RL78/G13

フラッシュ・セルフ・プログラミング 実行編

R01AN0718JJ0102
Rev.1.02
2013.02.15

要旨

このアプリケーションノートは，RL78/G13 マイクロコントローラで使用するフラッシュ・セルフ・プログラミング・ライブラリ Type01 の機能を理解し，それを用いたアプリケーション・システムを設計するユーザを対象としています。

このアプリケーションノートは，RL78 マイクロコントローラのコード・フラッシュ・メモリの書き換えを行うために使用するフラッシュ・セルフ・プログラミング・ライブラリ Type01 の使用方法を理解していただくことを目的としています。

動作確認デバイス

RL78/G13 (R5F100LE)

目 次

はじめに

対 象 者　本アプリケーションノートは，RL78/G13のフラッシュ・セルフ・プログラミング機能を使用し，アプリケーション・システムの設計を行うユーザを対象としています。

目 的　本アプリケーションノートは，RL78/G13のフラッシュ・セルフ・プログラミング機能を使用し，フラッシュ・メモリの書き換えを行うためのプログラム開発を行う方法を，ユーザに理解していただくことを目的としています。

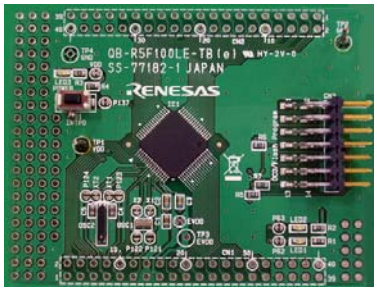
構 成　本アプリケーションノートは，大きく分けて次の内容で構成しています。

・概 要
・フラッシュ・セルフ・プログラミング・ライブラリ
・プログラムの書き換え例
・付録

本アプリケーションノートでは評価ボード「QB-R5F100LE-TB」を使用し，フラッシュ・セルフ・プログラミング機能を使用したプログラム例を紹介しています。そのため，本アプリケーションノートに付属しているサンプル・プログラムを使用される場合には，QB-R5F100LE-TBをご用意いただく必要があります。

また，QB-R5F100LE-TBにて評価を行うためには別途E1エミュレータや，UART-RS-232C変換機，外部電源等が必要です。E1エミュレータやQB-R5F100LE-TBのご購入，及びご質問等につきましては，お近くの販売店にご相談ください。

図　評価ボード QB-R5F100LE-TB



凡 例

データ表記の重み	：左が上位桁，右が下位桁
注	：本文中につけた注の説明
注意	：気をつけて読んでいただきたい内容
備考	：本文の補足説明
数の表記	：2進数…××××または××××B
	10進数…××××
	16進数…××××H

関連資料

IDE/タイトル	ドキュメントNo.
RL78ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01 ユーザーズマニュアル[注1]	R01US0050
RL78ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01 Ver.2.20 リリースノート[注1]	R20UT0777
CubeSuite+ V1.03.00 リリースノート[注2]	R20UT2259
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル 起動編[注2]	R20UT2133
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル RL78 設計編[注2]	R20UT2136
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル 解析編[注2]	R20UT2146
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル メッセージ編[注2]	R20UT2147
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル RL78 デバッグ編[注2]	R20UT2145
CubeSuite+ RL78,78K0R コンパイラ CA78K0R V1.50 リリースノート[注2]	R20UT2261
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル RL78, 78K0R コーディング編[注2]	R20UT2140
CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル RL78, 78K0R ビルド編[注2]	R20UT2143

注1. このドキュメントは，フラッシュ・セルフ・プログラミング・ライブラリType01 Ver.2.20のインストール時に，ツール本体と一緒に，PCにインストールしてください。「フラッシュ・セルフ・プログラミング・ライブラリType01 ユーザーズマニュアル」に記載されていない内容に関しては，「フラッシュ・セルフ・プログラミング・ライブラリ Type01 Ver.2.20 リリースノート」を参照してください。

2. このドキュメントは，弊社Webの「統合開発環境 CubeSuite+」のページからダウンロードしてください。

注意 上記関連資料は予告なしに内容の変更を行うことがあります。設計などには，必ず最新の資料をご使用ください。

第1章 概 要

このアプリケーションノートでは，RL78/G13内蔵のコード・フラッシュ・メモリの書き換えを，“RL78ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01 Ver.2.20”を使用し，フラッシュ・セルフ・プログラミングを行うための方法を紹介しています。

“RL78ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01”の詳細については，以下のユーザーズマニュアル，及びリリースノートをご参照ください。

・RL78ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01
ユーザーズマニュアル（資料番号：R01US0050）
・RL78ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01 Ver.2.20
リリースノート（資料番号：R20UT0777）

この章では，主にRL78/G13のフラッシュ・セルフ・プログラミング機能についての概要を説明します。

1. 1　RL78/G13のコード・フラッシュ・メモリについて

RL78/G13は，消去，書き込みを行うことができるコード・フラッシュ・メモリを内蔵しています。以下にRL78/G13のコード・フラッシュ・メモリの特長を示します。

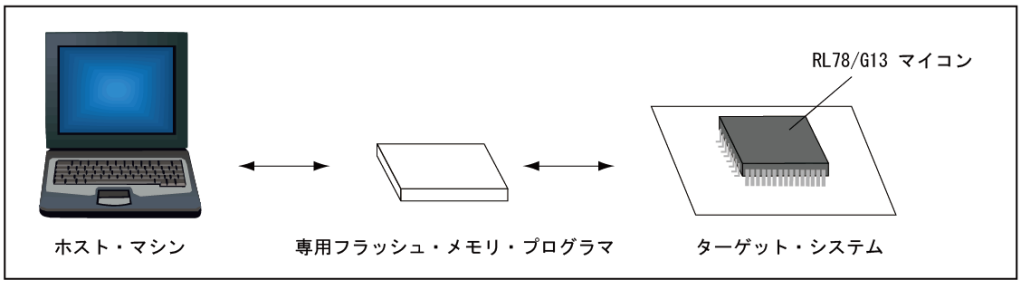
表1－1　コード・フラッシュ・メモリの特長

動作電源	同一電源による消去と書き込みが可能
最小の消去単位	1ブロック（ 1K = 1024バイト ）
最小の書き込み単位	1ワード（ 4バイト ）
セキュリティ	ブロック消去禁止，書き込み禁止，ブート領域の書き換え禁止
	マイコン出荷時の初期設定値はすべて許可
	フラッシュ・シールド・ウインドウ（FSW）により，指定したウインドウ範囲以外の書き込みおよび消去をフラッシュ・セルフ・プログラミング時のみ禁止に設定することが可能
	フラッシュ・セルフ・プログラミング・ライブラリにより設定変更可能

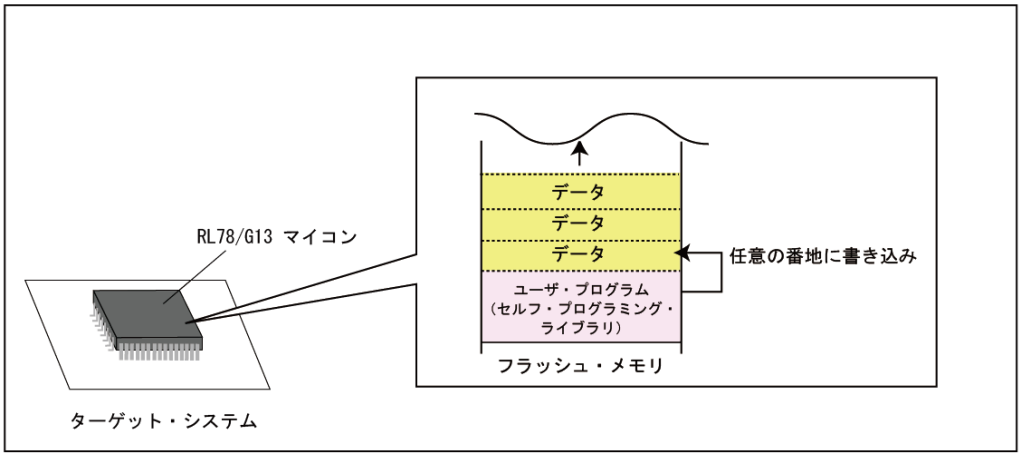
備考　ブート領域の書き換え禁止とFSW以外のセキュリティ設定は，フラッシュ・セルフ・プログラミング時は無効となります。

RL78/G13のコード・フラッシュ・メモリは，基板に実装した状態で書き換えることができます。コード・フラッシュ・メモリを書き換える方法としては，専用のフラッシュ・メモリ・プログラマで書き換える方法と，コード・フラッシュ・メモリに書き込んだプログラムによって書き換えるフラッシュ・セルフ・プログラミングがあります。

図1－1　コード・フラッシュ・メモリを書き換える手段



専用フラッシュ・メモリ・プログラマによる書き換え



セルフ・プログラミングによる書き換え

1. 2 RL78/G13のフラッシュ・セルフ・プログラミング

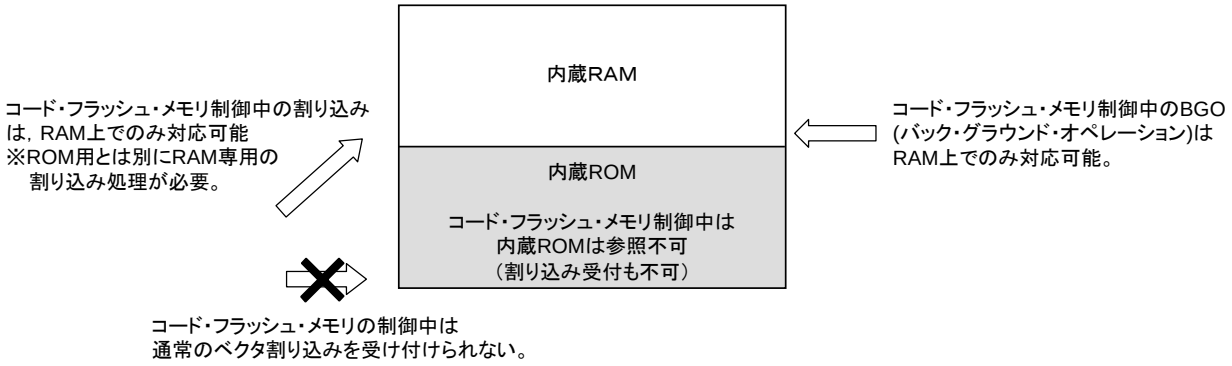
RL78/G13には，フラッシュ・セルフ・プログラミングを行うためのライブラリが用意されています。書き換えプログラムからフラッシュ・セルフ・プログラミング・ライブラリの各関数を呼び出すことでフラッシュ・セルフ・プログラミングを行います。

RL78/G13のフラッシュ・メモリは，00000H番地から1ブロック（1024バイト）ごとにブロック番号が割り付けられており，フラッシュ・メモリの消去はこのブロック単位で行います。

また，RL78/G13のフラッシュ・セルフ・プログラミングはシーケンサを使用し，フラッシュの書き換え制御を実行しますが，シーケンサの制御中はコード・フラッシュ・メモリを参照できません。そのため，割り込み等，シーケンサ制御中にユーザ・プログラムを動作させる必要がある場合，コード・フラッシュ・メモリの消去や書き込み，セキュリティ・フラグの設定等を行う時に，フラッシュ・セルフ・プログラミング・ライブラリの一部のセグメントや，書き換えプログラムをRAMに配置して制御を行う必要があります。シーケンサ制御中にユーザ・プログラムを動作させる必要が無い場合は，フラッシュ・セルフ・プログラミング・ライブラリや書き換えプログラムをROM(コード・フラッシュ・メモリ)上に配置して動作させる事が可能です。

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・ライブラリ，及び書き換えプログラムをROM(コード・フラッシュ・メモリ)上に配置し、書き換えを行う例を紹介します。

図1－2 コード・フラッシュ・メモリ書き換え中の状態



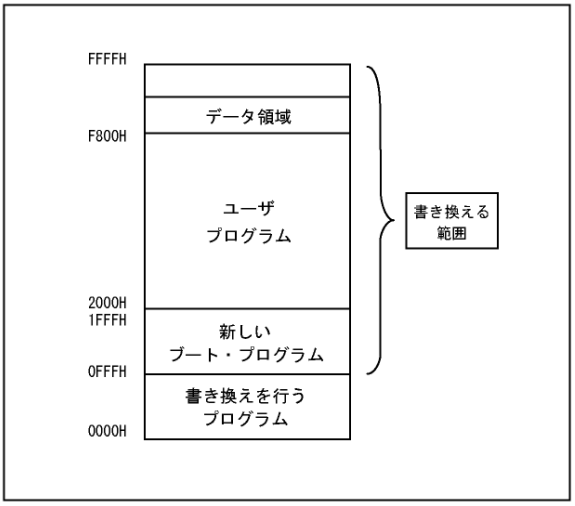
1.3 コード・フラッシュ・メモリの書き換え方法

フラッシュ・セルフ・プログラミング機能を使用する事により，コード・フラッシュ・メモリ上のプログラムやデータを書き換える事が出来ます。

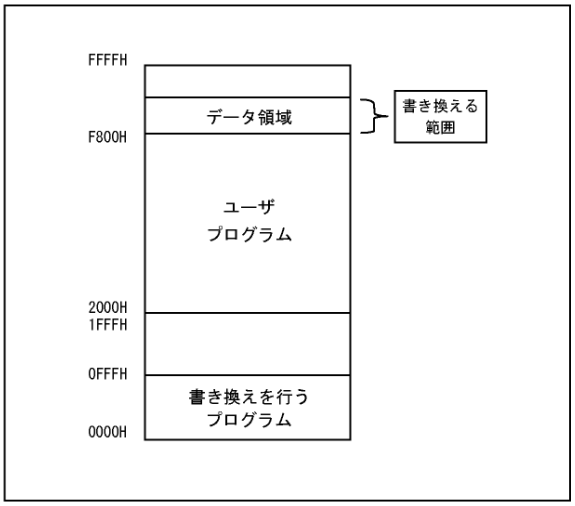
図1−3に全てのプログラムを書き換える場合，データを書き換える場合，また，処理毎に配置する領域を分ける事で部分的にプログラムを書き換える場合の例を記載します。

フラッシュ・セルフ・プログラミング機能を使用してコード・フラッシュ・メモリの書き換えを行う場合には，フラッシュ・セルフ・プログラミングを行うプログラムとそれ以外の機能のプログラムは別のブロックに配置する必要があります。

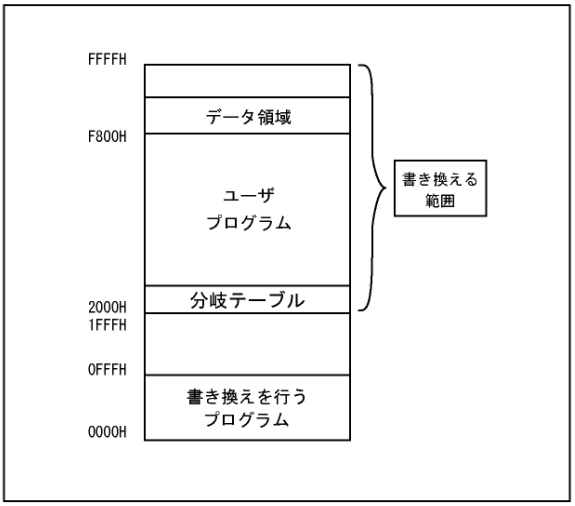
図1−3 コード・フラッシュ・メモリの書き換え例



プログラムの書き換え



データの書き換え



プログラムの部分書き換え

1.4　プログラムとデータの書き換え

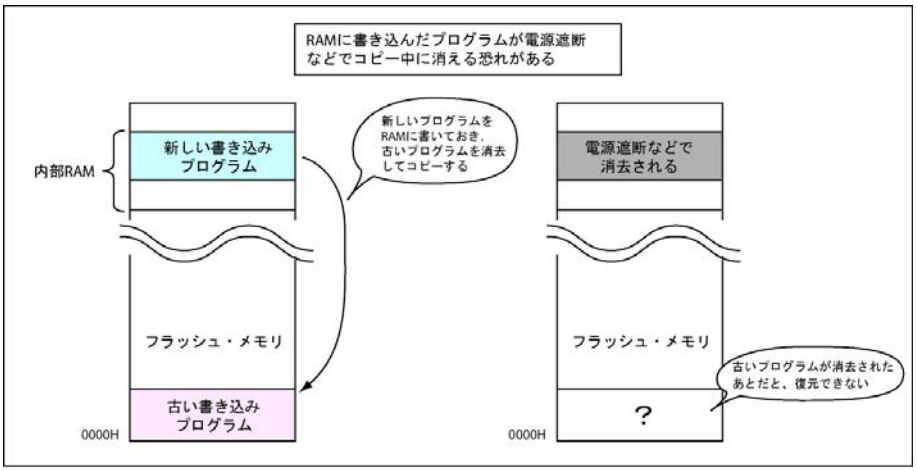
RL78/G13では，フラッシュ・メモリを安全に書き換えるための機能として，ブート・スワップ機能が用意されています。フラッシュ・メモリの0000H-0FFFHまでの領域がブート・クラスタ0，1000H-1FFFHまでの領域がブート・クラスタ1に割り当てられており，この2つの領域は，ブート・スワップを行うことによって切り替えることができます。このブート・スワップ機能を利用することにより，プログラムを安全に切り替えることができます。

RL78/G13にはブート・スワップ用のライブラリ関数が用意されています。本アプリケーションノートでは，FSL_InvertBootFlag関数を使用したブート・スワップの方法と，プログラムの書き換えの方法を記載します。

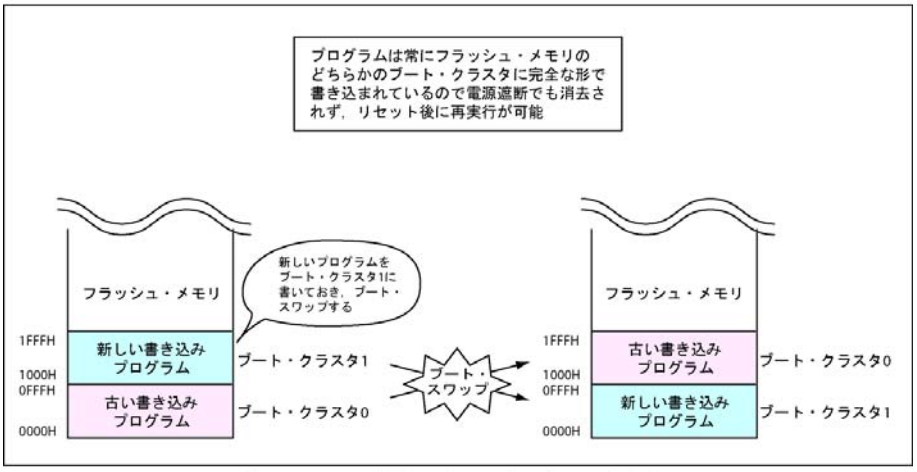
また、フラッシュ・セルフ・プログラミング機能を使用することにより,ユーザ・プログラムで使用するデータの書き換えを行うことができます。書き換えデータのテーブルを固定アドレスに配置することにより,ユーザ・プログラムの変更なしに安全にデータの書き換えを行うことが可能です。本アプリケーションノートでは,固定アドレスにデータ・テーブルを配置し,データを書き換える方法を記載します。

なお，「第2章　プログラムとデータの書き換え例」で使用しているプログラムは，フラッシュ・セルフ・プログラミングを行うプログラムが配置されたブロックの書き換えは行いません。プログラム例の詳細については，「第2　プログラムとデータの書き換え例」を参照してください。

図1−4　プログラムの書き換え（RAMに書く方法とブート・スワップで切り替える方法）



RAMを使ったプログラムの書き換え



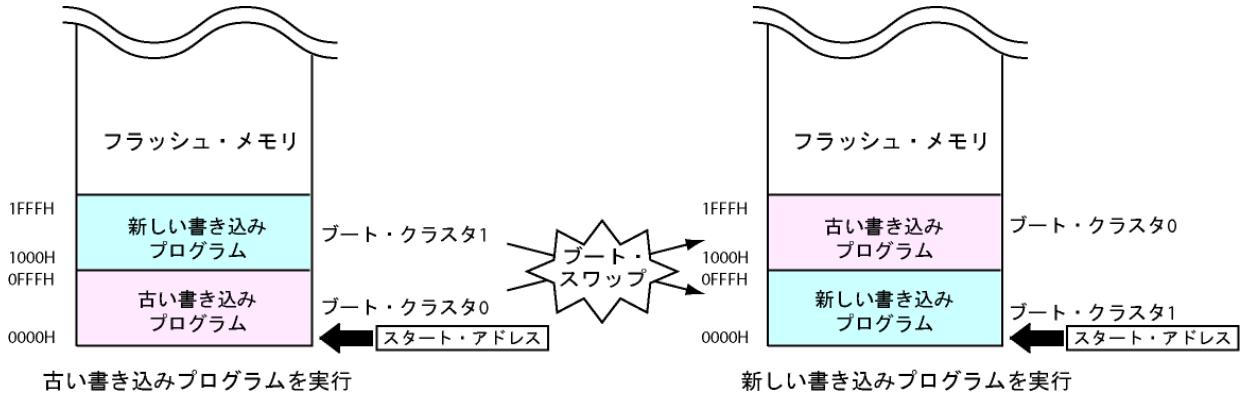
ブート・スワップ機能を使ったプログラムの書き換え

(1) ブート・スワップとは

ブート・スワップとは，ブート・クラスタ0とブート・クラスタ1を切り替えることです。切り替えることによって，それまでブート・クラスタ0（0000H-0FFFH）だった領域がブート・クラスタ1（1000H-1FFFH）に，ブート・クラスタ1（1000H-1FFFH）だった領域がブート・クラスタ0（0000H-0FFFH）になります。

ブート・スワップを行うためには，コード・フラッシュ・メモリの設定レジスタに所定の設定を行います。ブート・スワップには2種類の方法があり，設定直後に新しいプログラムに切り替わる方法と，ブート・フラグのみを書き換え，リセット後にプログラムを切り替える方法があります。本アプリケーションノートでは，リセット後にプログラムを切り替える方法を説明しています。

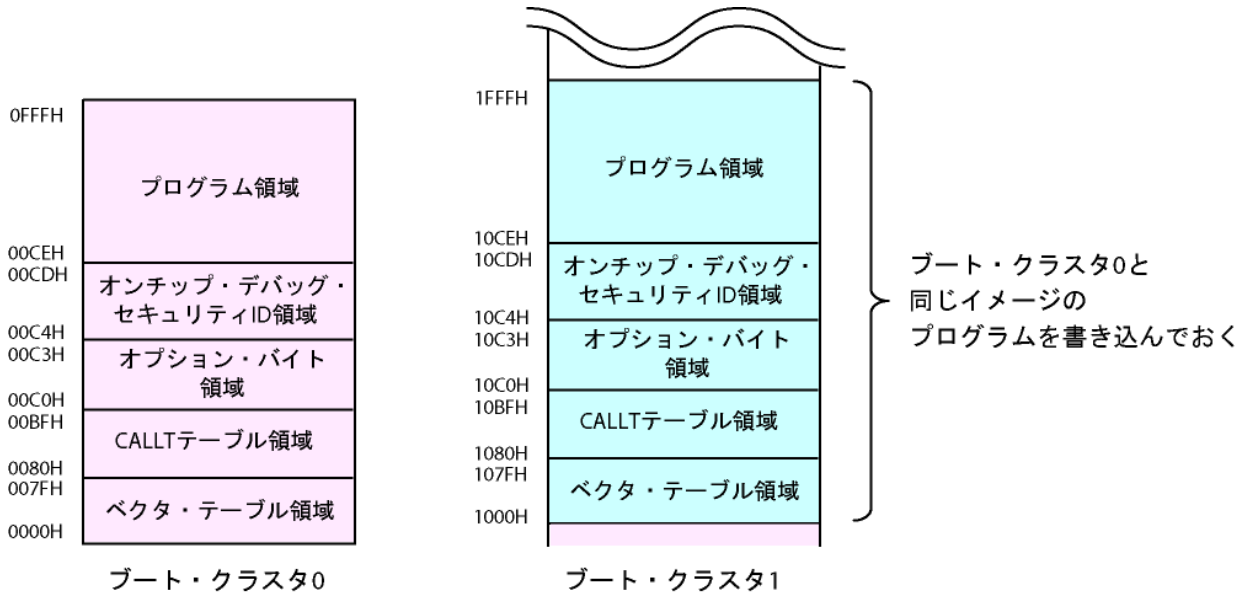
図1−5 ブート・スワップ



(2) ブート・クラスタ1に書き込んでおくプログラム

ブート・クラスタ1は，ブート・スワップ後はブート・クラスタ0に切り替わり，アドレスも0000H-0FFFHとなります。したがってブート・クラスタ1にあらかじめ書き込んでおく新しいプログラムは，ブート・クラスタ0に書き込まれているプログラムと同じ開始アドレス，構成のものを書き込みます。

図1−6 ブート・クラスタ1に書き込んでおくプログラム



(3) フラッシュ・シールド・ウインドウ（FSW）とは

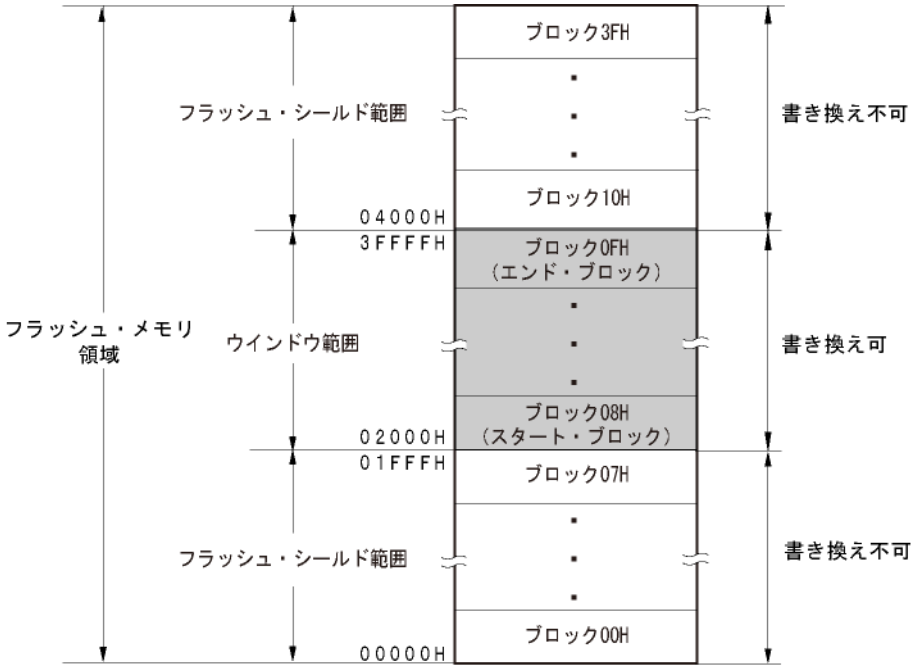
フラッシュ・シールド・ウインドウはフラッシュ・セルフ・プログラミング時のセキュリティ機能の一つで，指定したウインドウ範囲以外の書き込みおよび消去を，フラッシュ・セルフ・プログラミング時のみ禁止に設定する機能です。

オンボード／オフボード・プログラミング時は，ウインドウとして指定した範囲外にも書き込みおよび消去が可能です。

フラッシュ・シールド・ウインドウの設定は， FSL_SetFlashShieldWindow 関数で設定します。ウインドウ範囲は，スタート・ブロックとエンド・ブロックを指定することで設定できます。

図1−7 フラッシュ・シールド・ウインドウの設定例

（対象デバイス：R5F100LE，スタート・ブロック：08H，エンド・ブロック：0FHの場合）



1.5 再リンク機能

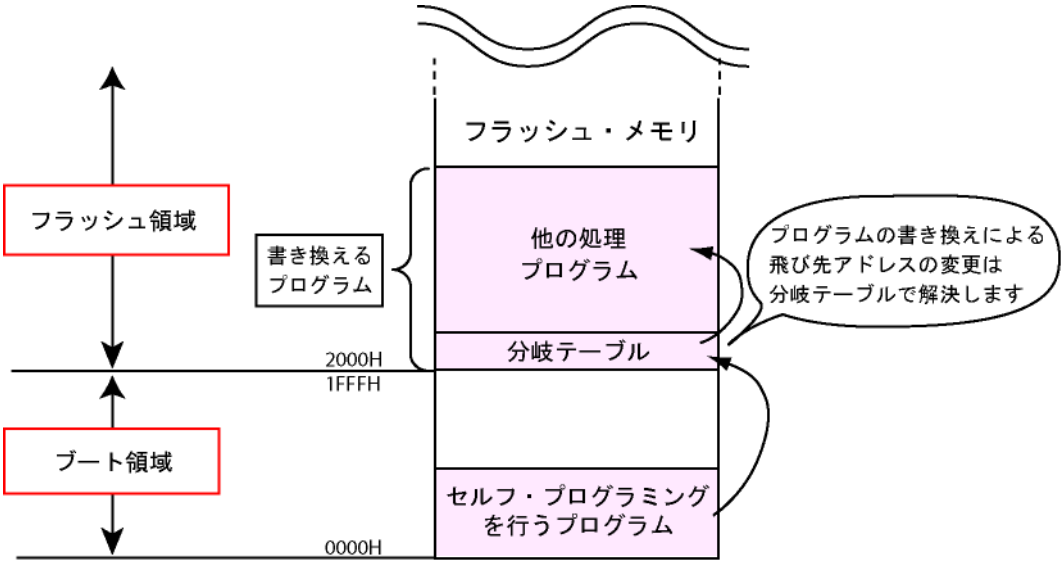
システムによっては，書き換え／取り換えが不可能な領域（ブート領域）に加え，フラッシュや外付けROMといった，書き換え／取り換えが可能な領域（フラッシュ領域）を使用することがあります。

CubeSuite+ではこのようなシステムにおいて，フラッシュ領域上のプログラムのみを変更したい場合，ブート領域上のプログラムの再構築を行わず，ブート領域とフラッシュ領域間の関数呼び出しを正常に行う機能を“再リンク機能”と呼んでいます。

RL78/G13では，この再リンク機能を使用することで，コード・フラッシュ・メモリに配置しているプログラムの一部分だけをフラッシュ・セルフ・プログラミング機能で書き換えることができます。コード・フラッシュ・メモリのフラッシュ領域の先頭アドレスに設定されている領域未満をブート領域，フラッシュ領域の先頭アドレスに設定した以降の領域をフラッシュ領域として二つに分け，フラッシュ・セルフ・プログラミングを行うプログラムをブート領域に，それ以外の書き換えられるプログラムをフラッシュ領域に配置する事で，ブート領域側から安全にフラッシュ領域を書き換える事が可能となります。

本アプリケーションノートでは，この機能を使用して書き換えを行います。プログラムの作成方法については，「**第2章　プログラムの構成方法と書き換え例**」を参照してください。

図1−8　ブート領域とフラッシュ領域の構成イメージ



・分岐テーブル

ブート領域からフラッシュ領域に配置された処理を実行するためには，使用するフラッシュ側の処理の配置情報をブート側で把握する必要があります。CubeSuite+では特定の領域にフラッシュ側の処理の配置情報を記録したテーブルを作成し，そのテーブルに登録されている情報を使用してブート側の処理から，フラッシュ側の処理を実行可能にします。このテーブルのことを“分岐テーブル”と呼びます。

分岐テーブルには，書き換えられるフラッシュ領域のプログラムの各関数の先頭アドレスや割り込みベクタを登録されています。プログラムが書き換わると，分岐テーブルも更新されます。これによってフラッシュ領域のプログラムの各関数の先頭アドレスが変更されても，ブート領域のプログラムからの関数呼び出しが可能になります。

第2章　プログラムの構成方法と書き換え例

この章では，フラッシュ・セルフ・プログラミング・ライブラリを使って，RL78/G13(R5F100LE)のコード・フラッシュ・メモリを書き換えるためのプログラムの構成方法や書き換えの例について紹介します。

2. 1　サンプル・プログラムの動作環境

本アプリケーションノートで使用するサンプル・プログラムは，ブート時の処理を行うブート・プログラム，フラッシュ・セルフ・プログラミングによってプログラム及びデータの書き換えを行う処理，ユーザ・プログラム(LEDの点滅処理)の3つから構成されています。

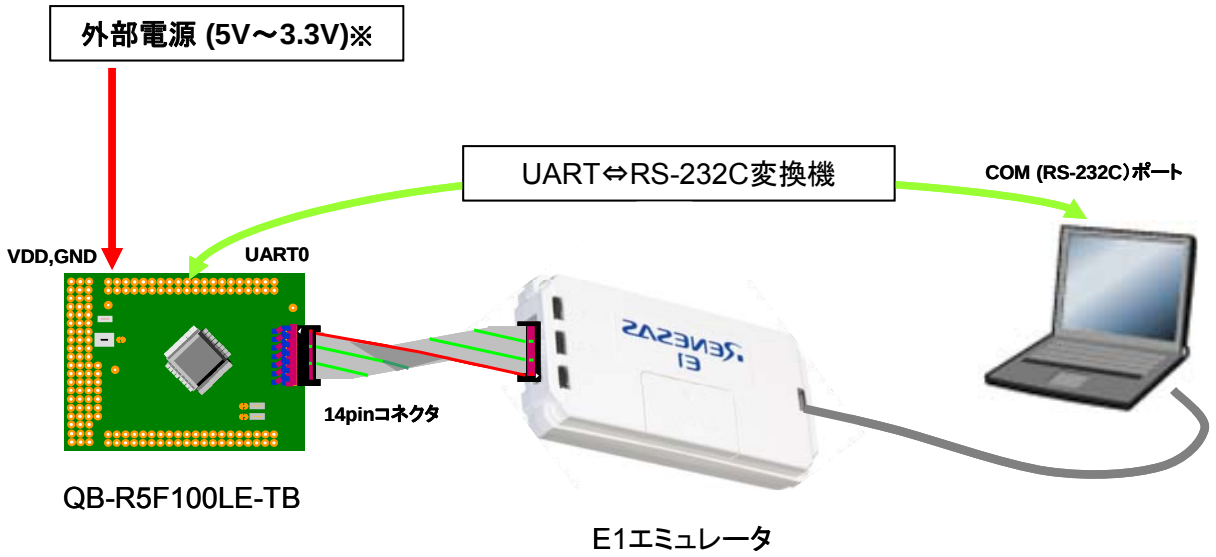
構成されているプログラムの処理のシーケンスは，起動時にブート・プログラムによってRL78/G13(R5F100LE)の基本的な初期化処理を行い，QB-R5F100LE-TB上に搭載されているスイッチの状態を確認する事で，プログラム及びデータの書き換えを行う処理を起動するか，LEDの点滅処理を実行するかを判別，選択します。

SW1を押下せずにターゲットの電源をON，もしくはリセットさせるとLED2が点灯し，プログラム及びデータの書き換えを行う処理が起動され，シリアル通信の受信待ちとなります。シリアル通信はホスト・マシンで動作する「SelfFlashWriter」（詳細は付録Aを参照してください）と行い，通信時にはLED1が点滅します。SelfFlashWriterよりシリアル通信でプログラム・コードのデータを受信し，プログラムを更新します。

SW1を押下した状態でターゲットの電源をON，もしくはリセットさせると，ユーザ・プログラムが起動され，LEDの点滅動作を実行します。起動後にSW1を押下すると，PC側にシリアル通信でASCⅡデータを送信し，LEDの点滅動作を遅くした上でWDTリセットを実行します。再度押下するとWDTが一度クリアされ，リセットまでの時間が延長されます。

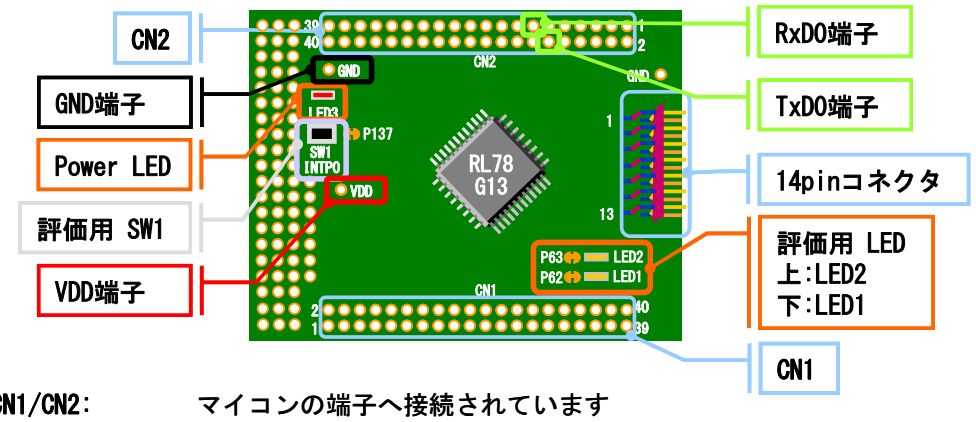
表2－1に使用しているRL78マイクロコントローラのI/Oと，図2－1，図2－2にサンプル・プログラムの動作環境，図2－3にプログラムの配置状態と構成を示します。

図2－1 サンプル・プログラムで使用しているQB-R5F100LE-TBの動作環境イメージ



※外部電源はマイクロコントローラ単体での動作確認に必要です。
E1エミュレータ使用時には必要ありません。

図2－2 サンプル・プログラムで使用しているQB-R5F100LE-TBの端子イメージ



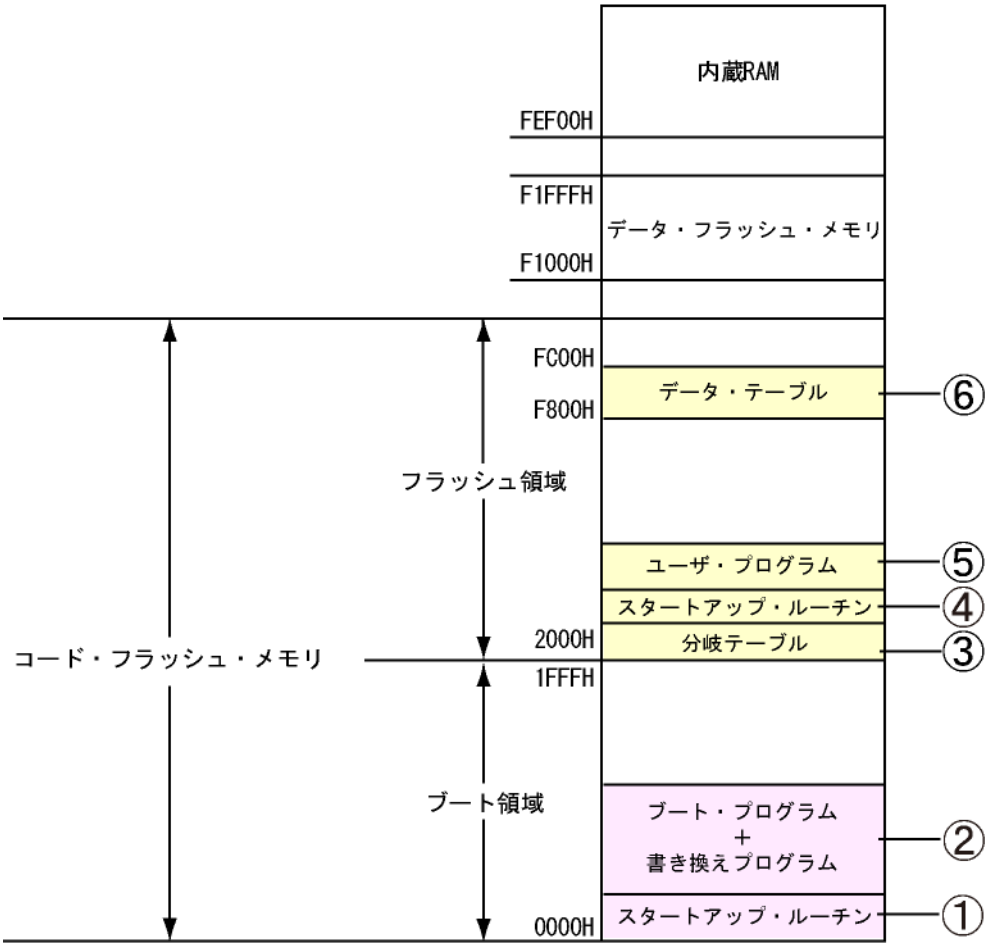
CN1/CN2: マイコンの端子へ接続されています
PowerLED(LED3)：電源が入った時に赤色に発光します
評価用LED1: ポート62(P62)がLOWで黄色に発光します
評価用LED2: ポート63(P63)がLOWで黄色に発光します
評価用SW1： INTP0に接続されています
14pinｺﾈｸﾀ： E1(別売)を接続し、オンチップ･デバッグや書き込み時に使用します
GND,VDD端子: ターゲット･ボードへ電源供給する場合の端子です
RxD0端子: UARTのRX端子。パソコンのCOMポートと接続してシリアル通信(データの受信)を行います※
TxD0端子: UARTのTX端子。パソコンのCOMポートと接続してシリアル通信(データの送信)を行います※

※パソコンのCOM(RS-232C)ポートと接続するためには別途レベル変換回路が必要です

表2－1 プログラムで使用しているQB-R5F100LE-TBの端子リスト

接続部	用　途	I/O
表示機1 LED1	＜ブート・プログラム＞ LED1，LED2を初期化，消灯する。	ポート62（P62端子）
表示機2 LED2	＜書き換えプログラム＞ LED2を常時点灯，LED1を通信時に点滅する。 ＜ユーザ・プログラム＞ LED1を常時点灯，LED2を一定の間隔で点滅。	ポート63 (P63端子)
スイッチ1 SW1	＜ブート・プログラム＞ ブート時に書き換え・プログラムを実行するか，ユーザ・プログラムを実行するかの判断に使用する。 ＜ユーザ・プログラム＞ PC側にASCⅡデータをシリアルで送信し，LED1の点滅間隔を遅くさせ，WDTリセットを実行。再度押下すると一度WDTをクリアする。	ポート137（P137端子）
シリアル通信 RxD0:P11(CN2:13pin) TxD0:P12(CN2:12pin)	RL78/G13マイクロコントローラのUART0ポートを使用し，パソコンとシリアル通信を行う。パソコンのCOM(RS-232C)ポートと接続するためには，別途レベル変換回路等が必要。 ＜通信仕様＞ ビット／秒：115200　データ・ビット長：8 パリティ：なし　ストップ・ビット：1　フロー制御：なし	RXD0(P11端子) TXD0(P12端子)

図2−3 コード・フラッシュ・メモリのサンプル・プログラム配置概要図



① ブート側スタートアップ・ルーチン

ブート領域側の初期化を行うスタートアップ・ルーチン

② ブート・プログラム＋書き換えプログラム

ブート時の初期化処理とフラッシュ・セルフ・プログラミング・ライブラリを使用したブート領域側のフラッシュ書き換えプログラム。

③ 分岐テーブル

ブート側からフラッシュ側へアクセスするための分岐テーブル領域

④ フラッシュ側スタートアップ・ルーチン

フラッシュ領域側の初期化を行うスタートアップ・ルーチン

⑤ ユーザ・プログラム

LED1を常時点灯，LED2を一定の間隔で点滅させるフラッシュ領域側のプログラム

⑥ データ・テーブル

ユーザ・プログラムで使用するPC側に送信するASCⅡデータが入っている領域。

図2−4 サンプル・プログラム(書き換えモード時)の動作環境（イメージ図）

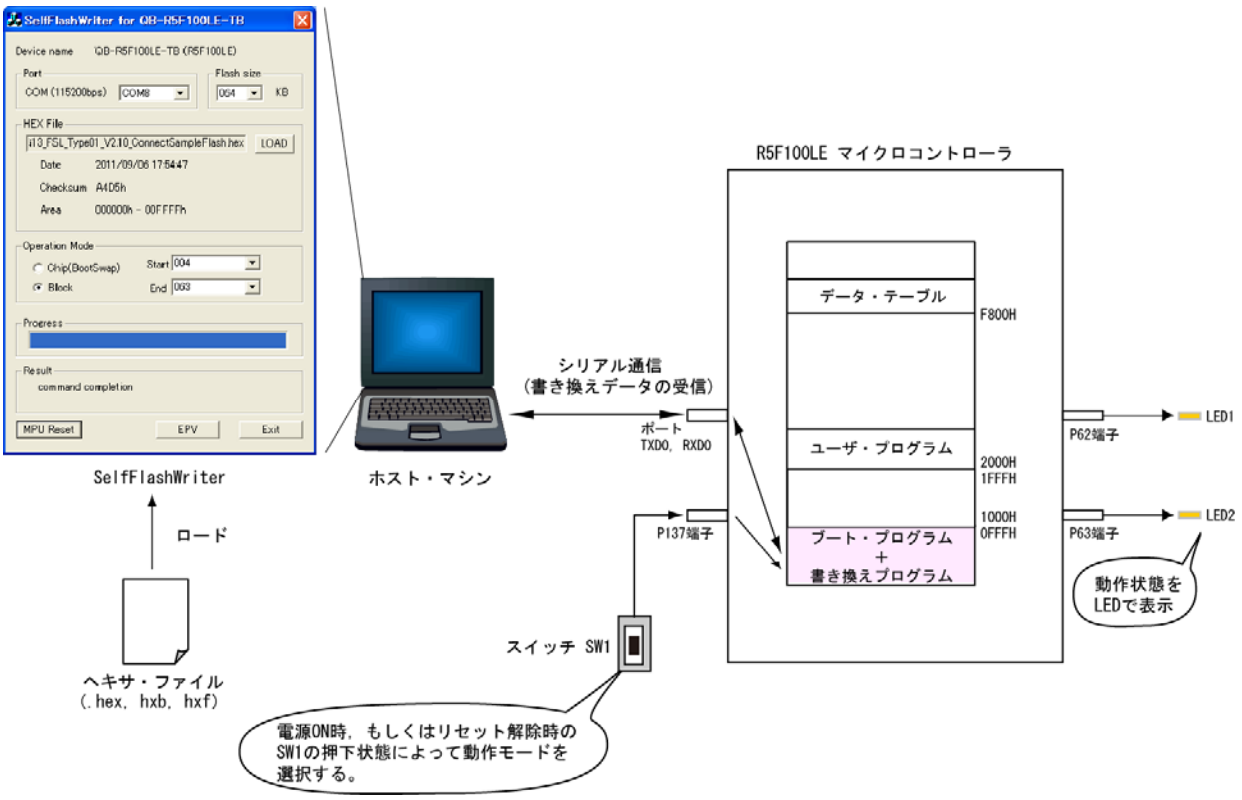
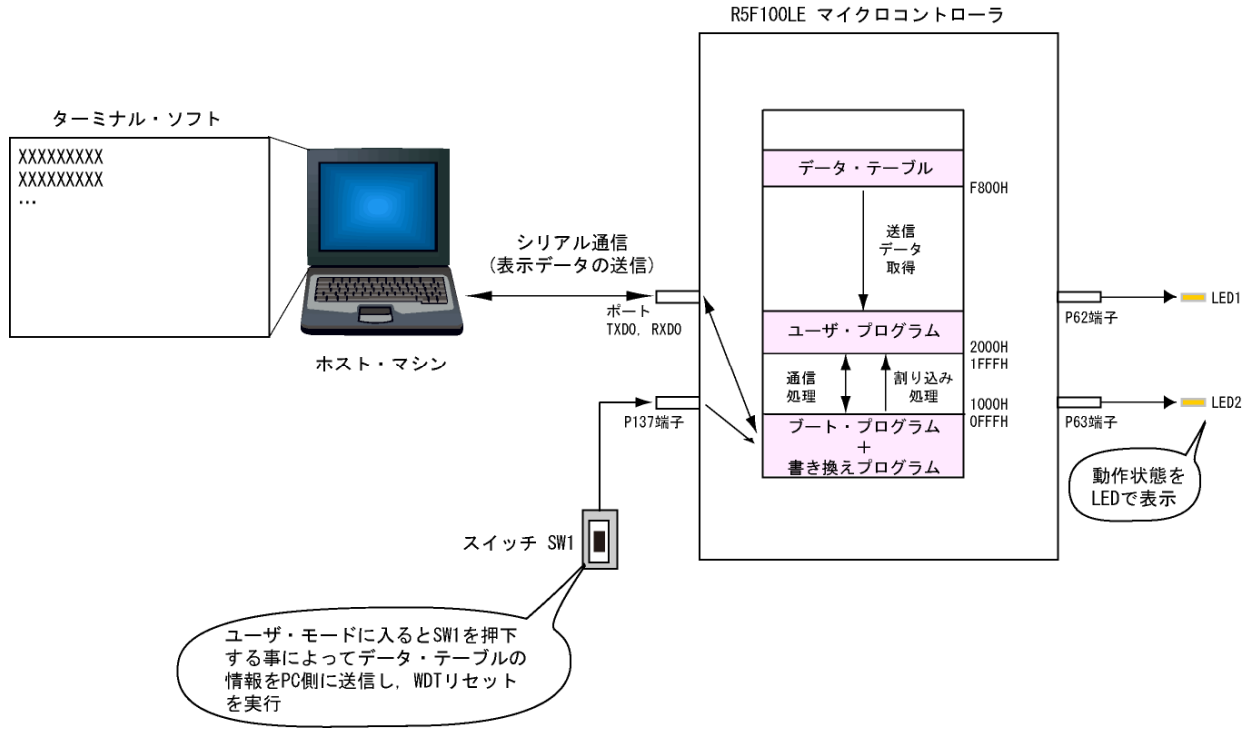


図2−5 サンプル・プログラム(ユーザ・モード時)の動作環境2（イメージ図）

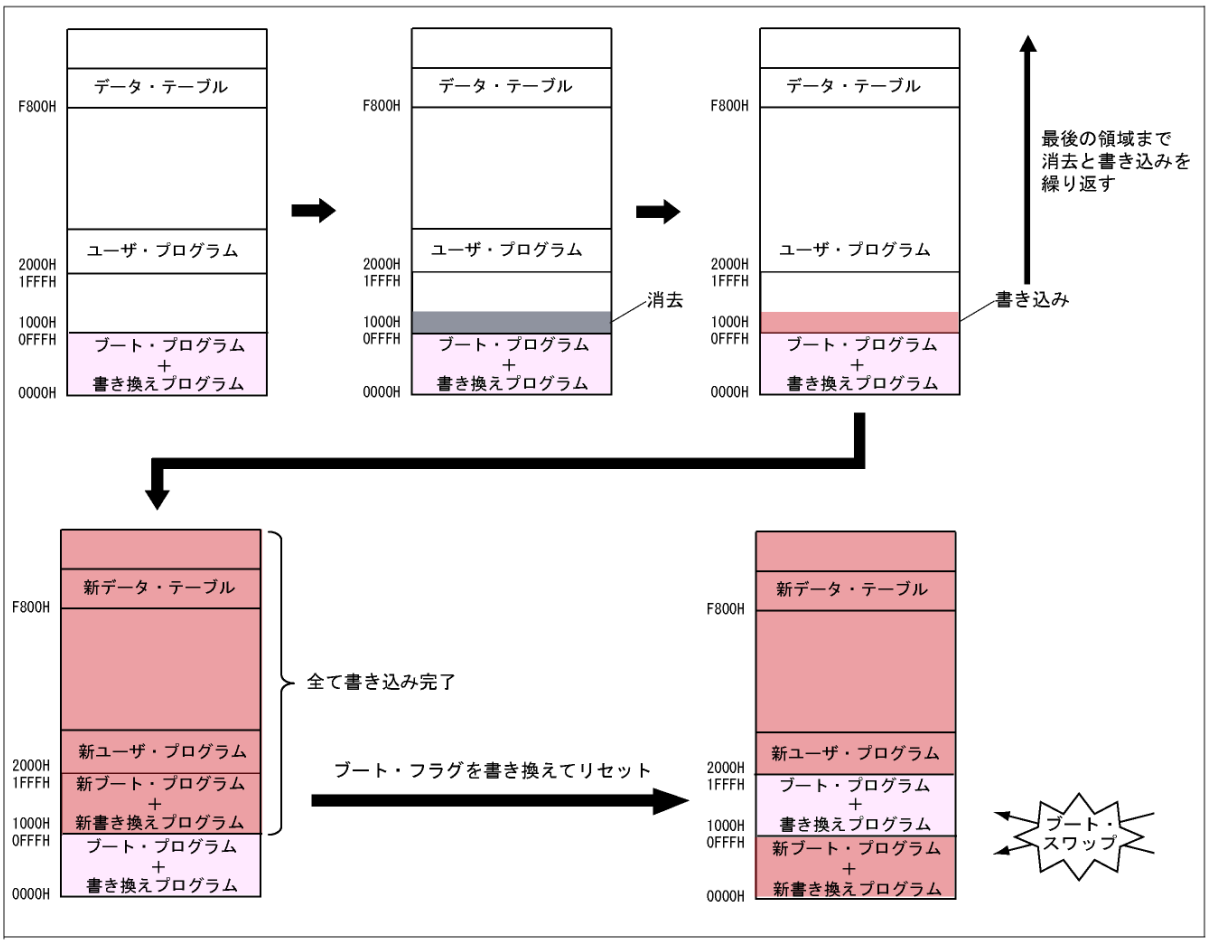


2.2　フラッシュ書き換えの動作フロー

サンプル・プログラムのプログラムの書き換え，及びデータの書き換えの動作フローを，図2－6，2－7，2－8に示します。

フラッシュ・セルフ・プログラミングを行うプログラムは，ブート・クラスタの領域（ブロック0～3）に配置しています。

図2－6　プログラムの全体書き換え動作（イメージ図）



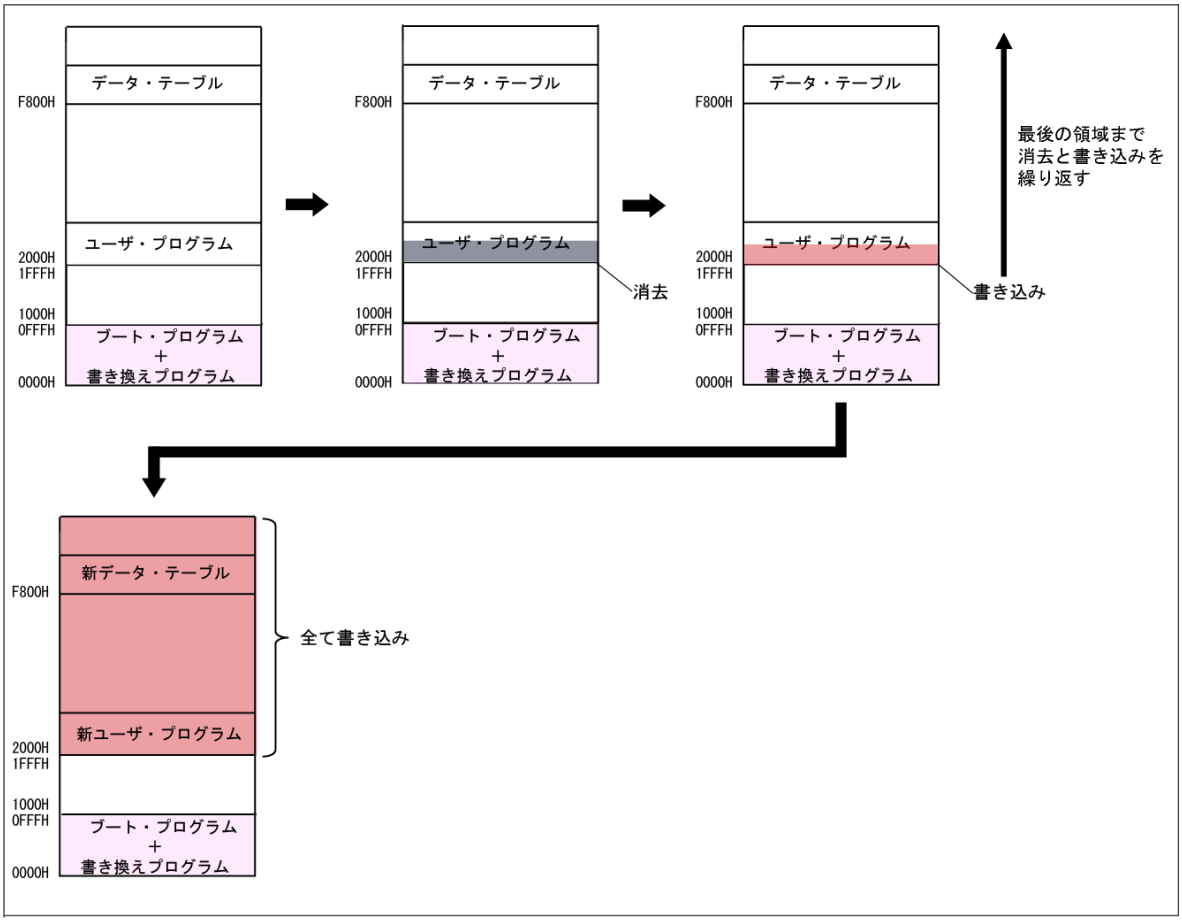
① 受信したプログラム・データを全て書き込む

RL78/G13(R5F100LE)の書き換えプログラム(フラッシュ・セルフ・プログラミングを行うプログラム)は，SelfFlashWriterから受信した新しいブート・プログラム，及び書き換えプログラムをブロック4～7（1000H-1FFFH）に書き込み，ユーザ・プログラムとデータ・テーブルをブロック8（2000H-FFFFH）以降に書き込みます。

② ブート・スワップを行う

SelfFlashWriterは，全てのデータの書き込みが完了した事を確認すると，BOOTSWAPコマンドを送信します。RL78/G13(R5F100LE)の書き換えプログラムは，フラッシュ・セルフ・プログラミング・ライブラリを使用してブート・スワップの設定を行い，正常終了後にリセットを実行します。リセット後にブート・スワップが実行され，新しいブート・プログラムが実行されます。

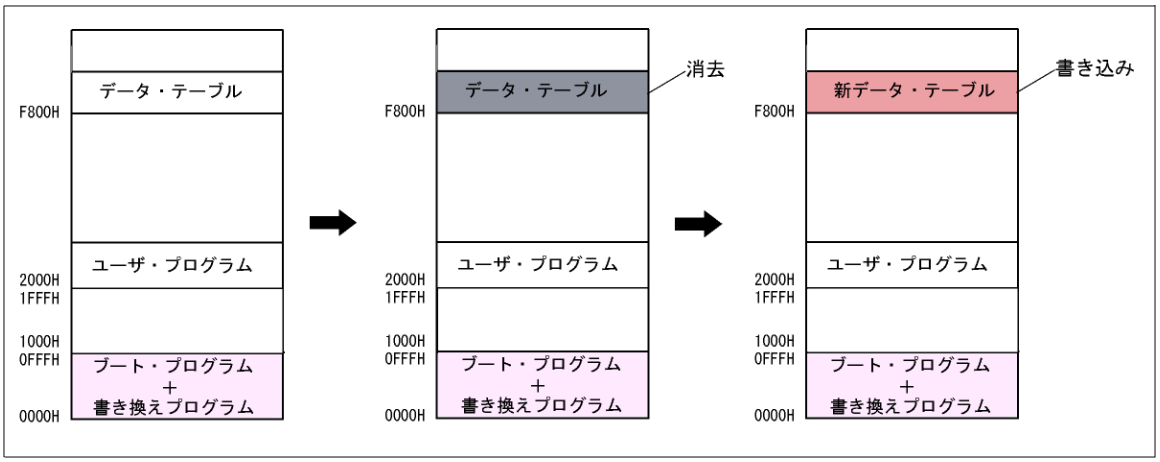
図2－7 プログラムの部分(フラッシュ領域)書き換え動作（イメージ図）



・ 受信したプログラム・データを全て書き込む

RL78/G13(R5F100LE)の書き換えプログラム(フラッシュ・セルフ・プログラミングを行うプログラム)は，SelfFlashWriterから受信した新しいユーザ・プログラムとデータ・テーブルをブロック8(2000H-FFFFH）以降に書き込みます。

図2－8 データの書き換え動作（イメージ図）



・ 受信したデータ・テーブルやプログラムを書き込む

RL78/G13(R5F100LE)の書き換えプログラム(フラッシュ・セルフ・プログラミングを行うプログラム)は，SelfFlashWriterから受信した新しいデータ・テーブルやプログラムを部分的に書き込みます。

2.3 サンプル・プログラムのファイル構成

サンプル・プログラムのファイル構成を表2－2に示します。CubeSuite+でプロジェクトを読み込む場合は，CubeSuite+を立ち上げ，「r_fsl_praxis01.mtpj」ファイルを読み込んでください。

また，サンプル用のプロジェクト・ファイルは「C:¥Program Files¥Renesas Electronics¥Flash Libraries」フォルダに，フラッシュ・セルフ・プログラミング・ライブラリ Type01 V2.20がインストールされている状態で作成されています。ライブラリのインストール先が異なる場合は，プロジェクトを立ち上げた後，ライブラリ関連ファイルの登録先を適宜変更してください。

表2－2 サンプル・プログラムのファイル構成（ フォルダ名：R01AN0718_PRAXIS01 ）

ファイル名			内 容
root		r_fsl_praxis01.mtpj	プロジェクト・ファイル
		r_fsl_praxis01_boot.mtsp	ブート側サブ・プロジェクト・ファイル
		r_fsl_praxis01_flash.mtsp	フラッシュ側サブ・プロジェクト・ファイル
	¥DefaultBuild	r_fsl_praxis01_boot.hex	ブート領域用プロジェクトHEX形式ファイル
		r_fsl_praxis01_flash.hex	フラッシュ領域用プロジェクト ヘキサ・ファイル ・全領域用ヘキサ・ファイル (hex)
		r_fsl_praxis01_flash.hxb	フラッシュ領域用プロジェクト ヘキサ・ファイル ・ブート領域用ヘキサ・ファイル (hxb)
		r_fsl_praxis01_flash.hxf	フラッシュ領域用プロジェクト ヘキサ・ファイル ・フラッシュ領域用ヘキサ・ファイル(hxf)
	¥TestData	r_fsl_praxis01_write_test.hex	テスト用HEX形式ファイル ・書き換え確認用ヘキサ・ファイル （LED1，LED2の表示方法反転）
		r_fsl_praxis01_boot_write_test.hxb	テスト用HEX(hxb)形式ファイル ・ブート領域用ヘキサ・ファイル (hxb) （LED1，LED2の表示方法反転）
		r_fsl_praxis01_flash_write_test.hxf	テスト用HEX(hxf)形式ファイル ・フラッシュ領域用ヘキサ・ファイル(hxf) （LED1，LED2の表示方法反転）
	¥inc	r_fsl_praxis01_com.h	プログラム共通ヘッダ・ファイル
		r_fsl_praxis01_BranchTable.h	分岐テーブル登録設定ファイル
		r_fsl_praxis01_BootSection.h	ブート領域側のセクション設定ファイル
		r_fsl_praxis01_FlashSection.h	フラッシュ領域側のセクション設定ファイル
	¥src ¥boot	r_fsl_praxis01_boot_main.c	ブート領域側メイン処理
		r_fsl_praxis01_boot_write.c	ブート領域側書き込み処理
	¥flash	r_fsl_praxis01_flash_main.c	フラッシュ領域側メイン処理
	¥dr	r_fsl_praxis01_boot_map.dr	ブート領域側リンク・ディレクティブ・ファイル
		r_fsl_praxis01_flash_map.dr	フラッシュ領域側リンク・ディレクティブ・ファイル

2.4 サンプル・プログラムのリソース

サンプル・プログラムのリソースの参考値を表2−3〜2−5に示します。
本数値は、参考値のため、実際の値とは異なっています。詳細なサイズに関しては、コンパイル時に生成されるmapファイルにて、ご確認ください。

表2−3 サンプル・プログラムの全体リソース(参考値)

領域名	ROM領域範囲	占有ROMサイズ	占有RAMサイズ	備考
ブート領域	0H〜FFFH	3000バイト	900バイト	割り込みベクタ，オプション・バイト，ライブラリの使用領域も含む。
フラッシュ領域	2000H〜FBFFH	600バイト	100バイト	OCDモニター領域は除く

表2−4 サンプル・プログラムのブート領域側のリソース(参考値)

使用領域		項目	合計サイズ
ROM		割り込みベクタ，オプション・バイト等の領域	3000バイト
		スタートアップ・ルーチン，ランタイム・ライブラリ等	
		標準ライブラリ(memcpy_f, memset_f, @stkinit)	
		フラッシュ・セルフ・プログラミング・ライブラリ	
		ブート・プログラム，書き込みプログラム，割り込み処理	
RAM		スタートアップ・ルーチン，ランタイム・ライブラリ等	750バイト
		ブート・プログラム，書き込みプログラム，割り込み処理	
	スタック(参考値)	ブート・プログラム，書き込みプログラム	150バイト
		割り込み処理	
		フラッシュ・セルフ・プログラミング・ライブラリ	

表2−5 サンプル・プログラムのフラッシュ領域側のリソース(参考値)

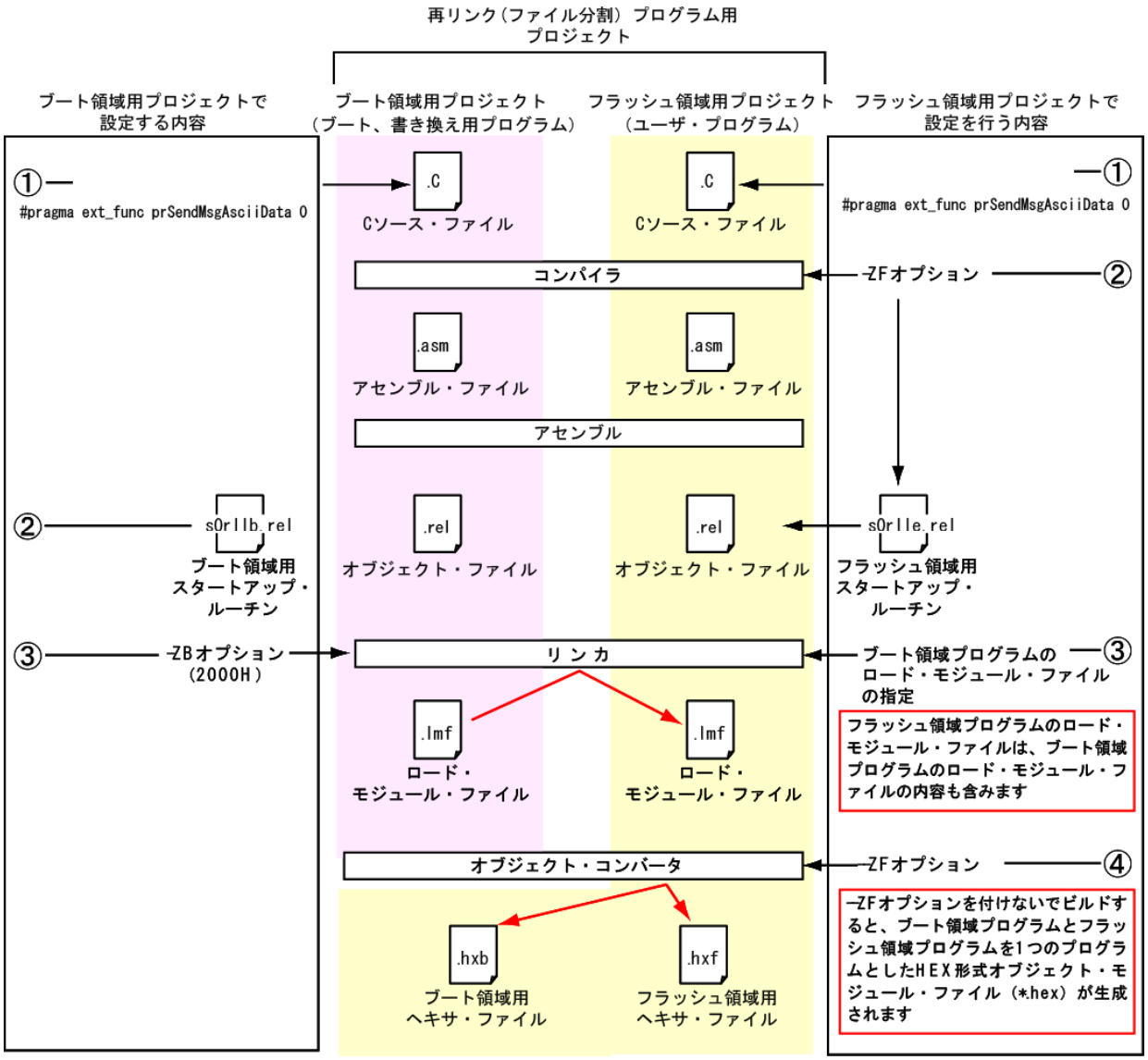
使用領域		項目	合計サイズ
ROM		分岐テーブル(ベクタ，分岐テーブル登録関数)	600バイト
		スタートアップ・ルーチン，ランタイム・ライブラリ等	
		ユーザ・プログラム	
		データ・テーブル(F800H〜FBFFH)	
RAM		ユーザ・プログラム	50バイト
	スタック(参考値)	ユーザ・プログラム	50バイト
		割り込み処理	

2.5 プロジェクトの構成（再リンク機能の設定）

RL78のアセンブラ，コンパイラのオプションを指定することで，１つのプログラムをブート領域とフラッシュ領域に分けて開発を行うことができます。

CubeSuite+で開発を行う場合には，別のプロジェクトに分けて作成します。また，ブート領域のプログラムのロード・モジュール・ファイル（.lmf）をフラッシュ領域のプログラムのロード・モジュール・ファイルにリンクさせて，プログラムを生成するので，先にブート領域のプログラムをビルドしておく必要があります。図2－9に再リンク機能の概要と，図2－10～図2－19設定に必要なCubeSuite+の操作方法を示します。

図2－9 CubeSuite+でのプロジェクト構成例と再リンク機能の各設定



(1) ブート領域用プロジェクト(ブート・プログラム，書き換えプログラム)の設定

① 拡張機能(#pragma)を使用してブート領域からフラッシュ領域への関数呼び出しを設定

分岐テーブルへの関数の登録を，Cソース内に記述します。登録する関数は，ユーザ・プログラムでも使用するSW1割り込み関数です。

この設定で，ブート領域のプログラムからフラッシュ領域のプログラム（関数）を実行することができます。

図2－10 ブート領域からフラッシュ領域への関数呼び出し機能(r_fsl_praxis01_BranchTable.h)

```
・
・
/*------------------------------------------------------------------------*/
/* (#pragma) branch table functions (ext_func)                            */
/*------------------------------------------------------------------------*/
#pragma ext_func prSendMsgAsciiData 0
・
・
・
```

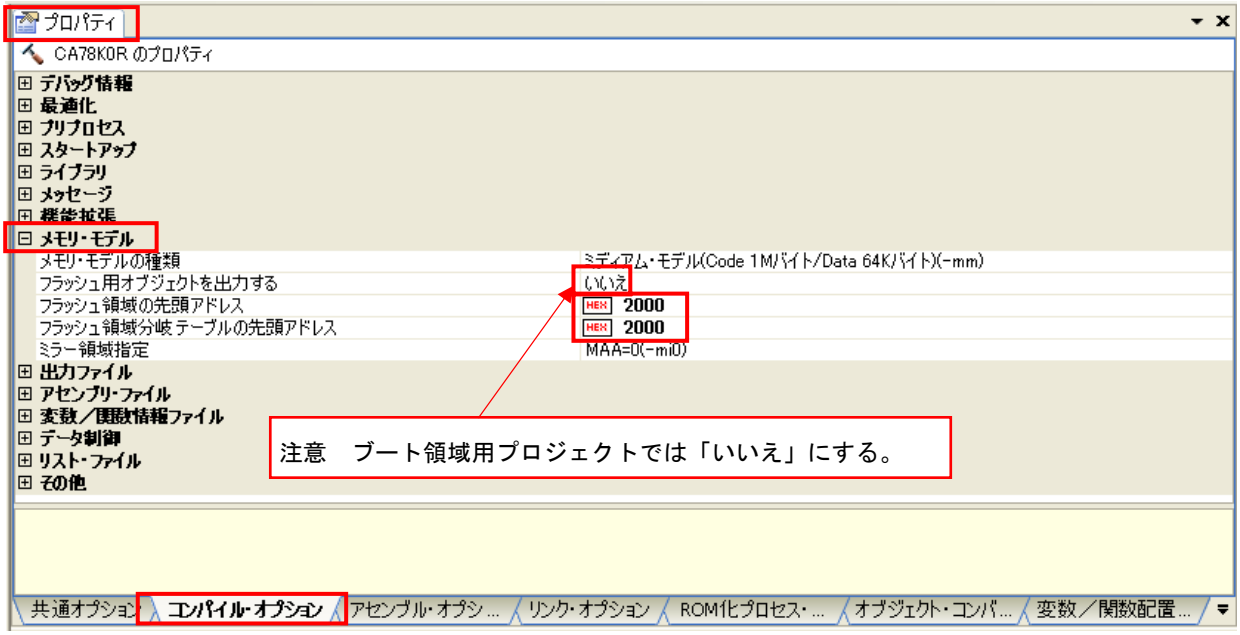
関数の分岐テーブルへの登録
#pragma ext_func 関数名 ID番号

フラッシュ領域に配置されている使用したい関数名を記述する。
※プロトタイプ宣言は別途必要。

② 分岐テーブルとフラッシュ領域の先頭アドレス指定

ブート領域用プロジェクトで使用する分岐テーブルとフラッシュ領域の先頭アドレスをCA78K0Rのコンパイル・オプションで設定します。

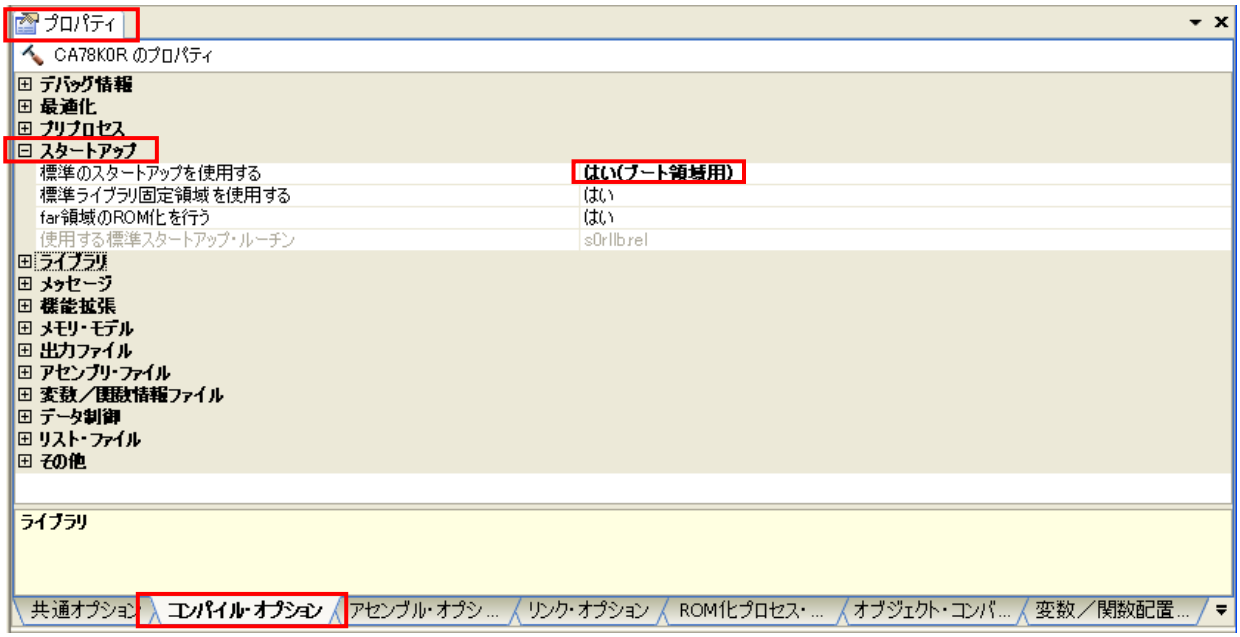
図2－11 分岐テーブルとフラッシュ領域の先頭アドレス設定



③ ブート領域用プロジェクトのスタートアップ・ルーチン指定

ブート領域用プロジェクトのスタートアップ・ルーチンを指定します。

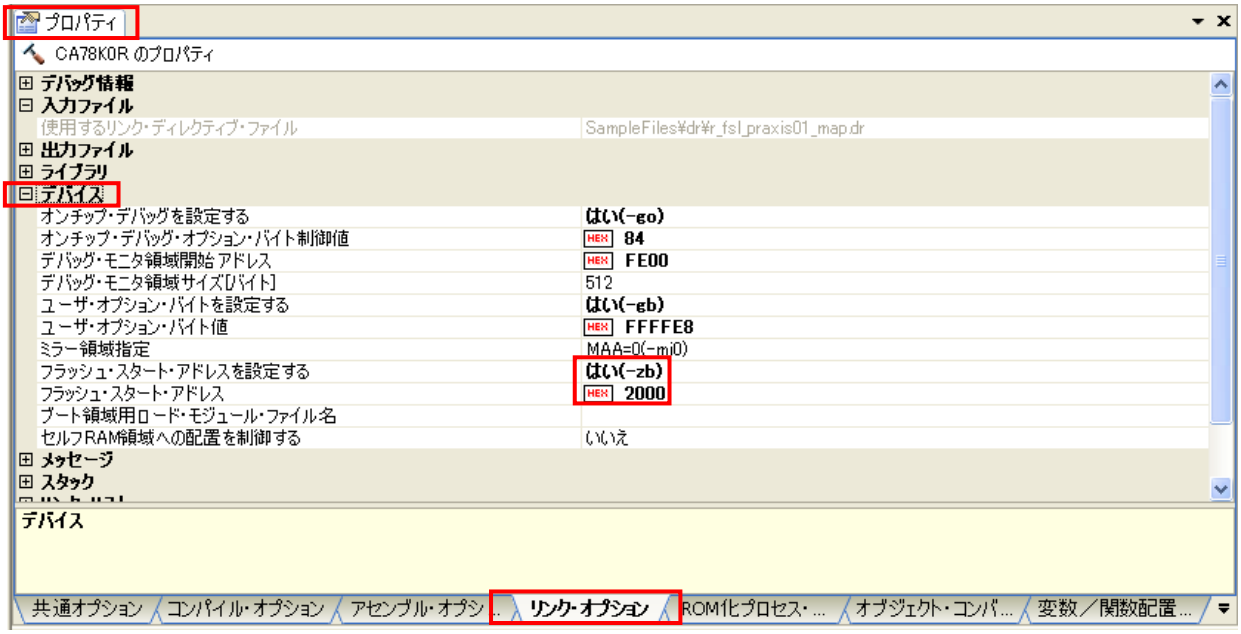
図2－12 ブート領域用のスタートアップ・ルーチンの指定



④ リンカ –ZBオプションの設定

リンカの-ZBオプションを設定し，フラッシュ領域の先頭アドレスを指定します。

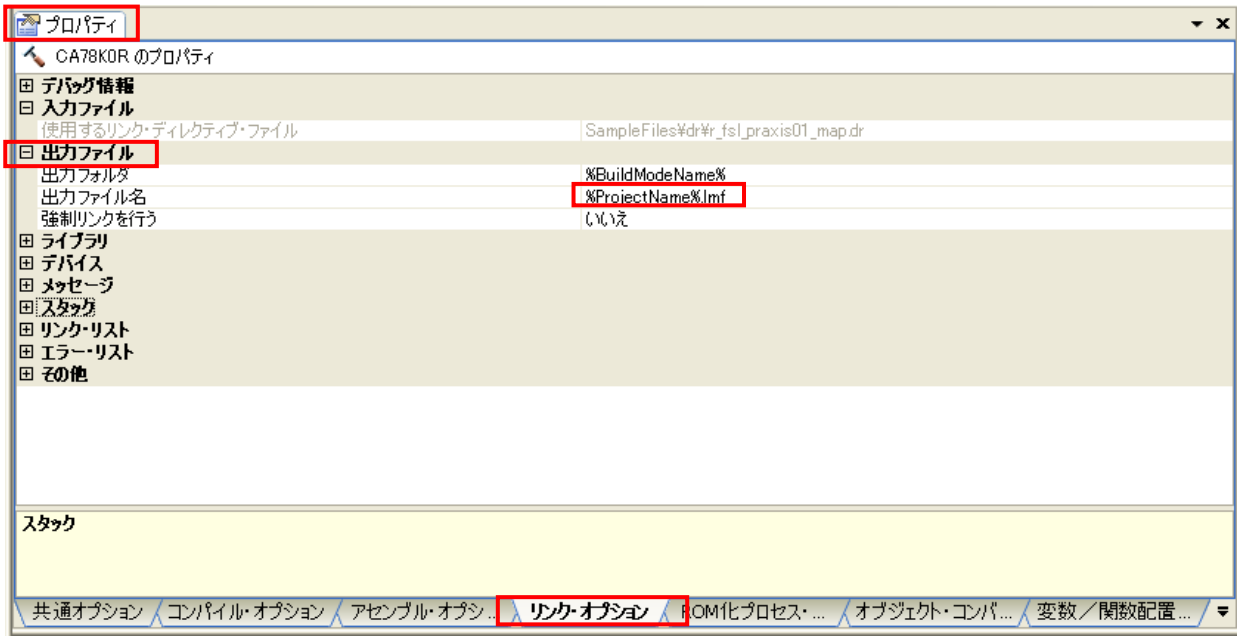
図2－13 リンカ –ZBオプションの指定



⑤ ブート領域用プロジェクトのロード・モジュール名の設定

フラッシュ領域用プロジェクトはブート領域用プロジェクトのロード・モジュールを使用し，全体のモジュールを生成するため，ブート領域用プロジェクトのロード・モジュールに個別の名称が必要な場合は名称を設定し，フラッシュ領域用プロジェクト側でロード・モジュールを指定できるように設定します。デフォルトの場合は，「プロジェクト名.lmf」となります。

図2－14 ブート領域用プロジェクトのロード・モジュール名の指定



⑥ フラッシュ領域から実行するブート領域側のプログラムの確認

ブート領域側に配置したプログラムをフラッシュ領域側から実行するためには，ブート領域側に配置する予定のプログラムが，ブート領域側のロード・モジュール・ファイル(*.lmf)に含まれている必要がありますが，対象プログラムがブート領域側で使用されていないと，ブート領域側のリンク時に，対象プログラムがリンクされない場合があります。

この状態でフラッシュ側から対象プログラムを実行しようとしても，ブート領域側のロード・モジュール・ファイル(*.lmf)に対象プログラムが含まれていないため，リンク・エラーとなってフラッシュ領域側のプログラムが作成できません。

フラッシュ領域側から実行する必要のあるブート領域側のプログラムがあり，かつ対象プログラムがブート領域側で使用されていない場合は，対象プログラムを実行するようなダミー関数を作成する等，対象プログラムがブート領域側のロード・モジュール・ファイル(*.lmf)に含まれるようにプログラムを作成してください。

(2) フラッシュ領域用プロジェクト（ユーザ・プログラム側）の設定

① 拡張機能(#pragma)を使用してブート領域からフラッシュ領域への関数呼び出しを設定

フラッシュ領域用プログラムの方にも分岐テーブルへの関数の登録を，Cソース内に記述します。登録する関数は，ユーザ・プログラムで使用するSW1割り込み関数です。

図2－15 ブート領域からフラッシュ領域への関数呼び出し機能(r_fsl_praxis01_flash_main.c)

r_fsl_praxis01_BranchTable.h：

```
・
・
/*------------------------------------------------------------------------------*/
/* (#pragma) branch table functions (ext_func)                                  */
/*------------------------------------------------------------------------------*/
#pragma ext_func prSendMsgAsciiData 0
・
```

関数の分岐テーブルへの登録
#pragma ext_func 関数名 ID番号

登録する関数本体

r_fsl_praxis01_flash_main.c:

```
/*------------------------------------------------------------------------------*/
/* Include common files                                                         */
/*------------------------------------------------------------------------------*/
/* (#pragma) 分岐テーブル登録ヘッダー   */
#include "r_fsl_praxis01_BranchTable.h"
```

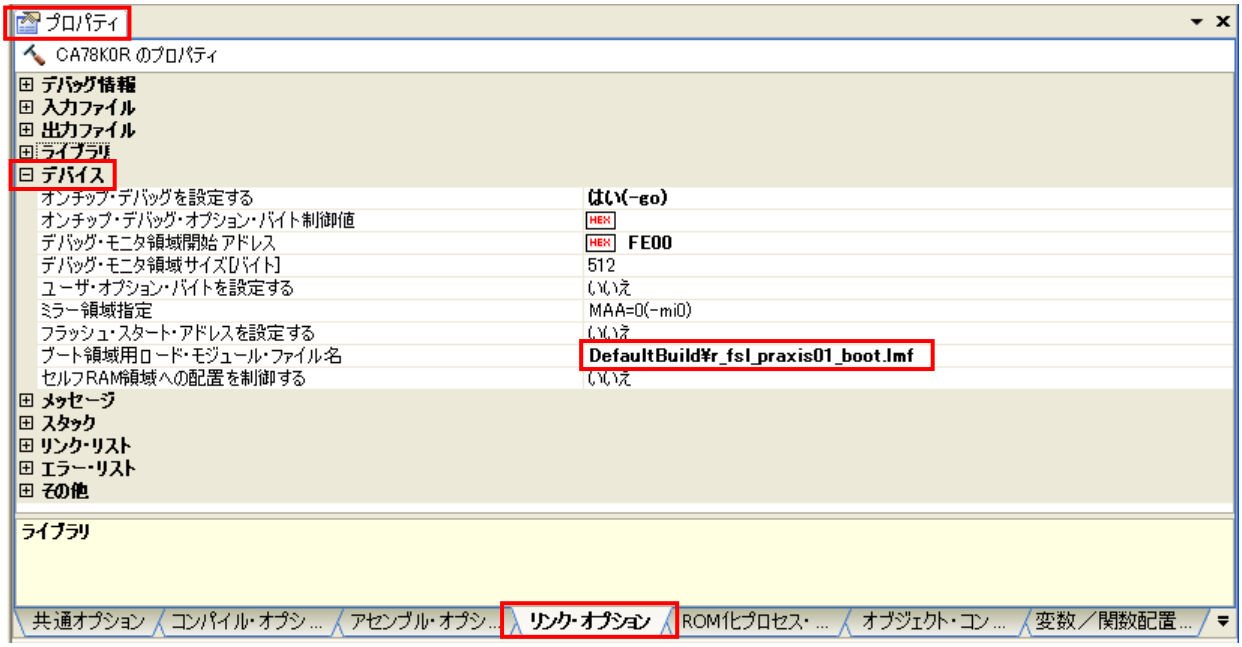
省略

```
void prSendMsgAsciiData( void )
{
    UH  duh_i;
    UB  dubSendData[5];

    /* ASCⅡデータ送信処理 */
    for( duh_i = 0 ; duh_i < sizeof( prFcubSendMsgData ) ; duh_i++ )
    {
        dubSendData[0] = prFcubSendMsgData[ duh_i ];
        prUartSendData( &dubSendData[0] );
    }

    /* スイッチ回数計算 */
    prDuhSwNum++;
    if( prDuhSwNum > 999 )
    {
        prDuhSwNum = 0;
    }

    /* 送信データの作成 */
    dubSendData[0] = (UB)( prDuhSwNum / 100 ) | 0x30;
    dubSendData[1] = (UB)( ( prDuhSwNum % 100 ) / 10 ) | 0x30;
    dubSendData[2] = (UB)( prDuhSwNum % 10  ) | 0x30;
    dubSendData[3] = '¥n';
    dubSendData[4] = '¥r';

    /* 数値データデータ送信処理 */
    for( duh_i = 0 ; duh_i < 5 ; duh_i++ )
    {
        prUartSendData( &dubSendData[duh_i] );
    }

#ifdef PR_USE_OCD_MODE

#else
    /* WDTリセットとWDT割り込みの停止 */
    PR_WD_INT_OFF();

    /* LEDの点滅速度を遅くする */
    if( prDuwLedTime <= PR_LED_DEFAULT_WAIT )
    {
        prDuwLedTime = PR_LED_DEFAULT_WAIT * PR_LED_WAIT_MAG;
    }
#endif
}
```

② ブート領域用プロジェクトのロード・モジュール・ファイルの設定

フラッシュ領域用プロジェクトで使用するブート領域用プロジェクトのロード・モジュール・ファイル（.lmf）を設定します。

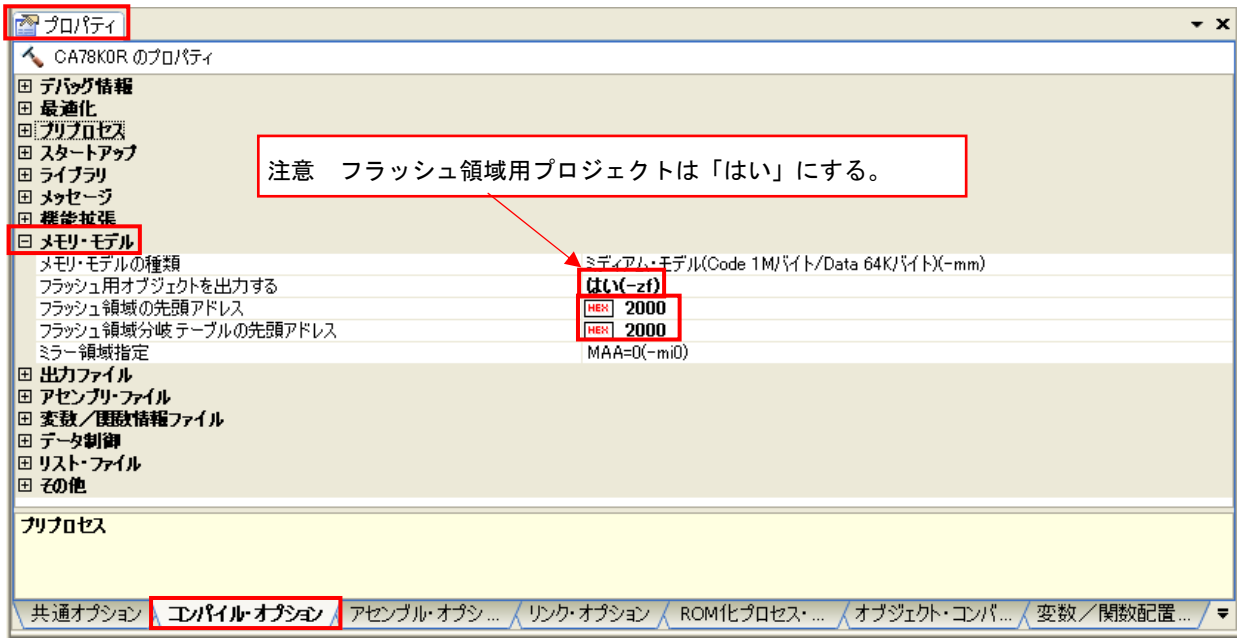
図2－16 ブート領域用プロジェクトのロード・モジュール・ファイルの設定



③ 分岐テーブルとフラッシュ領域の先頭アドレス指定

フラッシュ領域用プロジェクトで使用する分岐テーブルとフラッシュ領域の先頭アドレスを設定します。

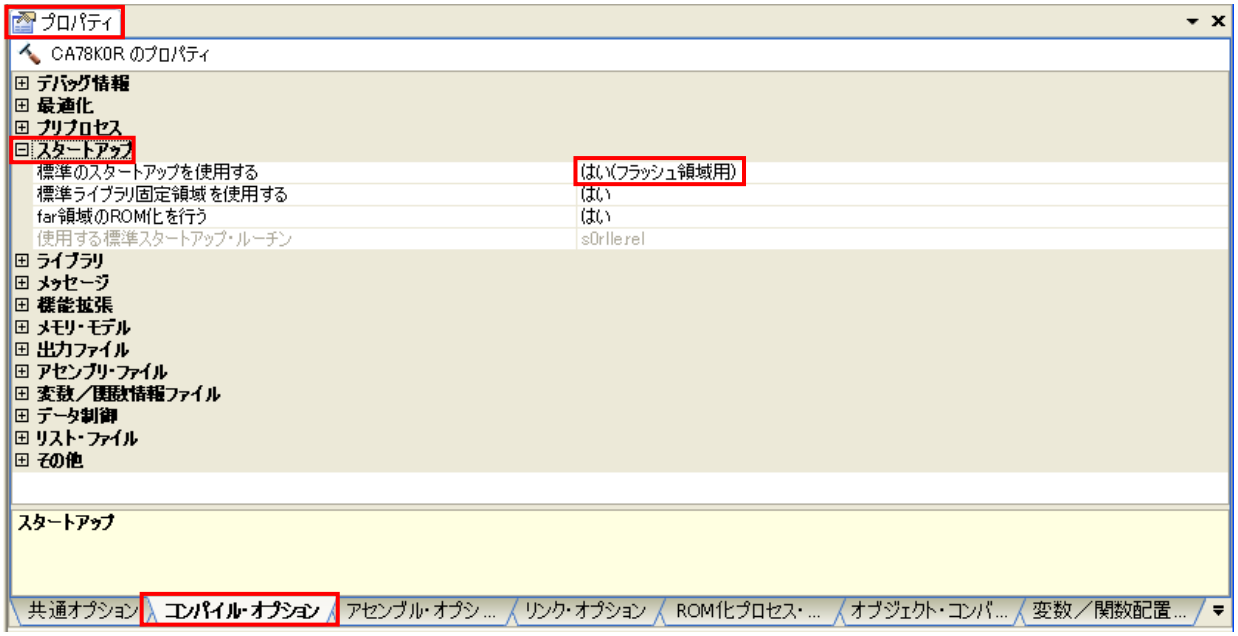
図2－17 分岐テーブルとフラッシュ領域の先頭アドレス指定



④ フラッシュ領域用プロジェクトのスタートアップ・ルーチン指定

フラッシュ領域用プロジェクトのスタートアップ・ルーチンを指定します。

図2－18 ブルとフラッシュ領域の先頭アドレス指定

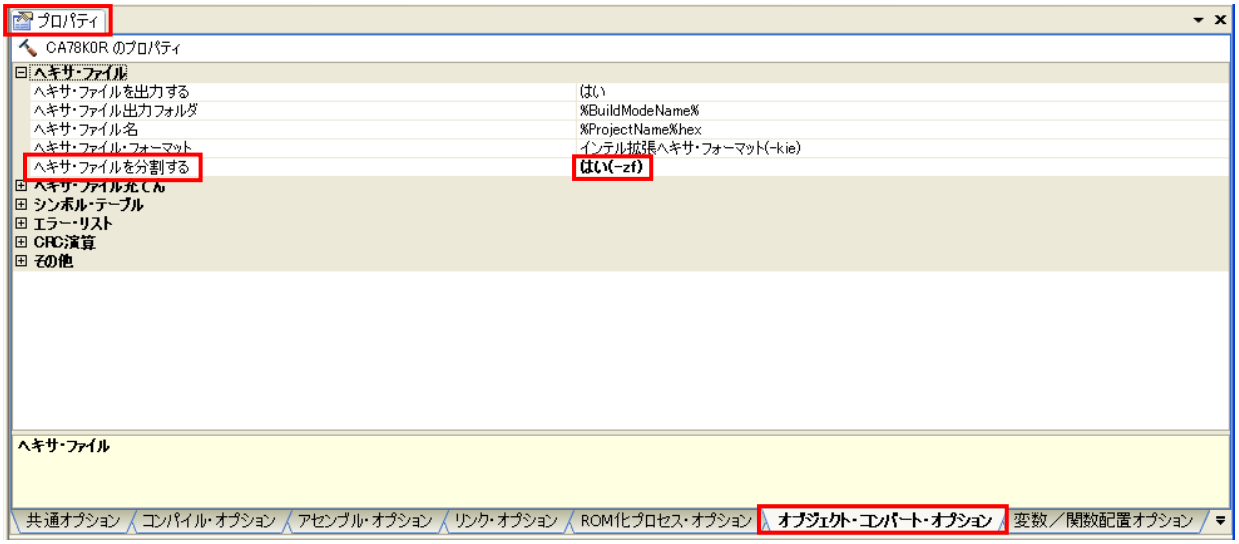


⑤ ヘキサ・ファイルの設定（ ヘキサ・ファイルの分割出力を行う場合のみ ）

オブジェクト・コンバータの-ZFオプションを設定します。このオプションを指定すると，ブート領域のプログラムとフラッシュ領域のプログラムを別々のHEX形式オブジェクト・モジュール・ファイルに分割出力します。

ブート領域プログラムの出力ファイルは拡張子.hxb，フラッシュ領域のプログラムの出力ファイルは拡張子.hxfとなります。

図2－19 オブジェクト・コンバータ -ZFオプションの設定

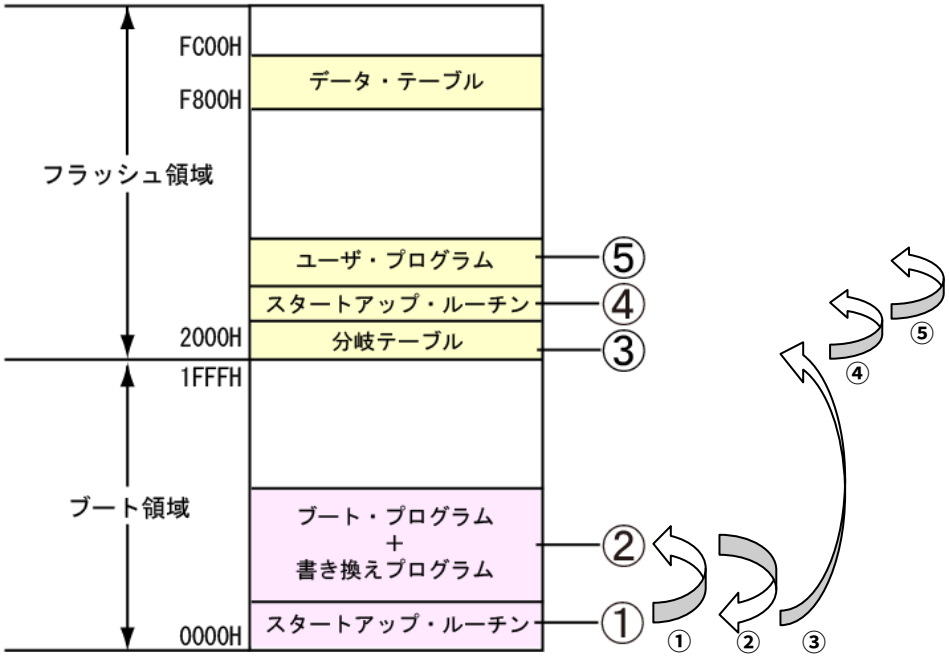


2.6 リセット解除後からメイン処理までの設定

再リンク機能を使用したプログラムは，ブート領域とフラッシュ領域のプログラムそれぞれにスタートアップ・ルーチンがあるため，リセット解除後からメイン処理までの動作は通常のプログラムと異なります。図2-20の①から③のように，ブート領域用スタートアップ・ルーチンとフラッシュ領域用スタートアップ・ルーチンが実行されるようプログラムを記述してください。

フラッシュ領域のメイン関数以降（下図の⑤）は，プログラムの仕様にしたがって実行してください。

図2－20 プログラムの動作順序



① ブート領域用のスタートアップ・ルーチン

リセット解除後はブート領域用のスタートアップ・ルーチンが実行されます。ブート領域用のデータ初期化後に，ブート領域のプログラムのメイン関数（boot_main() = ブート領域用のスタートアップ・ルーチンが起動するメイン関数 ）を実行します。

②ブート領域側のメイン関数(boot_main())

メイン関数（ boot_main() ）では，ブート・プログラムとして，QB-R5F100LE-TBの基本的な初期化処理を行い，QB-R5F100LE-TB上に搭載されているスイッチの状態を確認する事で，データの書き換えを行う処理を実行するかを判断します。ユーザ・プログラムを動作させる場合は，そのまま終了し，ブート領域用のスタートアップ・ルーチンの処理に戻ります。

③分岐テーブルへのジャンプ

メイン関数（ boot_main() ）が終了すると，ブート領域用のスタートアップ・ルーチンに戻り，フラッシュ領域側にある分岐テーブルへジャンプします。分岐テーブルが所定の位置に存在しない場合，正常に処理を実行することはできません。

④分岐テーブル

フラッシュ領域用のスタートアップ・ルーチンへジャンプします。

⑤ フラッシュ領域用のスタートアップ・ルーチン

ブート領域用のデータ初期化後に，フラッシュ領域用のメイン関数（ main()＝ フラッシュ領域用のスタートアップ・ルーチンが起動するメイン関数 ）へジャンプします。

以降は，プログラムの仕様にしたがって記述します。図2－21，2－22にサンプル・プログラムの処理内容を記載します。

図2－21　プログラムの実行順序

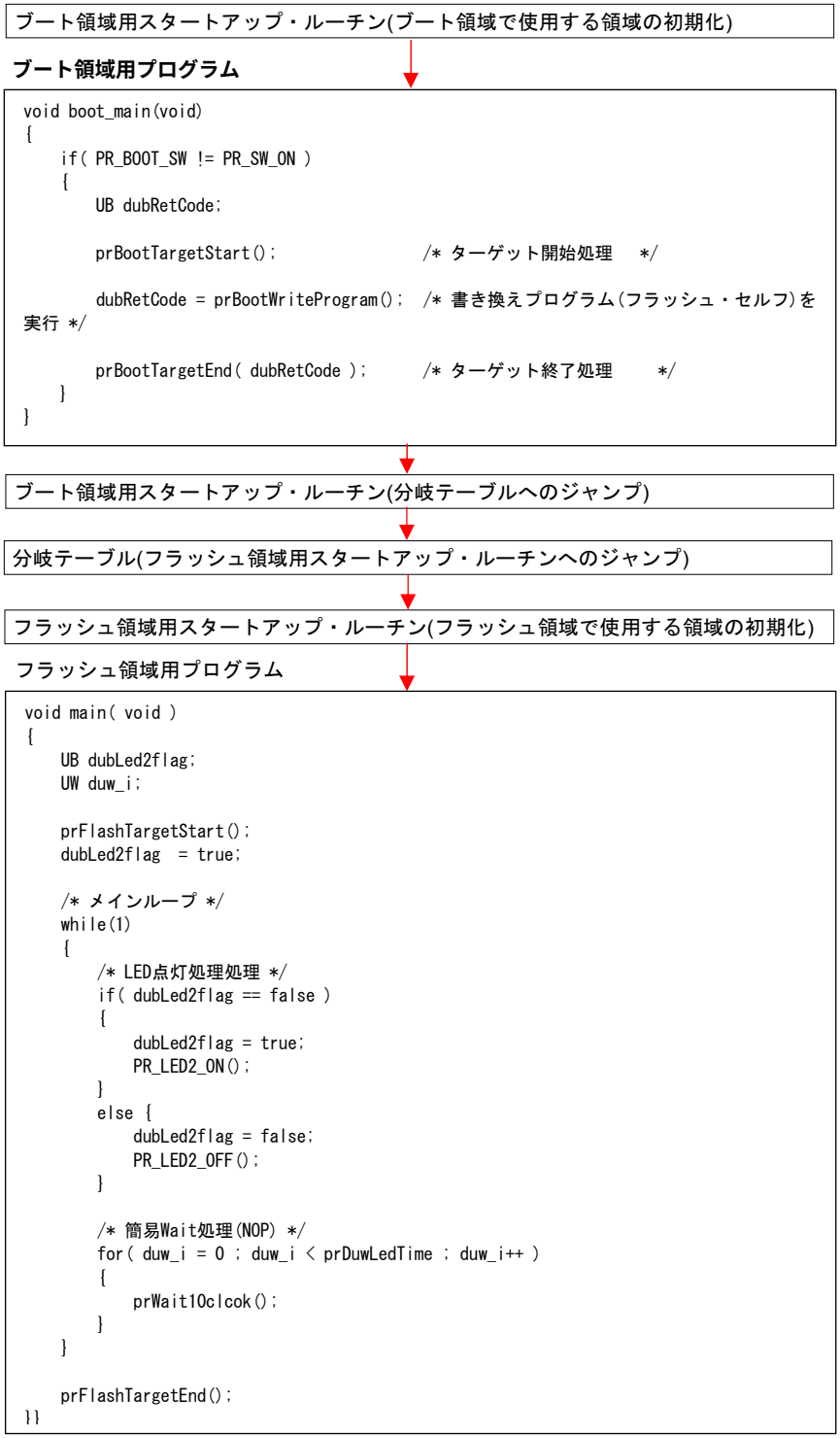
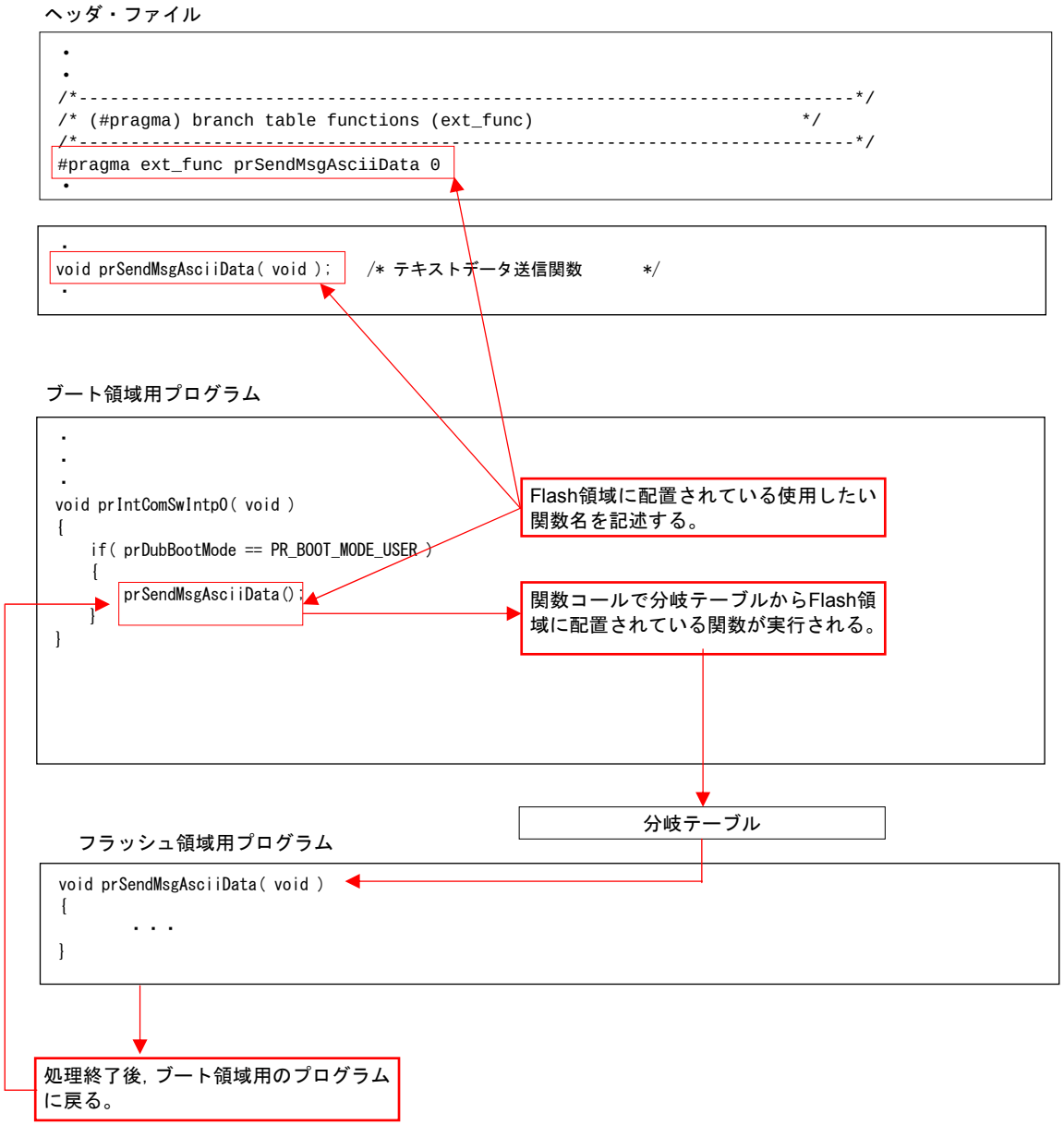


図2－22　分岐テーブルの使用方法例



2.7 メイン関数と各関数の詳細

プログラムのリストを以下に示します（主にフラッシュ・セルフ・プログラミングに関連するプログラムを掲載しています。その他のプログラムについては，サンプル・プログラムを参照してください）。

ヘッダ・ファイル（r_fsl_praxis01_com.h）で，スイッチ判定のプログラム等の内容を変更できます。仕様が異なるプログラムを用意しておくことで，フラッシュ・セルフ・プログラミングでプログラムが書き換わったことを確認できます（確認の方法については，「2.9 プログラム書き換えの評価方法」を参照してください）。

リスト2−1 ヘッダ・ファイル（r_fsl_praxis01_com.h）

```
・
・
・
/***********************/
/* サンプル内共通定義 */
/***********************/
/* 領域定義 */
#define PR_MAX_BLOCK_NUM        64          /* 最大ブロック数    */
#define PR_BLOCK_SIZE           0x400       /* ブロック容量      */
#define PR_WORD_SIZE            4           /* ワード・サイズ    */

/* SWの判別定義 */
#define PR_SW_ON                0           /* スイッチの極性    */

/* 動作モード */
#define PR_BOOT_MODE_UNKNOWN    0           /* ブート不明        */
#define PR_BOOT_MODE_WRITE      1           /* 書き換えモード    */
#define PR_BOOT_MODE_USER       2           /* ユーザ・モード    */
・
・
・
```

ブート時のスイッチ極性を変更する場合はスイッチの極性を変更する。

リンク・ディレクティブ・ファイルの設定により，フラッシュ・セルフ・プログラミングを行う書き換えプログラムはブロック0～3の領域(ブート・クラスタ0)に配置され，ユーザ・プログラムはブロック8以降に配置されます。

フラッシュ・セルフ・プログラミングを行う場合は，フラッシュ・セルフ・プログラミング・ライブラリで制限されているRAM領域以外に配置されるように，リンク・ディレクティブ・ファイルで指定を行う必要があります。フラッシュ・セルフ・プログラミング・ライブラリ使用時に使用するRAM領域については，フラッシュ・セルフ・プログラミング・ライブラリのユーザーズ・マニュアルを参照してください。

リスト2－2　ブート領域側リンク・ディレクティブ・ファイル（r_fsl_praxis01_boot_map.dr）

```
;*************************************************************
; Redefined ROM area
;*************************************************************
; -------------------------------------------------------------
; Redefined default data segment ROM
; -------------------------------------------------------------
MEMORY ROM       : ( 000000H, 001000H )   ← ブート領域の定義

; -------------------------------------------------------------
; Define new memory entry for OCD Monitor area
; -------------------------------------------------------------
MEMORY OCD_ROM   : ( 00FC00H, 000400H )   ← OCDモニター領域の定義

;*************************************************************
; Redefined RAM area
;*************************************************************
; -------------------------------------------------------------
; Define new memory entry for self-RAM
; -------------------------------------------------------------
MEMORY SELFRAM   : ( 0FEF00H, 000400H )   ← セルフRAM領域
                                            →セルフの制限領域を標準“RAM”領域から切り離す

; -------------------------------------------------------------
; Redefined default data segment RAM
; -------------------------------------------------------------
MEMORY RAM       : ( 0FF300H, 000B20H )   ← 標準“RAM”領域の再定義

; -------------------------------------------------------------
; Define new memory entry for saddr area
; -------------------------------------------------------------
MEMORY RAM_SADDR : ( 0FFE20H, 0001E0H )   ← SADDR(ショート･アドレッシング・レジスタ)領域
                                            →セルフの制限領域を標準“RAM”領域から切り離す
```

リスト2－3　フラッシュ領域側リンク・ディレクティブ・ファイル（r_fsl_praxis01_flash_map.dr）

```
;*************************************************************
; Redefined ROM area
;*************************************************************
; -------------------------------------------------------------
; Redefined default data segment ROM
; -------------------------------------------------------------
MEMORY ROM       : ( 000000H, 00F800H )   ← フラッシュ領域の定義(ブート側に重ねて設定が必要)

; -------------------------------------------------------------
; Define new memory entry for OCD Monitor area
; -------------------------------------------------------------
MEMORY ROM_DATA  : ( 00F800H, 000400H )   ← データ・テーブル領域の定義

; -------------------------------------------------------------
; Define new memory entry for OCD Monitor area
; -------------------------------------------------------------
MEMORY OCD_ROM   : ( 00FC00H, 000400H )   ← OCDモニター領域の定義(フラッシュ側でも設定が必要)

;*************************************************************
; flash segment
;*************************************************************
; -------------------------------------------------------------
; Merge FLAS_CNF segment
; -------------------------------------------------------------
MERGE FLAS_CNF := ROM_DATA                ← データ・テーブルの配置設定
```

ブート・プログラムのメイン関数では，スイッチSW1の状態にしたがって書き換えプログラムか，ユーザ・プログラムのどちらかを起動します。

リスト2－4　メイン関数（r_fsl_praxis01_boot_main.c）

```
/***********************************************************************
* Outline           : boot_main
* Include           : none
* Declaration       : void boot_main(void)
* Function Name     : boot_main
* Description       : none
* Argument          : none
* Return Value      : none
* Calling Functions : start-up routine( boot project )
***********************************************************************/
void boot_main(void)
{
    if( PR_BOOT_SW != PR_SW_ON )
    {
        UB dubRetCode;

        prBootTargetStart();                /* ターゲット開始処理    */

        dubRetCode = prBootWriteProgram();  /* 書き換えプログラム(フラッシュ・セルフ)を実行 */

        prBootTargetEnd( dubRetCode );      /* ターゲット終了処理      */
    }
}
```

書き換えプログラム(フラッシュ・セルフ・プログラミングのプログラム)

ユーザ･プログラム(LED点灯プログラム)へ遷移

書き換えプログラムはフラッシュ・セルフ・プログラミング・ライブラリを使用することを前提にしているので，初期起動時にフラッシュ・セルフ・プログラミング・ライブラリを初期化し，書き換え実行が可能な状態に遷移します。

フラッシュ・セルフ・プログラミング・ライブラリの初期化が正常に終了すると，タイマと通信ポートを設定し，コマンドの受信待ちに移行します。

リスト2−5　書き換えプログラミングのメイン関数（r_fsl_praxis01_boot_write.c）

```
UB prBootWriteProgram( void )
{
    省　略
    /* 書き込みパラメータの設定     */
    dtyWriteBuff.fsl_data_buffer_p_u08 = prDubWriteBuffer;
    dtyWriteBuff.fsl_word_count_u08    = PR_WRITE_SIZE;

    /* フラッシュ・セルフ・プログラミング・ライブラリの開始処理 */
    dubSelfResult = prFslStart();

    if( dubSelfResult == FSL_OK )
    {
        /*-- UART1のポートを通信用に初期化 115200bps --*/
        prUartinit();

        /* フラッシュ領域を2000H未満としてRL78でmemsetを使用する場合,   */
        /* far扱いの標準関数として指定した上で使わなくてはならないため、*/
        /* memsetではなく、memset_fで使用する必要があります。           */
        memset_f( prDubWriteBuffer, 0x00, PR_MSG_PACKET_SIZE );

        /* 通信ループ */
        while( duhSelfLoop == true )
        {
            /*-- SelfFlashWriterから受信 --*/
            do
            {   /* Uart コマンド・メッセージ受信 */
                dubMsgResult = prUartRcvMsg( &prDubMsgBuffer[0], &dubCommnad );

                if( dubMsgResult != PR_MSG_RET_NORM_END )
                {   /* コマンドに異常がある場合はSelfFlashWriterにエラーを送信 */
                    prUartSendMsg( dubCommnad, dubMsgResult );
                }
            }
            while( dubMsgResult != PR_MSG_RET_NORM_END );

            /*-- コマンド内容に従って処理 --*/
            switch( dubCommnad )
            {
                                {
                    省　略
                /* 該当しないコマンドは捨てる */
                default:
                    dubMsgResult =  PR_MSG_RET_PRM_ERR;
                    prUartSendMsg( dubCommnad, dubMsgResult );
                    break;


            }
        }
        dubReturn = true;
    }
    else {
        dubReturn = false;
    }

    /* フラッシュ・セルフ・プログラミング終了 */
    prFslEnd();
    return dubReturn;
}
```

書き込み専用のプログラムなので，最初にフラッシュ・セルフ・プログラミングが実行可能なように初期化処理を実行する。失敗した場合は通信をせず，終了する。

ブート領域にミラー領域が含まれていない場合は，memsetはmemset_fで実行する。

SelfFlashWriterから受信したコマンドによって処理を分岐。実行するコマンド処理については次ページ以降に記載する。

SelfFlashWriterからコマンドを受信したら，各コマンドにしたがって処理を行います。対応している受信コマンドはWRITE，DATA，IVERIFY，BOOTSWAP，RESETです。その他のコマンドについてはエラーとして返します。

以下はWRITEコマンドを受信した場合の処理です。

- 受信したブロック，アドレス，サイズの情報を，受信バッファからメモリに確保しておきます。
- 受信したデータのパラメータをチェックし，問題がなければ指定されたブロックのブランク・チェック，及び必要な場合は消去処理を実行したあと，SelfFlashWriterへ実行結果を送信します。

＜SelfFlashWriterからのWRITEコマンド＞

書き込むブロック，アドレス，サイズを送信します。

WRITEコマンドのフォーマット

スタート・コード	データ長	コマンド	データ			チェックサム
0x01	0x0008	0x05	Block	Address	Size	1バイト

リスト2－6　書き換えプログラムのWRITEコマンド処理（r_fsl_praxis01_boot_write.c）

```
                /*-- WRITEコマンド --*/
                case PR_MSG_COMM_WRITE:
                {
                    UB  dub_i;
                    UB  dubStartEraseBlock;
                    UW  duwStartWriteAddress;
                    UB  dubBlockLength;

                    /*-- 受信データ(書き込むブロック，アドレス，サイズ)をバッファから保存 --*/
                    dubStartEraseBlock    = prDubMsgBuffer[ PR_MSG_BLOCK_NUM ];
                    duwStartWriteAddress  = ( (UW)( prDubMsgBuffer[ PR_MSG_ADDR_HI  ] ) ) << 16;
                    duwStartWriteAddress |= ( (UW)( prDubMsgBuffer[ PR_MSG_ADDR_MID ] ) ) <<  8;
                    duwStartWriteAddress |= ( (UW)( prDubMsgBuffer[ PR_MSG_ADDR_LOW ] ) );
                    duwWriteSize          = ( (UW)( prDubMsgBuffer[ PR_MSG_SIZE_HI  ] ) ) << 8;
                    duwWriteSize         |= ( (UW)( prDubMsgBuffer[ PR_MSG_SIZE_LOW ] ) );
                    duwEndWriteAddress    = duwStartWriteAddress + duwWriteSize  - 1;
                    dubBlockLength        = (UB)( ( duwWriteSize - 1 ) / PR_BLOCK_SIZE ) + 1;

                    /* パラメーターチェック(0-3ブロックの保護，0サイズ書き込みの無効など。) */
                    if( ( dubStartEraseBlock >= 4                ) &&   /* 0-3ブロックは消去不可とする。*/
                        ( dubStartEraseBlock <  PR_MAX_BLOCK_NUM ) &&   /* 最大ブロック以上は不可       */
                        ( duwWriteSize       != 0                ) )    /* 書き込みサイズ0は不可        */
                    {
                        /* 書き込み対象ブロックの状態確認と消去処理 */
                        for( dub_i = 0 ; dub_i < dubBlockLength ; dub_i++ )
                        {
#ifdef PR_USE_OCD_MODE      /* OCD時はモニター領域に対しては何もしない。  */
                            if( ( dubStartEraseBlock + dub_i ) != PR_OCD_MONITOR_BLOCK )
                            {
#endif
                                DI();
                                dubSelfResult = FSL_BlankCheck( dubStartEraseBlock + dub_i );

                                /* 対象ブロックがブランク状態ではない場合 */
                                if( dubSelfResult == FSL_ERR_BLANKCHECK )
                                {

                                    dubSelfResult = FSL_Erase( dubStartEraseBlock + dub_i );
                                }

                                EI();

#ifdef PR_USE_OCD_MODE      /* OCD時はモニター領域に対しては何もしない。  */
                            }
                            else {
                                dubSelfResult = FSL_OK;
                            }
#endif
                        }

                        /* 書き込み対象アドレスの設定 */
                        duwWriteAdressIndex = duwStartWriteAddress;

                        /* フラッシュ・セルフ・プログラミングの結果を送信パラメータに変換 */
                        dubMsgResult = prFslErrorCheck( dubSelfResult );
                    }
                    else {
                        dubMsgResult = PR_MSG_RET_PRM_ERR;
                    }

                    /* 結果送信 */
                    prUartSendMsg( dubCommnad, dubMsgResult );
                    break;
                }
```

書き込み対象ブロックの状態チェックと消去

ライブラリ関数コール

ステータスを送信

WRITEコマンドを受信したあと，DATAコマンドを受信します。1回のDATAコマンドで256バイトのデータを受信し，4回で1024バイト（1ブロック）を受信することになります。

DATAコマンドで受信した書き換えデータは，WRITEコマンドの設定に従って対象ブロックに書き込まれます。

- WRITEコマンドによって指定されたブロックに256バイトの書き込みを行います。
- 次の書き込みのために，256バイト分開始アドレスを足しておきます。
- SelfFlashWriterへ実行結果を送信します。

＜SelfFlashWriterからのDATAコマンド＞

書き込むデータを送信します。

DATAコマンドのフォーマット

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0102	0x06	256バイト	1バイト

リスト2−7　書き換えプログラムのDATAコマンド処理（r_fsl_praxis01_boot_write.c）

```
      /*-- DATAコマンド --*/
                /*-- DATAコマンド --*/
                case PR_MSG_COMM_DATA:

                    /* 書き込み対象アドレスが終了アドレスより小さいかを判断 */
                    if( duwWriteAdressIndex <= duwEndWriteAddress )
                    {
#ifdef PR_USE_OCD_MODE  /* OCD時はモニター領域に対しては何もしない。  */
                        if( duwWriteAdressIndex < PR_OCD_MONITOR_ADDR )
                        {
#endif
                            /* 書き込みデータをデータ・バッファにコピー */
                            memcpy_f( &prDubWriteBuffer[0], &prDubMsgBuffer[0], PR_MSG_PACKET_SIZE );

                            dtyWriteBuff.fsl_destination_address_u32 = duwWriteAdressIndex;
                            DI();
                            dubSelfResult = FSL_Write( &dtyWriteBuff );
                            EI();

#ifdef PR_USE_OCD_MODE  /* OCD時はモニター領域に対しては何もしない。  */
                        }
                        else {
                            dubSelfResult = FSL_OK;
                        }
#endif

                        /* フラッシュ・セルフ・プログラミングの結果を送信パラメータに変換 */
                        dubMsgResult = prFslErrorCheck( dubSelfResult );

                        /*書き込み対象アドレスを書き込みサイズ分更新*/
                        duwWriteAdressIndex += PR_MSG_PACKET_SIZE;
                    }
                    else {
                        dubMsgResult = PR_MSG_RET_ERR_END;
                    }

                    /* 結果送信 */
                    prUartSendMsg( dubCommnad, dubMsgResult );

                    break;
```

ブート領域にミラー領域が含まれていない場合は，memsetはmemset_fで実行する。

ライブラリ関数コール

1ブロックの書き込みが終了すると，SelfFlashWriterから対象ブロックに対してのIVERIFYコマンドが送信されます。IVERIFYコマンドを受信すると，対象ブロックに対してIVERIFY処理を実行します。

- WRITEコマンドによって指定されたブロックにIVERIFY処理を実行します。
- SelfFlashWriterへ実行結果を返信します。

＜SelfFlashWriterからのIVERIFYコマンド＞

IVERIFYを行うブロック番号を送信します。

IVERIFYコマンドのフォーマット

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0003	0x0B	Block	1バイト

リスト2－8　書き換えプログラムのIVERIFYコマンド処理（r_fsl_praxis01_boot_write.c）

```
/*-- IVERIFYコマンド --*/
case PR_MSG_COMM_IVERIFY:

    /* ベリファイ処理 */
    DI();
    dubSelfResult = FSL_IVerify( prDubMsgBuffer[ PR_MSG_IVERIFY_BLOCK ] );
    EI();

    /* フラッシュ・セルフ・プログラミングの結果を送信パラメータに変換して結果送信 */
    dubMsgResult = prFslErrorCheck( dubSelfResult );
    prUartSendMsg( dubCommnad, dubMsgResult );
    break;
```

ライブラリ関数コール

SelfFlashWriterはChip Modeで書き換えを行った場合，全ての書き換えデータを送信，正常終了を確認するとBOOTSWAPコマンドを送信します。
以下はBOOTSWAPコマンドを受信した場合の処理です。

- ブート・フラグの書き換えを実行します。
- SelfFlashWriterへ実行結果を送信します。
- リセットを実行します。

＜SelfFlashWriterからのBOOTSWAPコマンド＞

BOOTSWAPコマンドを送信します。

BOOTSWAPコマンドのフォーマット

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0002	0x08	なし	1バイト

リスト2－9　書き換えプログラムのBOOTSWAPコマンド処理（r_fsl_praxis01_boot_write.c）

```
                /*-- BOOTSWAPコマンド --*/
                case PR_MSG_COMM_BOOTSWAP:

                /* OCD時は実行不可 */
#ifdef PR_USE_OCD_MODE
                    /* OCD中の場合は何もせず正常終了 */
                    prUartSendMsg( dubCommnad, PR_MSG_RET_NORM_END );
#else
                    /* OCD中で無ければ処理を実行 */
                    /* ブート・フラグの書き換え処理 */
                    DI();
                    dubSelfResult = FSL_InvertBootFlag();
                    EI();

                    /* フラッシュ・セルフ・プログラミングの結果を送信パラメータに変換して結果送信 */
                    dubMsgResult  = prFslErrorCheck( dubSelfResult );
                    prUartSendMsg( dubCommnad, dubMsgResult );

                    /* 正常終了であれば強制リセット処理 */
                    if( dubMsgResult == PR_MSG_RET_NORM_END )
                    {
                        /* UART通信終了処理 */
                        prUartEnd();

                        /* 強制リセット処理 */
                        FSL_ForceReset();
                    }
#endif
                    break;
```

ライブラリ関数コール

その他，SelfFlashWriterのResetボタンを押下すると，RESETコマンドが送信されます。

以下はRESETコマンドを受信した場合の処理です。

- SelfFlashWriterへ受信結果を送信します。
- リセットを実行します。

＜SelfFlashWriterからのRESETコマンド＞

RESETコマンドを送信します。

RESETコマンドのフォーマット

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0002	0x07	なし	1バイト

リスト2－10　書き換えプログラムのRESETコマンド処理（r_fsl_praxis01_boot_write.c）

```
                /*-- RESETコマンド --*/
                case PR_MSG_COMM_RESET:
                /* OCD時は実行不可 */
#ifdef PR_USE_OCD_MODE
                    /* OCD中の場合は何もせず正常終了 */
                    prUartSendMsg( dubCommnad, PR_MSG_RET_NORM_END );
#else
                    /* OCD中で無ければ処理を実行 */
                    /* 受信結果を送信    */
                    dubMsgResult = PR_MSG_RET_NORM_END;
                    prUartSendMsg( dubCommnad, dubMsgResult );

                    /* UART通信終了処理 */
                    prUartEnd();

                    /* 強制リセット処理 */
                    FSL_ForceReset();          <-- ライブラリ関数コール
#endif
                    break;
```

2.8 デバッグ時の注意事項

サンプル・プログラムの評価を行う場合，次の（1），（2），（3），（4）に示す内容に注意して下さい。

(1) オンチップ・デバッグ中のブート・スワップについて

ブート・クラスタ0，及びブート・クラスタ1に配置されているプログラムは，スワップ後にアドレスが変更される為，ブート・クラスタ0，及びブート・クラスタ1に書き込まれているプログラムが，同一のプログラムで無い限り，プログラムの実行をデバッガ上で正常に確認できなくなります。

正常に動作できなくなった場合は，デバッガを終了し，一旦ターゲットの電源を落とした上で，再接続してください。

(2) オンチップ・デバッグ中のリセットや書き換えプログラムの動作確認について

オンチップ・デバッグ中にデバッガ側ではなく，プログラム側でリセットを行う場合，FSL_ForceReset()関数や不正命令実行によるソフト・リセットを行うと事はできません。

また，オンチップ・デバッグを使用する場合は，コード・フラッシュ・メモリの領域の一部にモニタープログラムを配置する必要があり，その領域を書き換えてしまうと，オンチップ・デバッグが正常に動作できなくなります。

サンプル・プログラムはROM動作を想定して作成しているため，オンチップ・デバッグを使用してサンプル・プログラムの全領域書き換え等の確認を行う場合，上述の内容により，一部デバッガにて正常に動作できないプログラムがあります。

オンチップ・デバッグを使用してサンプル・プログラムを確認する場合は，ヘッダ・ファイル（r_fsl_praxis01_com.h）にある「#define PR_USE_OCD_MODE」を有効にしてください。

ただし，この設定を行うとプログラム中のリセット処理と，オンチップ・デバッグ用のモニタープログラム領域に対する書き換え処理は行われなくなります。

リスト2－11 書き換えプログラムのRESETコマンド処理（r_fsl_praxis01_com.h）

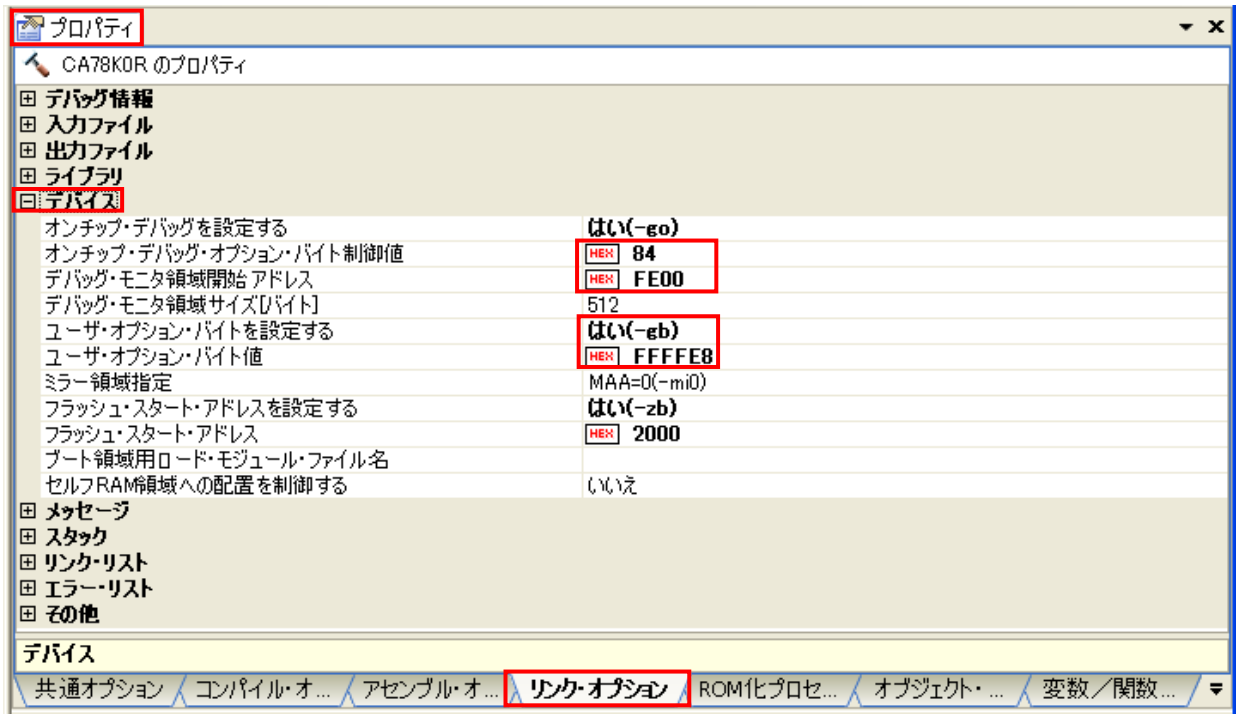
```
/*********************************************************/
/* サンプル・プログラム・プログラムスイッチ用シンボル */
/*********************************************************/
/* QB-R5F100LE-TB単体使用時ON可能 */
#if 1
#define __QB_R5F100LE_TB__                  /* R5F100LE(64Kbyte) TBボード */
/*#define PR_USE_OCD_MODE*/                 /* オンチップ・デバッグ使用時 */

/* ターゲット無し */                                  コメント・アウトを外す
#else
#define __NON_TARGET__
#endif

/* OCD使用判別 */
#ifdef PR_USE_OCD_MODE
    #define PR_OCD_MONITOR_BLOCK    0x3F    /* OCDモニター領域を含むブロック          */
    #define PR_OCD_MONITOR_ADDR     0xFC00  /* OCDモニター領域を含むブロックのアドレス */

#endif
```

(3) オプション・バイトとオンチップ・デバッグの設定について

サンプル・プログラムの通信処理は高速オンチップ・オシレータ(高速OCO)を32MHzに設定する事で正常に動作します。付属のプロジェクト・ファイルでは，ブート領域側プロジェクトにオプション・バイトの値を「FFFFE8」に指定し，WDTの設定と共に高速オンチップ・オシレータを32MHzに設定していますが，プロジェクトを最初から作成する場合は，デフォルトでは設定されていません。ユーザ側で作成したプロジェクトへサンプル・プログラムを組み込む場合は，ブート領域側のオプション・バイトの値を「FFFFE8」に設定してください。

また，オンチップ・デバッグを行う場合はブート領域側のオンチップ・デバッグ設定を「はい」に指定してください。

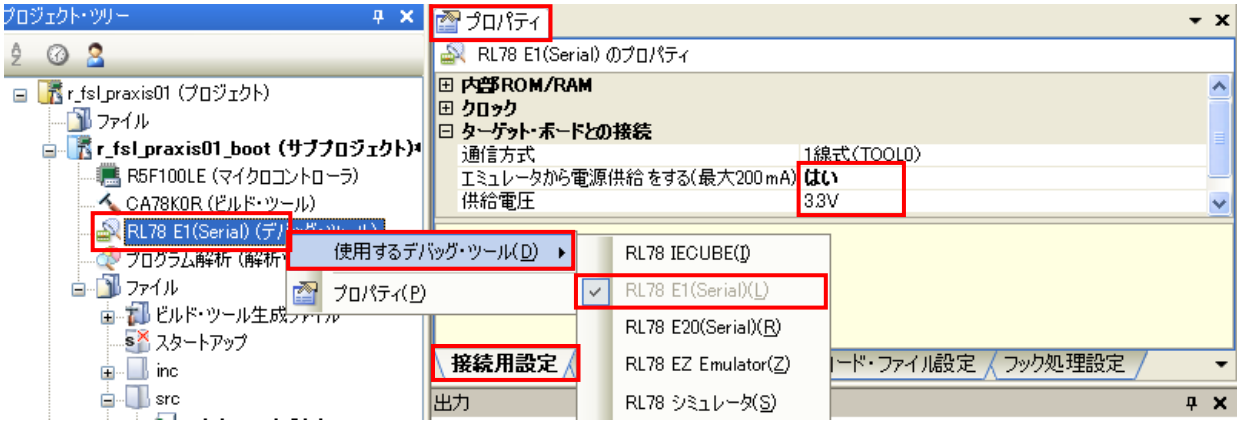
図2－23　オプション・バイトの設定



(4) E1エミュレータの設定と電源供給について

デバッグ時にE1エミュレータを使用する場合，プロジェクト・ファイルのデバッグツールをデフォルトのシミュレータからE1エミュレータに変更してください。また，E1エミュレータから電源を供給する場合は，「エミュレータから電源供給をする」を「はい」に指定してください。

図2－24　E1エミュレータの設定



2.9 プログラム書き換えの評価方法

フラッシュ・セルフ・プログラミングを使用して，仕様の異なるプログラムを書き込むことで，プログラムが書き換わることを確認することができます。次の（1），（2）に示す方法でプログラムを評価してください。

(1) 全領域プログラムのヘキサ・ファイルをQB-R5F100LE-TBに書き込む

フラッシュ領域プログラムのプロジェクトフォルダ¥DefaultBuild以下のファイル「r_fsl_praxis01_flash.hex」を，フラッシュ・メモリ・プログラマなどでRL78/G13に書き込みます。（このファイルは，ブート領域とフラッシュ領域の両方のプログラムを含んだヘキサ・ファイルになります。）

QB-R5F100LE-TBのSW1を押下したままリセットを行うと，ユーザ・プログラムが起動しLED1が常時点灯し，LED2が点滅します。その状態でSW1を押下すると，パソコン側へASCⅡデータが送信されるので，ターミナルソフト等で表示データを確認します。その後，自動的にWDTによるリセットが発生し， SelfFlashWriterからの通信待ちとなり，LED2が常時点灯，LED1が消灯します。

(2) SelfFlashWriterでプログラムを書き込む

（1）で書き込んだプログラムで書き込みプログラミングを実行します。

図2－25のようにSelfFlashWriterより書き換え用のプログラム「r_fsl_praxis01_write_test.hex」を指定します。フラッシュ領域を書き換える場合は「Operation Mode」を「Block」にチェックし，「Start」ブロック番号008，「Stop」ブロック番号063を設定してください（フラッシュ領域全体を指定します）

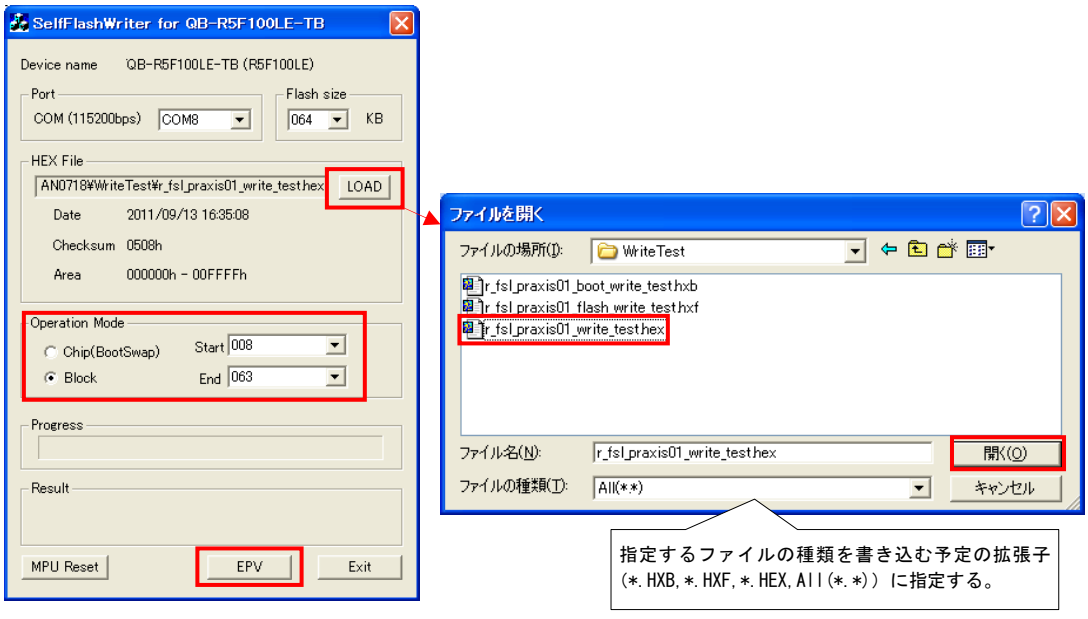
「EPV」ボタンをクリックすると通信が始まり，フラッシュ領域のプログラムのみが書き換えられます。

書き込みが終了したら，QB-R5F100LE-TBのSW1を押下した状態でリセットし，ユーザ・プログラムのみがLED1とLED2の表示方法が変わることを確認してください。このとき，書き込みプログラム側のLED1とLED2の表示は変更されません。

全領域を書き換える場合は「Operation Mode」を「Chip」にチェックし，「EPV」ボタンをクリックすると書き換えが始まります。書き込みが終了したら自動的にリセットが実行されます。書き換え後，ユーザ・プログラムと書き換えプログラムのLEDの表示が両方とも逆になることを確認してください。

備考：「EPV」実行後は、「EXIT」ボタンをクリックし、SelfFlashWriteを一度終了させてください。

図2－25 SelfFlashWriterによる書き込み



2. 10 データ書き換えの評価方法

フラッシュ・セルフ・プログラミングを使用して，データ・テーブルを更新し，フラッシュ領域用プログラム(ユーザ・プログラム)で使用されるデータが変更されることを確認します。次の（1），（2）に示す方法で，評価を行ってください。

（1）全領域プログラムのヘキサ・ファイルをQB-R5F100LE-TBに書き込む

フラッシュ領域プログラムのプロジェクトフォルダ¥DefaultBuild以下のファイル「r_fsl_praxis01_flash.hex」を，フラッシュ・メモリ・プログラマなどでRL78/G13に書き込みます。（このファイルは，ブート領域とフラッシュ領域の両方のプログラムを含んだヘキサ・ファイルになります。）

QB-R5F100LE-TBのSW1を押下したままリセットを行うと，ユーザ・プログラムが起動しLED1が常時点灯し，LED2が点滅します。その状態でSW1を押下すると，パソコン側へASCⅡデータが送信されるので，ターミナルソフト等で表示データを確認します。その後，自動的にWDTによるリセットが発生し，SelfFlashWriterからの通信待ちとなり，LED2が常時点灯，LED1が消灯します。

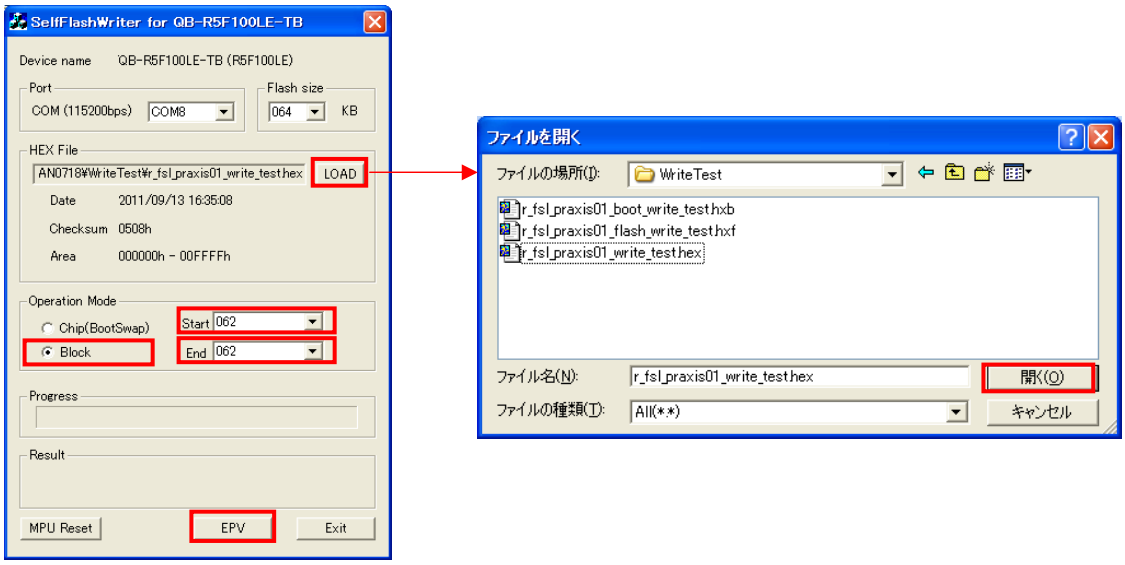
（2）SelfFlashWriterでフラッシュ領域のデータ・テーブルを更新する

（1）で書き込んだプログラムで書き込みプログラミングを実行します。図2－26のようにSelfFlashWriterよりフラッシュ領域用プログラム「r_fsl_praxis01_write_test.hex」を指定します。「Operation Mode」をBlock書き換えモードに変更し，StartブロックとEndブロックに62を指定して「EPV」ボタンをクリックすると通信が始まります。フラッシュ・セルフ・プログラミングのプログラムにより，プログラムのデータ・テーブルが書き換えられます。

書き換え後にQB-R5F100LE-TBのSW1を押下した状態でリセットし，ユーザ・プログラムを起動して，SW1押下時にパソコン側へ送信されるASCⅡデータだけが変更されることを確認します。

備考：「EPV」実行後は、「EXIT」ボタンをクリックし、SelfFlashWriteを一度終了させてください。

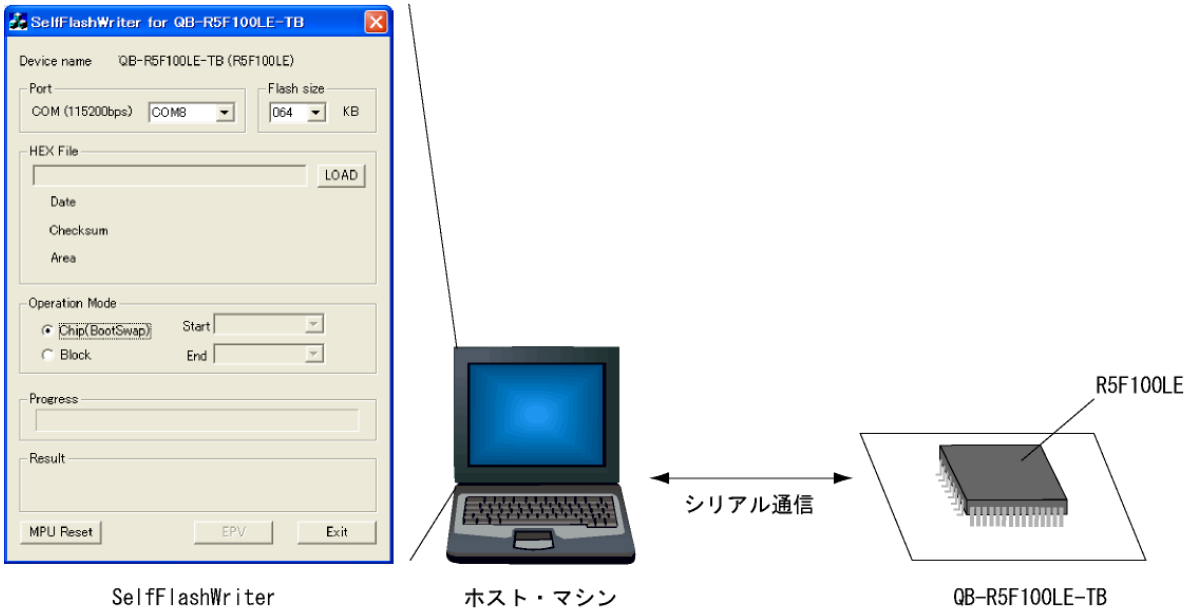
図2－26 SelfFlashWriterによる書き込み



付録A　SelfFlashWriter

SelfFlashWriterは，PCから指定された機械語ファイル（*.hex）を，フラッシュ・メモリの最小消去単位（1ブロック=1024バイト）に分割してシリアル通信で送信するGUIです。フラッシュ・セルフ・プログラミングを評価するための仮想的な治具として使うことができます。

図A－1　SelfFlashWriterとQB-R5F100LE-TBの接続イメージ



（1）使用環境

SelfFlashWriterは以下の環境でご使用ください。

表A－1　SelfFlashWriterの使用環境

CPU	Pentium® III 500MHz 以上
対応OS	Windows® 2000/ Windows XP®/ Windows Vista®/ Windows® 7
メモリ	512Mバイト以上
HDD容量	約7Mバイト

（2）通信仕様

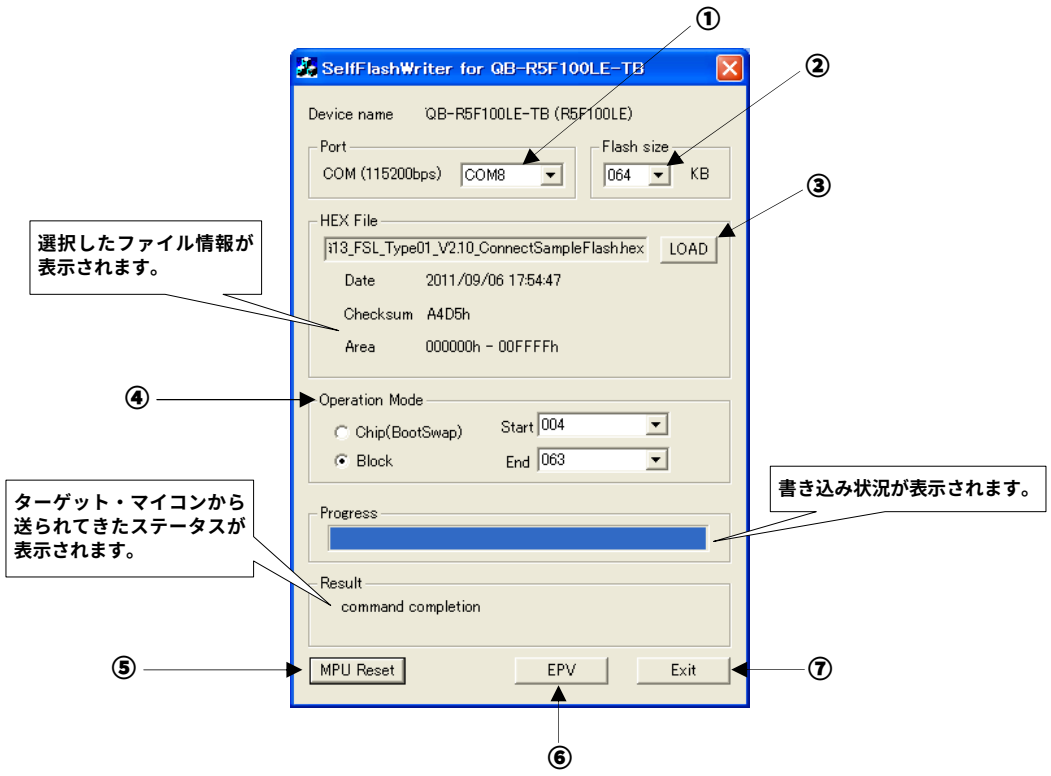
SelfFlashWriterは以下のシリアル通信仕様でRL78/G13(R5F100LE)と通信します。

表A－2　SelfFlashWriterの通信仕様

ビット／秒	115200
データ・ビット長	8
パリティ	なし
ストップ・ビット	1
フロー制御	なし

（3）各機能

図A－2 SelfFlashWriterの各機能



① COMポートの選択

通信ポートを1～16まで，プルダウンメニューから選択することができます。

② フラッシュ・メモリサイズの設定

RL78/G13(64kByte以下)のフラッシュ・メモリのサイズを指定します。

③ LOADボタン

RL78/G13(R5F100LE)に書き込む機械語ファイル（*.hex）を指定します。

④ Operation Mode チェック

Chip： チップ全体の書き換えを行う場合に選択します。このモードをチェックした状態で「EPV」ボタンを押すと，書き換えデータを送信し，全ての書き換えが終了した後，BOOTSWAPコマンドを送信します。

備考：「Block」をチェックし、「Start」ブロック番号に004、「Stop」ブロック番号に063(最終のブロック番号)を指定してください。

Block： ブロック単位で書き換えを行う場合に選択します。書き換えるブロックの範囲は，START，ENDのプルダウンで指定します。機械語ファイル（*.hex）の開始，終了アドレスに関わらず，START，ENDで設定したアドレス範囲を書き換えます（機械語ファイルにない範囲を指定した場合は消去（FFhで書き換え）します）。尚，ブロック0-3の書き換えはできません。

⑤ MPU Resetボタン

RL78/G13(R5F100LE)にRESETコマンドを送信し，RL78/G13(R5F100LE)側でリセットを行います。

⑥ EPV（Erase-Program-Verify）ボタン

RL78/G13(R5F100LE)へ各コマンドを送信します。RL78/G13(R5F100LE)ではフラッシュ・メモリの消去，データの書き込み，内部ベリファイを行います。

備考：「EPV」実行後は、「EXIT」ボタンをクリックし、SelfFlashWriteを一度終了させてください。

⑦ Exitボタン

SelfFlashWriterを終了します。

(4) 通信コマンド

次に各通信コマンドの内容とフォーマットを示します。

① SelfFlashWriterが送るコマンド

＜WRITEコマンド＞

書き込むブロック，アドレス，サイズを送信します。

スタート・コード	データ長	コマンド	データ			チェックサム
0x01	0x0008	0x05	Block	Address	Size	1バイト

＜DATAコマンド＞

書き込むデータを256バイト送信します。

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0102	0x06	256 バイト	1バイト

＜IVERIFYコマンド＞

IVERIFYコマンドを送信します。

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0003	0x0B	Block	1バイト

＜RESETコマンド＞

RESETコマンドを送信します。

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0002	0x07	なし	1バイト

＜BOOTSWAPコマンド＞

BOOTSWAPコマンドを送信します。Operation ModeでChipが選択されている場合に使用されます。

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0002	0x08	なし	1バイト

② RL78/G13(R5F100LE)が送るコマンド

以下のコマンドは，SelfFlashWriterからのコマンドに対してRL78/G13(R5F100LE)が返すコマンドです。このフォーマットにしたがってコマンドを返すようにプログラムを作成してください。

＜DATA_REVコマンド＞

SelfFlashWriterからの書き込み関連のコマンド（WRITEコマンド，DATAコマンド）に対して送るACK（返答）コマンドです。

スタート・コード	データ長	コマンド	データ	チェックサム
0x01	0x0003	返答コマンド	Status注	1バイト

注 Statusは，SelfFlashWriterが送ったコマンドの実行結果を表します。詳細は，表A−3 ステータス一覧を参照してください。

表A−3 ステータス一覧表

ステータス名	値	内 容
正常終了	0x00	正常終了
異常終了	0x01	異常終了
パラメータ・エラー	0x05	通信フォーマットのパラメータ・エラー
プロテクト・エラー	0x10	指定ブロックがブート領域に含まれており，ブート領域書き換え禁止が設定されている
消去エラー	0x1A	消去エラー
書き込みエラー	0x1C	書き込みデータが正しく書き込めない
ベリファイ・エラー	0x1D	書き込み後のベリファイ・エラー
チェックサム・エラー	0xFF	通信フォーマットのチェックサム・エラー

(5) 通信コマンドのやりとり

SelfFlashWriterとRL78/G13(R5F100LE)との通信コマンドのやりとりは次の①，②のようになります。

RL78/G13(R5F100LE)側が必ず①，②に示すやりとりのとおりにコマンドを返すよう，プログラムを作成してください。

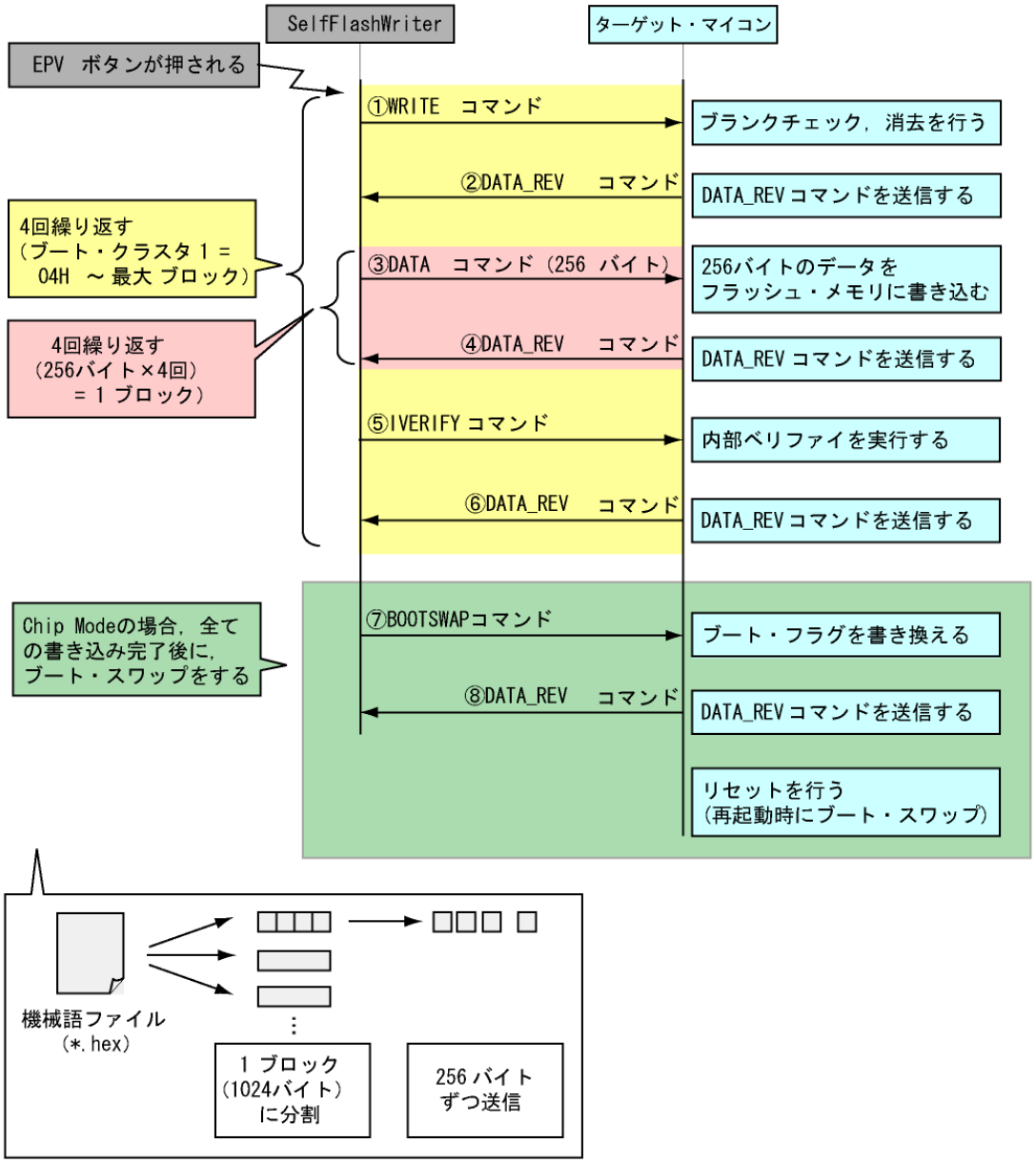
① Operation Mode でBlockを選択した場合の「EPV」実行フロー

SelfFlashWriterで指定した機械語ファイルを書き込みます。機械語ファイルは，ブロック単位（1024バイト）に分けられ，1回のWRITEコマンドで1ブロック分を書き込みます。送信するときは，さらに256バイトずつ4回に分けて送られます。SelfFlashWriterからの各コマンド送信のあとは，RL78/G13(R5F100LE)側からのDATA_REV応答待ちとなり，応答が来ない場合はタイムアウト・エラーとなります。

② Operation Mode でChipを選択した場合の「EPV」実行フロー

このモードで書き込む場合は，RL78/G13(R5F100LE)が，ブート・スワップ機能を利用して，プログラムの全領域を書き換えます。SelfFlashWriterは，全ての書き換えが終了した後，BOOTSWAPコマンドを送信します。SelfFlashWriterからの各コマンド送信のあとは，RL78/G13(R5F100LE)側からの応答待ちとなり，応答が来ない場合はタイムアウト・エラーとなります。

図A－3 EPV実行フロー



〔メ　　モ〕

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/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2011.09.30	—	新規作成
1.01	2012.11.22	全頁	アプリケーションノートのフォント、及びフォーマット調整を実施
1.02	2013.02.15	全頁	フラッシュ・セルフ・プログラミング・ライブラリ Type01 Ver.2.20に対応 CubeSuite+をV1.03に変更

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・電話番号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） (03)5201-5307

■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 2.0