ARP タイマがタイムアウトしていれば ARP タイマをリセットして、ARP 処理関数を実行します。
引　数	なし
リターン値	なし

onram_memcpy		
概　要	RAM 上配置 memcpy 関数処理	
ヘッダ		
宣　言	void * onram_memcpy (void * out, const void * in, size_t n)	
説　明	引数 n で指定したバイトサイズ分を引数 in の指す領域から引数 out の指す領域へコピーします。	
引　数	void * out	: コピー先アドレス
	const void * in	: コピー元アドレス
	size_t n	: データサイズ
リターン値	引数 out の値	

onram_memcmp		
概　要	RAM 上配置 memcmp 関数処理	
ヘッダ		
宣　言	int onram_memcmp (const void * s1, const void * s2, size_t n)	
説　明	引数 s1 の指す領域の最初の引数 n で指定したバイトサイズ分を引数 s2 の指す領域と比較します。	
引　数	const void * s1	: 比較データ 1 の先頭アドレス
	const void * s2	: 比較データ 2 の先頭アドレス
	size_t n	: 比較サイズ
リターン値	0	: 一致
	> 0	: s1 のほうが大きい
	< 0	: s1 のほうが小さい

onram_memset		
概　要	RAM 上配置 memset 関数処理	
ヘッダ		
宣　言	void * onram_memset (const void * s, int c, size_t length)	
説　明	引数 s の指す領域の最初の length バイトに、unsigned char 型に変換した引数 c の値をコピーします。	
引　数	const void * s	: 設定先の先頭アドレス
	int c	: 設定値
	size_t length	: データ長
リターン値	引数 s	

clock_init	
概　要	システムクロックタイム初期化処理
ヘッダ	
宣　言	void ostm0_init (void)
説　明	OS タイマをインターバル・タイマ・モードで使用するための初期化を行います。 1ms 毎にタイマ・アンダフロー割り込み(INTOSTM0)が発生するように OSTM コンペア・レジスタを設定します。 また、クロック時刻変数を初期化して割り込みレベルの設定と許可を行い、カウントを開始します。
引　数	なし
リターン値	なし

clock_time	
概　要	システムクロックタイム取得処理
ヘッダ	
宣　言	clock_time_t clock_time (void)
説　明	グローバル変数のシステムクロックタイムの値をリターンします。
引　数	なし
リターン値	-2147483648～2147483647：システムクロックタイム

ostm0_isr	
概　要	OS タイマ割り込み処理
ヘッダ	
宣　言	void ostm0_isr (void)
説　明	グローバル変数のシステムクロックタイムの値をインクリメントします。
引　数	なし
リターン値	なし

ether_port_init	
概　要	イーサネット・コントローラ用ポート機能初期化処理
ヘッダ	
宣　言	void ether_port_init (void)
説　明	イーサネット・サンプルドライバで使用するポートの機能の設定を行います。
引　数	なし
リターン値	なし

hbus_init	
概　要	H バス初期化処理
ヘッダ	
宣　言	void hbus_init (void)
説　明	H バスを使用するための初期設定を行います。
引　数	なし
リターン値	なし

msleep	
概　要	ミリ秒単位スリープ処理
ヘッダ	
宣　言	void msleep (uint32_t limit_time)
説　明	指定された時間のタイマをセットして、タイマが指定の時間を経過するまで、何もしないループをします。指定の時間を経過したら終了します。
引　数	uint32_t limit_time　　：スリープ時間（ms）
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 6.5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。

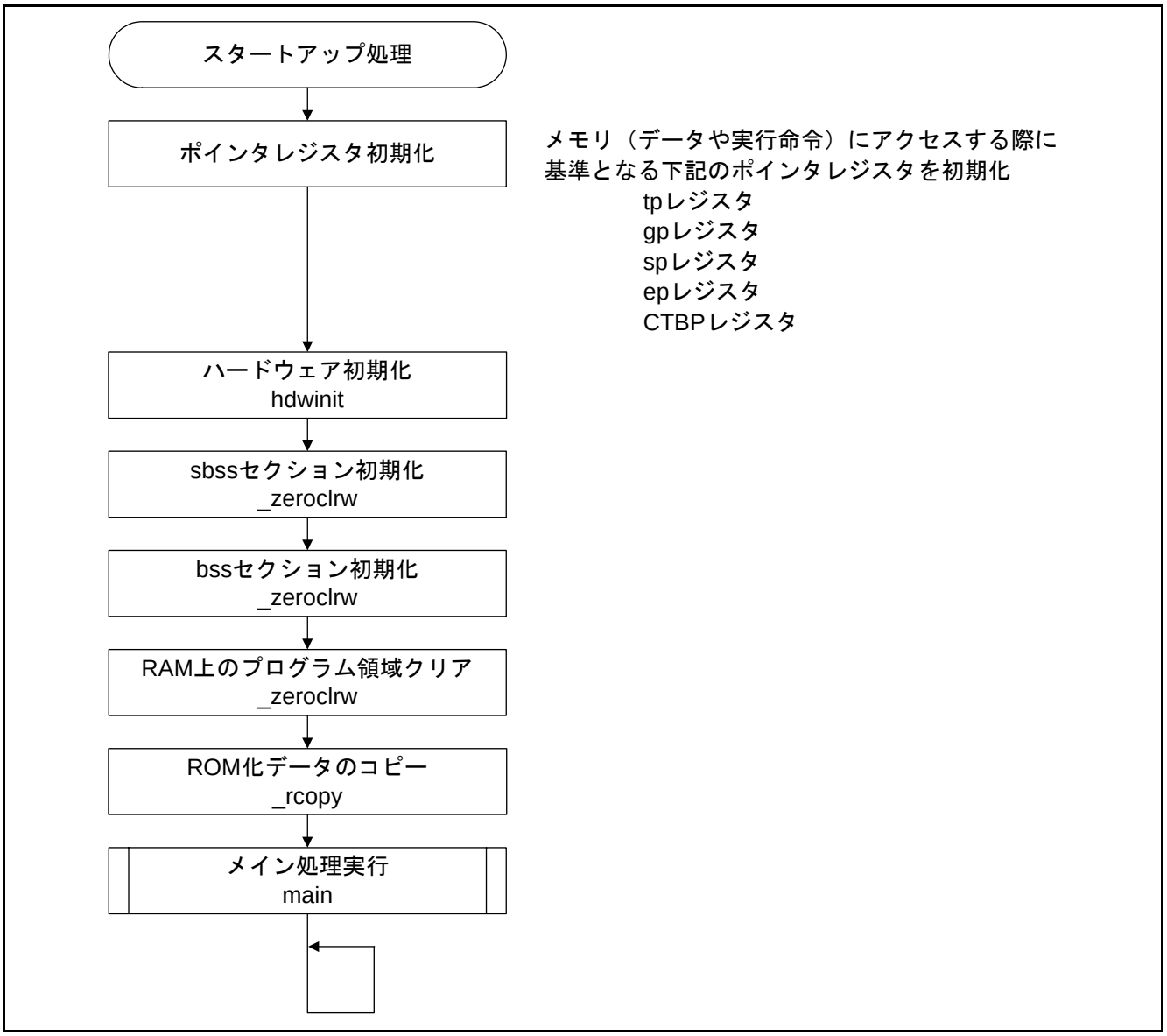


図6.5　スタートアップ・ルーチンの処理

6.7.2 メイン処理

図 6.6に メイン処理のフローチャートを示します。

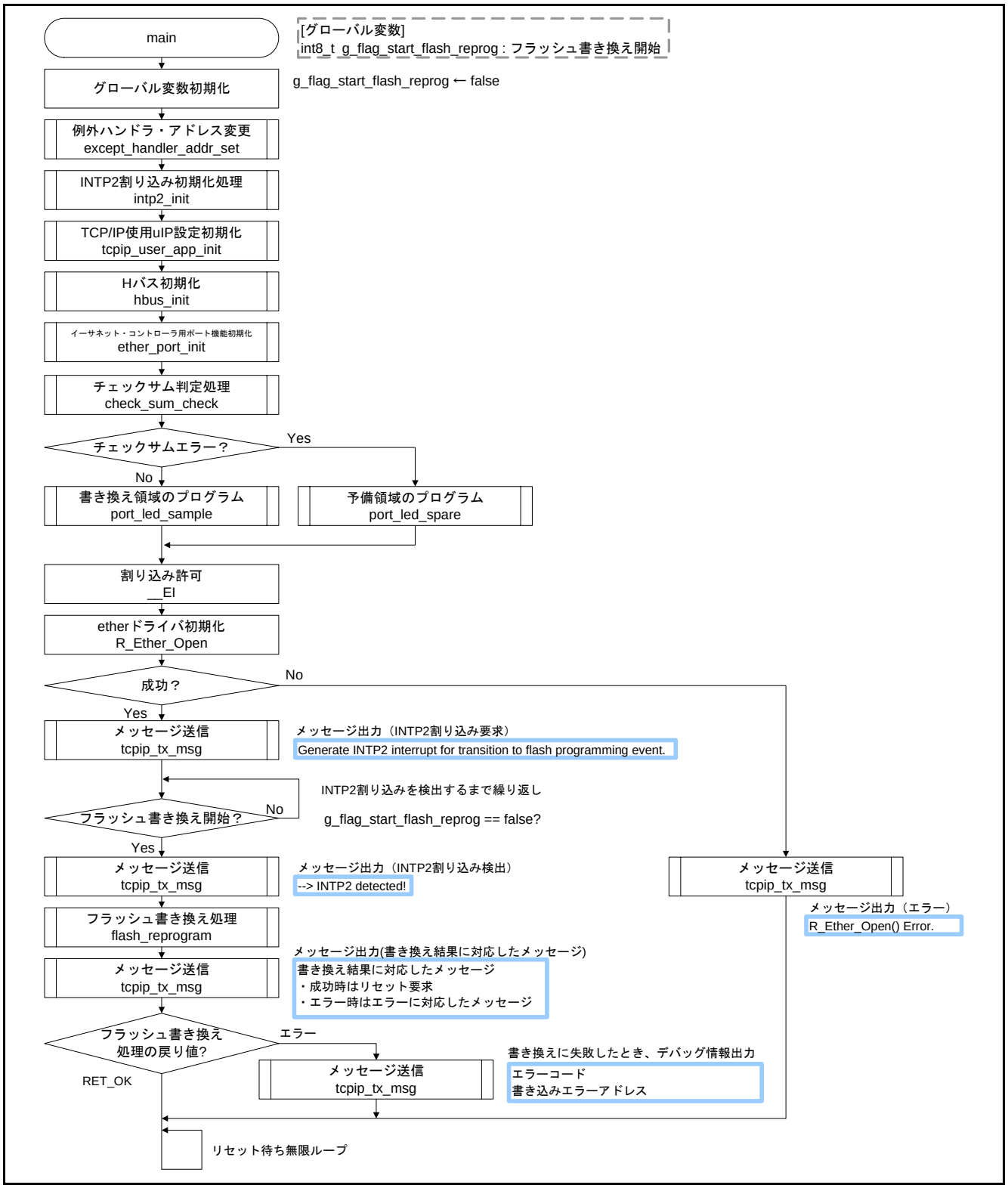
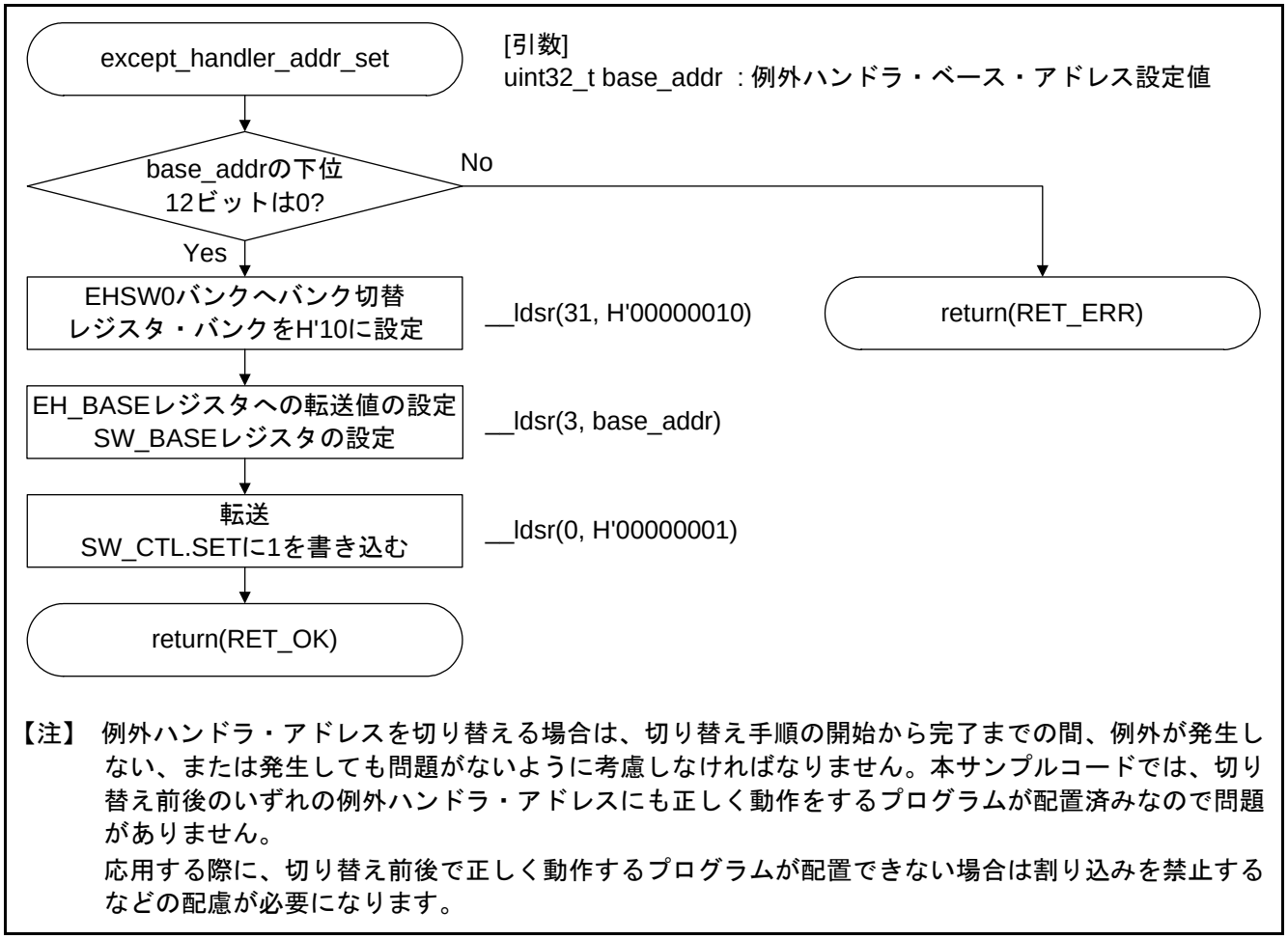


図6.6 メイン処理

6.7.3　例外ハンドラ・アドレス切り替え処理

図 6.7に 例外ハンドラ・アドレス処理のフローチャートを示します。



【注】 例外ハンドラ・アドレスを切り替える場合は、切り替え手順の開始から完了までの間、例外が発生しない、または発生しても問題がないように考慮しなければなりません。本サンプルコードでは、切り替え前後のいずれの例外ハンドラ・アドレスにも正しく動作をするプログラムが配置済みなので問題がありません。
応用する際に、切り替え前後で正しく動作するプログラムが配置できない場合は割り込みを禁止するなどの配慮が必要になります。

図6.7　例外ハンドラ・アドレス処理

6.7.4　書き換え領域のチェックサム判定処理

図 6.8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。

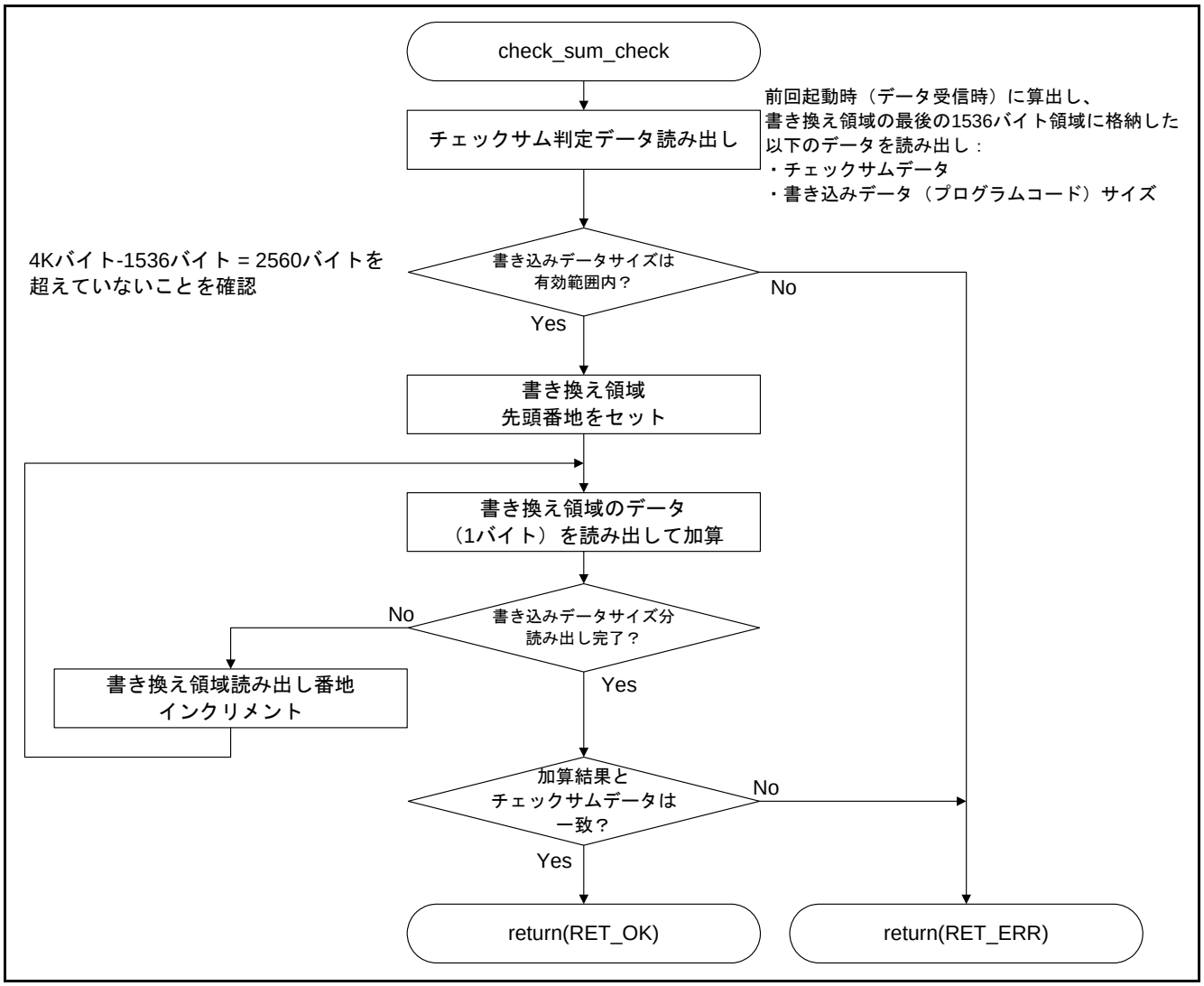


図6.8　書き換え領域のチェックサム判定処理

6.7.5　INTP2 割り込み初期化処理

図 6.9に INTP2 割り込み初期化処理のフローチャートを示します。

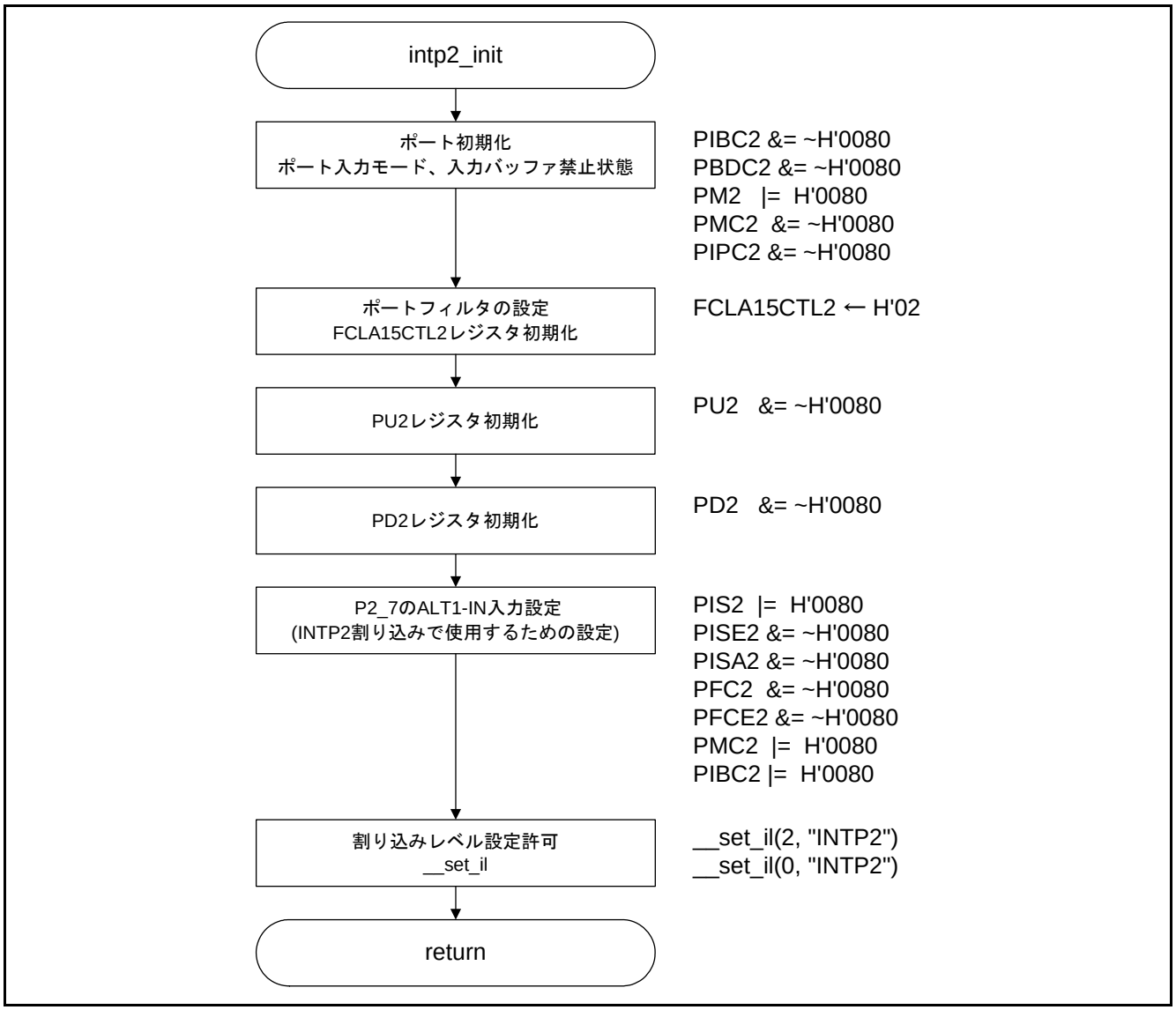


図6.9　INTP2 割り込み初期化処理

6.7.6　INTP2 割り込み処理

図 6.10に INTP2 割り込み処理のフローチャートを示します。

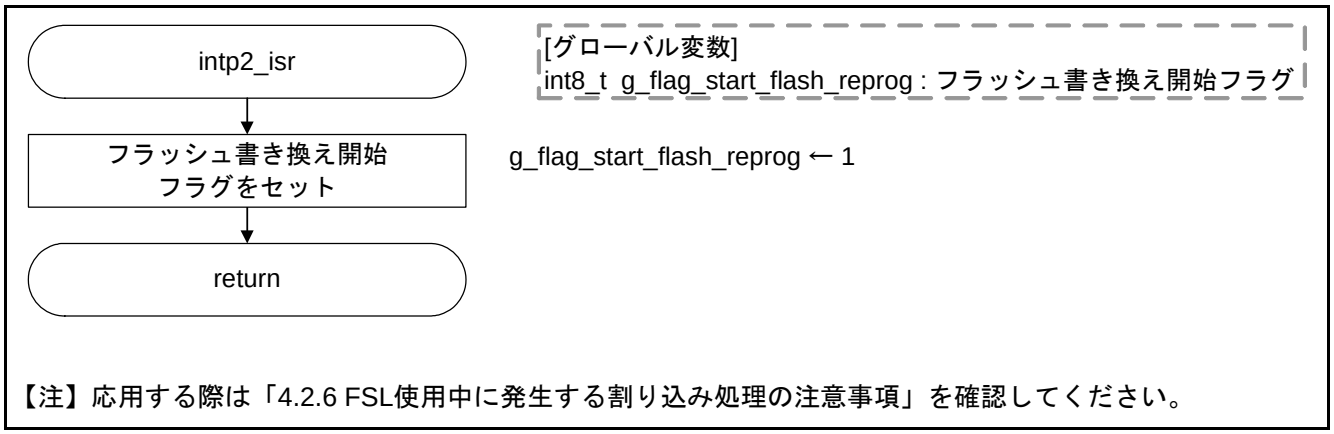


図6.10　INTP2 割り込み処理

6.7.7　フラッシュ書き換え処理

図 6.11、図 6.12に フラッシュ書き換え処理のフローチャートを示します。

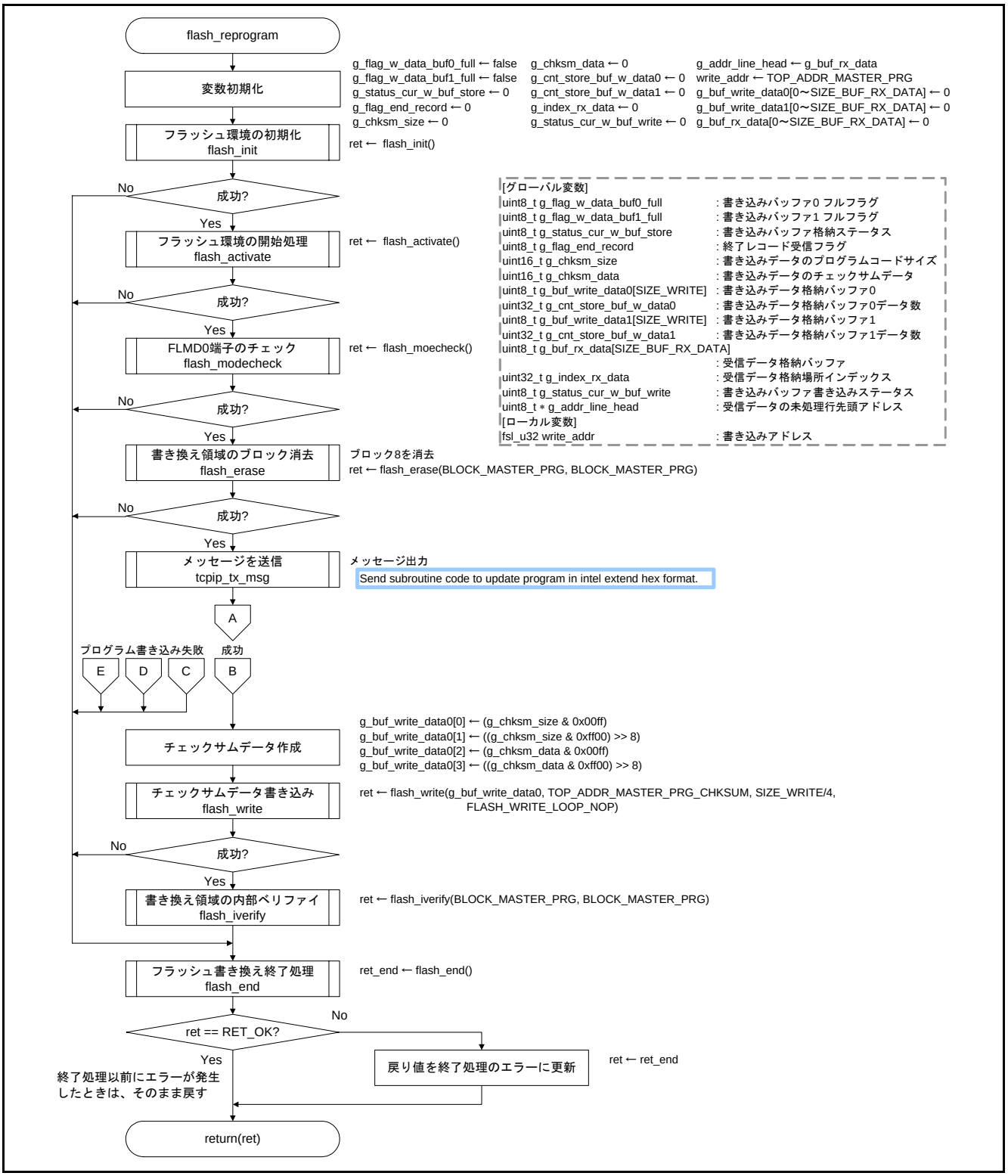


図6.11　フラッシュ書き換え処理（1／2）

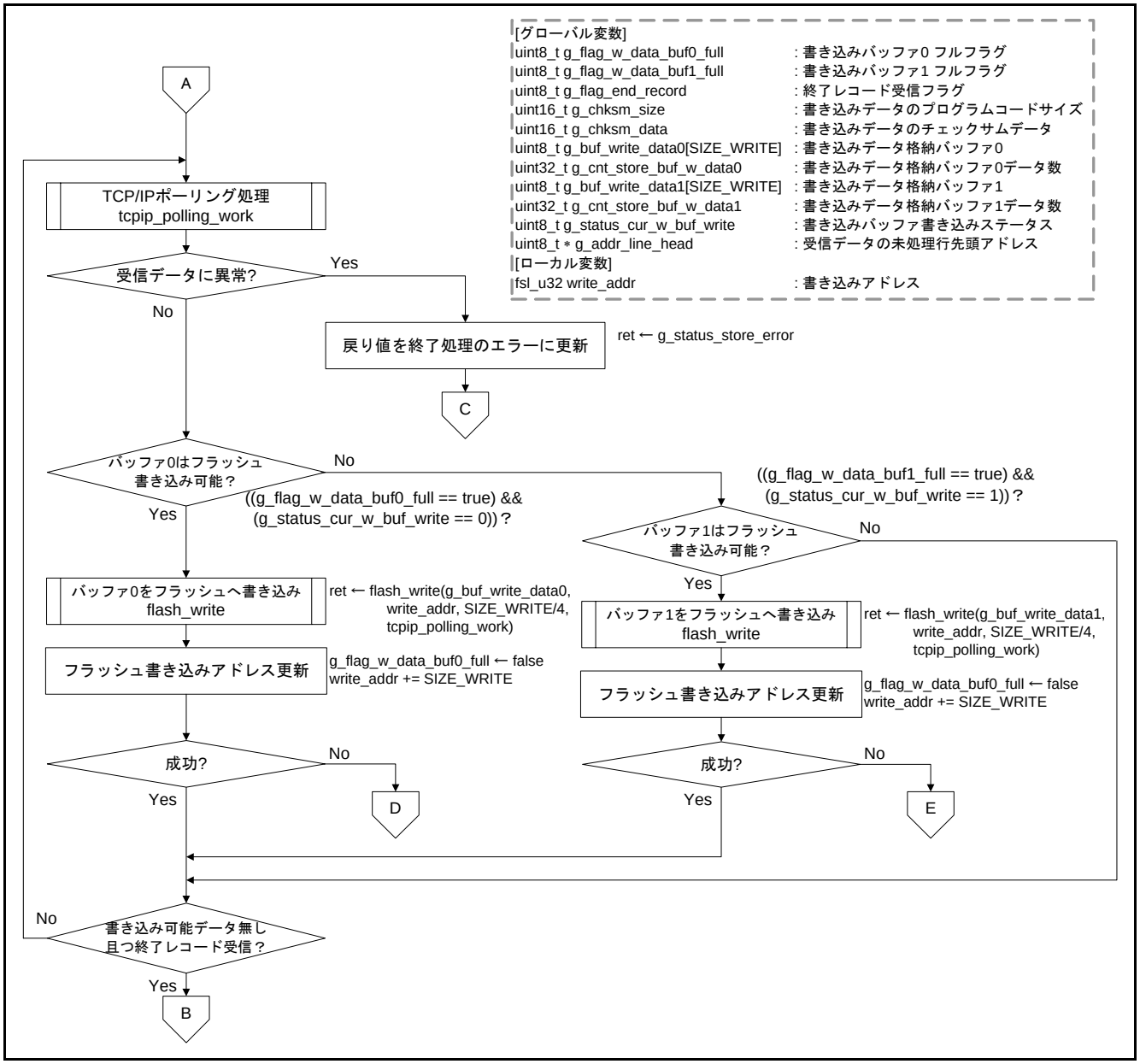


図6.12 フラッシュ書き換え処理（2／2）

6.7.8　フラッシュ環境の初期化処理

図 6.13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。

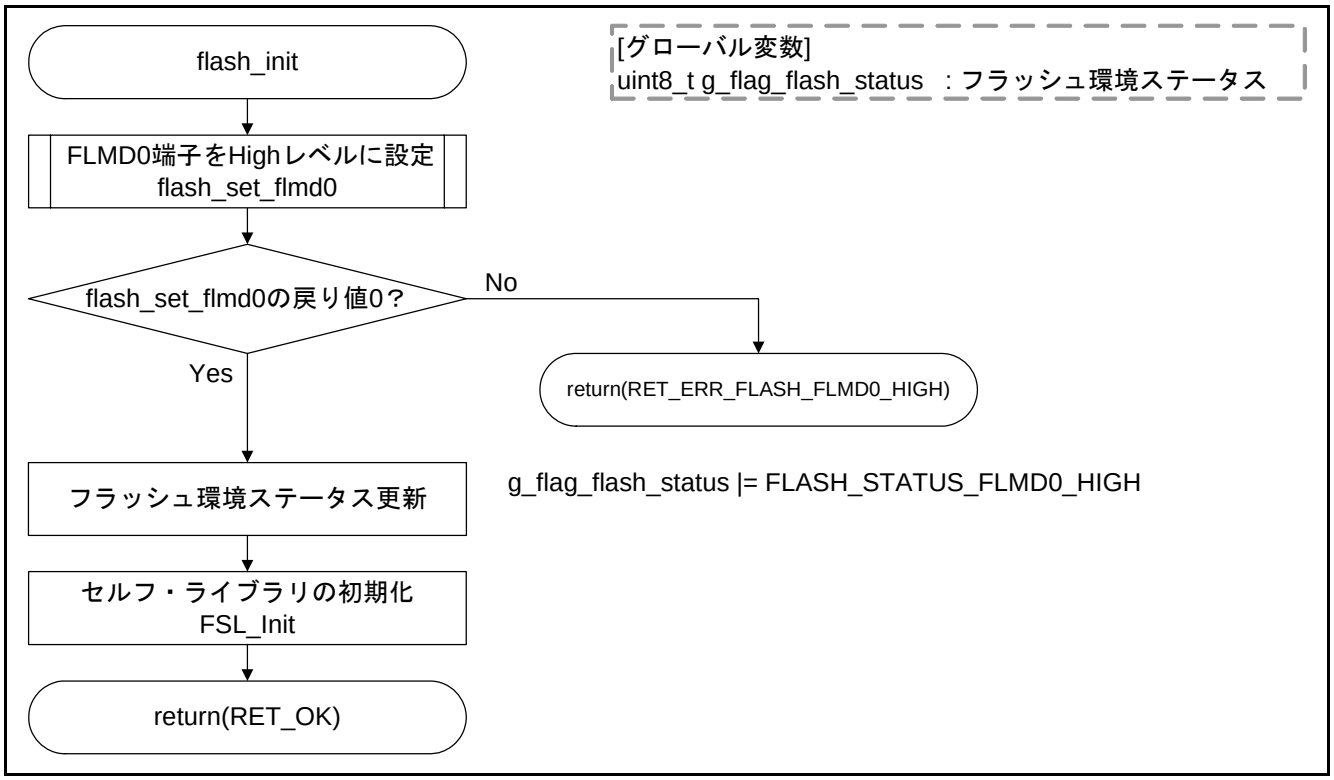


図6.13　フラッシュ環境の初期化処理

6.7.9　フラッシュ環境の開始処理

図 6.14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。

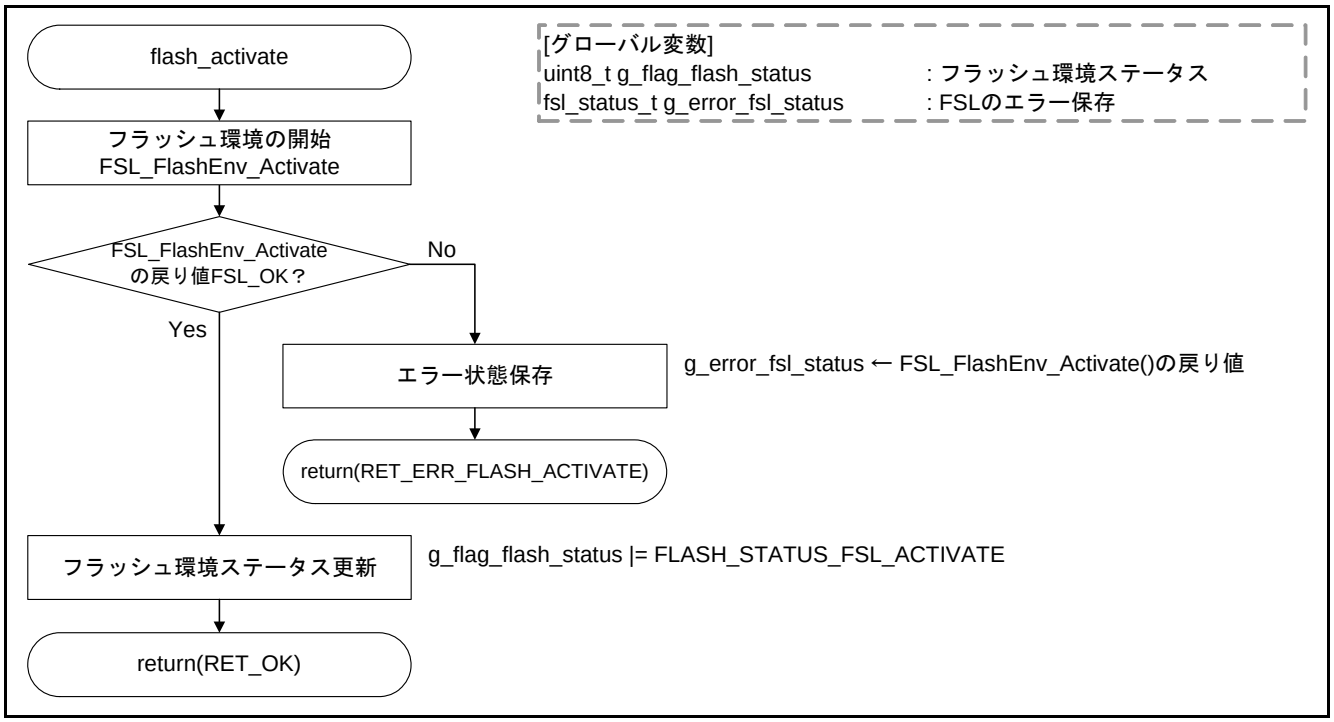


図6.14　フラッシュ環境の開始処理

6.7.10　FSL による FLMD0 端子のチェック処理

図 6.15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。

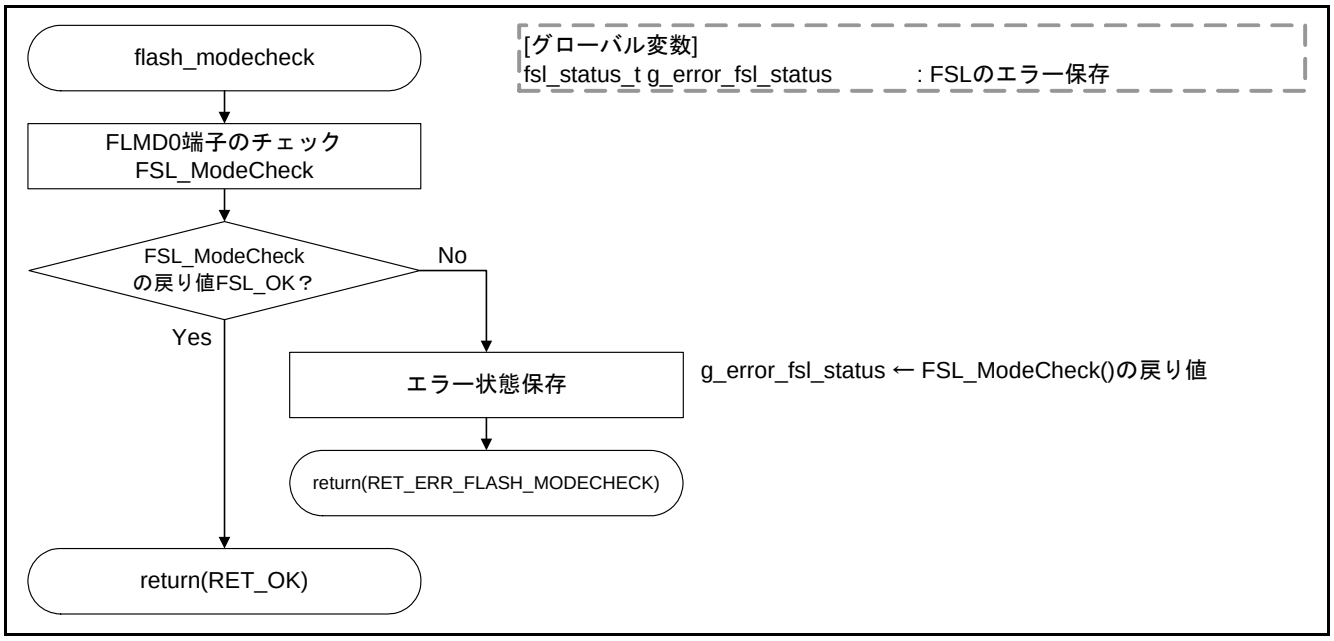


図6.15　FSL による FLMD0 端子のチェック処理

6.7.11　指定ブロックの消去処理

図 6.16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。

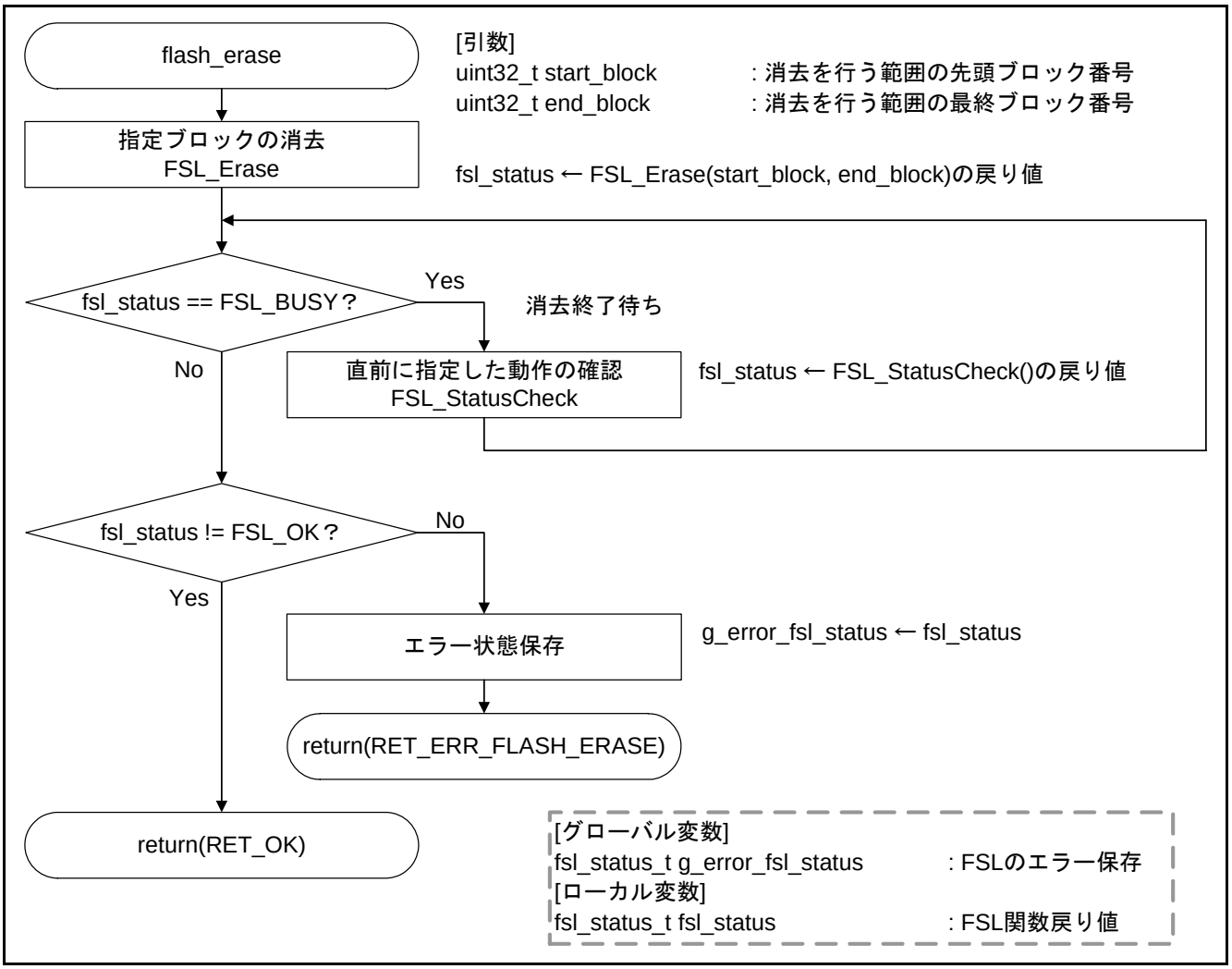


図6.16　指定ブロックの消去処理

6.7.12　指定アドレスからの書き込み処理

図 6.17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。

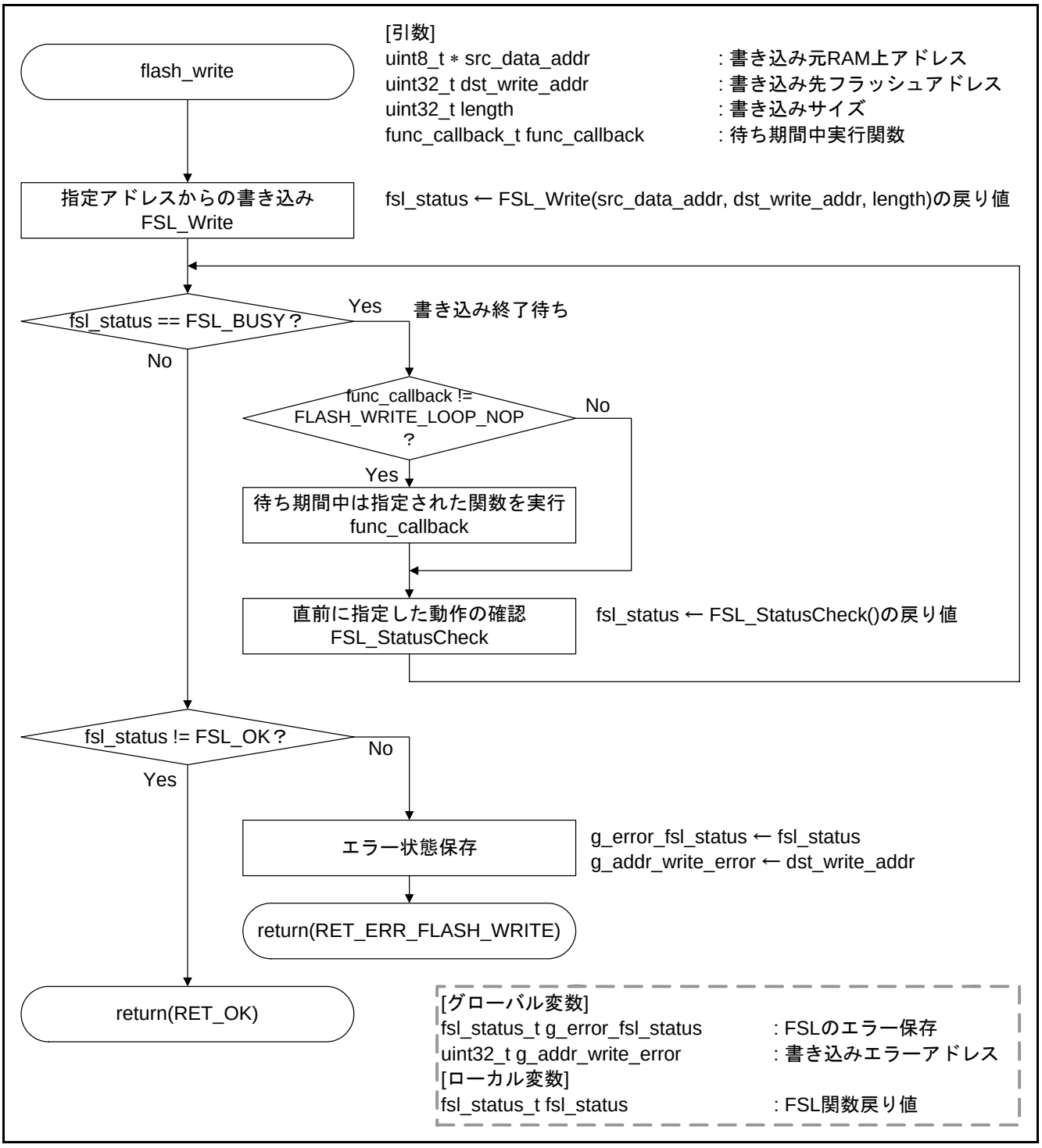


図6.17　指定アドレスからの書き込み処理

6.7.13　指定ブロックの内部ベリファイ処理

図 6.18に 指定ブロックからの内部ベリファイ処理のフローチャートを示します。

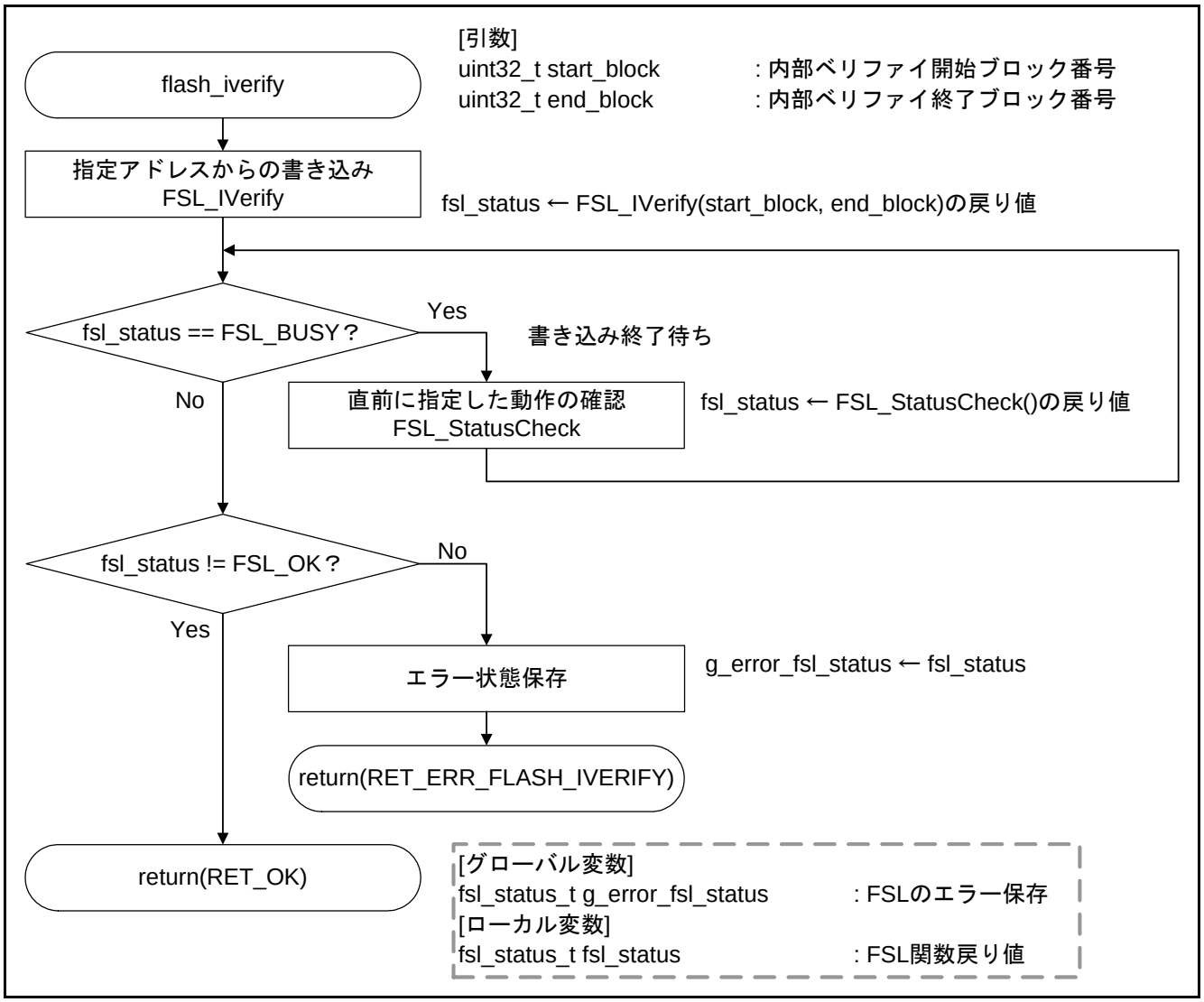


図6.18　指定ブロックからの内部ベリファイ処理

6.7.14 フラッシュ環境終了処理

図 6.19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。

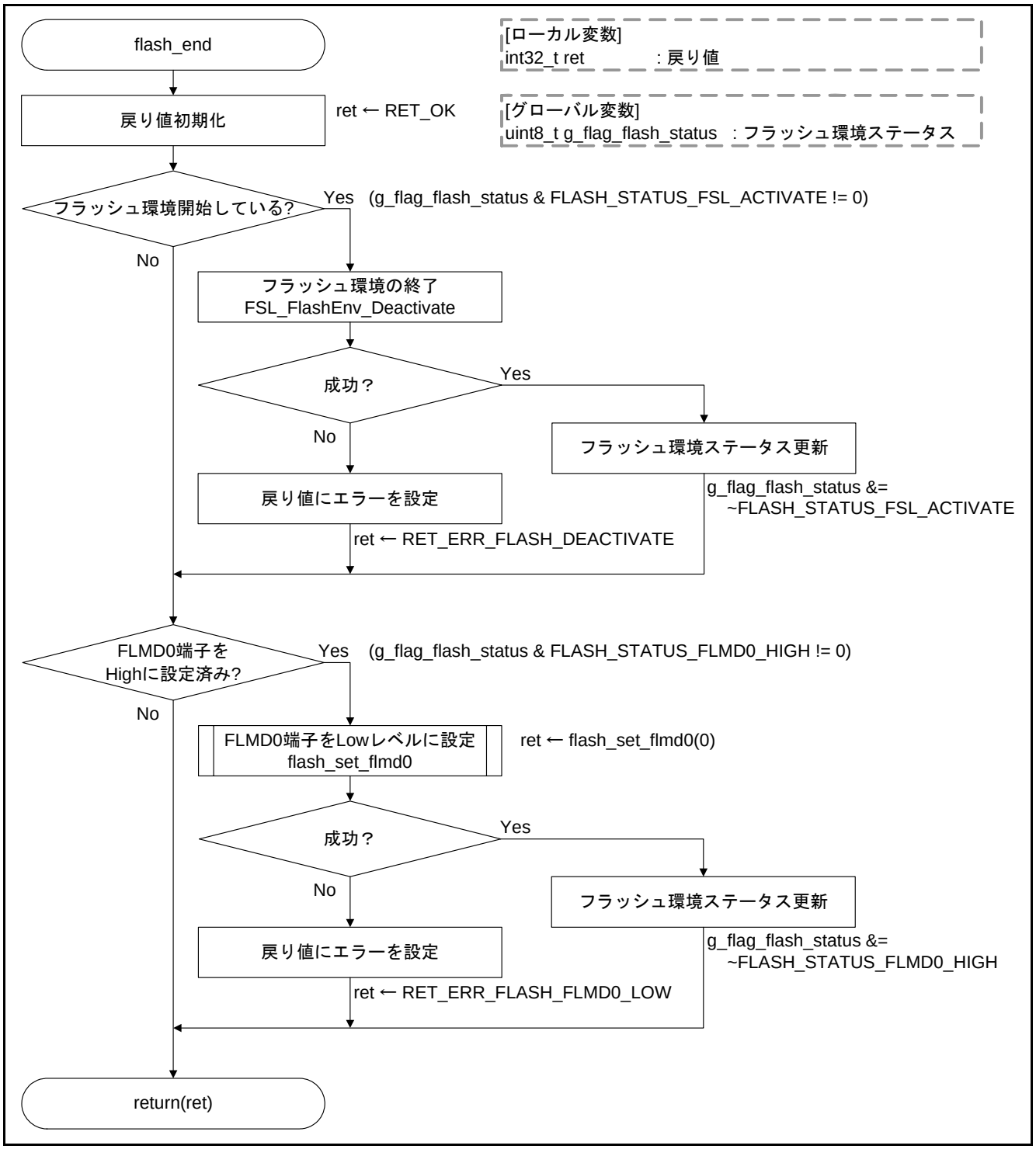


図6.19 フラッシュ環境終了処理

6.7.15　FLMD0 端子レベル設定処理

図 6.20に FLMD0 端子レベル設定処理のフローチャートを示します。

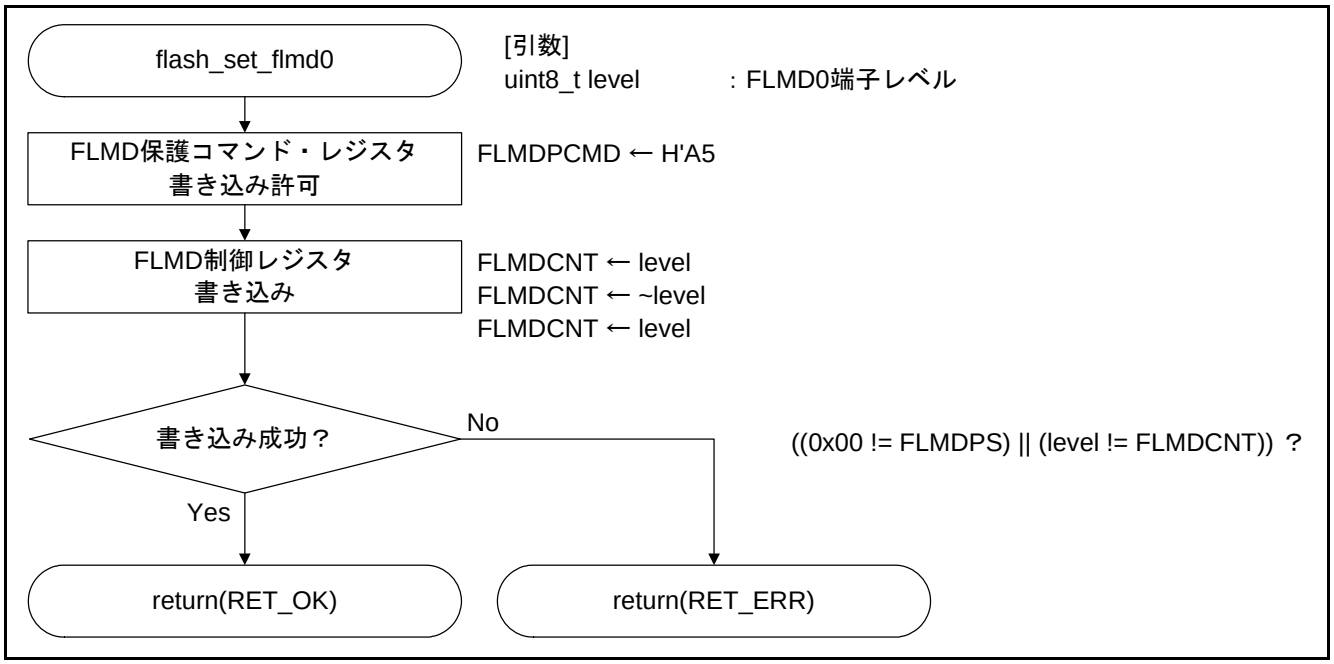


図6.20　FLMD0 端子レベル設定処理

6.7.16　受信データ格納処理

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。

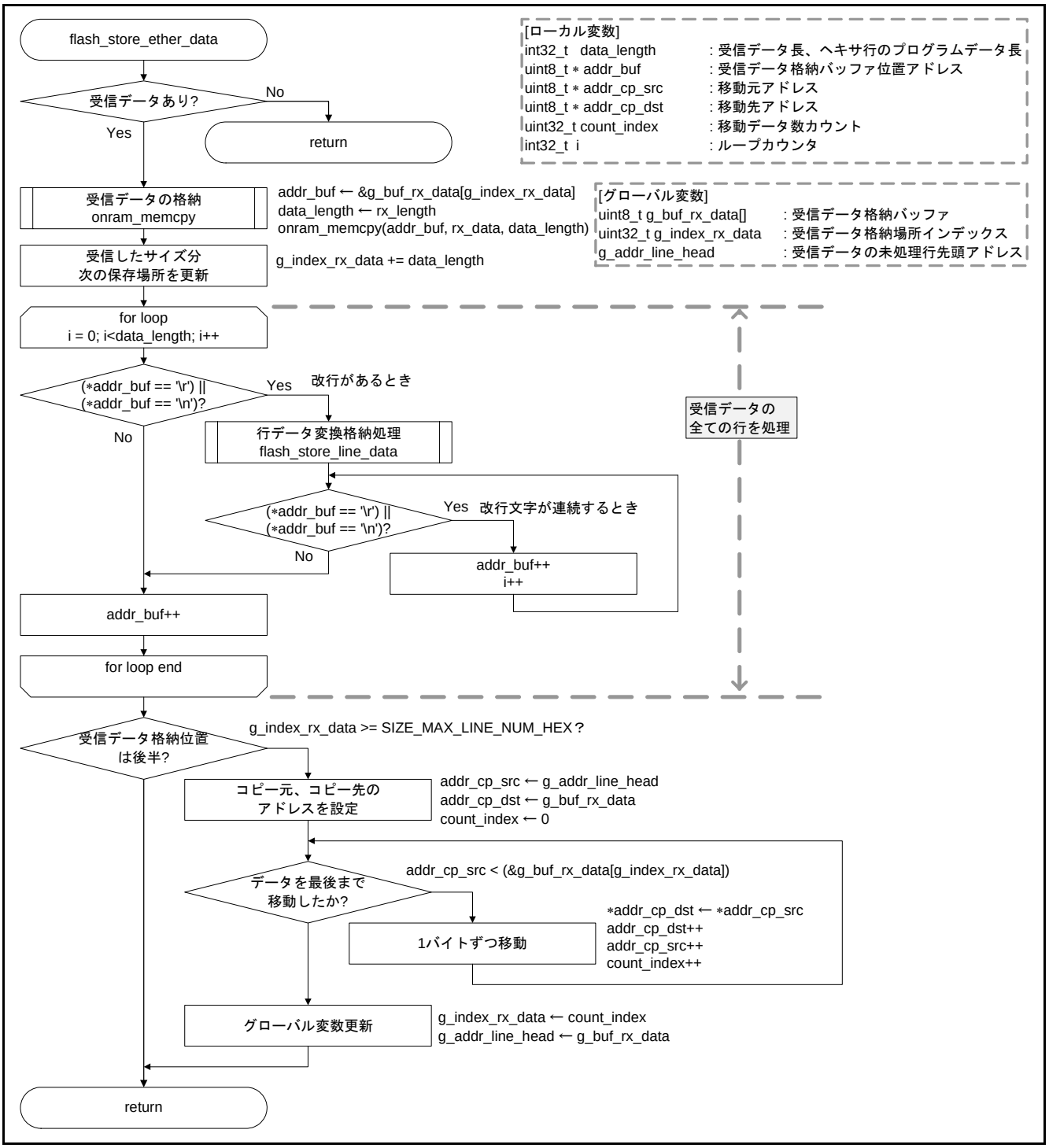


図6.21　受信データ格納処理

6.7.17　行データ変換格納処理

図 6.22、図 6.23に 行データ変換格納処理のフローチャートを示します。

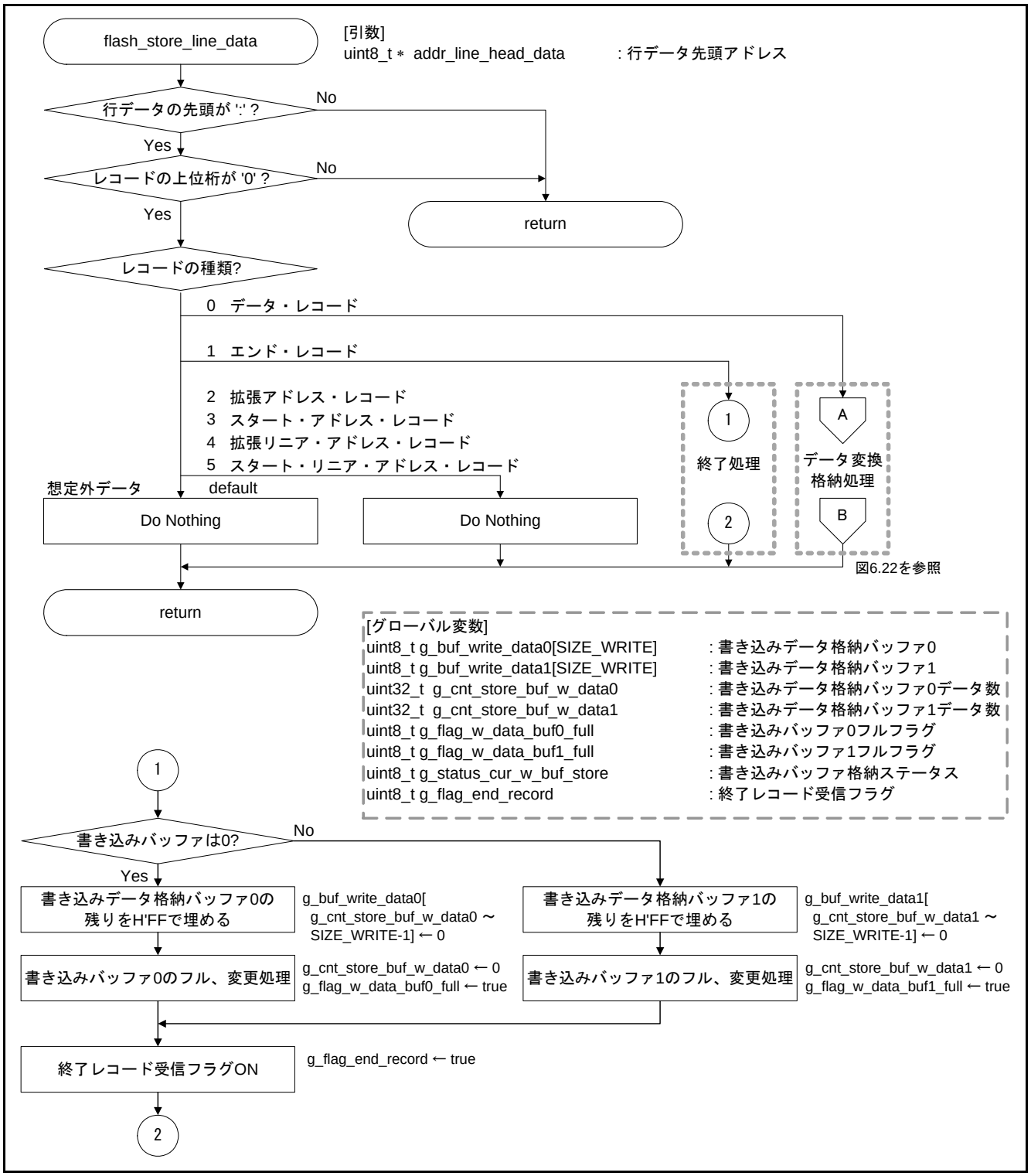


図6.22　行データ変換格納処理（1／2）

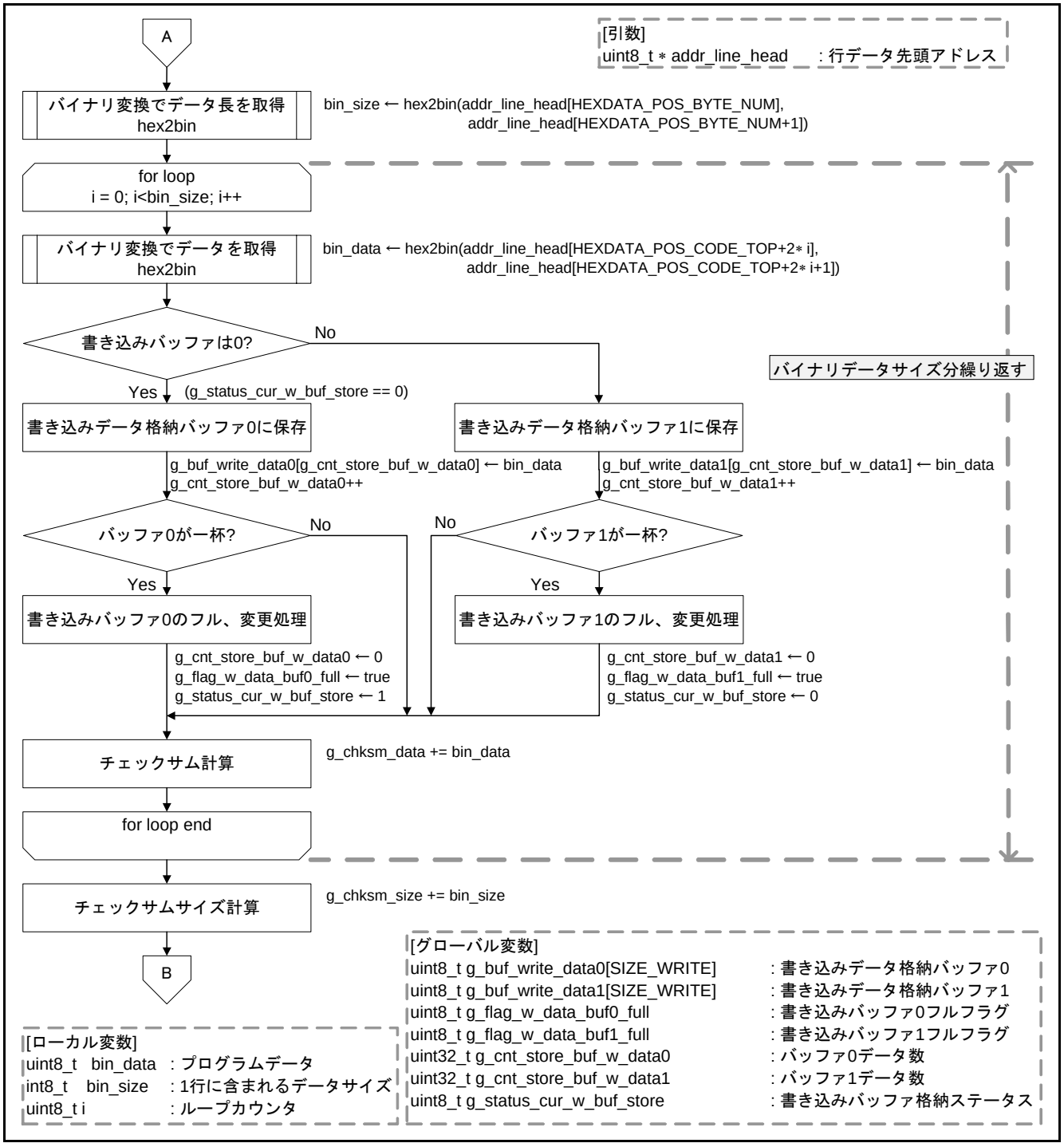


図6.23　行データ変換格納処理（2／2）

6.7.18　テキストバイナリ変換処理

図 6.24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。

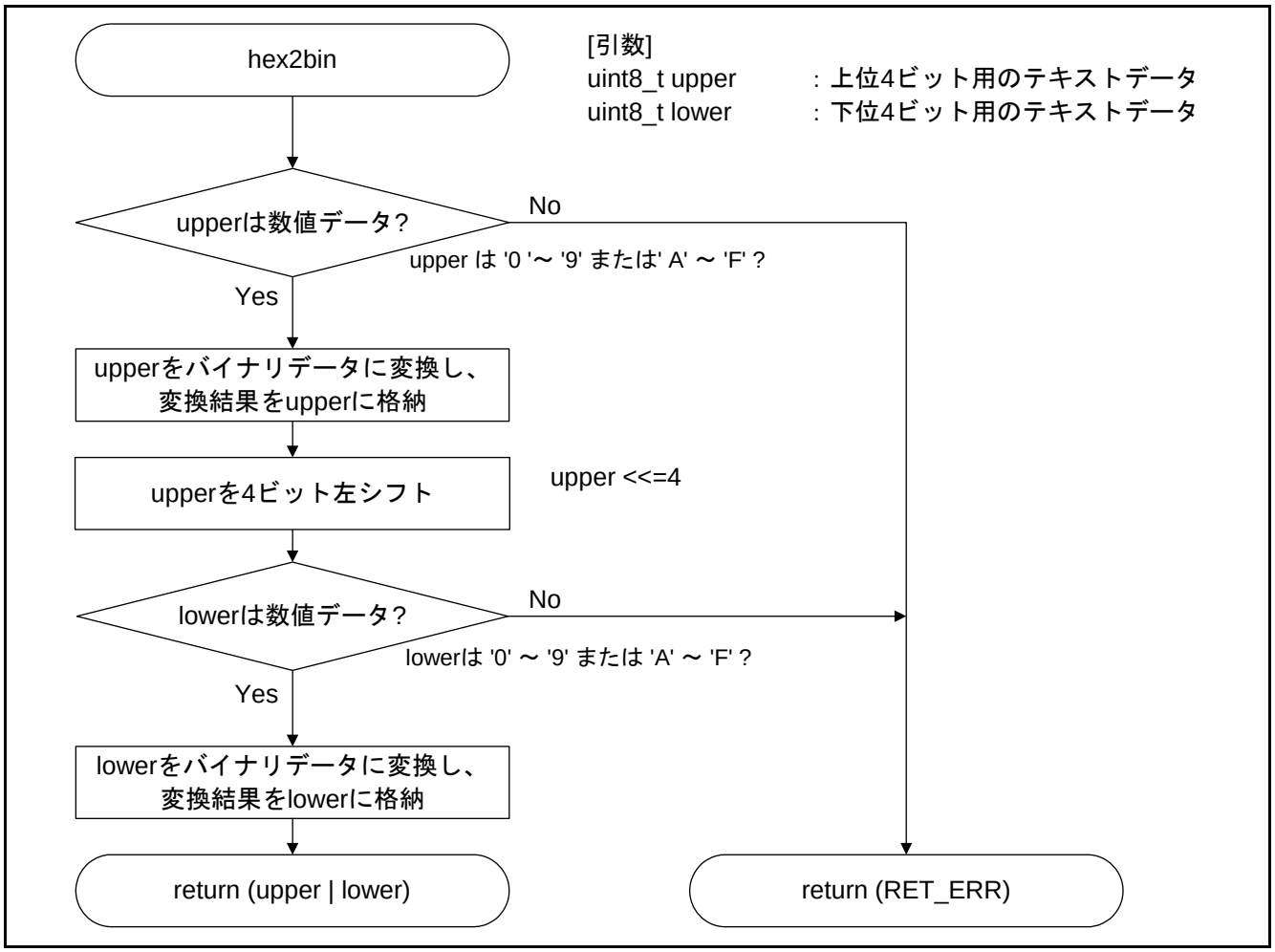


図6.24　テキストバイナリ変換処理

6.7.19　定周期 LED 点滅用 TAUA0 初期化処理（書き換え領域、予備領域サンプル関数）

図 6.25に 定周期LED点滅用TAUA0 初期化処理（書き換え領域、予備領域サンプル関数）のフローチャートを示します。

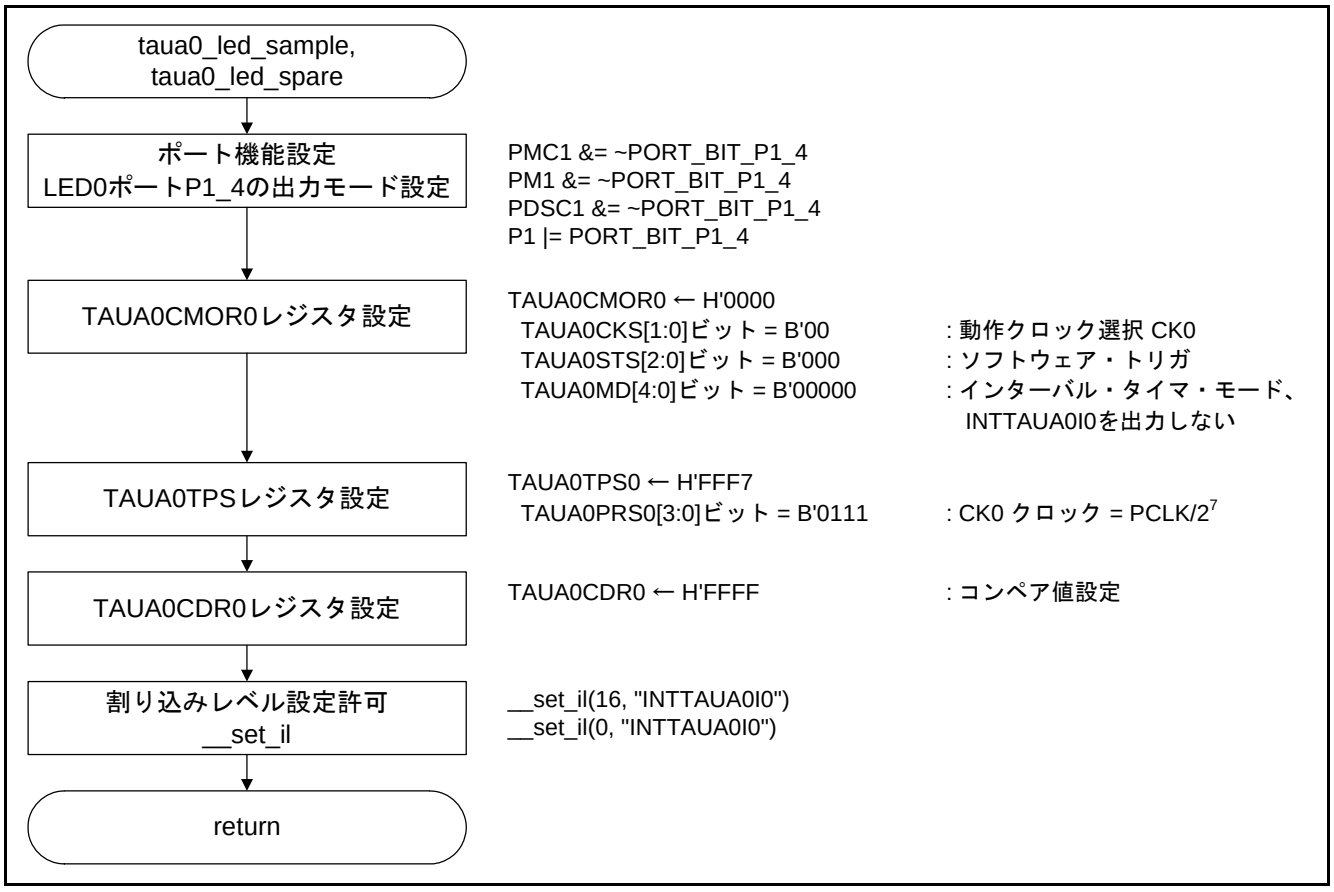
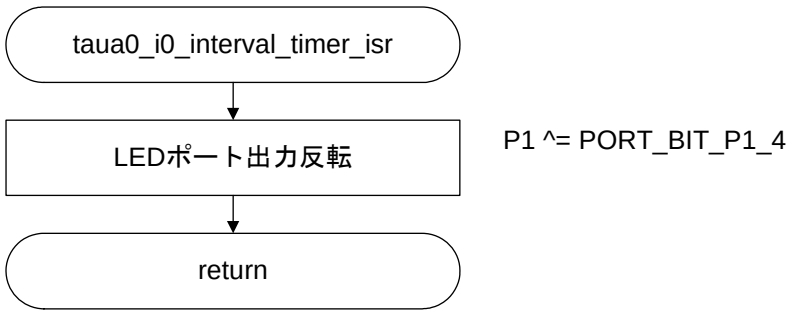


図6.25　定周期 LED 点滅用 TAUA0 初期化処理（書き換え領域、予備領域サンプル関数）

6.7.20　TAUA インターバル・タイマ割り込み処理

図 6.26に TAUA0 インターバル・タイマ割り込み処理のフローチャートを示します。



【注】応用する際は「4.2.6 FSL使用中に発生する割り込み処理の注意事項」を確認してください。

図6.26　TAUA0 インターバル・タイマ割り込み処理

6.7.21　TCP/IP 使用 uIP 設定初期化処理

図 6.27に TCP/IP使用uIP設定初期化処理のフローチャートを示します。

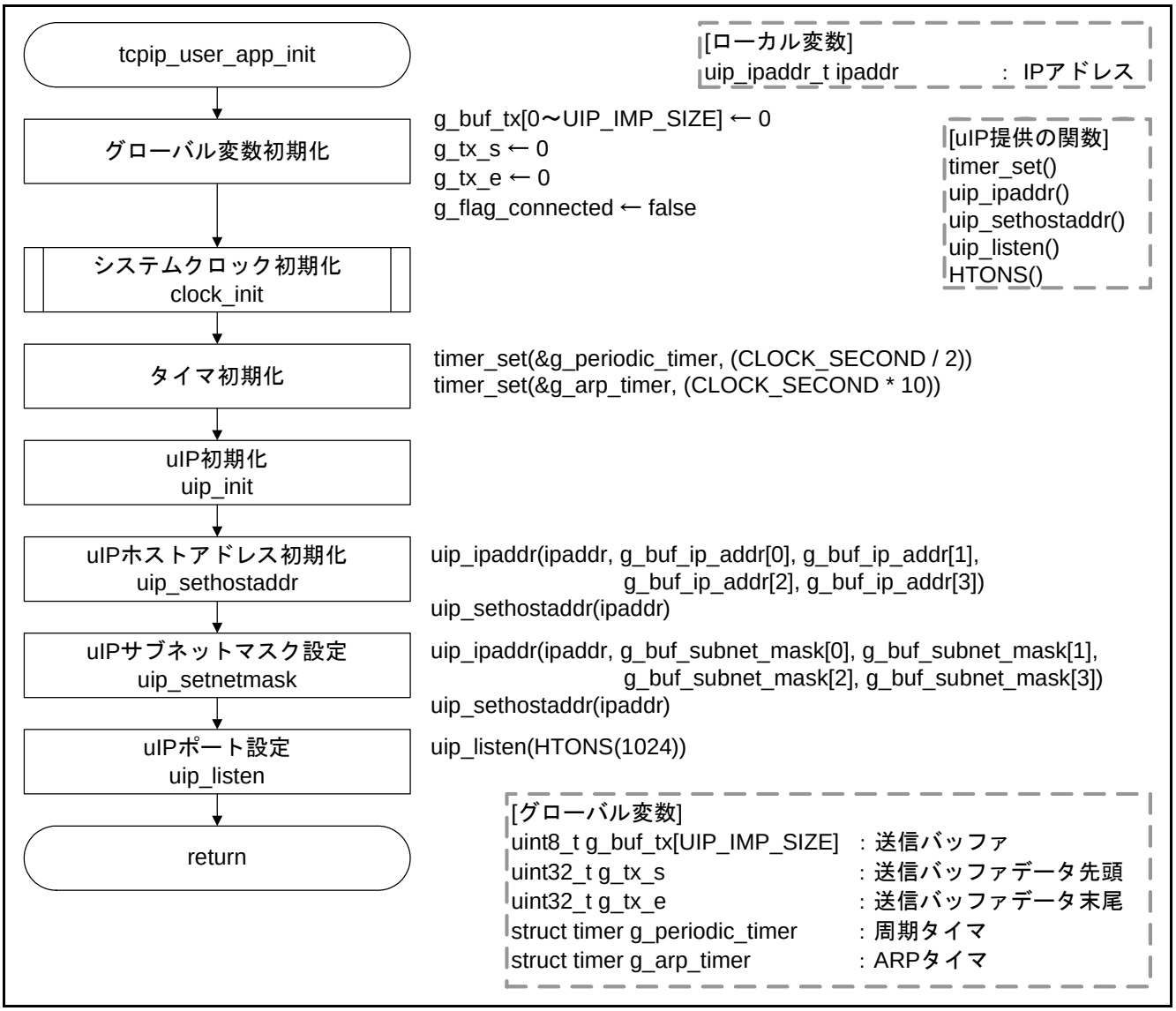


図6.27　TCP/IP 使用 uIP 設定初期化処理

6.7.22　TCP/IP イベント処理

図 6.28に TCP/IPイベント処理のフローチャートを示します。

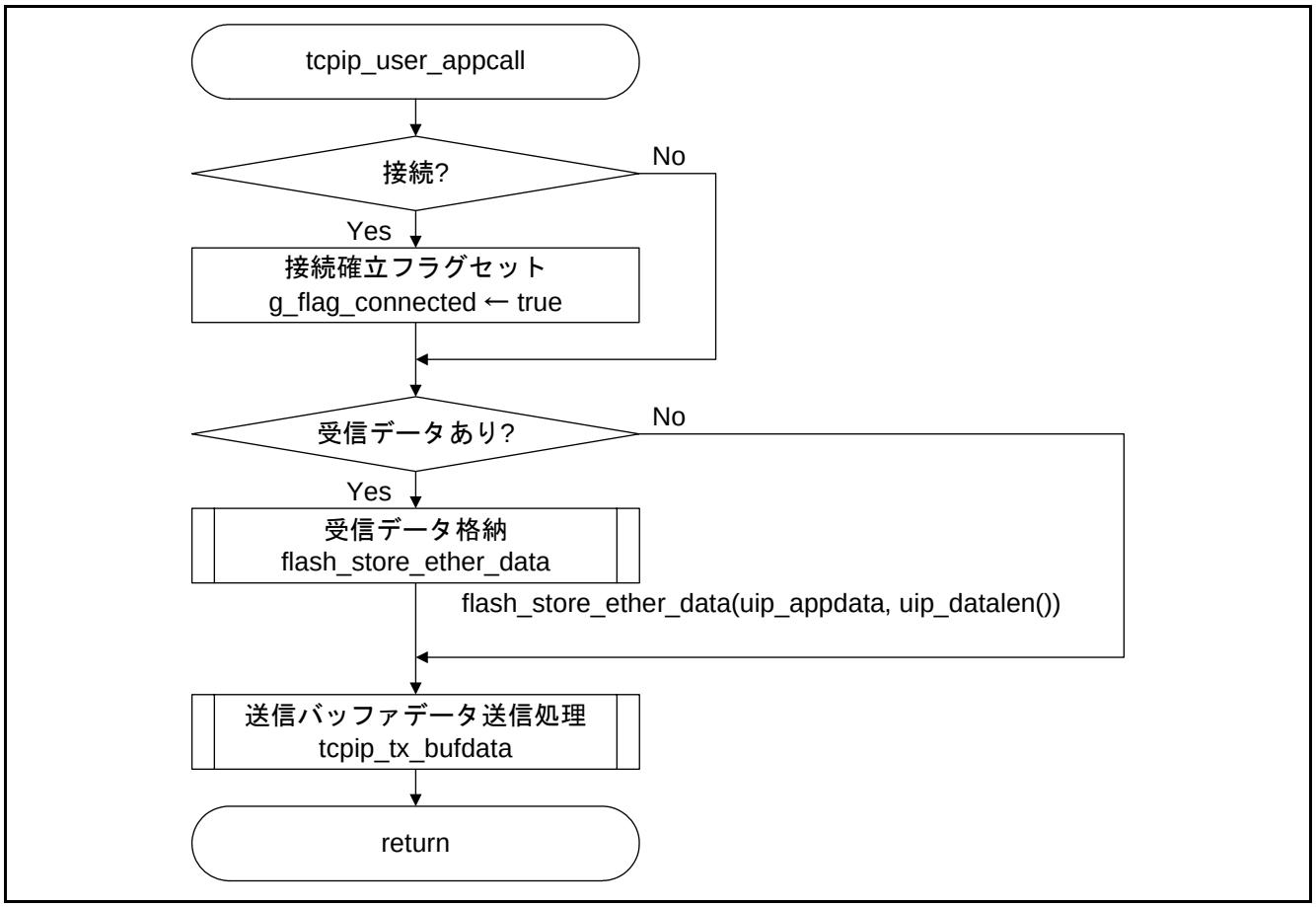


図6.28　TCP/IP イベント処理

6.7.23　TCP/IP 送信メッセージ格納処理

図 6.29に TCP/IP送信メッセージ格納処理のフローチャートを示します。

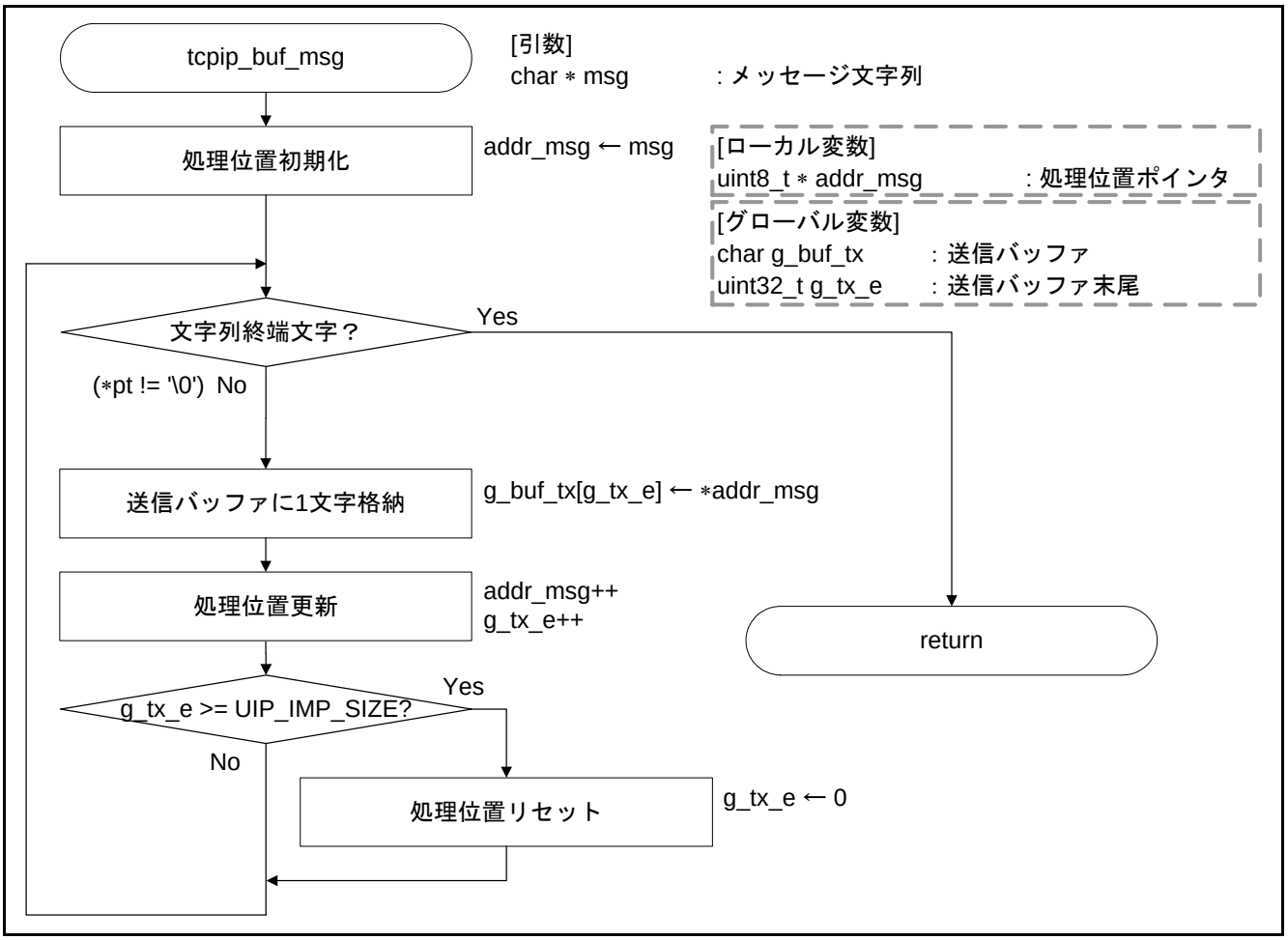


図6.29　TCP/IP 送信メッセージ格納処理

6.7.24　TCP/IP バッファデータ送信処理

図 6.30に TCP/IPバッファデータ送信処理のフローチャートを示します。

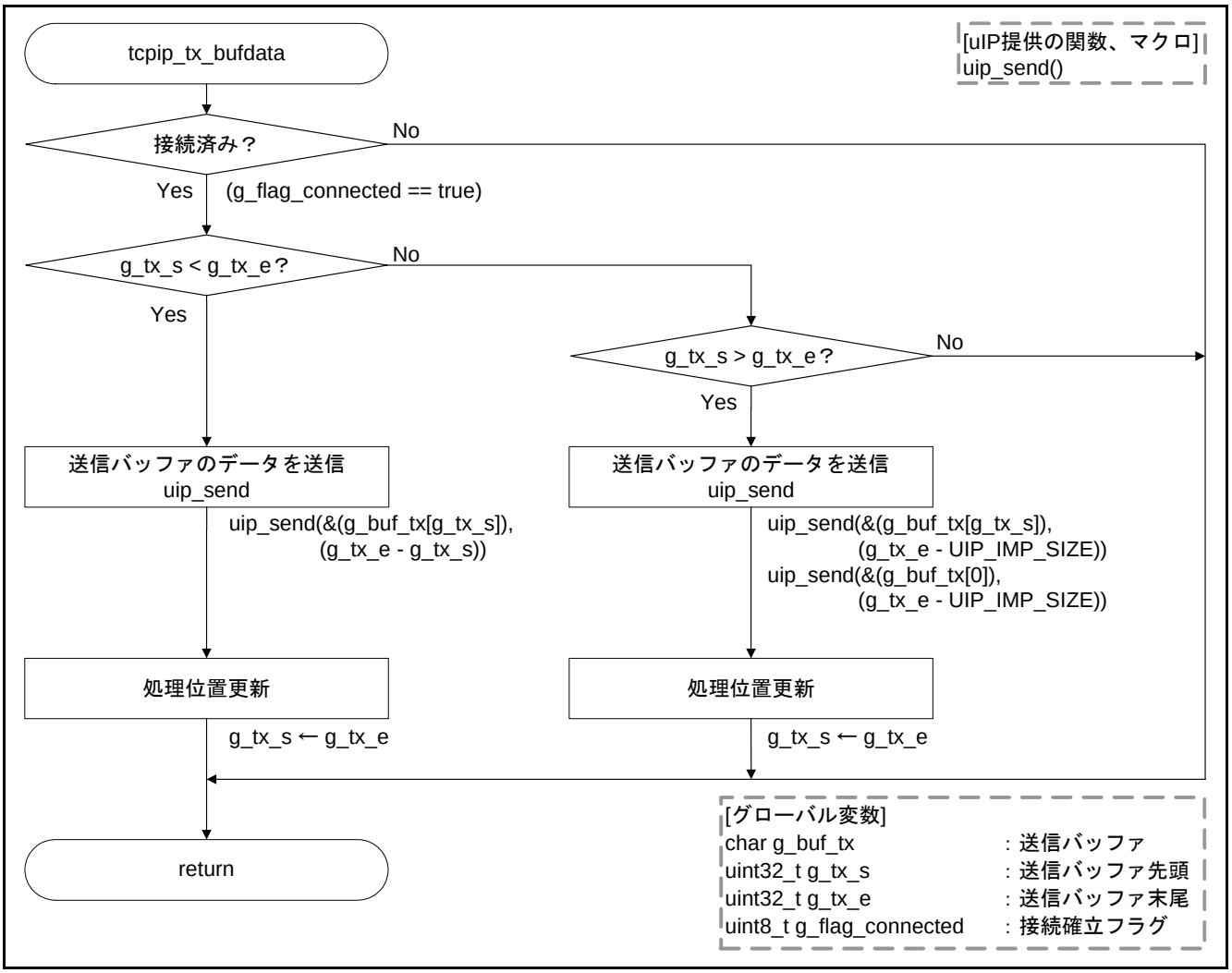


図6.30　TCP/IP バッファデータ送信処理

6.7.25　TCP/IP ポーリング処理

図 6.31に TCP/IPポーリング処理のフローチャートを示します。

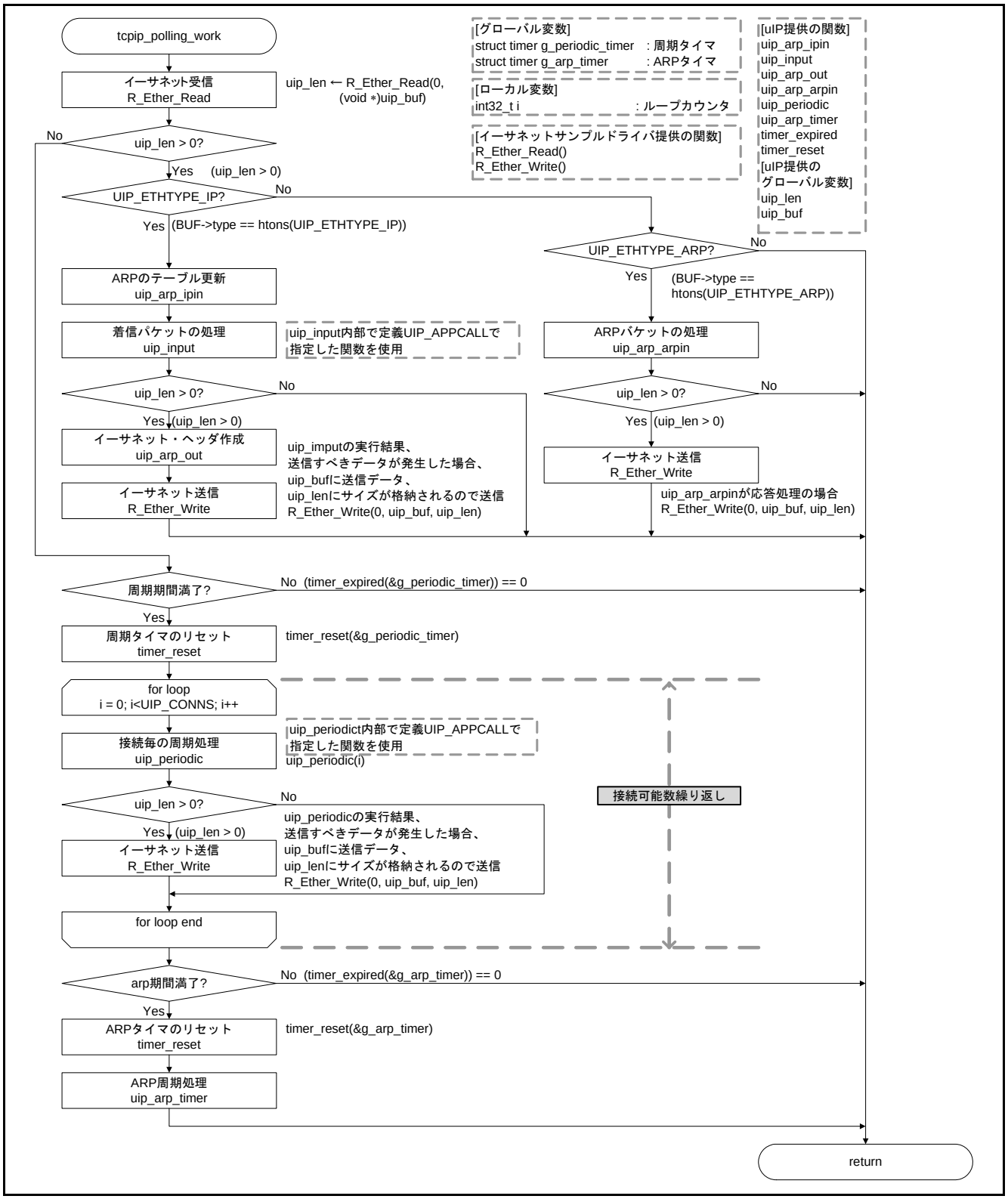


図6.31　TCP/IP ポーリング処理

6.7.26　RAM 上配置 memcpy 関数処理

図 6.32に RAM上配置memcpy関数処理のフローチャートを示します。

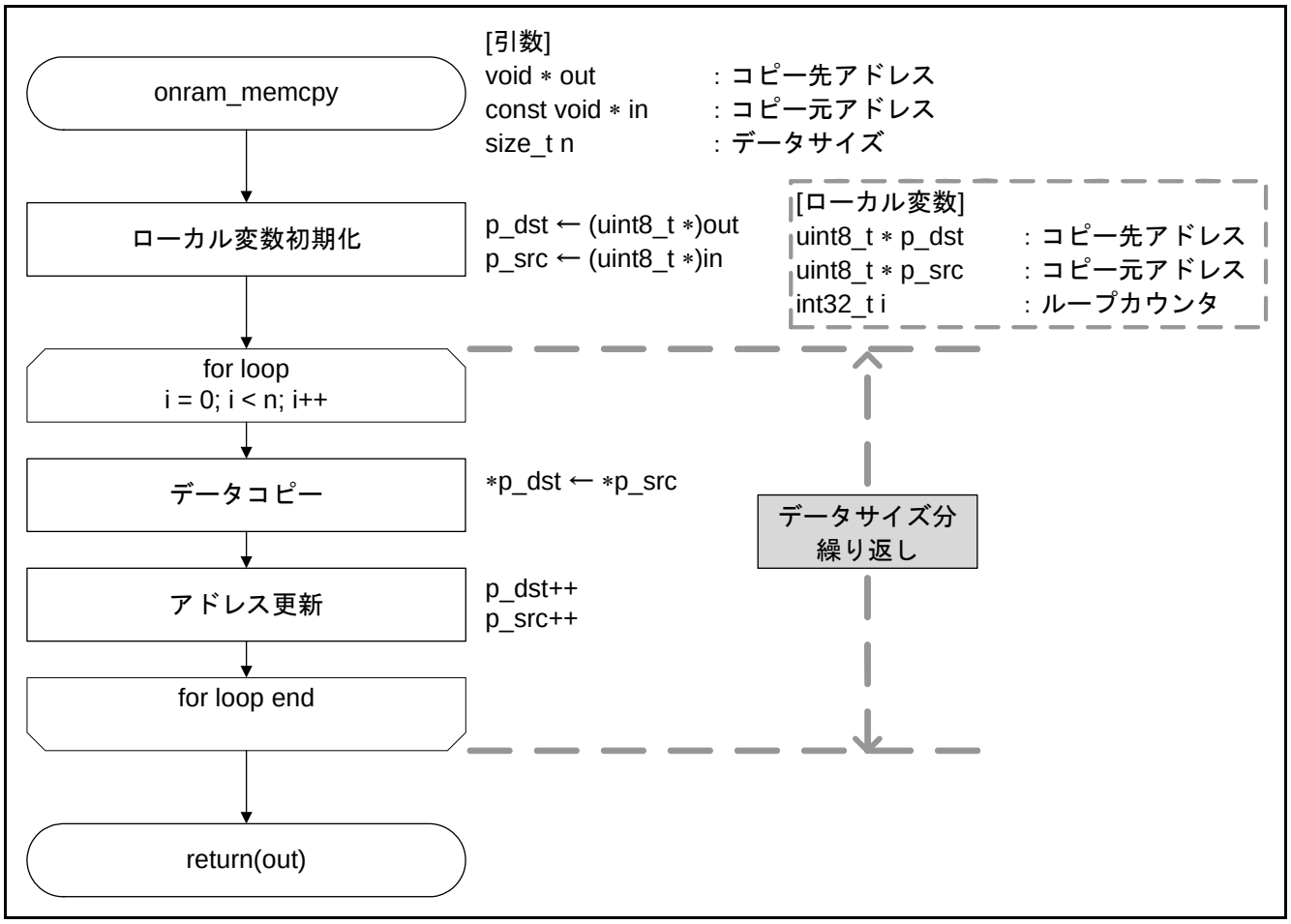


図6.32　RAM 上配置 memcpy 関数処理

6.7.27　RAM 上配置 memcmp 関数処理

図 6.33に RAM上配置memcmp関数処理のフローチャートを示します。

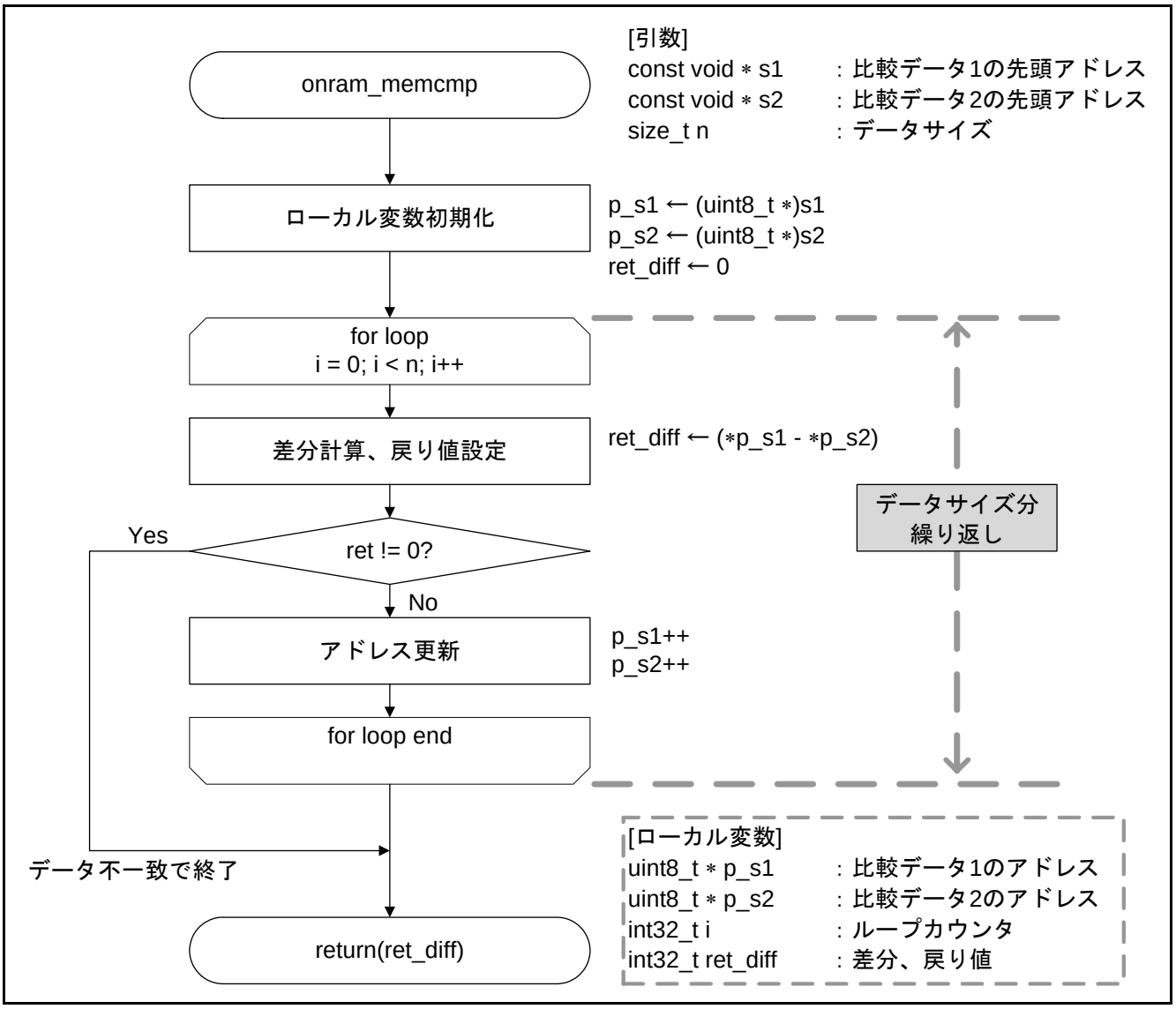


図6.33　RAM 上配置 memcmp 関数処理

6.7.28　RAM 上配置 memset 関数処理

図 6.34に RAM上配置memset関数処理のフローチャートを示します。

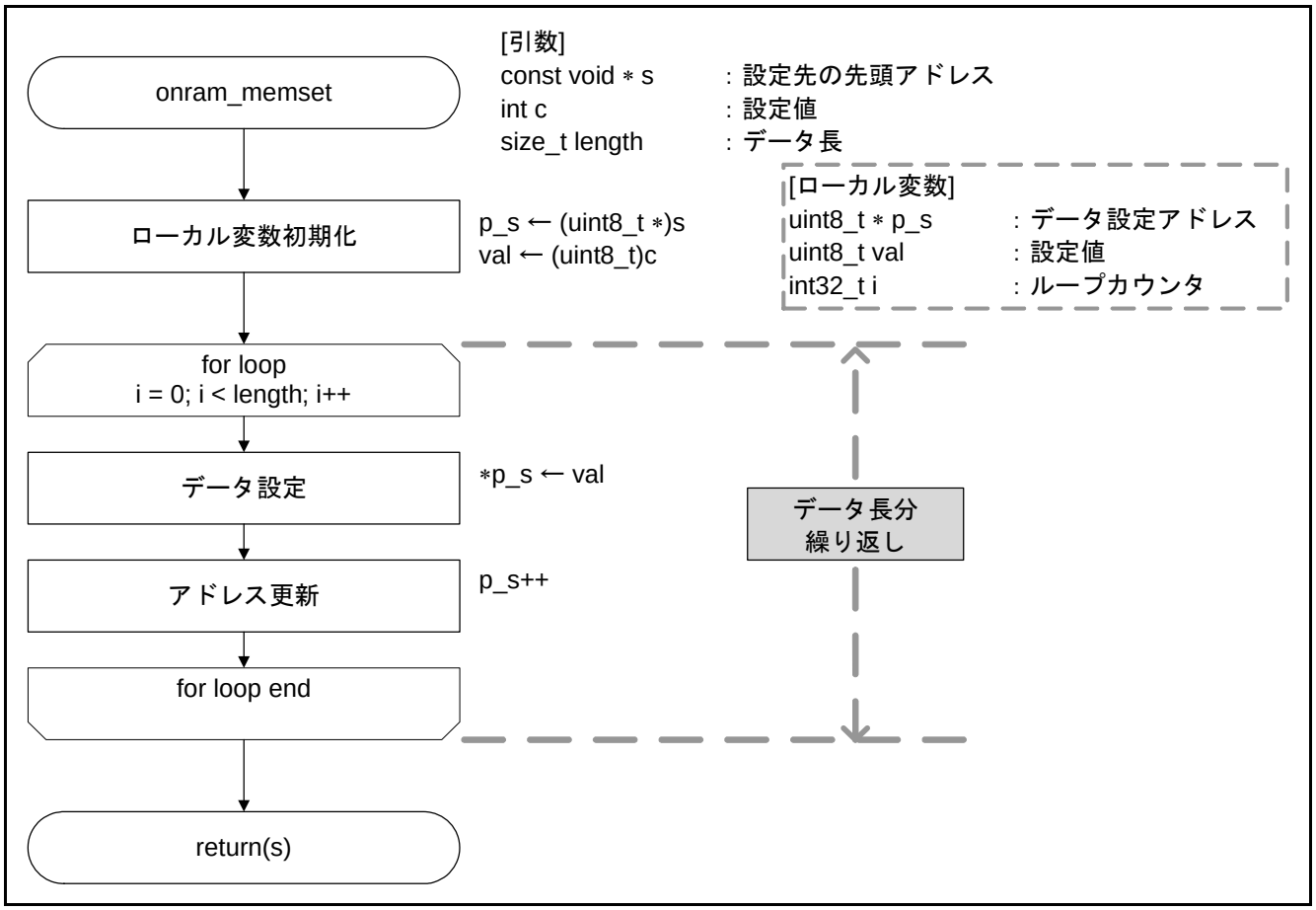


図6.34　RAM 上配置 memset 関数処理

6.7.29　システムクロックタイム初期化処理

図 6.35に クロック時間初期化処理のフローチャートを示します。

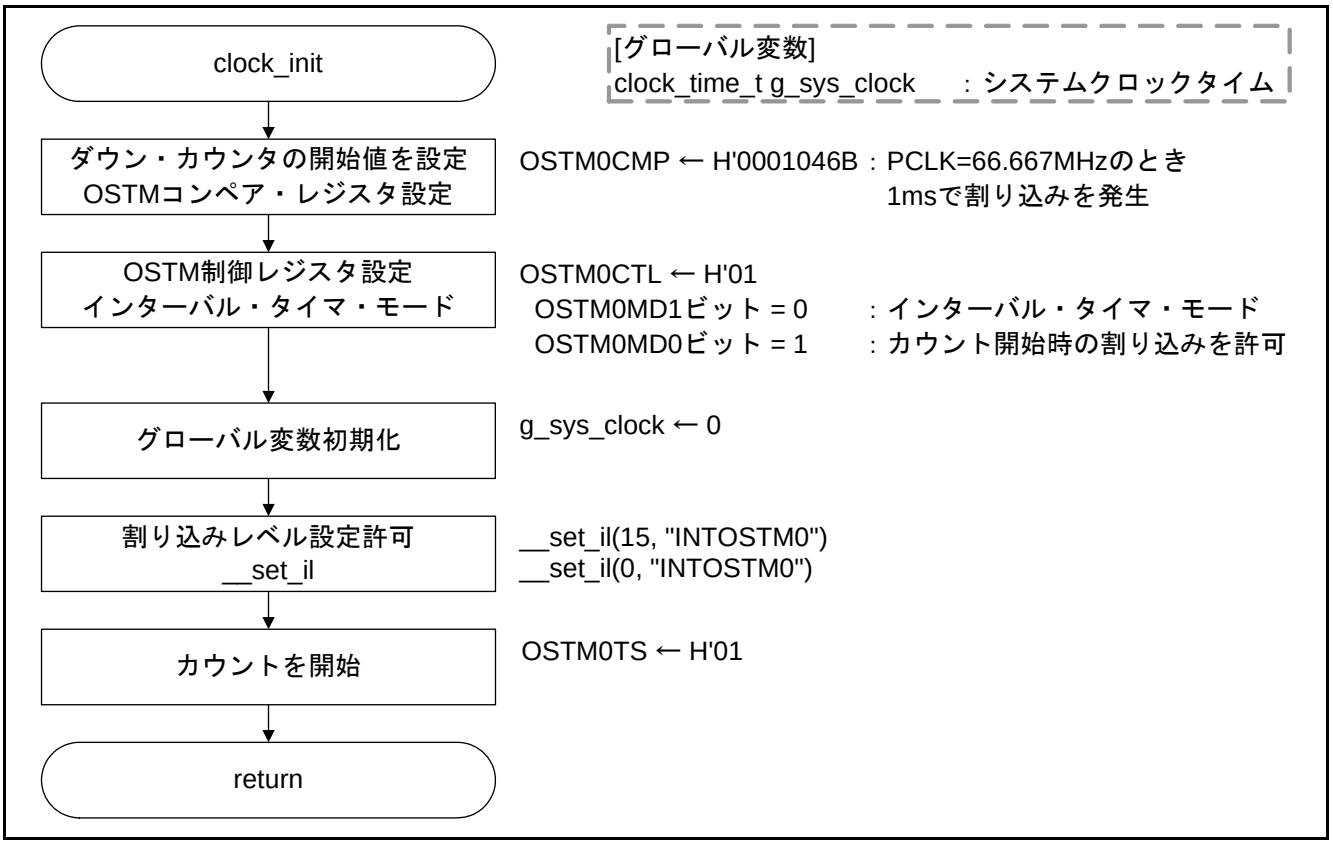


図6.35　クロック時間初期化処理

6.7.30　システムクロックタイム取得処理

図 6.36に クロック時間取得処理のフローチャートを示します。

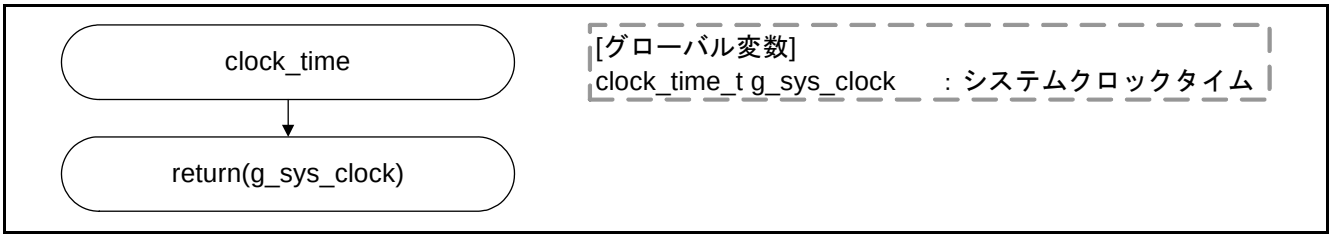
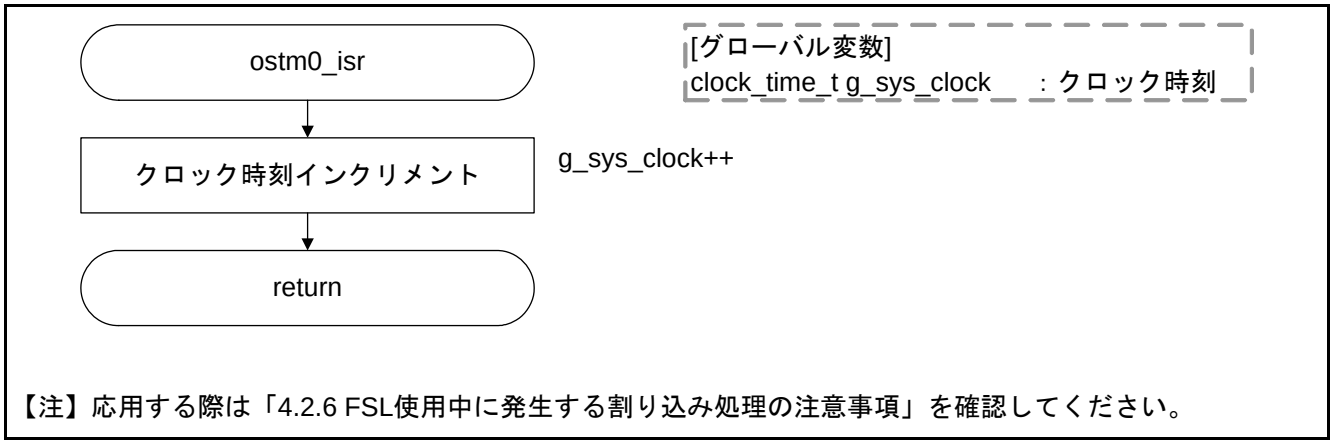


図6.36　クロック時間取得処理

6.7.31　OS タイマ割り込み処理

図 6.37に OSタイマ割り込み処理のフローチャートを示します。



【注】応用する際は「4.2.6 FSL使用中に発生する割り込み処理の注意事項」を確認してください。

図6.37　OS タイマ割り込み処理

6.7.32　イーサネット・コントローラ用ポート機能初期化処理

図 6.38にイーサネット・コントローラ用ポート機能初期化処理のフローチャートを示します。

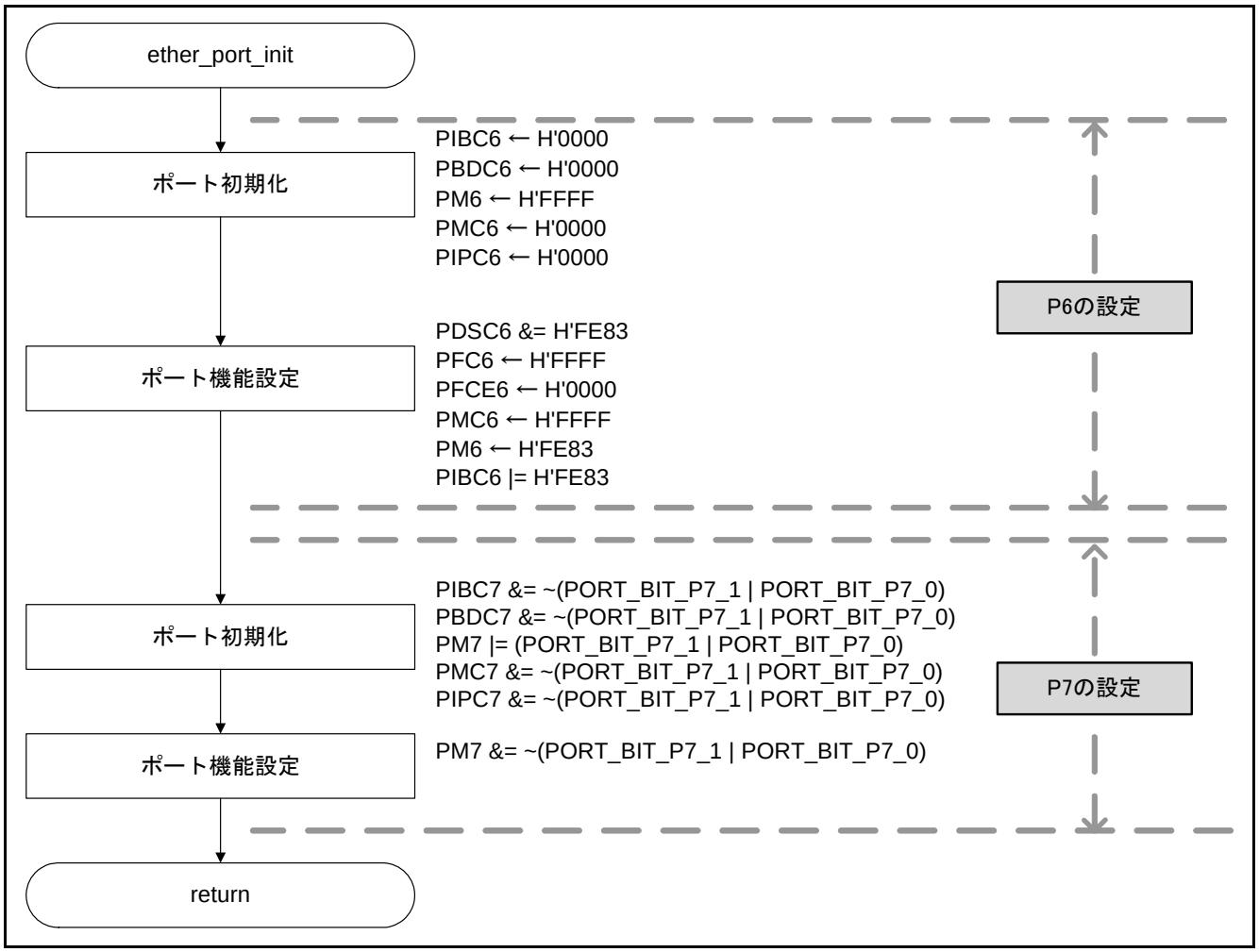


図6.38　イーサネット・コントローラ用ポート機能初期化処理

6.7.33　H バス初期化処理

図 6.39に Hバス初期化処理のフローチャートを示します。

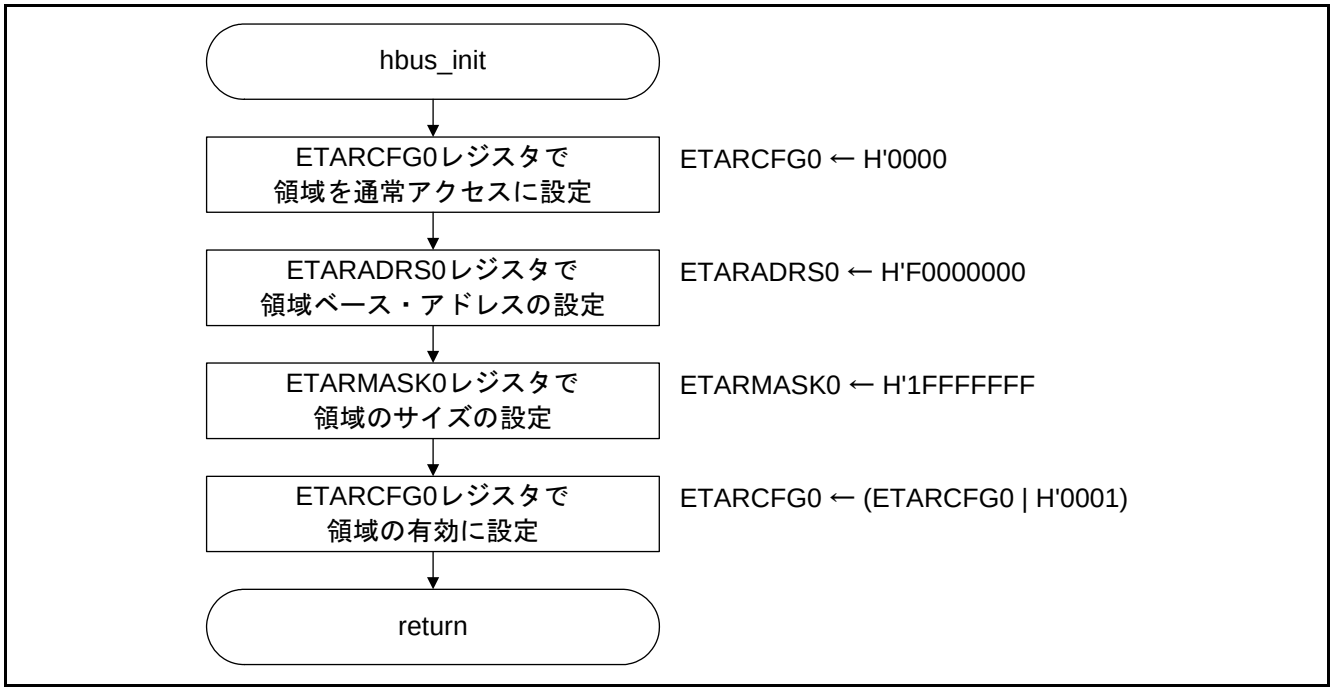


図6.39　H バス初期化処理

6.7.34　ミリ秒単位スリープ処理

図 6.40に ミリ単位スリープ処理のフローチャートを示します。

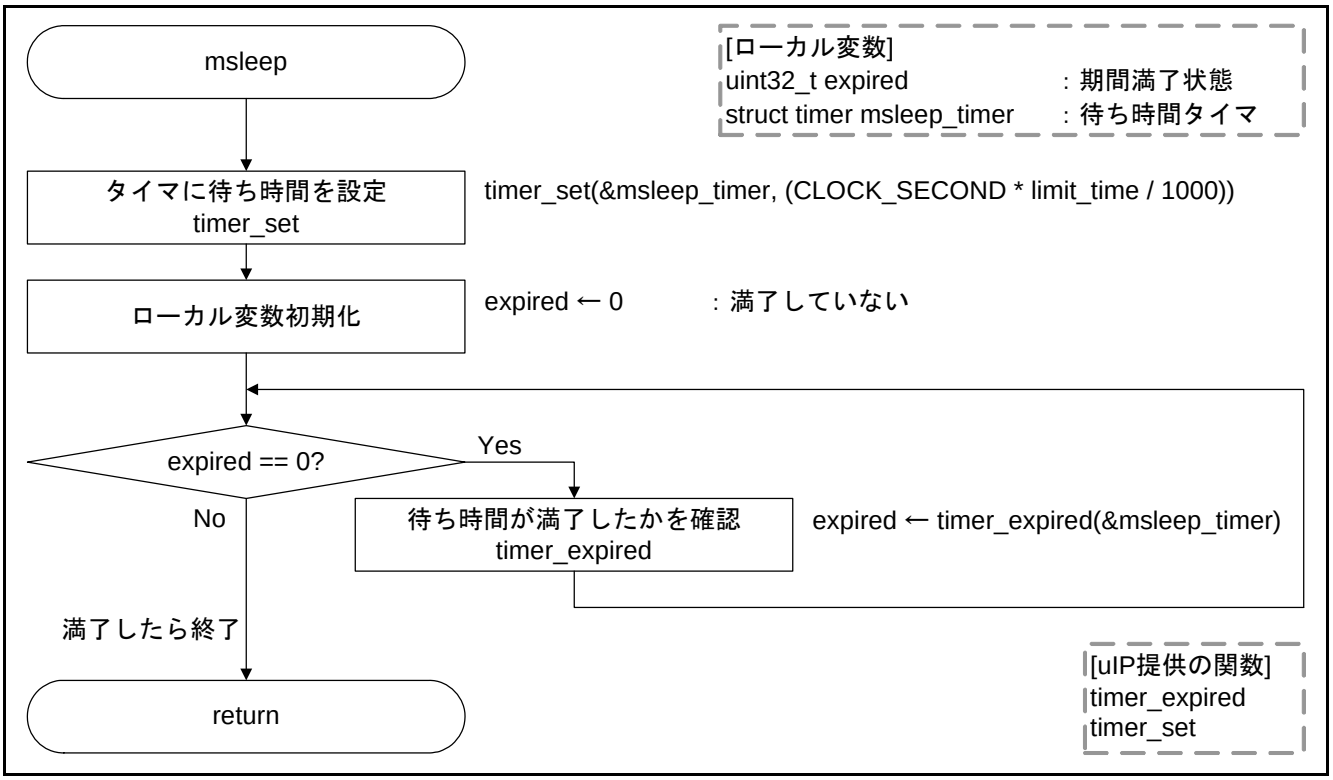


図6.40　ミリ単位スリープ処理

7.　操作概要

本サンプルコードでは、イーサネット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。ここではイーサネット通信ホスト機器として PC を使用して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。

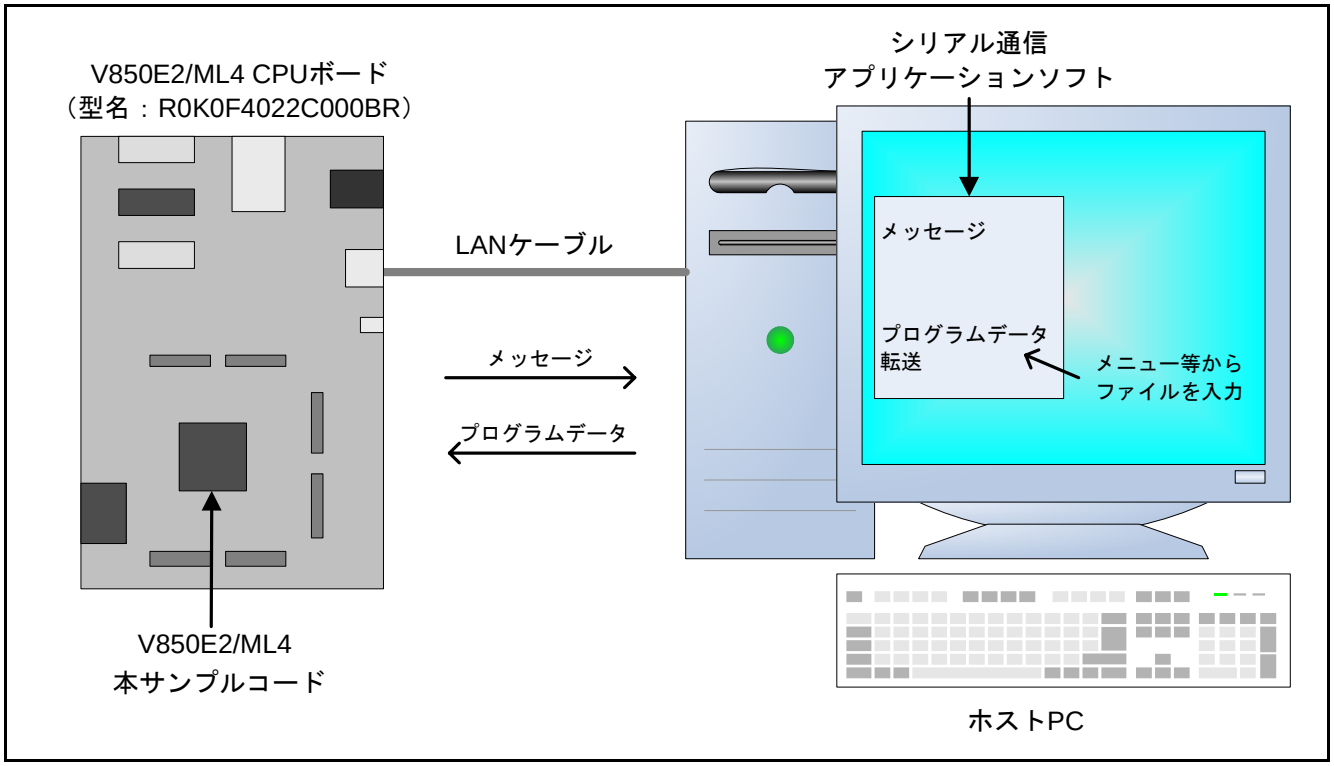


図7.1　サンプルコード使用のためのハードウェア構成例

CPU ボード（Ethernet コネクタ（J2））とホスト PC を、LAN ケーブルで接続してください。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。

上記の設定完了後、本サンプルコードの CPU ボードの電源を入れてください。ホスト PC が CPU ボードを認識してから、VT100 互換端末エミュレータ（シリアル通信アプリケーションソフト）を起動してください。また、本サンプルコードを実行する際は、ホスト PC と CPU ボードが同じネットワークになるように、あらかじめホスト PC の IP アドレスとサブネットマスクを設定する必要があります。

端末エミュレータ起動後、接続の設定を行います。端末エミュレータのホストアドレス（通信相手のアドレス）には CPU ボードの IP アドレスを設定し、ポート番号には 1024 を指定してください。接続タイプやサービスが設定可能な端末エミュレータを使用する場合は、SSH や Telnet などのサービスを設定せずに、TCP/IP、IPv4 の Raw データで通信を行うように設定してください。

表 7.1、表 7.2、表 7.3に接続に必要な各種設定例を示します。

表7.1　設定例：IP アドレス（CPU ボード）とサブネットマスク用変数の（初期）設定値

変数名	設定値
g_buf_ip_addr[]	{192, 168, 1, 76}
g_buf_subnet_mask[]	{255, 255, 255, 0}

【注】設定値は環境に合わせて適宜変更してください。

表7.2　設定例：ホスト PC のローカルエリアネットワーク設定

TCP/IPv4 設定項目	設定値
IP アドレス	192. 168. 1. 77
サブネットマスク	255. 255. 255. 0（配列変数 g_buf_subnet_mask の初期値）

表7.3　設定例：端末エミュレータ上の接続設定

設定項目	設定値
接続方法	TCP/IP
ホストアドレス（CPU ボードの IP アドレス）	192. 168. 1. 76（配列変数 g_buf_ip_addr の初期値）
ポート番号	1024
サービス・プロトコル	IPv4

本サンプルコードを実行（CPU ボードを起動）すると、V850E2/ML4 は予備領域に格納したプログラムを実行し、CPU ボード上の LED を一定周期で点滅させます。この状態でホスト PC 上の端末エミュレータ接続を確立すると、V850E2/ML4 はホスト PC に対し「Generate INTP2 interrupt for transition to flash programming event.」というメッセージを送信します。

端末エミュレータ上で上記メッセージ表示を確認後、CPU ボード上の INTP2 外部割り込み用スイッチ（SW5）を押すと、V850E2/ML4 はホスト PC に対し「--> INTP2 detected!」というメッセージを送信します。INTP2 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、最初にアップデート領域を消去します。消去が完了すると、V850E2/ML4 はホスト PC 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 PC からのデータ受信待ち状態に入ります。

ホスト PC 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などを利用してください。該当ファイル（v850e2ml4_sample_host_send.hex など）を選択して送信すると、本サンプルコードは受信したファイルをプログラムデータに変換し、フラッシュ・メモリ領域のプログラムを書き換えます。

書き換え（アップデート）完了後、V850E2/ML4 はホスト PC 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。メッセージ表示を確認後、端末エミュレータを終了してから対象ボードを再起動してください。

CPU ボードを再起動すると、CPU 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 総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 総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（R20UT0778JJ0100）
（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。）

uIP 組み込み TCP/IP スタック uip-1.0 リファレンスマニュアル

"The uIP Embedded TCP/IP Stack" The uIP 1.0 Reference Manual (June 2006, Adam Dunkels, Swedish Institute of Computer Science)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

http://japan.renesas.com

お問合せ先

http://japan.renesas.com/contact/

改訂記録	V850E2/ML4　アプリケーションノート　イーサネット･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	2013.08.06	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理

【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。

CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

2. 電源投入時の処置

【注意】電源投入時は，製品の状態は不定です。

電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。

外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。

同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. リザーブアドレス（予約領域）のアクセス禁止

【注意】リザーブアドレス（予約領域）のアクセスを禁止します。

アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

4. クロックについて

【注意】リセット時は、クロックが安定した後、リセットを解除してください。

プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。

リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

5. 製品間の相違について

【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。

同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

RENESAS

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/

Colophon 2.1