お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

H8/300L SLP シリーズ

SLP ユーザモードプログラミング (UserMP)

要旨

このアプリケーションノートでは, SLP MCU を使ってユーザモードでフラッシュメモリのプログラミングを実行する方法を説明します。このドキュメントには，以下のソースコードを含みます。

1. ユーザモードプログラミングカーネル
2. ユーザモードデモプログラム
3. フラッシュプログラミング GUI (TCL/TK ベースのソフトウェア)

動作確認デバイス

H8/38024F

目次

1. 概要

フラッシュ MCU には，ブートモードとユーザモードの 2 つの動作モードがあります。

ブートモードでは，MCU はシリアルポートで外部と通信します。MCU 起動時の初期状態ではプログラムがないので，シリアルポートを使って MCU のフラッシュメモリへ書き込みます。ブートモードでのフラッシュメモリへの書き込みは，アプリケーションノート “In-circuit boot mode programming”に詳しく説明してあります。（このモードでは，MCU にブートモードカーネルがあるので，ユーザはコードを書き込む必要がありません。）

プログラムを書き込まれると，MCU はユーザモードで起動してプログラムを実行することができます。ユーザモードでは，フラッシュメモリへの書き込みや (再) 書き込みができます。しかし，ユーザはユーザカーネル，ホストとのインタフェースプログラム，およびホスト制御ソフトウェア (ホストはパソコンでも他の組み込みシステムでも良い) を準備しなければなりません。

1.1 ブートモードでの書き込み

ブートモードでは，空白のデバイスをインサーキットで自動書き込みする機能，または書き込み前にチップを自動消去してデバイスに再書き込みする機能が使用できます。チップのリセット状態からブートモードに入ると，LSI 内のブートプログラム (もともとチップに内蔵されている) が開始し，以下の機能を実行します。

1. 外部ホストとのシリアルポートのボーレートの自動検出
2. シリアルポートを介して，ユーザから提供されたブートカーネルの RAM へのダウンロード
3. ブートプログラム内の消去プログラムの実行，またそれに伴うすべてのフラッシュメモリ内容の消去
4. ダウンロードしたブートカーネルの実行

以下の例では,1 線プログラミングのデバイスのブートモードへの移行とブートモードの終了を示します。(デバイスの詳細については，ハードウェアマニュアルを参照してください。)

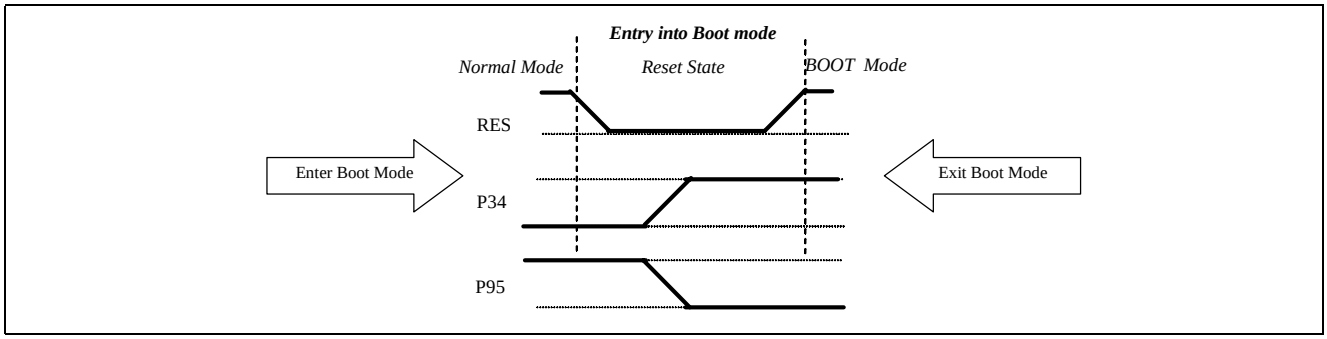

図1 ブートモードへの遷移タイミング

ブートカーネルを実行すると，チップ全体が消去され，書き込みができる状態になります。ブートカーネルはユーザによって提供されるため，どのような機能も実行させることができますが，チップが空白であるため，書き込み機能を含むことが必要です。ブートカーネルは，データのダウンロード用にシリアルポートを使用するなど，必要なデータ (パラレルインタフェース，CAN バス等) を獲得するために，チップに内蔵する他の周辺機能を使用することもできます。書き込み動作が完了すると，必要なモードピンをリセットして通常動作用の値に設定してください。また，チップもリセットされます。

1.2 ユーザモードでの書き込み

ユーザモードでフラッシュメモリに書き込むときには, MCU をリセットせずにフラッシュメモリの一部を変更するだけで，バージョンやデータの更新などのフレキシブルな対応ができます。ユーザが通信方法を決定できるので，シリアルポートや他のデータメディアとしている通信チャネルが考えられます。

ユーザモードで書き込みをするには，以下のソフトウェアが必須です。

- **ホストコントローラ (GUI)** は MCU と通信するシステムの 1 つです。MCU のフラッシュメモリに焼き付けるデータを提供して，MCU のフラッシュメモリに，連続したデータを「焼き付ける」ことができます。このアプリケーションノートでは，ホストコントローラとして PC の GUI を使用しています。GUI のソフトウェアは TCL/TK スクリプトで記述されています。
- **ユーザモードホストインタフェースルーチン (UI)** はホストとのインタフェースを行うソフトウェアであり，通信チャネルや転送プロトコルを決定します。
- **ユーザモードフラッシュカーネル (KERNEL)** はフラッシュ書き込みのメインコントローラです。消去と書き込みの処理の詳細 (0.35µm フラッシュメモリへの書き込みアルゴリズム)を含みます。
- **アプリケーションソフトウェア (APPLICATION)** は特定の組み込みシステムのタスクを実行するユーザのターゲットアプリケーションプログラムです。

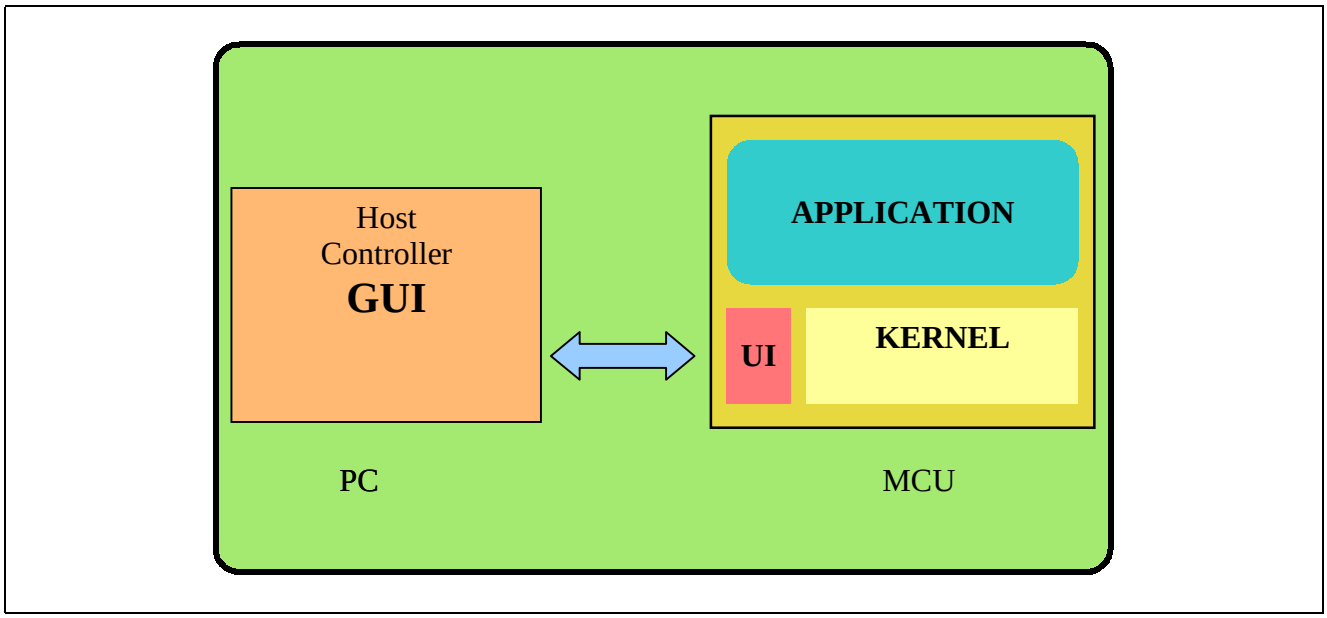

図2 ユーザモードでの書き込みの概要

2. GUI

ブートモードおよびユーザモードの両モードで，GUI は以下の処理をします。

1. S レコード形式出力ファイルをバイナリーフォーマットにデコードする。
2. MCU 内の UI ルーチンとの通信を確立する。
3. シリアルポートを介して MCU に機械語コードをダウンロードする。

この GUI では 2 種類のフラッシュメモリのモードをサポートします。

1. ブートモード
2. ユーザモード

2.1 GUI の概要

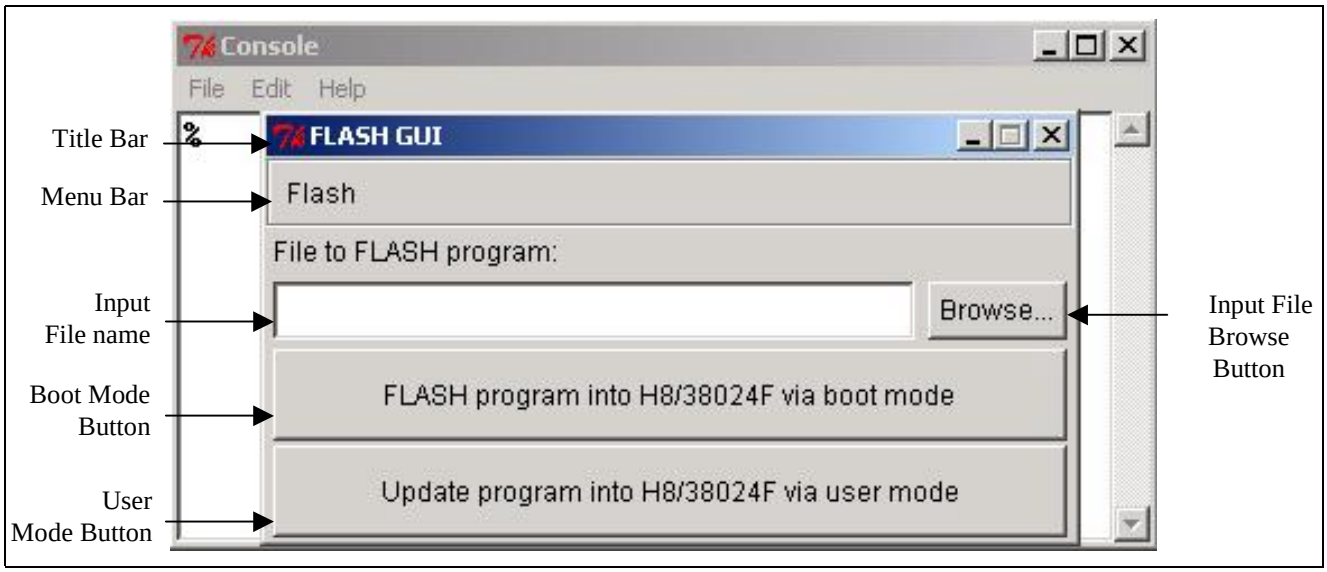

図3 フラッシュ GUI ダイアログボックス

メニューバー: Flash

フラッシュメニューバーの “Quit” をクリックすると，フラッシュ GUI を終了します。

入力ファイル名

ユーザが，フラッシュメモリにダウンロードする S レコード形式ファイルを選択できます。

入力ファイルブラウズボタン

このコマンドで Windows ファイルを開くための標準のダイアログボックスを立ち上げます。ユーザは S レコードファイルを一度に 1 つだけ選択できます。

ブートモードボタン

このコマンドにより，現在入力された S レコードファイルをダウンロードします。フラッシュ書き込み動作により，選択した S レコードファイルからのデータを対象となるフラッシュメモリへ書き込みます。この動作は，ブートモードでおこなわれるため，ユーザは，ブートモードに入るには，ターゲットデバイスをリセットしなければいけないことに注意してください。

ユーザモードボタン

このコマンドはブートモードのリセットをせずに現在入力された S レコードファイルでターゲットデバイスを更新します。注意：ユーザは，ソフトウェア更新動作中に，(フラッシュ ROM 内の) インタフェースソフトウェアを上書きまたは消去してはいけません。フラッシュメモリ全体を上書きする必要があるときは，インタフェースソフトウェアを ROM ではなく RAM に格納することを推奨します。

2.2 GUI スクリプト言語

2.2.1 TCL/TK の概要

TCL (Tool Command Language) (“tickle”) はシンプルなインタプリタ的なプログラミング言語です。

TCL の主な特長を以下に示します。

- 高級スクリプト言語
  高級プログラミング言語の経験者にとって TCL は他の言語と類似しています。
- インタプリタ
  コードをコンパイルやリンクなしで直接コーディングできます。
- 拡張可能
  ユーザが独自のコマンドを追加して TCL 言語を拡張できます。
- アプリケーションに組み込み可能
  TCL インタプリタは各種アプリケーションへの組み込み用として開発されました。TCL をアプリケーションに簡単に組み込むことができ, まるでそのアプリケーションのために作成されたかのように自然に融合します。
- 多くのプラットフォームで実行可能
  多少の変更は必要ですが，Windows，UNIX，Macintosh のプラットフォームをサポートします。
- 無償
  TCL のソースプログラムはインターネットにもあり，商用のアプリケーションでも無償で使用でます。

TK ツールキットは, TCL スクリプト言語と共に動作するウィンドウプログラミング用のグラフィカルユーザインタフェースツールです。最終的には他のウィンドウシステムへのポートもありますが，X ウィンドウシステム用に設計されています。TK は他のウィンドウ操作のツールキットと多くの共通した概念を持ちますが，ユーザが TK を開始するにあたって，グラフィカルユーザインタフェースについての知識はほとんど必要としません。

TK は TCL のコマンドセットを持ち，それが「ウィジェット」を作成し操作します。ウィジェットはグラフィカルユーザインタフェースのウィンドウの一種であり, 独特の表示や動作をします。ウィジェットとウィンドウは同じ意味で使われることも多いです。ウィジェットは，ボタン，スクロールバー，メニュー，およびテキストウィンドウを含みます。

図4 TCL/TK スクリプト解釈プログラム

2.2.2 TCL/TK LICENSE TERMS

This software is copyrighted by the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

2.2.3 TCL/TK スクリプト解釈プログラムのインストール

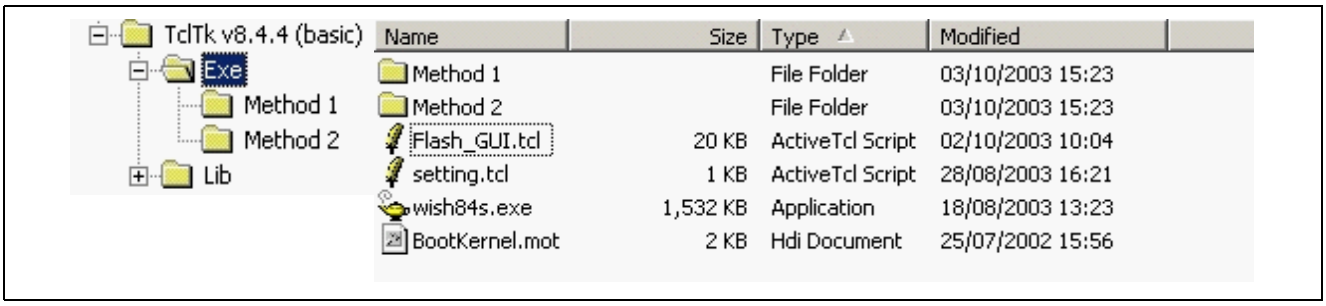

図5 TclTk v8.4.4 (basic) フォルダの中身

2.2.4 TCL/TK スクリプト解釈プログラムの実行

wish84s.exe をダブルクリックすると，TCL/TK スクリプト解釈プログラムが実行されます。

図6 TCL/TK コンソール

[File → Sources…] をクリック → Flash_GUI.tcl を選択→ [Open] をクリックしてください。

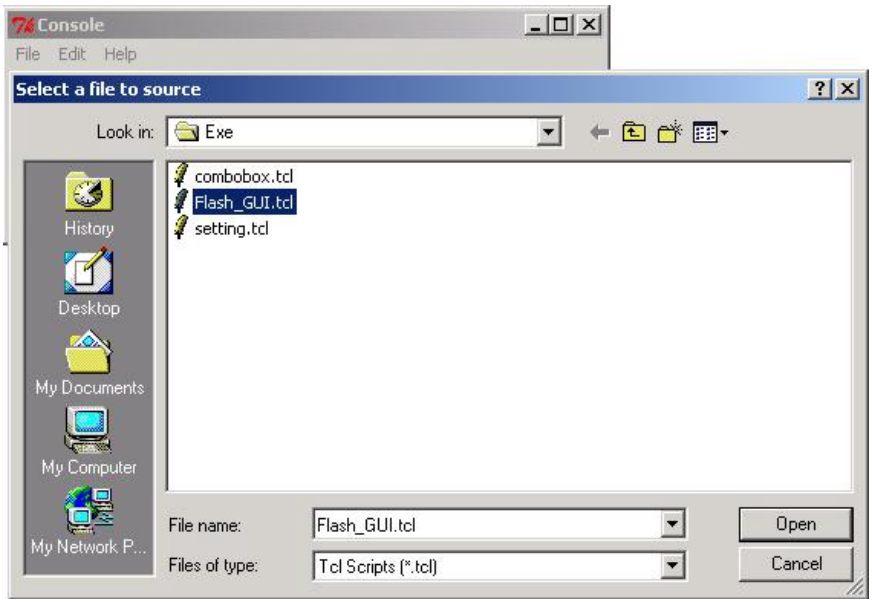

図7 Flash_GUI.tcl ファイルを開く

2.3 GUI の構成

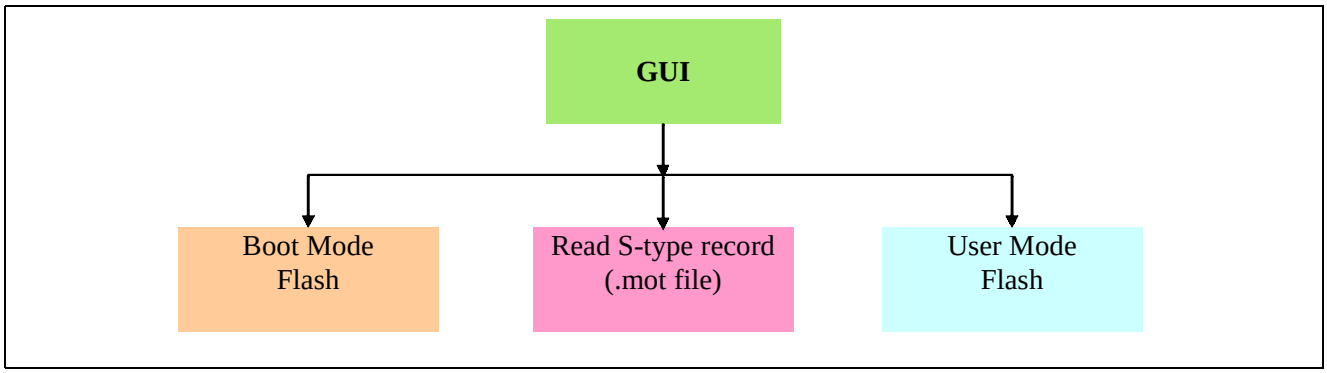

図8 GUI 概要

3 つの基本ソフトウェアモジュールがあります。

1. S タイプレコードの読み出し
   a. S タイプレコードフォーマット (.mot) をアブソリュートバイナリフォーマット (.bin) に変換
   b. バイナリフォーマットのデータを 128 バイトのブロックに分割
   c. 空ブロック情報 (空ブロックはデータ 0xFF を 128 バイト分持つ) を確認
2. ブートモードフラッシュ
   a. ブートモードでフラッシュカーネルファイルの読み出し
   b. PC シリアルポートを介して MCU とのブートモードによる接続を確立
   c. ユーザのターゲットプログラムファイルの読み出し
   d. ユーザのターゲットプログラムを MCU のフラッシュメモリへダウンロード
3. ユーザモードフラッシュ
   a. MCU へ書き込みコマンド (文字「U」) を送信
   b. ユーザのターゲットプログラムファイルの読み出し
   c. ユーザのターゲットプログラムを MCU のフラッシュメモリへダウンロード

3. UI (ユーザインタフェース)

UI は，MCU との通信チャネルやデータ転送プロトコルを決めるインタフェースルーチンです。

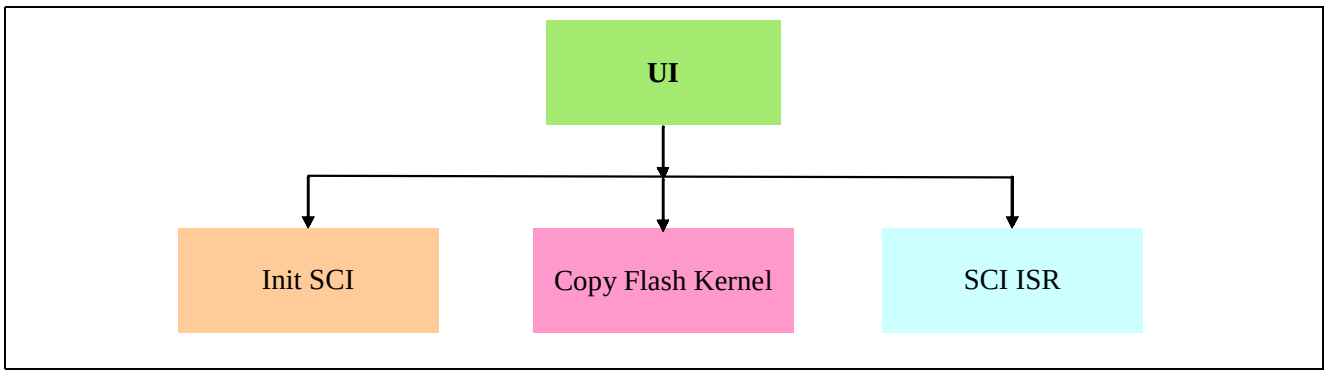

図9　UI の概要

UI の構成

3 つの主なモジュールがあります。

1. Init SCI (SCI の初期化)
   a. 受信割り込みを許可して MCU 内蔵のシリアルコミュニケーションインタフェース (SCI) 部を初期化
   b. SCI のボーレートを 38400bps に設定
2. フラッシュカーネルのコピー
   a. フラッシュカーネルを ROM から RAM へコピー
      注意：フラッシュ書き込み/消去カーネルは RAM 領域で実行すること。
3. SCI ISR (SCI 割り込みサービスルーチン)
   a. SCI 割り込み要求に対する割り込みサービスルーチン
   b. PC から書き込みコマンド (文字「U」) を受信する
   c. フラッシュカーネルを ROM から RAM へコピーする
   d. 開始アドレスと PC からの 128 バイトのブロックデータを PC から受信する
   e. 消去ブロック開始アドレスを検出した場合，フラッシュ消去ルーチンの呼び出す
   f. フラッシュ書き込みルーチンの呼び出し (動作完了時，文字「a」を返し，動作失敗時，文字「n」を返す)
   g. フラッシュアドレスの終了 (0x8000) を検出するまでステップ d. を繰り返す
   h. データ有効フラグが有効であることを確認後，プログラムリセット開始ポイント [PowerON_Reset()] へ飛ぶ

4. カーネル

カーネルは，H8/38024F マイコン用フラッシュメモリ書き込みルーチンです。

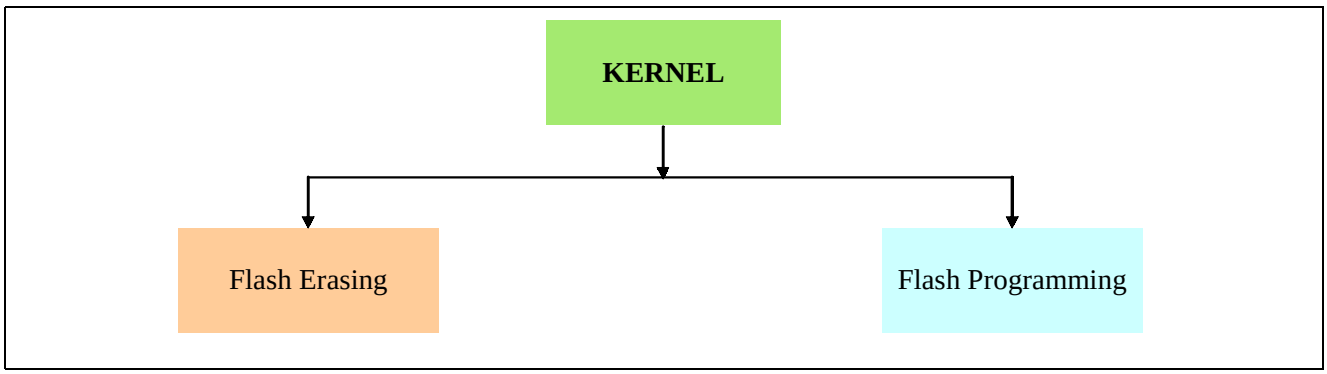

図10　カーネルの概要

カーネルの構成

1. フラッシュ消去
   a. フラッシュ消去はブロック単位で行われます (例：消去ブロック 1, 2, 3, と 4)
   b. フラッシュメモリの消去手順を以下に示します。
      - フラッシュブロックを消去します。
      - メモリがイレースベリファイモードになります。
      - フラッシュの内容が読み出されます。
      - 読み出された内容をすべて 1 の消去値と比較します。
   c. ブロックのビットが 1 以外の場合は，ブロックを再消去します。この処理を消去が完了するまで，または，消去回数の上限に達するまで繰り返します。.
2. フラッシュ書き込み
   a. フラッシュ書き込みの前にフラッシュの消去が必要です。
   b. フラッシュメモリの書き込みは 128 バイト単位で行われ，128 バイトの境界から開始します。
   (例： 0x0000, 0x0080, 0x0100,… , 0x7F00, 0x7F80)
   c. ファイル kernel.c の関数 prog_flash_line_128 を呼び出すことにより，128 バイトブロックでフラッシュを書き込むことができます。
   d. この関数に最初に渡されるパラメータは，128 バイト境界にあるフラッシュメモリ書き込み開始アドレスです。
   e. 2 番目のパラメータは書き込むデータのポインタです。

5. アプリケーション

アプリケーションモジュールとはユーザのターゲットアプリケーションのことです。このアプリケーションノートには, (SLP 38024F CPU ボード上の) H8/38024F MCU のポート 9 に接続された 2 つの LED を制御するいくつかの簡易アプリケーションプログラムが記載されます。

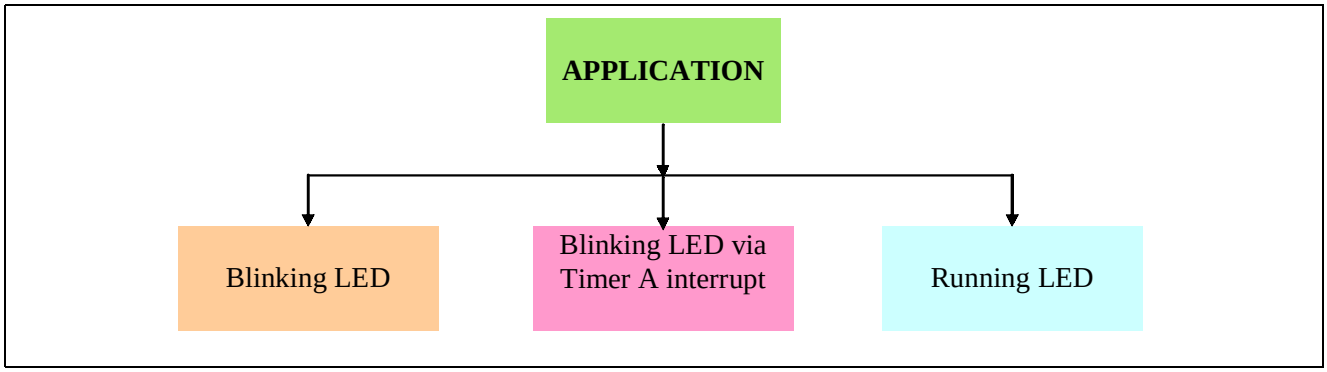

図11 アプリケーションの概要

アプリケーションの構成

LED ドライバなしで直接 LED を駆動できる大電流ポートである H8/38024F のポート 9 を使用しています。

1. LED の点滅
   a. 2 つの LED がポート 9 の 2 ピンと 3 ピンに接続されています。
   b. MCU が実行中に, アプリケーションプログラムはポート 9 の 2 ピンと 3 ピンを一定の遅延でトグルします。
2. タイマ A 割り込みを使った LED の点滅
   a. 2 つの LED がポート 9 の 2 ピンと 3 ピンに接続されています。
   b. タイマ A オーバーフロー割り込みによりポート 9 の 2 ピンと 3 ピンがトグルされます。
3. LED の点灯
   a. 2 つの LED がポート 9 の 2 ピンと 3 ピンに接続されています。
   b. MCU が実行中に, アプリケーションプログラムはポート 9 の 2 ピンと 3 ピンを一定の遅延で交互にトグルします。

6. 通信プロトコル

下図に，ユーザモードでの GUI (PC) と UI (MCU) 間の通信プロトコルを示します。ブートモードの書き込みの詳細は，ハードウェアマニュアルを参照してください。

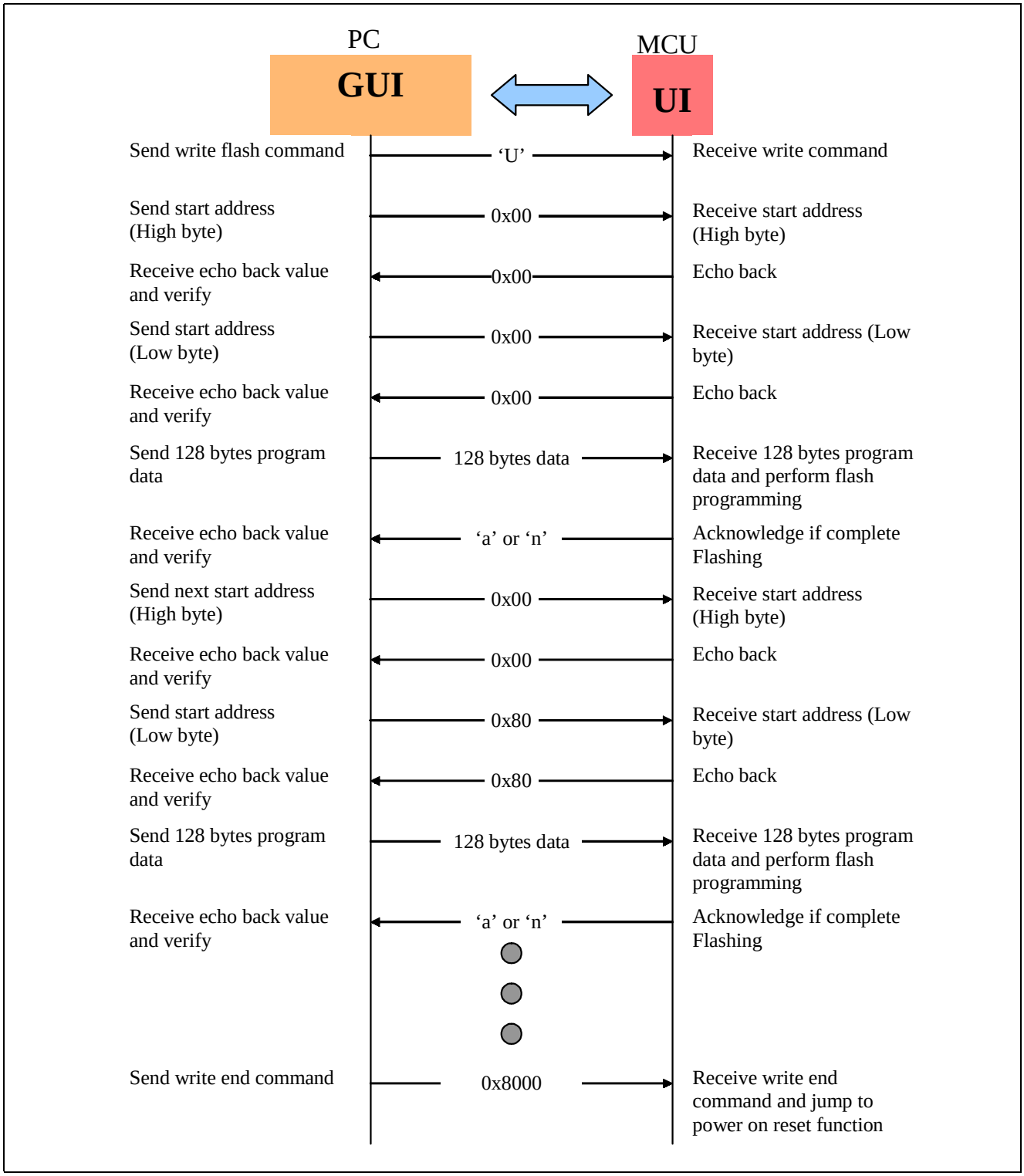

図12　通信プロトコルの遷移図

7. MCU コーディングの適用

ユーザモードの実行と書き込みの期間中，MCU 内にプログラムが存在しなければなりません。このプログラムは 3 つの主要な部分で構成されます。

1. フラッシュカーネル (KERNEL)
2. ホストインタフェースプログラム (UI)
3. ユーザアプリケーションプログラム (APPLICATION)

プログラミング性を維持するために，いかなるフラッシュ手順の後にも，フラッシュカーネルとホストインタフェースプログラムは，MCU 内に存在し続けなければなりません。フラッシュ手順の主な目的は，

1. 新しいデータの更新，または
2. ユーザアプリケーションプログラムの新バージョンへの更新です。

2 種類の動作が可能です。

1. 全ブロック
   → MCU フラッシュがすべて消去され，まったく新しいアプリケーションコードがカーネルとホストインタフェースプログラムと共に書き込まれます。しかし，これはブートモードの書き込みと同じであるため，一般的な書き込み手順ではありません。
2. 部分的ブロック
   → MCU フラッシュの一部が消去され，新しいコードまたはデータが更新されます。
   → 新規データの生成は簡単ですが，ユーザは新規コードの生成には特別な注意を払う必要があります。

部分的ブロックのユーザ書き込みの詳細を以下に示します。

1. データの更新
2. コードの更新
   → 方法 1
   → 方法 2

7.1 データの更新

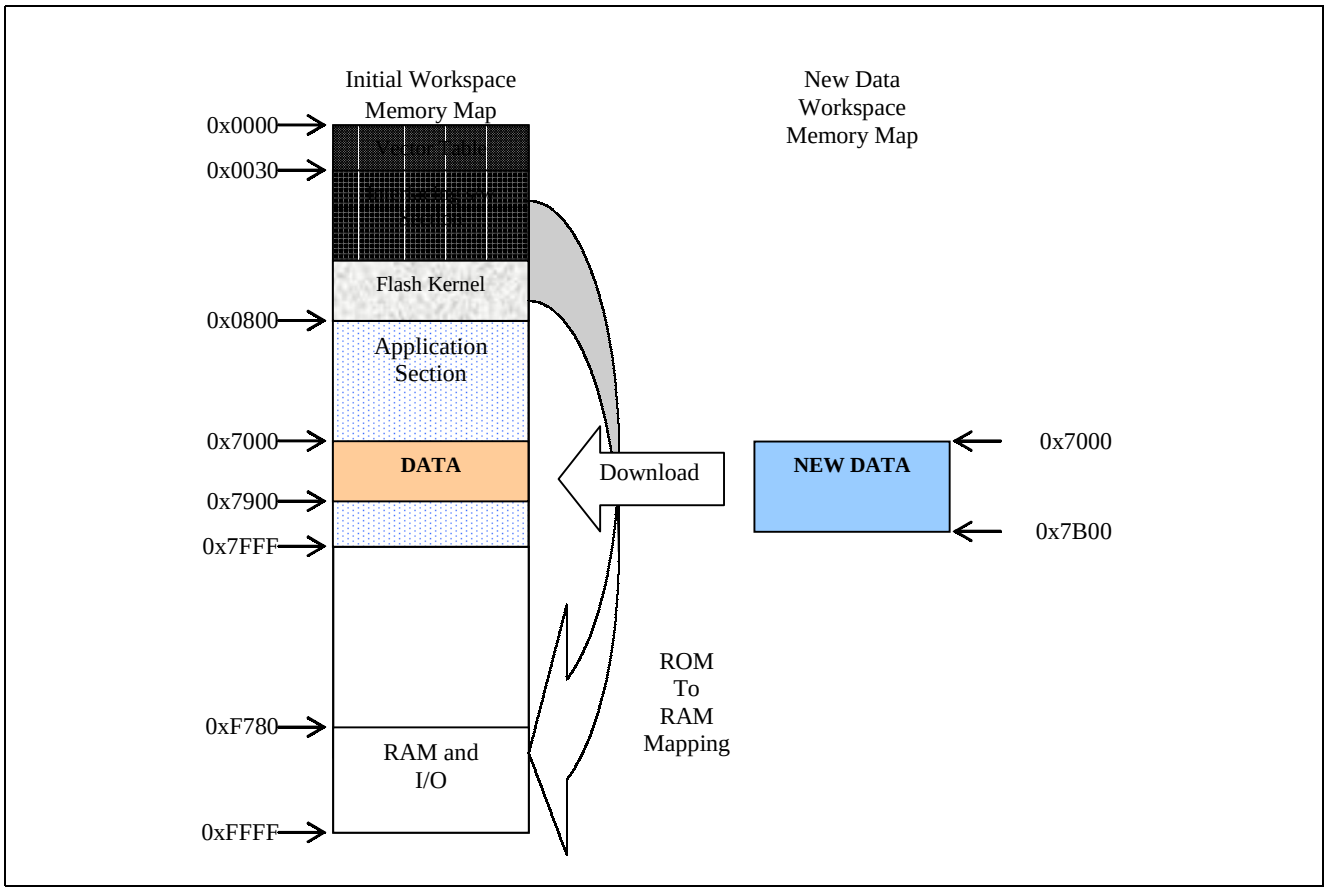

図13　データ更新のメモリマップ

データ更新の手順

1. C またはアセンブリプログラムのための空白のワークスペースを作成する。
2. データを宣言 (静的定数…　またはデータ …)する。
3. セクションの宣言とアドレスの定義をする。
4. コンパイルして S レコードファイルを生成する。
5. または，統合化環境ツール HEW の Save as コマンドで S レコードファイルを生成する。

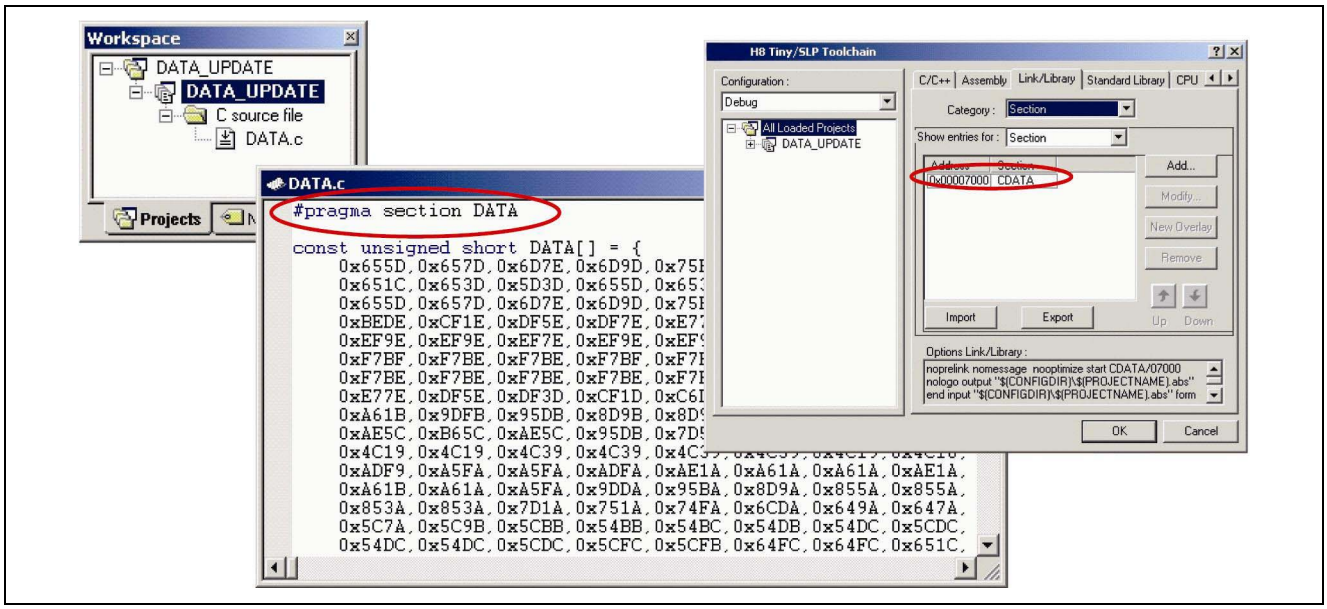

図14　新規データワークスペースの生成

7.2 コードの更新

初期ワークスペースと混在させるためには，新規ワークスペースで生成したコードはいくつか考慮すべき点があります。

1. 初期化変数
2. スタック
3. 定数
4. 新規ワークスペースへのエントリポイント
5. 初期ワークスペースへのエントリポイント

7.2.1 方法 1 [M1]

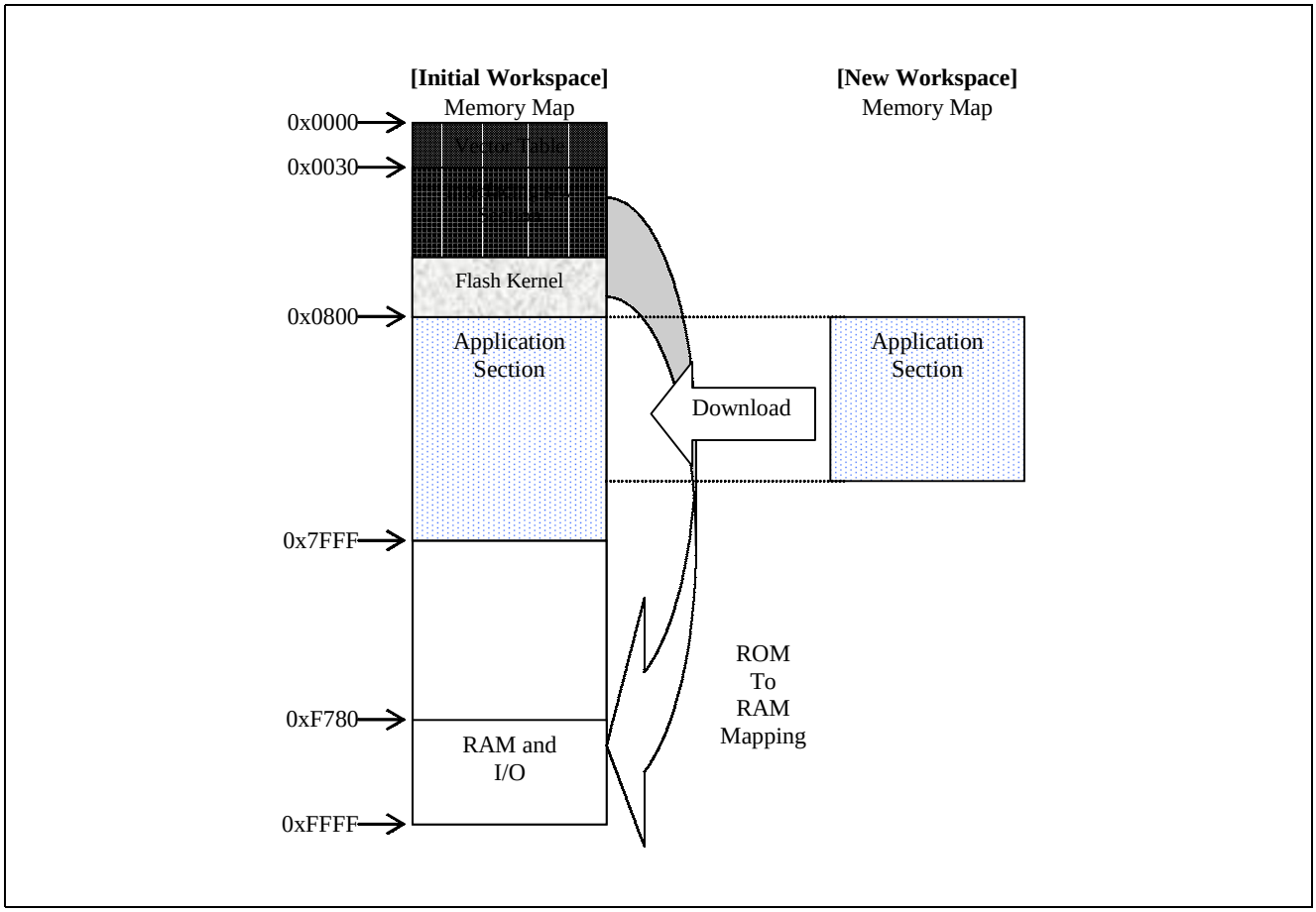

図15 コード更新方法 1 のメモリマップ

[初期ワークスペース] と [新規ワークスペース] の処理の手順

1. 電源投入シーケンス
2. メイン関数に入る
3. SCI を初期化する
4. アプリケーションプログラムに飛ぶ

【注】 この方法は既存のワークスペースへの変更が少ないときにのみ使用してください (例：プッシュボタンや新規アルゴリズムの操作への新規機能の追加，他の値の追加など)。初期ワークスペースの定数，変数，割り込みベクタテーブルに変更がないことが重要です。変更が必要な場合は，方法 2 を使用してください。

フラッシュ手順

1. PC GUI で Download コマンドを起動する。
2. SCI 割り込みを有効にする。
3. MCU インタフェースルーチン (UI) から SCI 割り込みサービスルーチンに飛び，以下の処理を行う。
   a. フラッシュカーネルを ROM 領域から RAM 領域へコピー。
   b. フラッシュカーネル用のデータを取得。
4. フラッシュカーネルによりフラッシュメモリへ書き込み。
5. 完了後,
   a. UI はパワーオンリセット機能に飛び，全ワークスペースを初期化する。または,
   b. ユーザはハードウェアリセット信号をアサートして新規アプリケーションを実行する。または,
   c. ユーザはウォッチドッグタイマを使用して内部リセットを発生させ，全 I/O ポートをハイインピーダンス状態に初期化する。

方法 1 [初期ワークスペース] を生成するための手順

1. SLP ツールチェーンに基づき，新規ワークスペース (アプリケーション) を作成する。
2. コードを書く (このコードにセクション名を付ける) 。
3. HEW の [Option/ Toolchain/ Linker/ Section] にセクションアドレスを宣言する。
4. コンパイルして S レコードファイルを生成する。

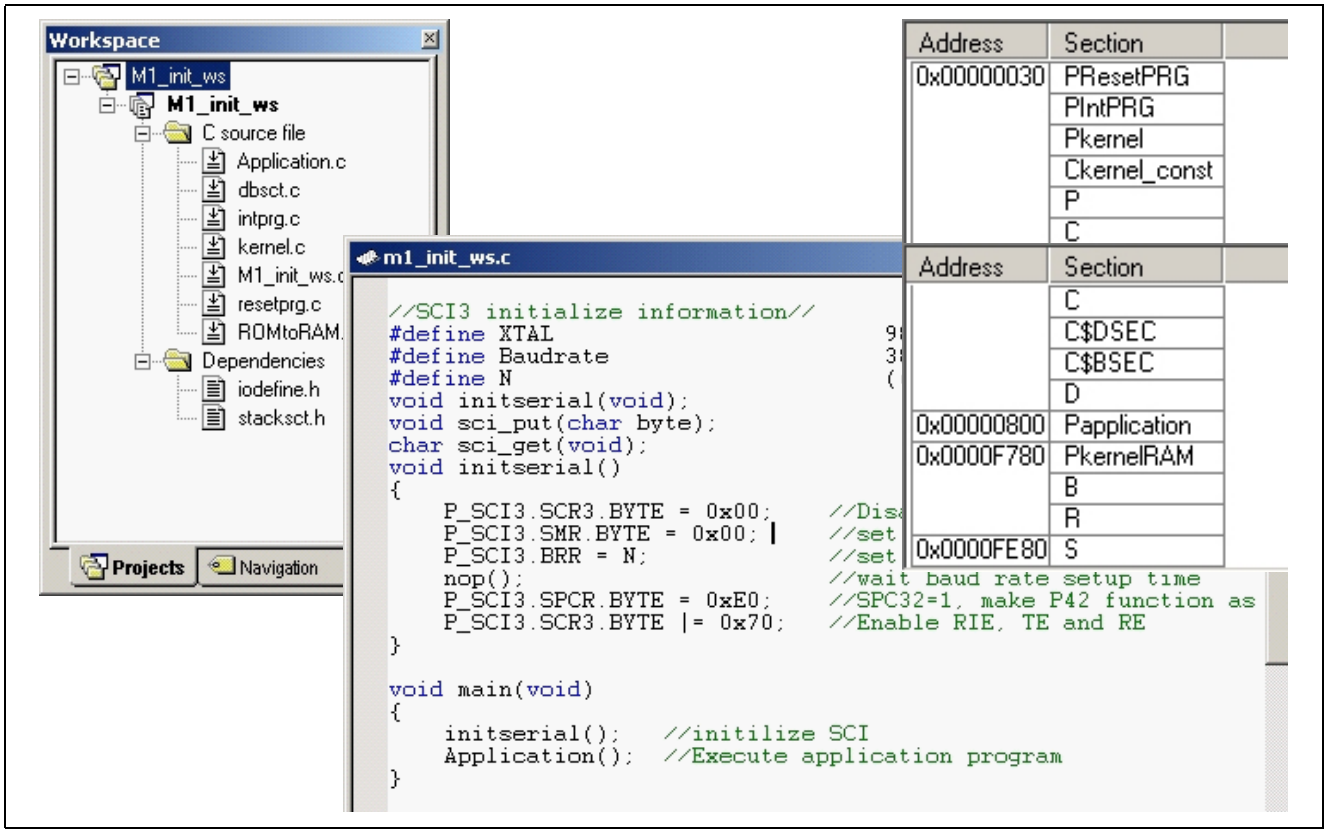

図16　方法 1 [初期ワークスペース] の生成

方法 1 [新規ワークスペース]を生成するための手順

SLP をもとに新規ワークスペース (空のアプリケーション) を作成する。

1. コードを書く (そのコードにセクション名を付ける) 。
2. HEW の [Option/ Toolchain/ Linker/ Section] にセクションアドレスを宣言する。
3. 初期ワークスペースのフォルダからファイル iodefine.h を新規ワークスペースのフォルダにコピーする。
4. [\Method 1\M1_init_ws\M1_init_ws\iodefine.h] を [\Method 1\M1_new_ws\M1_new_ws] にコピーする。
5. コンパイルして S レコードファイルを生成する。

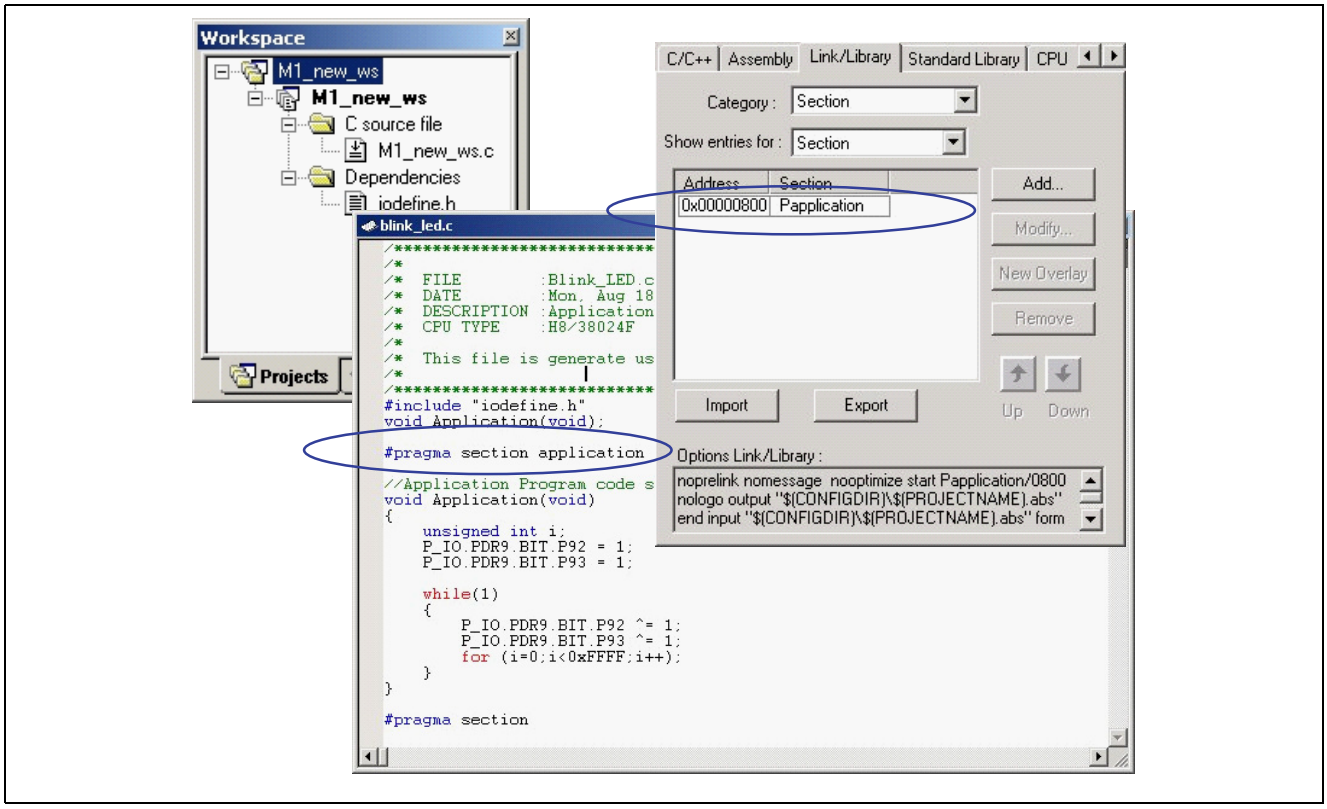

図17　方法 1 [新規ワークスペース] の生成

注意事項：

新規アプリケーションには以下の制限があります。

1. 割り込みが制御できない。
2. ユーザが初期データをコピーしなければならない。

7.2.2 方法 2 [M2]

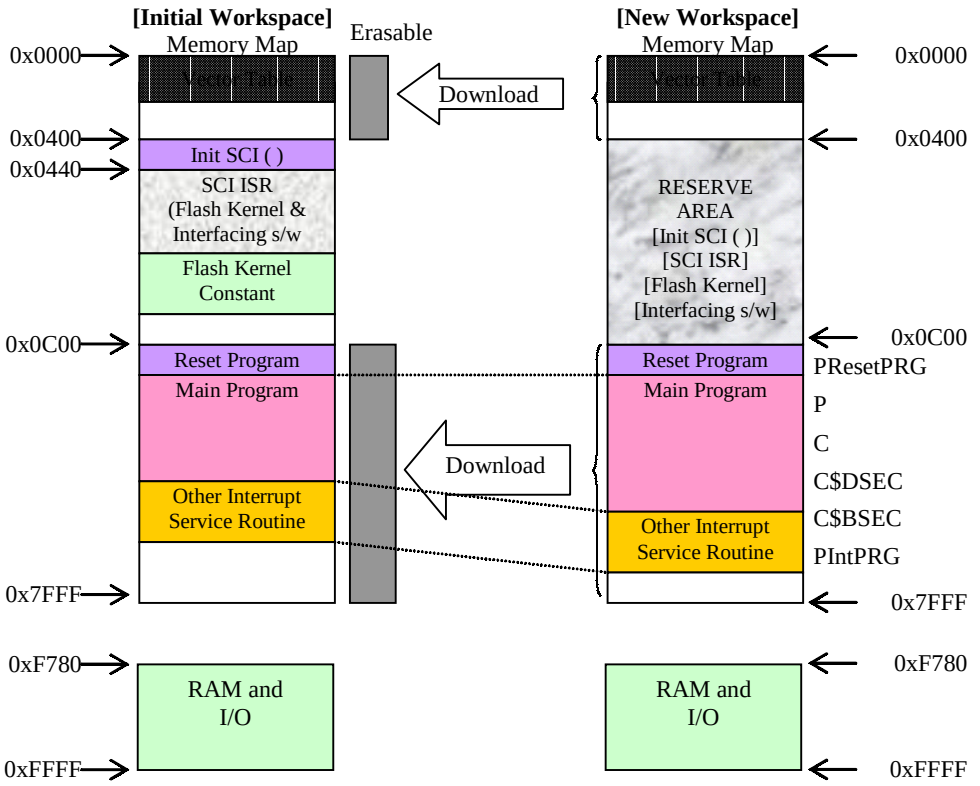

図18 コード更新方法 2 のメモリマップ

フラッシュ手順

1. PC GUI で Download コマンドを起動する。
2. SCI 割り込みを有効にする。
3. MCU インタフェースルーチン (UI) から SCI 割り込みサービスルーチンに飛び，以下の処理を行う。
   a. フラッシュカーネルを ROM 領域から RAM 領域へコピーする。
   b. フラッシュカーネル用のデータを取得する。
4. フラッシュカーネルによりフラッシュメモリへ書き込む。
5. 完了後,
   a. UI はパワーオンリセット機能に飛び，全ワークスペースを初期化する。または,
   b. ユーザはハードウェアリセット信号をアサートして新規アプリケーションを実行する。または,
   c. ユーザはウォッチドッグタイマを使用して内部リセットを発生させ，全 I/O ポートをハイインピーダンス状態に初期化する。

方法 2 [初期ワークスペース] を生成するための手順

SLP をもとに新規ワークスペース (アプリケーション) を作成する。

1. コードを書く (このコードにセクション名を付ける) 。
2. HEW の [Option/ Toolchain/ Linker/ Section] にセクションアドレスを宣言する。
3. コンパイルして S レコードファイルを生成する。
4. コンパイラのを設定する：
   a. 最適化 = 速度重視の最適化 (SCI ISR の register save ライブラリオプションを削除するため)
   b. [新規ワークスペース] による上書きを避けるため，カーネル定数セクションを追加する。

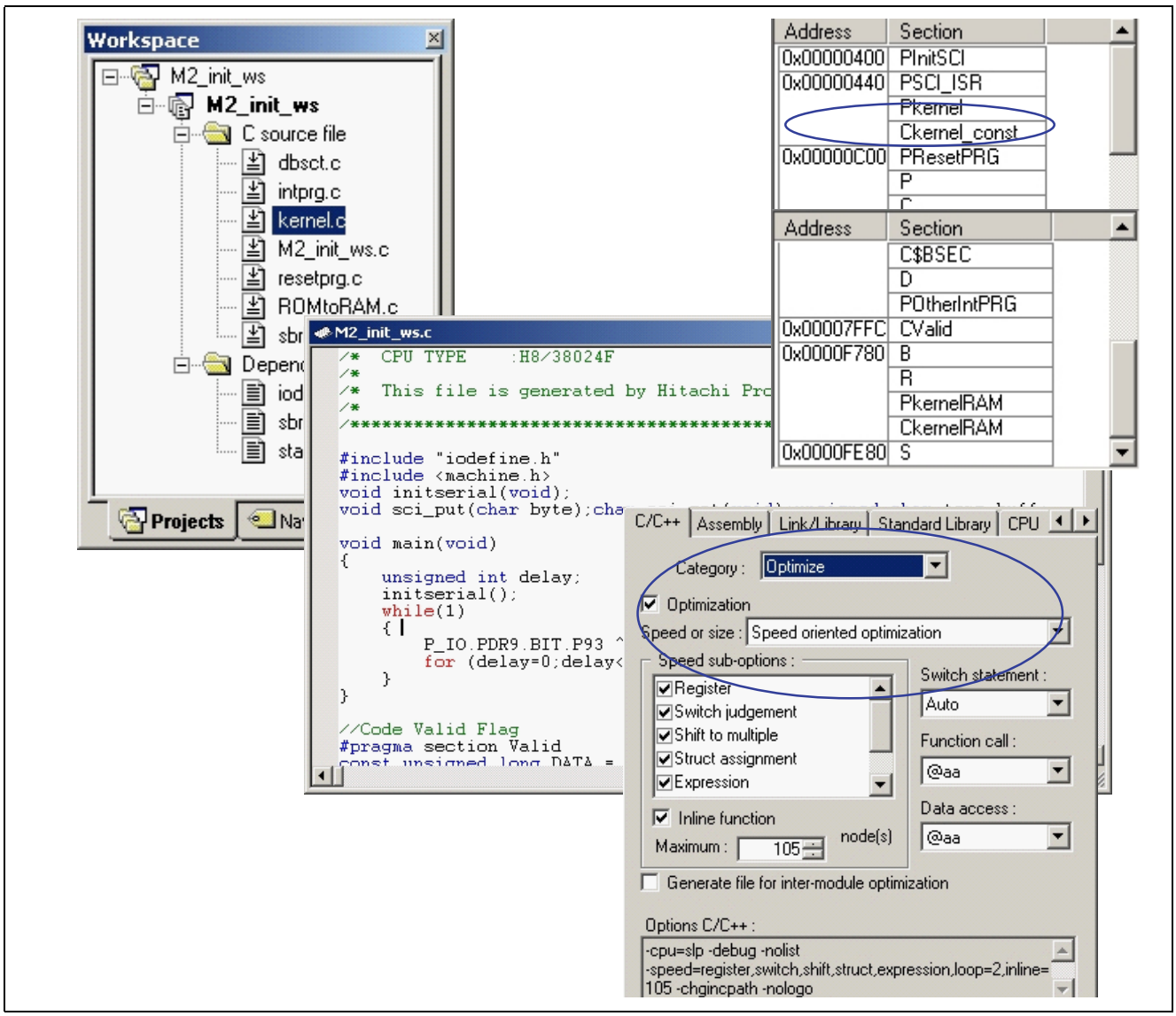

図19 方法 2 [初期ワークスペース] の生成

方法 2 [新規ワークスペース] の重要な注意事項

1. H'0400 to H'0C00 (フラッシュブロック 1 と 2) をリザーブする。
   → 最初のアプリケーションフラッシュカーネルやインタフェースソフトウェアへの上書きを避けるため
2. H'0C00 (フラッシュブロック 3) にリセットルーチンを固定する。
3. リセットルーチンの後にメインおよび他の ISR を固定する。
4. Init SCI ( ) へのアクセスは，H'0400 への関数呼び出しで可能にする。
5. H'0440 に SCI 割り込みサービスルーチンを固定する。
   → 割り込みハンドラ (intprg.c) を使って実現する
   例：

```
#pragma section SCI_ISR
      static const unsigned short DATA = 0x0440;
#pragma section
```

方法 2 [新規ワークスペース] を生成するための手順

1. SLP をもとに新規ワークスペース (アプリケーション) を作成する。
2. 新規ワークスペースコードを書く。
3. HEW の [Option/ Toolchain/ Linker/ Section] にセクションアドレスを宣言する。
   詳細は下図参照のこと。
4. コンパイルして S レコードファイルを生成する。

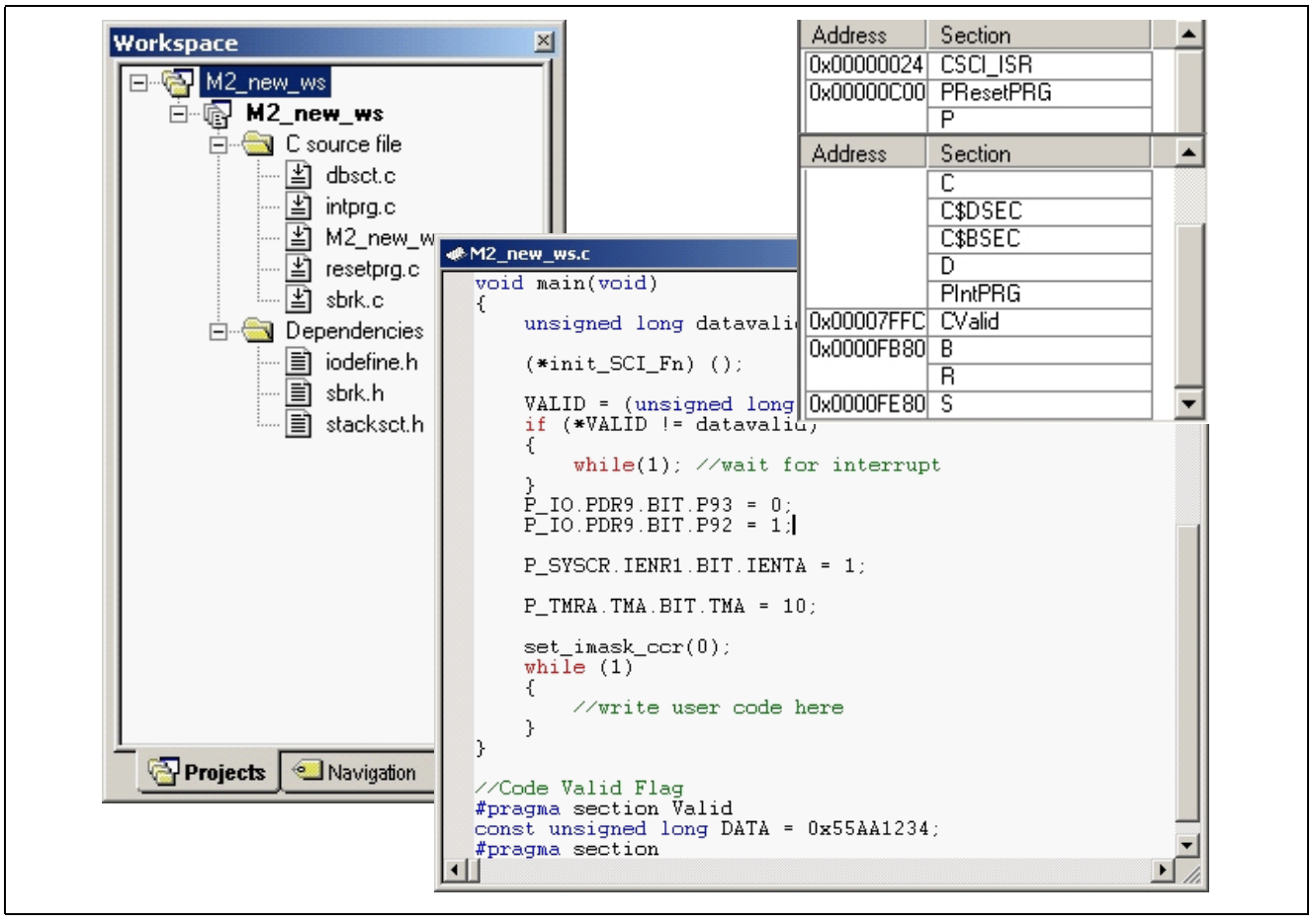

図20　方法 2 [新規ワークスペース] の生成

8. 全体の動作と考察

この章では，アプリケーションノートに必要なセットアップとフラッシュ GUI の動作を紹介します。

8.1 環境設定

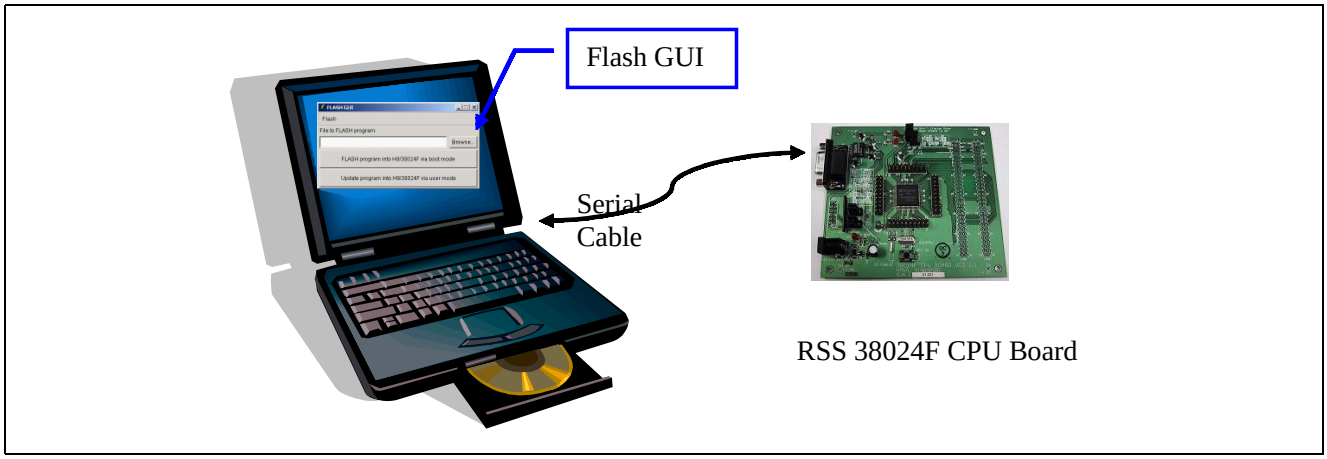

図21　ユーザモードによる（再）書き込み用の環境設定

RSS 38024F CPU ボードがない場合の簡単な接続の図を示します。

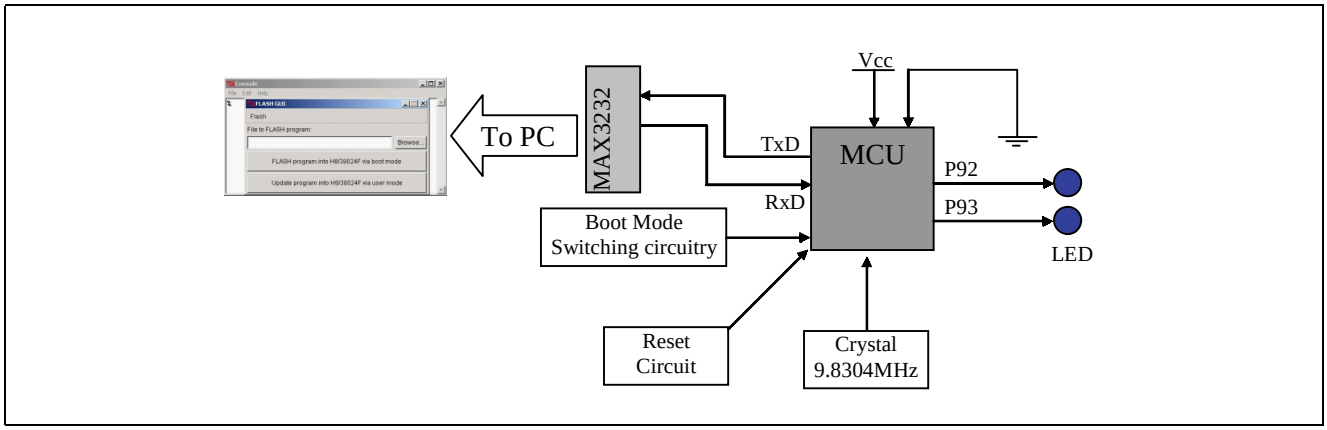

図22　ユーザモードによる書き込み用デモボードのブロック図

8.2 GUI を使用した書き込み

8.2.1 方法 1 の紹介

ブートモードの書き込み

1. Flash_GUI.tcl を開く。
2. ダウンロードファイル *M1_init_ws.mot* を選択する。
3. H8/38024F MCU をブートモードに切り換え* リセットボタンを押下する。
4. フラッシュ GUI のブートモードボタン Flash program into H8/38024F via boot mode をクリックするとダウンロードが開始される。
5. ブートモード書き込みが完了すると，“Program downloaded!”のメッセージが表示される。.
6. H8/38024F MCU をユーザモードに切り換え* リセットボタンを押下する。
7. ポート 9 に接続された 2 つの LED が連続して点滅し, M1_init_ws プログラムが動作中であることを示す。

ユーザモードの書き込み

1. 入力 S レコードファイルとして *M1_new_ws.mot*” を選択する。
2. Update program into H8/38024F via user mode をクリックして **[新規ワークスペース]** をダウンロードする。
3. “Program downloaded!”のメッセージボックスが表示される。
4. 2 つの LED，D3，D4 が交互に点灯して，新規アプリケーションプログラムが動作中であることを示す。

ユーザモードの再書き込み

1. 入力 S レコードファイルとして *M1_App1.mot* を選択する。
2. Update program into H8/38024F via user mode をクリックして新規アプリケーションプログラムをダウンロードする。
3. “Program downloaded!”のメッセージボックスが表示される。
4. 2 つの LED，D3，D4 が点灯して，新規アプリケーションプログラムが動作中であることを示す。

ユーザモードでは，ユーザは MCU をリセットせずに異なるアプリケーションプログラムをダウンロードすることができます。

注意： *ブートモードとユーザモードとの切り換え用ジャンパ設定は，38024F CPU Board Quick Start Guide を参照してください。

8.2.2 方法 2 の紹介

方法 2 は，8.2.1 章と同様にアクセス可能です。ファイル名は異なります。

例：

“M1_init_ws.mot”	➔	*“M2_init_ws.mot”*
“M1_new_ws.mot”	➔	*“M2_new_ws.mot”*
“M1_APP1.mot”	➔	*“M2_APP1.mot”*

実行結果は同じです。

9. プログラムリスト

H8/38024F SLP MCU 用の HEW プロジェクトジェネレータを使用して作成したプログラムリストを添付します。無償の SLP/Tiny ツールチェーンを使用しています。

9.1 方法 1 [初期ワークスペース] プログラムリスト

9.1.1 方法 1 [初期ワークスペース] メインルーチン

下図に，m1_init_ws.c のフローチャートを示します。続いて，プログラムリストを示します。

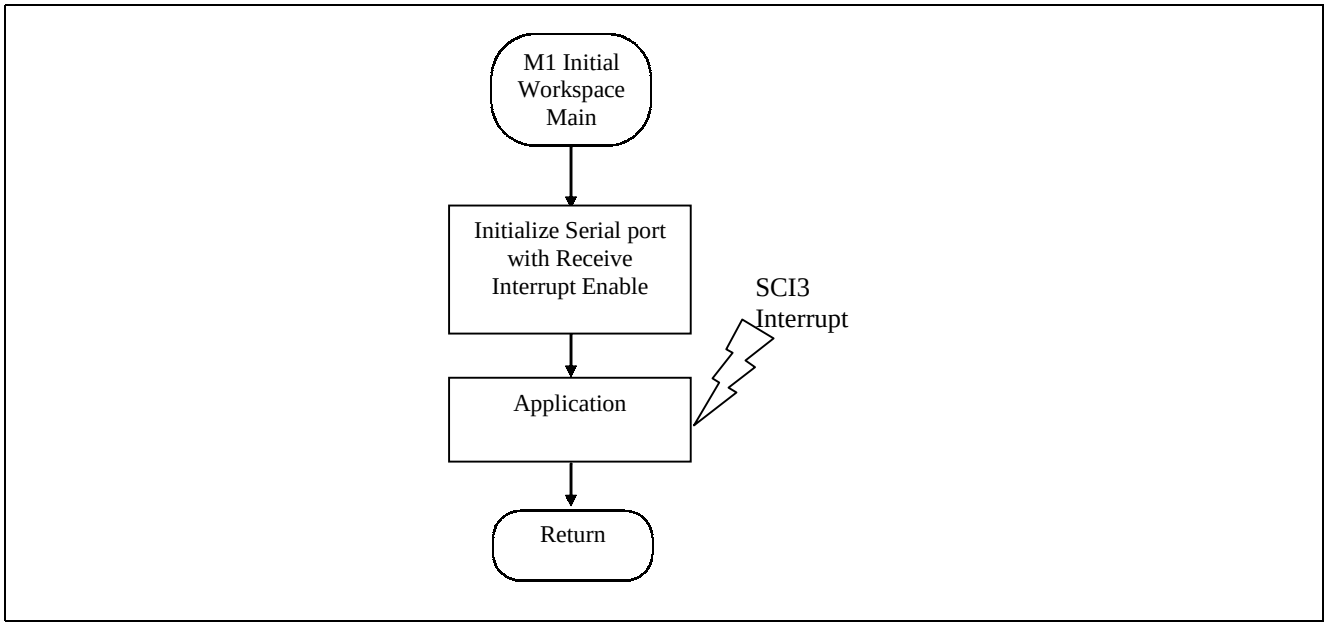

図23 方法 1 [初期ワークスペース] メインルーチンのフローチャート

```
/*********************************************************************/
/*                                                                   */
/*  FILE       :M1_init_ws.c                                         */
/*  DATE       :Mon, Sep 29, 2003                                    */
/*  DESCRIPTION :Main Program                                        */
/*  CPU TYPE   :H8/38024F                                            */
/*                                                                   */
/*  This file is generated by Renesas Project Generator (Ver.2.1).   */
/*                                                                   */
/*********************************************************************/
#include "iodefine.h"
#include <machine.h>

//Flash function prototype
extern void copyfunc(void);
extern unsigned char prog_flash_line_128 (unsigned long t_address, union
char_rd_datum_union *p_data);
extern unsigned char erase_block (unsigned char block_num);
extern int  *_PkernelBegin, *_PkernelEnd, *_Pkernel_RAMBegin;
extern int  *_CkernelBegin, *_CkernelEnd, *_Ckernel_RAMBegin;

//function prototype
void copyfunc(void);
extern void Application(void);

//SCI3 initialize information//
#define XTAL                    9830400L
#define Baudrate                38400L
#define N                       ((XTAL) / (64L*1L*Baudrate)) - 1L
void initserial(void);
void sci_put(char byte);
char sci_get(void);
void initserial()
{
   P_SCI3.SCR3.BYTE = 0x00;  //Disable TIE,TE,RE,MPIE,TEIE,RIE,
   P_SCI3.SMR.BYTE = 0x00;   //set Async, 8 data, none parity, 1 stop, clk n=0
   P_SCI3.BRR = N;           //set baud rate = N
   nop();                    //wait baud rate setup time
   P_SCI3.SPCR.BYTE = 0xE0;  //SPC32=1, make P42 function as TXD32
   P_SCI3.SCR3.BYTE |= 0x70; //Enable RIE, TE and RE
}

void main(void)
{
   initserial();             //initilize SCI
   Application();            //Execute application program
}
```

```
void copyfunc(void)
{
   register int *p, *q;
   for (p=_PkernelBegin, q=_Pkernel_RAMBegin;p<_PkernelEnd;p++,q++)
   {
      *q=*p;
   }

   for (p=_CkernelBegin, q=_Ckernel_RAMBegin;p<_CkernelEnd;p++,q++)
   {
      *q=*p;
   }
}

void sci_put(char byte)
{
   while(P_SCI3.SSR.BIT.TDRE==0);
   P_SCI3.TDR=byte;
   while(P_SCI3.SSR.BIT.TEND==0);
}

char sci_get(void)
{
   while(P_SCI3.SSR.BIT.RDRF==0){}       //Wait until RDRF = 1
   if ((P_SCI3.SSR.BYTE & 0x38) ==0)     //Check for SCI error
   {
   return P_SCI3.RDR;
   }
   else return 0xFF;               //If error occur return 0xFF
   if(P_SCI3.SSR.BIT.RDRF==1) P_SCI3.SSR.BIT.RDRF=0;
}

#pragma section
```

9.1.2 方法 1 [初期ワークスペース] アプリケーションルーチン

下図に Application.c のフローチャートを示します。続いて，プログラムリストを示します。

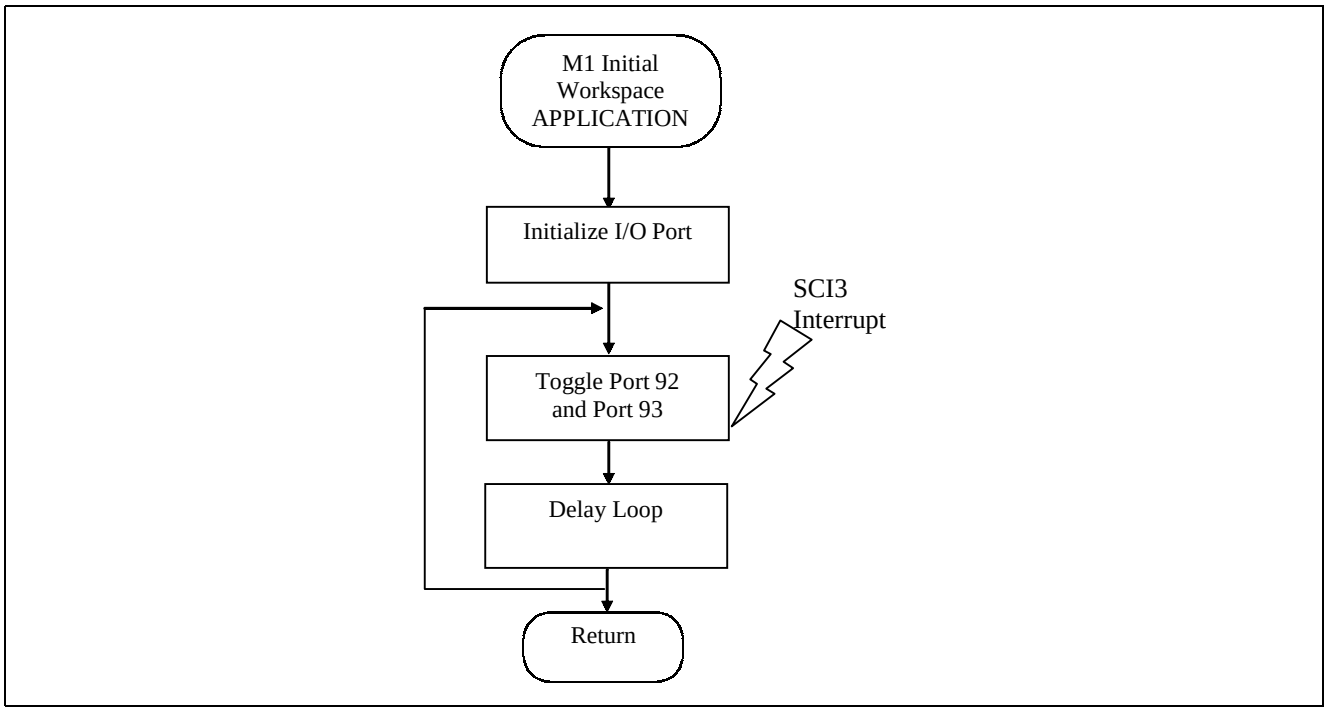

図24 方法 1 [初期ワークスペース] アプリケーションルーチンのフローチャート

```
#include "iodefine.h"
//Section define for application program
#pragma section application
void Application(void);
//Application Program code start
//Blinking LED application
void Application(void)
{
   unsigned int i;
   P_IO.PDR9.BIT.P92 = 1;
   P_IO.PDR9.BIT.P93 = 1;

   while(1)
   {
      P_IO.PDR9.BIT.P92 ^= 1;
      P_IO.PDR9.BIT.P93 ^= 1;
      for (i=0;i<0xFFFF;i++);
   }
}
#pragma section
```

9.1.3 方法 1 [初期ワークスペース] 割り込みサービスルーチン

下図に SCI 割り込みサービスルーチンのフローチャートを示します。続いて，プログラムリストを示します。

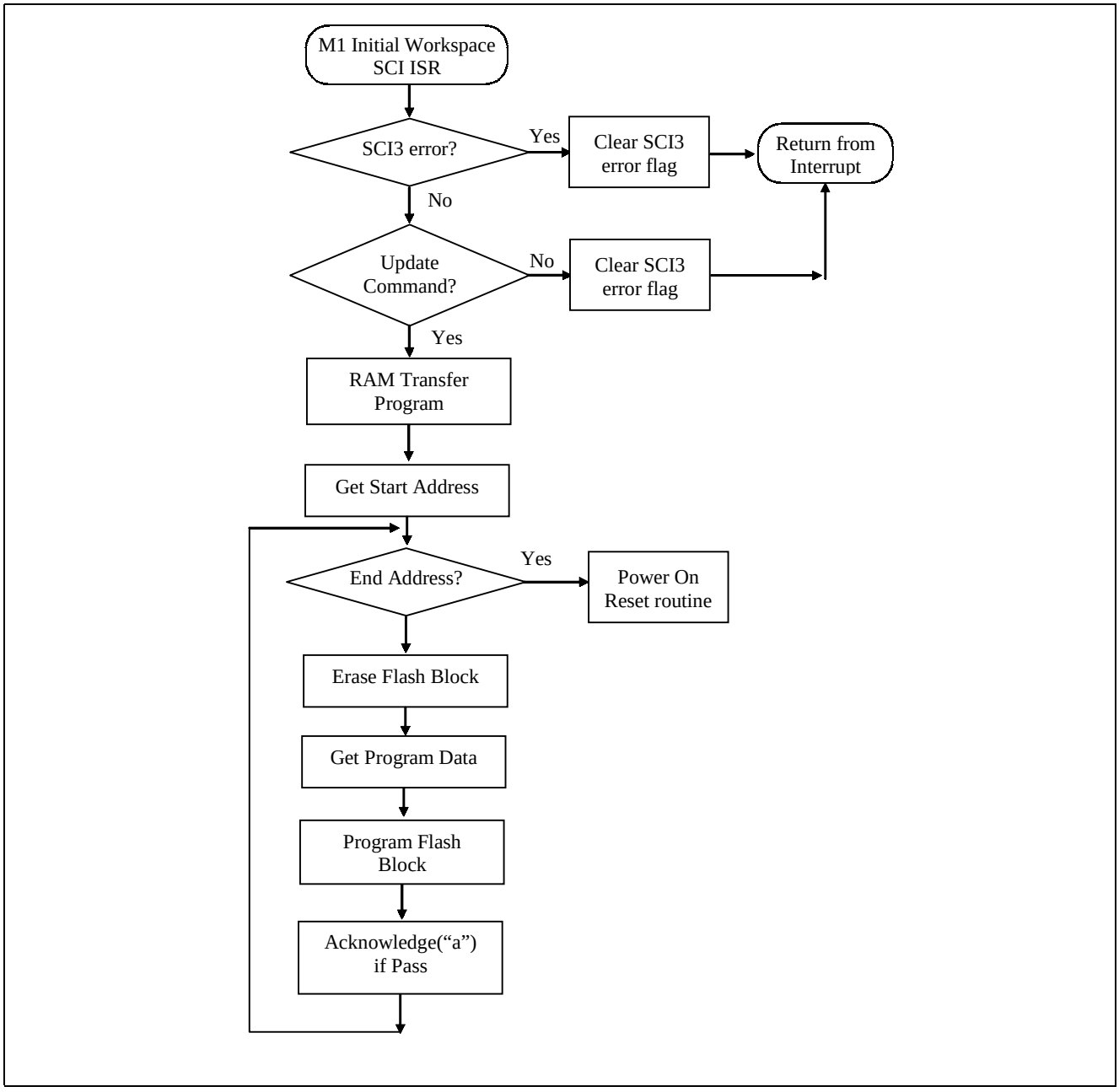

図25　方法 1 [初期ワークスペース] 割り込みサービスルーチンのフローチャート

```
/************************************************************************/
/*                                                                      */
/*  FILE       :intprg.c                                                */
/*  DATE       :Mon, Sep 29, 2003                                       */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE   :H8/38024F                                               */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.2.1).     */
/*                                                                      */
/************************************************************************/
#include "iodefine.h"
#include <machine.h>

//SCI function prototype
extern void sci_put(char byte);
extern char sci_get(void);
extern unsigned char temp_buff;

//Flash function prototype
extern unsigned char prog_flash_line_128 (unsigned long t_address, union
char_rd_datum_union *p_data);
extern unsigned char erase_block (unsigned char block_num);
extern void PowerON_Reset(void);

#pragma section IntPRG
// vector 1 Reserved
.
.
__interrupt(vect=16) void INT_TimerG(void) {/* sleep(); */}
// vector 17 Reserved

// vector 18 SCI3
__interrupt(vect=18) void INT_SCI3(void)
{
   unsigned short start_address;
   unsigned char prog_data_addr[128],count1;
   unsigned char temp_buff;
   if ((P_SCI3.SSR.BYTE & 0x38) == 0)    //Check for SCI error
   {
      if(P_SCI3.RDR=='U')
      {
            copyfunc();
            while(1)
            {
               //GET START ADDRESS
               temp_buff = sci_get();
               start_address = (unsigned short) (temp_buff <<8); //high byte
               sci_put(temp_buff);

               temp_buff = sci_get();
               start_address = start_address | (unsigned short) (temp_buff);
                                                                  //low byte
               sci_put(temp_buff);

               if    (start_address == 0x0000)      {erase_block (0);}
               else if (start_address == 0x0400)    {erase_block (1);}
               else if (start_address == 0x0800)    {erase_block (2);}
```

```
                else if (start_address == 0x0c00)     {erase_block (3);}
                else if (start_address == 0x1000)     {erase_block (4);}
                else if (start_address == 0x8000)     {PowerON_Reset();}
                                                      //end of flash programming
                else  nop(); //invalid start address

                for(count1=0;count1<128;count1++)
                {
                   prog_data_addr[count1] =  sci_get();
                }

                if(prog_flash_line_128 (start_address, (union
                      char_rd_datum_union * ) prog_data_addr)==0x01)
                {
                   sci_put('a');
                }
                else sci_put('n');
             }

       }
       else  return; // if not Update flash command then do nothing
    }
    else
    {
       //SCI error occur
       if (P_SCI3.SSR.BIT.OER == 1)
       temp_buff = P_SCI3.RDR;
       temp_buff = P_SCI3.RDR;
       P_SCI3.SSR.BYTE=0x84;
       sci_put('e');
    }
}
// vector 19 ADI
__interrupt(vect=19) void INT_ADI(void) {/* sleep(); */}
// vector 20 Direct Transition
__interrupt(vect=20) void INT_Direct_Transition(void) {/* sleep(); */}
```

9.1.4 方法 1 [新規ワークスペース] アプリケーションルーチン

下図に m1_new_ws.c のフローチャートを示します。続いて，プログラムリストを示します。

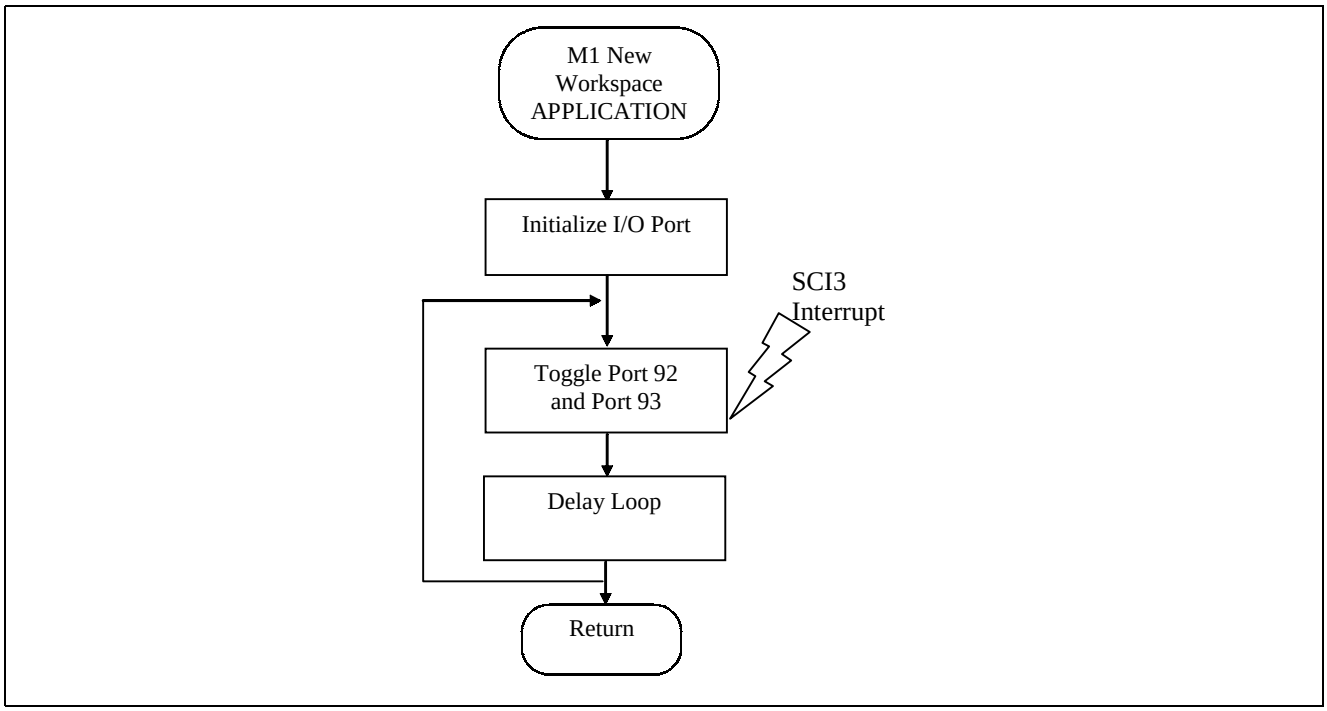

図26 方法 1 [新規ワークスペース] アプリケーションルーチンのフローチャート

```
#include "iodefine.h"
void Application(void);

#pragma section application

//Application Program code start
void Application(void)
{
   unsigned int i;
   P_IO.PDR9.BIT.P92 = 1;
   P_IO.PDR9.BIT.P93 = 0;

   while(1)
   {
      P_IO.PDR9.BIT.P92 ^= 1;
      P_IO.PDR9.BIT.P93 ^= 1;
      for (i=0;i<0xFFFF;i++);
   }
}

#pragma section
```

9.2 方法 2 [初期ワークスペース] プログラムリスト

9.2.1 方法 2 [初期ワークスペース] メインルーチン

下図に m2_init_ws.c のフローチャートを示します。続いて，プログラムリストを示します。

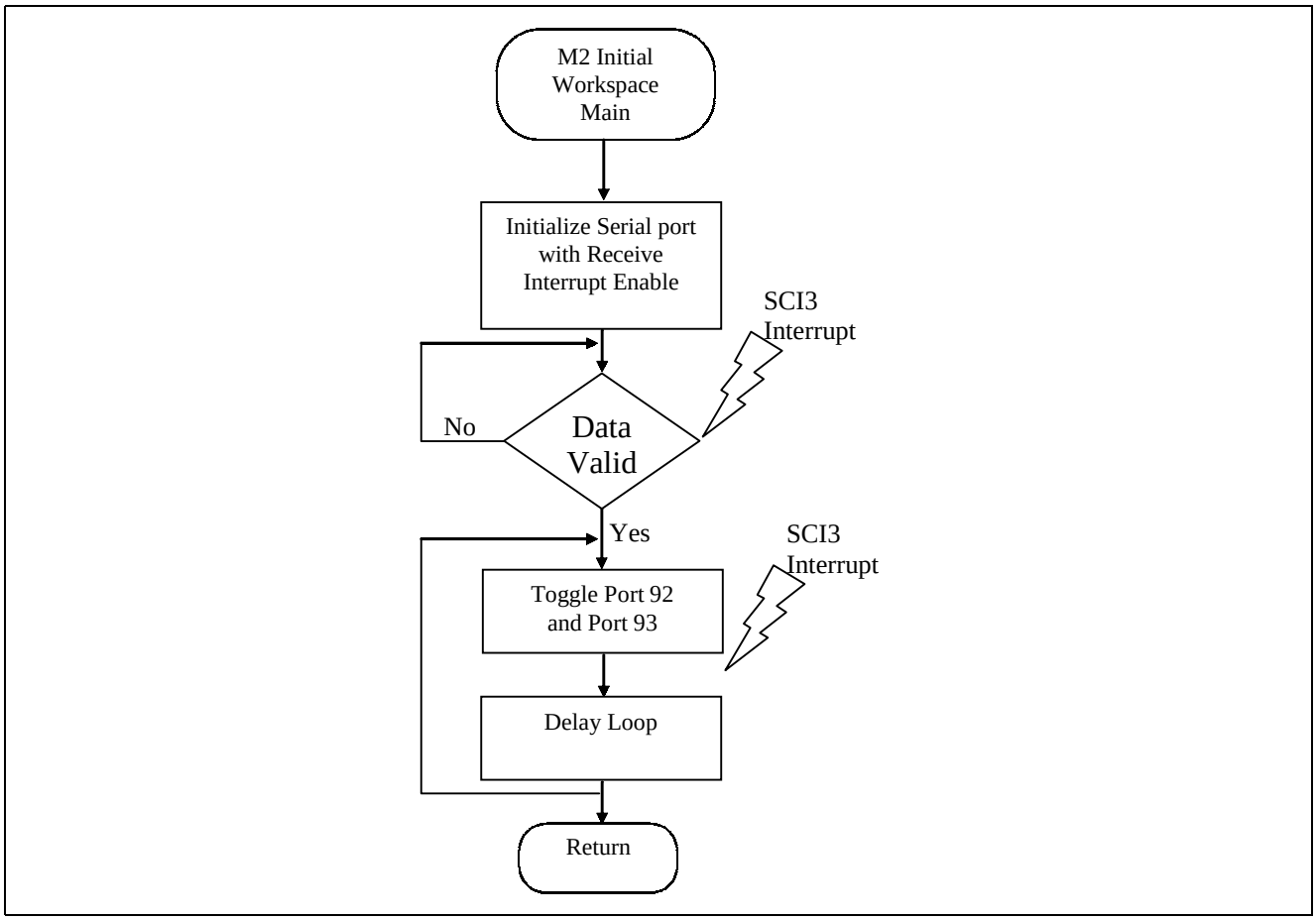

図27 方法 2 [初期ワークスペース] メインルーチンのフローチャート

```
/*********************************************************************/
/*                                                                   */
/*  FILE       :M2_init_ws.c                                         */
/*  DATE       :Mon, Sep 29, 2003                                    */
/*  DESCRIPTION :Main Program                                        */
/*  CPU TYPE   :H8/38024F                                            */
/*                                                                   */
/*  This file is generated by Renesas Project Generator (Ver.2.1).   */
/*                                                                   */
/*********************************************************************/

#include "iodefine.h"
#include <machine.h>
void initserial(void);
void sci_put(char byte);char sci_get(void);unsigned char temp_buff;

void main(void)
{
   unsigned int delay;
   unsigned long datavalid = 0x55AA1234, *VALID;

   initserial();
   VALID = (unsigned long *)0x7FFC;
   if (*VALID != datavalid)
   {
      while(1); //wait for interrupt
   }
   P_IO.PDR9.BIT.P93 = 1;
   P_IO.PDR9.BIT.P92 = 1;

   while(1)
   {
      P_IO.PDR9.BIT.P93 ^= 1;
      P_IO.PDR9.BIT.P92 ^= 1;
      for (delay=0;delay<0xFFFF;delay++);
   }
}

//Code Valid Flag fixed at last address (0x7FFC-0x7FFF)
#pragma section Valid
const unsigned long DATA = 0x55AA1234;
#pragma section
```

```
//Init SCI routine fixed at address 0x0400
#pragma section InitSCI
//SCI3 initialize information
#define XTAL                9830400L
#define Baudrate            38400L
#define N                   ((XTAL) / (64L*1L*Baudrate)) - 1L

//unsigned char *addr, temp;
void initserial()
{
   P_SCI3.SCR3.BYTE = 0x00;  //Disable TIE,TE,RE,MPIE,TEIE,RIE,
   P_SCI3.SMR.BYTE = 0x00;         //set Async, 8 data, none parity, 1 stop, clk
n=0
   P_SCI3.BRR = N;                 //set baud rate = 9600
   nop();                    //wait baud rate setup time
   P_SCI3.SPCR.BYTE = 0xE0;  //SPC32=1, make P42 function as TXD32
   P_SCI3.SCR3.BYTE |= 0x70; //Enable RIE, TE and RE
   set_imask_ccr(0);
}
//SCI3 initialize information end//
#pragma section
```

9.2.2 方法 2 [初期ワークスペース] SCI 割り込みサービスルーチン

下図に m2_init_ws.c のフローチャートを示します。続いて，プログラムリストを示します。

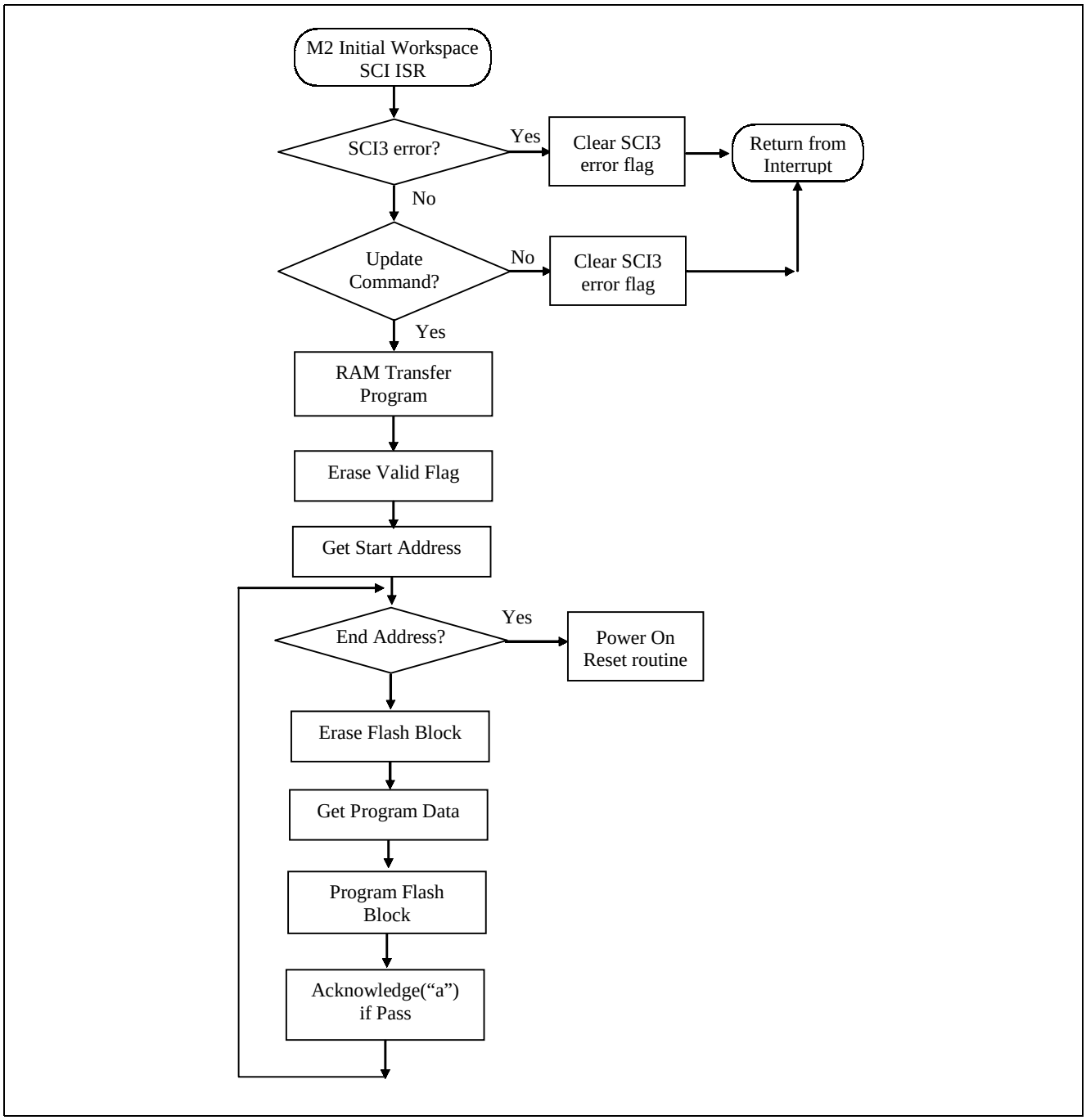

図28 方法 2 [初期ワークスペース] SCI 割り込みサービスルーチンのフローチャート

```
/************************************************************************/
/*                                                                      */
/*  FILE       :intprg.c                                                */
/*  DATE       :Mon, Sep 29, 2003                                       */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE   :H8/38024F                                               */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.2.1).     */
/*                                                                      */
/************************************************************************/
#include "iodefine.h"
#include <machine.h>

//SCI function prototype
void sci_put(char byte);char sci_get(void);
extern unsigned char temp_buff;

//Flash function prototype
void copyfunc(void);
extern unsigned char prog_flash_line_128 (unsigned long t_address, union
char_rd_datum_union *p_data);
extern unsigned char erase_block (unsigned char block_num);
extern int  *_PkernelBegin, *_PkernelEnd, *_Pkernel_RAMBegin;
extern int  *_CkernelBegin, *_CkernelEnd, *_Ckernel_RAMBegin;

extern void PowerON_Reset(void);

#pragma section OtherIntPRG
// vector 1 Reserved
.
.

// vector 19 ADI
__interrupt(vect=19) void INT_ADI(void) {/* sleep(); */}
// vector 20 Direct Transition
__interrupt(vect=20) void INT_Direct_Transition(void) {/* sleep(); */}

//SCI ISR section fixed at 0x0440
#pragma section SCI_ISR
// vector 18 SCI3
__interrupt(vect=18) void INT_SCI3(void)
{

   unsigned short start_address;
   unsigned char prog_data_addr[128],count1;

   if ((P_SCI3.SSR.BYTE & 0x38) == 0)    //Check for SCI error
   {
      if(P_SCI3.RDR=='U')
      {
            copyfunc();

            erase_block (4); //erase Valid Flag

            while(1)
            {
               //GET START ADDRESS
```

```
                temp_buff = sci_get();
                start_address = (unsigned short) (temp_buff <<8); //high byte
                sci_put(temp_buff);

                temp_buff = sci_get();
                start_address = start_address | (unsigned short) (temp_buff);
                                                                  //low byte
                sci_put(temp_buff);

                if    (start_address == 0x0000) {erase_block (0);}
                else if (start_address == 0x0400) {erase_block (1);}
                else if (start_address == 0x0800) {erase_block (2);}
                else if (start_address == 0x0c00) {erase_block (3);}
                else if (start_address == 0x1000) {erase_block (4);}
                else if (start_address == 0x8000)
                      {PowerON_Reset();}//end of flash programming
                else  nop(); //invalid start address

                for(count1=0;count1<128;count1++)
                {
                   prog_data_addr[count1] =  sci_get();
                }
                if(prog_flash_line_128 (start_address, (union
                      char_rd_datum_union * ) prog_data_addr)==0x01)
                {
                   sci_put('a');
                }
                else sci_put('n');
             }
       }
       else  return; // if not Update flash command then do nothing
    }
    else
    {
       //SCI error occur
       if (P_SCI3.SSR.BIT.OER == 1)
       temp_buff = P_SCI3.RDR;
       temp_buff = P_SCI3.RDR;
       P_SCI3.SSR.BYTE=0x84;
       sci_put('e');
    }
}
```

```
void sci_put(char byte)
{
   while(P_SCI3.SSR.BIT.TDRE==0){}
   P_SCI3.TDR=byte;
   while(P_SCI3.SSR.BIT.TEND==0){}
}

char sci_get(void)
{
   while(P_SCI3.SSR.BIT.RDRF==0){}       //Wait until RDRF = 1
   if ((P_SCI3.SSR.BYTE & 0x38) ==0)     //Check for SCI error
   {
   return P_SCI3.RDR;
   }
   else return 0xFF;               //If error occur return 0xFF
   if(P_SCI3.SSR.BIT.RDRF==1) P_SCI3.SSR.BIT.RDRF=0;
}

void copyfunc(void)
{
   register int *p, *q;
   for (p=_PkernelBegin, q=_Pkernel_RAMBegin;p<_PkernelEnd;p++,q++){*q=*p;}
   for (p=_CkernelBegin, q=_Ckernel_RAMBegin;p<_CkernelEnd;p++,q++){*q=*p;}
}
#pragma section
```

9.2.3 方法 2 [新規ワークスペース] メインルーチン

下図に m2_new_ws.c のフローチャートを示します。続いて，プログラムリストを示します。

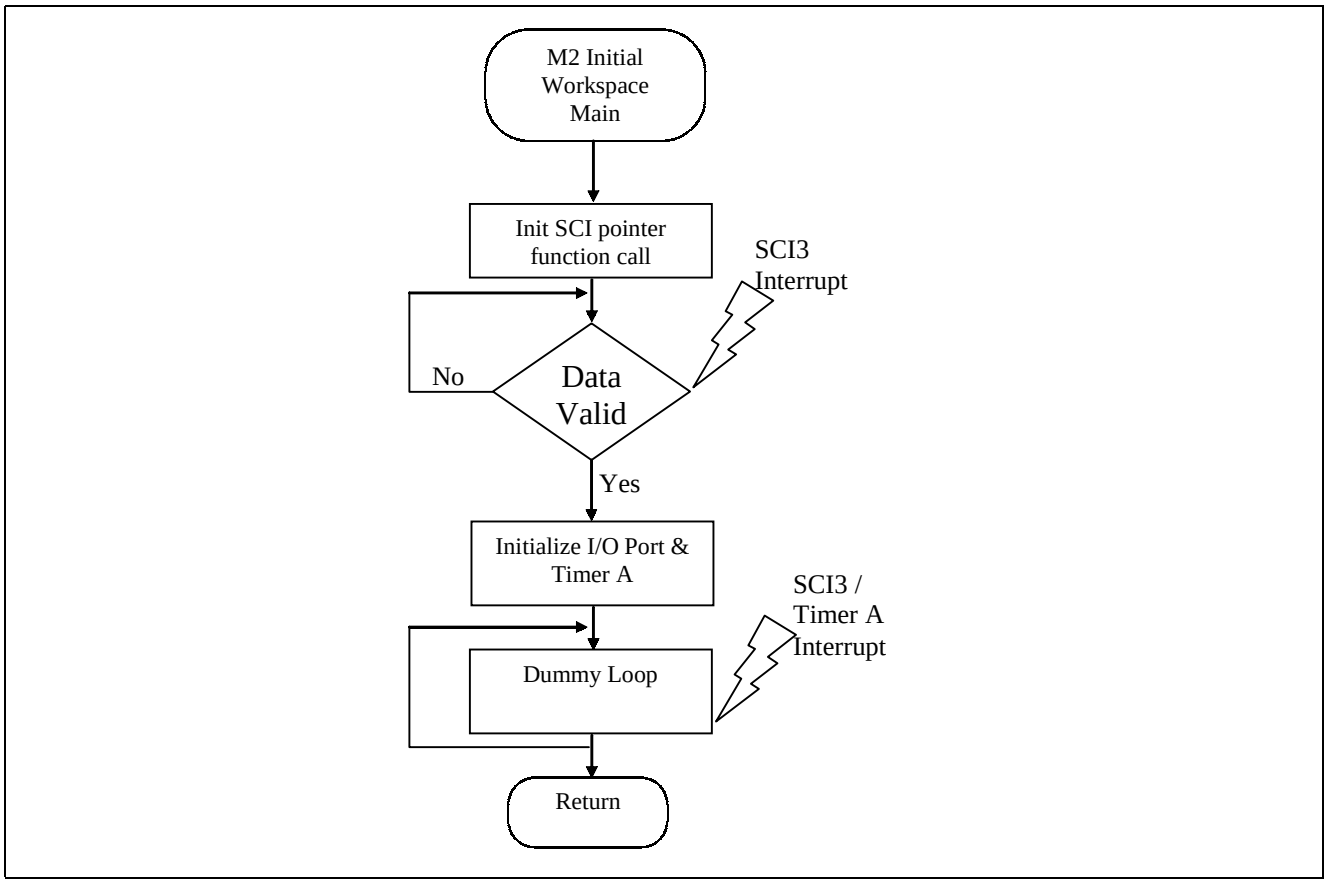

図29 方法 2 [新規ワークスペース] メインルーチンのフローチャート

```
/*********************************************************************/
/*                                                                   */
/*  FILE       :M2_new_ws.c                                          */
/*  DATE       :Mon, Sep 29, 2003                                    */
/*  DESCRIPTION :Main Program                                        */
/*  CPU TYPE   :H8/38024F                                            */
/*                                                                   */
/*  This file is generated by Renesas Project Generator (Ver.2.1).   */
/*                                                                   */
/*********************************************************************/
#include "iodefine.h"
#include <machine.h>

//pointer function call to init SCI
typedef  void         (*init_SCI_FnPtr)(void);
#define init_SCI_Fn   (init_SCI_FnPtr)((unsigned short *)(0x0400))

void main(void)
{
   unsigned long datavalid = 0x55AA1234, *VALID;
   unsigned int delay = 0;

   (*init_SCI_Fn) ();

   VALID = (unsigned long *)0x7FFC;
   if (*VALID != datavalid)
   {
      while(1); //wait for interrupt
   }
   P_IO.PDR9.BIT.P93 = 1;
   P_IO.PDR9.BIT.P92 = 1;

   P_SYSCR.IENR1.BIT.IENTA = 1;

   P_TMRA.TMA.BIT.TMA = 10;

   set_imask_ccr(0);
   while (1)
    {
      //write user code here
   }
}
//Code Valid Flag
#pragma section Valid
const unsigned long DATA = 0x55AA1234;
#pragma section
```

9.2.4 方法 2 [新規ワークスペース] タイマ A 割り込みサービスルーチン

下図に intprg.c のフローチャートを示します。続いて，プログラムリストを示します。

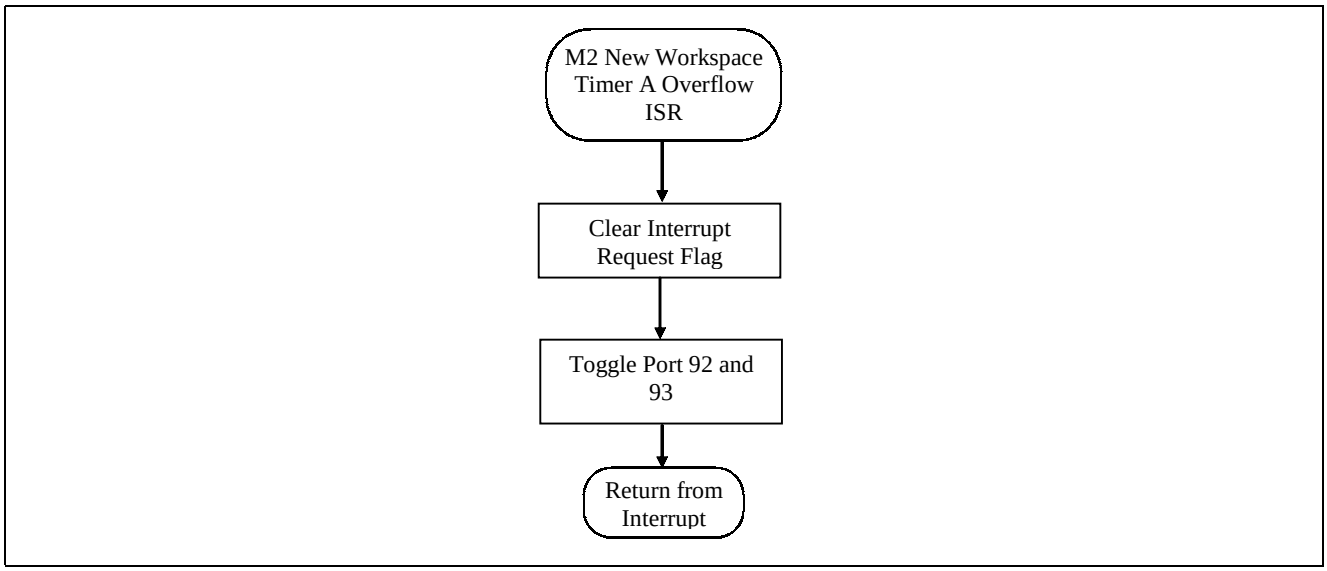

図30 方法 2 [新規ワークスペース] タイマ A 割り込みサービスルーチンのフローチャート

```
/************************************************************************/
/*                                                                      */
/*  FILE       :intprg.c                                                */
/*  DATE       :Mon, Sep 29, 2003                                       */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE   :H8/38024F                                               */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.2.1).     */
/*                                                                      */
/************************************************************************/
#include "iodefine.h"
#include <machine.h>
#pragma section IntPRG
// vector 1 Reserved
.
.
// vector 10 Reserved

// vector 11 Timer A Overflow
__interrupt(vect=11) void INT_TimerA(void)
{
   unsigned int delay = 0;
   if (P_SYSCR.IRR1.BIT.IRRTA == 1)
   P_SYSCR.IRR1.BIT.IRRTA = 0;
   P_IO.PDR9.BIT.P93 ^= 1;
   P_IO.PDR9.BIT.P92 ^= 1;
}
.
.
__interrupt(vect=16) void INT_TimerG(void) {/* sleep(); */}
// vector 17 Reserved

// vector 18 SCI3
// vector 19 ADI
__interrupt(vect=19) void INT_ADI(void) {/* sleep(); */}
// vector 20 Direct Transition
__interrupt(vect=20) void INT_Direct_Transition(void) {/* sleep(); */}

//Insert SCI ISR vector address as 0x0440
#pragma section SCI_ISR
static const unsigned short DATA = 0x0440;
//__interrupt(vect=18) void INT_SCI3(void) {/* sleep(); */}
```

9.3 KERNEL のプログラムリスト

9.3.1 フラッシュカーネルプログラム

下図に kernel.c のフローチャートを示します。続いて，プログラムリストを示します。

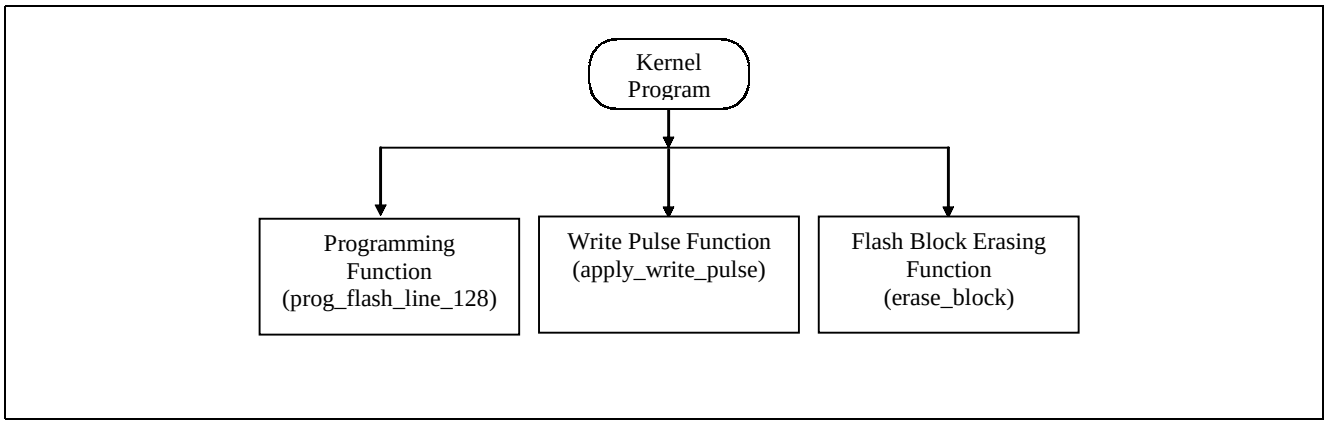

図31 フラッシュカーネルプログラムのフローチャート

注意： 詳細は，アプリケーションノート Flash Memory Programming Mode **を参照してください。**

```
// Renesas H8/38024F example flash programming and erasing routines
//
// kernel.c
//
// Clock speed = 9.8304MHz
// H8/38024F uses SCI3 for user mode
// Kernel start address - 0xF780


#include "iodefine.h"                    // IO header file
#include <machine.h>

// H8/38024F specific
#define FLASH_SWE P_ROM.FLMCR1.BIT.SWE
#define FLASH_PSU P_ROM.FLMCR1.BIT.PSU
#define FLASH_P   P_ROM.FLMCR1.BIT.P
#define FLASH_PV  P_ROM.FLMCR1.BIT.PV
#define FLASH_EBR1   P_ROM.EBR.BYTE
#define FLASH_ESU P_ROM.FLMCR1.BIT.ESU
#define FLASH_E   P_ROM.FLMCR1.BIT.E
#define FLASH_EV  P_ROM.FLMCR1.BIT.EV
#define  FLASH_FENR  P_ROM.FENR.BIT.FLSHE

// H8/38024F specific
#define MAX_FLASH_ADDR                   0x8000
#define FLASH_LINE_SIZE                  128
#define NO_OF_FLASH_BLOCKS               5
#define XTAL                             9830400L
#define MAX_PROG_COUNT                   1000
#define MAX_ERASE_ATTEMPTS               100
#define BLANK_VALUE                      0xFFFF      // 0xFFFFFFFF for SH,
                                                     //0xFFFF for H8S/300H

// array below should contain the start addresses of the flash memory blocks
// final array element should contain the end address of the flash memory (+1)

#pragma section kernel_const //only applicable for M2_init_ws
                             //additional constant section define needed

const unsigned long eb_block_addr [NO_OF_FLASH_BLOCKS + 1] =  {
   0x00000000L,
   0x00000400L,
   0x00000800L,
   0x00000C00L,
   0x00001000L,
   0x00008000L /* max flash address + 1 */
};

#define BLANK                            1
#define NOT_BLANK                        2
#define PROG_PASS                        0x01
#define PROG_FAIL                        0x02
#define ERASE_PASS                       0x01
#define ERASE_FAIL                       0x02
// delay values
// note this is xtal frequency specific
// these values are for the H8/38024F Timer F with a clock divider of 4
```

```
#define ONE_USEC                        ((1L * XTAL) / 8000000L)
#define TWO_USEC                        ((2L * XTAL) / 8000000L)
#define FOUR_USEC                       ((4L * XTAL) / 8000000L)
#define FIVE_USEC                       ((5L * XTAL) / 8000000L)
#define TEN_USEC                        ((1L * XTAL) / 800000L)
#define TWENTY_USEC                     ((2L * XTAL) / 800000L)
#define THIRTY_USEC                     ((3L * XTAL) / 800000L)
#define FIFTY_USEC                      ((5L * XTAL) / 800000L)
#define ONE_HUNDRED_USEC                ((1L * XTAL) / 80000L)
#define TWO_HUNDRED_USEC                ((2L * XTAL) / 80000L)
#define TEN_MSEC                        ((1L * XTAL) / 800L)

// typedef for reading the flash memory
// should be the size of the data bus connection to the flash memory
typedef unsigned short read_datum;

// function prototypes
unsigned char prog_flash_line_128 (unsigned long t_address, union
char_rd_datum_union *p_data);
void delay (unsigned short);
void init_delay_timer (void);
unsigned char erase_block (unsigned char block_num);
void apply_write_pulse(unsigned short prog_pulse);
extern void sci_put(char byte);
// variables
volatile unsigned long delay_counter;


union char_rd_datum_union {
   unsigned char c[FLASH_LINE_SIZE];
   read_datum u[FLASH_LINE_SIZE / sizeof (read_datum)];
} prog_data;

//DEFINE SECTION FOR KERNEL PROGRAM
#pragma section kernel

/*************************************************************
/*
/*    FUNCTION    :  prog_flash_line_128
/*    DESCRIPTION :  program 128 bytes of flash memory
/*    INPUT       :  flash start address,
/*                     program data pointer
/*    OUTPUT         : PROG_PASS if programming is successful
/*                     PROG_FAIL if programming is unsucessful
/*    Other information:
/*    t_address is the start address for the flash line to
/*    be programmed and must be on a flash line boundary e.g.
/*    multiple of 128 (this is not checked and so must be
/*    ensured by the caller) data to be programmed should be
/*    passed to this function in the form of a 'char_rd_datum_union'
/*    union pointer data must be written to the flash in byte units
/*
/*    Please note that for the H8/38024F during the dummy write,
/*    setting the PSU  and P bits no RTS intructions are permitted.
/*    Therefore no functions calls are allowed.
/*
/*    For this reason at these points in this function the code from
```

```
/*    the 'delay' function has been inlined to eliminate any RTS
/*    instructions. For further information on this see the Flash ROM
/*    section of the H8/38024F hardware manual version 4 or later.
/*
/************************************************************/

// Program 128 bytes functions start here
unsigned char prog_flash_line_128 (unsigned long t_address, union
char_rd_datum_union *p_data)
{
   unsigned char i;
   unsigned short n_prog_count;
   // loop counter for programming attempts (0 -> MAX_PROG_COUNT)
   unsigned short d;
   // variable used for various loop counts
   unsigned short ax;
   // loop counter for incrementing 'uc_v_write_address'

   // pointer (an unsigned short produces more efficient code than unsigned
   // char in this case)
   unsigned char m;
   // flag to indicate if re-programming is required (1=yes, 0=no)
   unsigned char *dest_address;          // pointer for writing to flash
   unsigned char *uc_v_write_address;
   // pointer for writing to address to be verified
   read_datum *ul_v_read_address;        // pointer for reading verify address
   union char_rd_datum_union additional_prog_data, re_program_data;
                                         // storage on stack for intermediate
                                         // programming data
   //Init Timer F start
   // 16 bit timer F counter, System clock / 4 selected
   P_TMRF.TCRF.BYTE = 0x86;

   //TCF cleared when TCF and OCRF match
   P_TMRF.TCSRF.BIT.CCLRH = 1;
   //Init Timer F end

   // enable access to the flash registers
   FLASH_FENR = 1;

   // enable flash writes
   FLASH_SWE = 1;

   // wait tSSWE (1 us)
   delay(ONE_USEC);

   // copy data from program data area to reprogram data area
   for (d=0; d<FLASH_LINE_SIZE; d++)
   {
      re_program_data.c[d] = p_data->c[d];
   }

   // program the data in FLASH_LINE_SIZE (128) byte chunks
   for (n_prog_count=0; n_prog_count<MAX_PROG_COUNT; n_prog_count++)
   {
      // clear reprogram required flag
      m = 0;
```

```
// copy data from reprogram data area into the flash with byte wide
// access
dest_address = (unsigned char *) t_address;

for (d=0; d<FLASH_LINE_SIZE; d++)
{
   *dest_address++ = re_program_data.c[d];
}

// to minimise code space the code to apply a write pulse has been
// placed into a separate function called 'apply_write_pulse'
if (n_prog_count < 6)
{
   apply_write_pulse(THIRTY_USEC);
}
else
{
   apply_write_pulse(TWO_HUNDRED_USEC);
}

// verify the data via word wide reads
uc_v_write_address = (unsigned char *) t_address;
ul_v_read_address = (read_datum *) t_address;

// enter program verify mode
FLASH_PV = 1;

// wait tSPV (4 us)
delay (FOUR_USEC);


// read data in read_datum size chunks
// verify loop
for (d=0; d<(FLASH_LINE_SIZE / sizeof(read_datum)); d++)
{
   // dummy write of H'FF to verify address
   *uc_v_write_address = 0xff;

   // see note at beginning of function
   // no RTS allowed here so 'apply_write_pulse' function inlined

   P_TMRF.OCRF.BYTE.H = (TWO_USEC)>>8;
   P_TMRF.OCRF.BYTE.L = (TWO_USEC);


   // Clear compare match flag
   P_TMRF.TCSRF.BIT.CMFH = 0;

   // Clear counter and start the timer F
   P_TMRF.TCF.BYTE.H = 0;
   P_TMRF.TCF.BYTE.L = 0;

   // Loop until we have a compare match
   while (P_TMRF.TCSRF.BIT.CMFH == 0);

   // increment this pointer to get to next verify address
```

```
      for (ax=0; ax<sizeof(read_datum); ax++)
      uc_v_write_address++;

      // read verify data
      // check with the original data
      if (*ul_v_read_address != p_data->u[d])
      {
         // 1 or more bits failed to program
         //
         // set the reprogram required flag
         m = 1;
      }

      //Enable watchdog timer
      P_WDT.TCSRW.BYTE = 0x5A;
      P_WDT.TCW = 0x00;
      P_WDT.TCSRW.BYTE = 0xF4;

      // check if we need to calculate additional programming data
      if (n_prog_count < 6)
      {
         // calculate additional programming data
         // simple ORing of the reprog and verify data
         additional_prog_data.u[d] = re_program_data.u[d] |
         *ul_v_read_address;
      }

      // calculate reprog data
      re_program_data.u[d] = p_data->u[d] | ~(p_data->u[d] |
      *ul_v_read_address);

      // increment the verify read pointer
      ul_v_read_address++;

      //Disable watchdog timer
      P_WDT.TCSRW.BYTE = 0xF2;
   } // end of verify loop
   // exit program verify mode
   FLASH_PV = 0;

   // check if additional programming is required
   if (n_prog_count < 6)
   {
      // perform additional programming
      //
      // copy data from additional programming area to flash memory
      dest_address = (unsigned char *) t_address;
      for (d=0; d<FLASH_LINE_SIZE; d++)
      {
         *dest_address++ = additional_prog_data.c[d];
      }

      apply_write_pulse(TEN_USEC);
   }
   // check if flash line has successfully been programmed
   if (m == 0)
   {
```

```
            // program verified ok
            //
            // disable flash writes
            FLASH_SWE = 0;

            // wait tCSWE (100 us)
            delay (ONE_HUNDRED_USEC);

            // end of successful programming
            // disable access to the flash registers
            FLASH_FENR = 0;
            return (PROG_PASS);
         }

      }  // end of for loop (n<MAX_PROG_COUNT) at this point we have made
         // MAX_PROG_COUNT programming attempts

      // failed to program after MAX_PROG_COUNT attempts
      // disable flash writes
      FLASH_SWE = 0;

      // wait tCSWE (100 us)
      delay (ONE_HUNDRED_USEC);

      // end of failed programming
      // disable access to the flash registers
      FLASH_FENR = 0;
      return (PROG_FAIL);
   }
   // Program 128 bytes functions end here
```

```
/*********************************************************
/*
/*  FUNCTION    :apply_write_pulse
/*  DESCRIPTION :Applies programming pulse to flash memory
/*  INPUT       :prog_pulse = 30us, 200us or 10us
/*  OUTPUT      :None
/*********************************************************/
// apply_write_pulse functions start here
void apply_write_pulse(unsigned short prog_pulse)
{

   //Enable watchdog timer
   P_WDT.TCSRW.BYTE = 0x5A;
   P_WDT.TCW = 0x00;
   P_WDT.TCSRW.BYTE = 0xF4;

   // enter program setup mode
   FLASH_PSU = 1;

   // no RTS allowed here so 'apply_write_pulse' function inlined

   P_TMRF.OCRF.BYTE.H = FIFTY_USEC>>8;
   P_TMRF.OCRF.BYTE.L = FIFTY_USEC;

   // Clear compare match flag
   P_TMRF.TCSRF.BIT.CMFH = 0;

   // Clear counter and start the timer F
   P_TMRF.TCF.BYTE.H = 0;
   P_TMRF.TCF.BYTE.L = 0;

   // Loop until we have a compare match
   while (P_TMRF.TCSRF.BIT.CMFH == 0);


   // start programming pulse
   FLASH_P = 1;

   // no RTS allowed here so 'apply_write_pulse' function inlined

   P_TMRF.OCRF.BYTE.H = prog_pulse>>8;
   P_TMRF.OCRF.BYTE.L = prog_pulse;

   // Clear compare match flag
   P_TMRF.TCSRF.BIT.CMFH = 0;

   // Clear counter and start the timer F
   P_TMRF.TCF.BYTE.H = 0;
   P_TMRF.TCF.BYTE.L = 0;

   // Loop until we have a compare match
   while (P_TMRF.TCSRF.BIT.CMFH == 0);

   // stop programming
   FLASH_P = 0;

   // delay (FIVE_USEC);
```

```
    P_TMRF.OCRF.BYTE.H = FIVE_USEC>>8;
    P_TMRF.OCRF.BYTE.L = FIVE_USEC;

    // Clear compare match flag
    P_TMRF.TCSRF.BIT.CMFH = 0;

    // Clear counter and start the timer F
    //P_TMRF.TCF.WORD = 0;
    P_TMRF.TCF.BYTE.H = 0;
    P_TMRF.TCF.BYTE.L = 0;

    // Loop until we have a compare match
    while (P_TMRF.TCSRF.BIT.CMFH == 0);


    // exit program setup mode
    FLASH_PSU = 0;

    // wait tCPSU (5 us)
    // delay (FIVE_USEC);
    P_TMRF.OCRF.BYTE.H = FIVE_USEC>>8;
    P_TMRF.OCRF.BYTE.L = FIVE_USEC;

    // Clear compare match flag
    P_TMRF.TCSRF.BIT.CMFH = 0;

    // Clear counter and start the timer F
    //P_TMRF.TCF.WORD = 0;
    P_TMRF.TCF.BYTE.H = 0;
    P_TMRF.TCF.BYTE.L = 0;

    // Loop until we have a compare match
    while (P_TMRF.TCSRF.BIT.CMFH == 0);
    //Disable watchdog timer
    P_WDT.TCSRW.BYTE = 0xF2;
}
// apply_write_pulse functions end here
```

```
/*********************************************************
/*
/*  FUNCTION    :erase_block
/*  DESCRIPTION :Erase flash memory block
/*  INPUT       :block_num = 0,1,2,3,4
/*  OUTPUT      :ERASE_PASS is attempt is successful
/*           ERASE_FAIL is attempt fails
/*********************************************************/
// erase block functions start here
unsigned char erase_block (unsigned char block_num)
{
   unsigned char erase, ax, x;
   unsigned long attempts;
   read_datum *ul_v_read;
   unsigned char *uc_v_write;

   //Init Timer F start
   // 16 bit timer F counter, System clock / 4 selected
   P_TMRF.TCRF.BYTE = 0x86;

   //TCF cleared when TCF and OCRF match
   P_TMRF.TCSRF.BIT.CCLRH = 1;

   // check that block is not already erased
   erase = BLANK;
   for (attempts=eb_block_addr[block_num]; attempts<eb_block_addr[block_num +
   1]; attempts++)
   {
      if ( *(unsigned char *) attempts != 0xff)
         erase = NOT_BLANK;
   }

   if (erase == BLANK)
      return ERASE_PASS;
   else
   {
      // block needs erasing
      //
      // enable access to the flash registers
      FLASH_FENR = 1;

      // enable flash writes
      FLASH_SWE = 1;

      // wait tSSWE (1us)
      delay (ONE_USEC);

      // initialise the attempts counter
      // 0 as we check for less than MAX (not <= MAX)
      attempts = 0;

      // set the correct EB bit in correct EBR register
      FLASH_EBR1 = 1<<block_num;
      erase = 0;
      while ( (attempts < MAX_ERASE_ATTEMPTS) && (erase == 0) )
      {
         // increment the attempts counter
```

```
attempts++;
// enter erase setup mode
FLASH_ESU = 1;

// wait tSESU (100 us)
delay (ONE_HUNDRED_USEC);

// start erasing
FLASH_E = 1;

// wait tSE (10 ms)
delay (TEN_MSEC);

// stop erasing
FLASH_E = 0;

// wait tCE (10 us)
delay (TEN_USEC);

// exit erase setup mode
FLASH_ESU = 0;

// wait tCESU (10 us)
delay (TEN_USEC);

// enter erase verify mode
FLASH_EV = 1;

// wait tSEV (20 us)
delay (TWENTY_USEC);

// verify flash has been erased
// setup the pointers for reading and writing the flash
ul_v_read = (read_datum *) eb_block_addr [block_num];
uc_v_write = (unsigned char *) eb_block_addr [block_num];

erase = 1;
while ( (erase == 1) && ( ul_v_read < (read_datum *) eb_block_addr
[block_num + 1] ) )
{
   // this loop will exit either when one word is not erased ('erase'
   // becomes 0)
   // or all addresses have been read as erased ('erase' stays as 1)
   // if 'erase' stays as 1 the outer while loop will exit as the
   // block has been erased
   //
   // dummy write
   *uc_v_write = 0xff;

   // see note at beginning of function
   // no RTS allowed here so 'apply_write_pulse' function inlined
   P_TMRF.OCRF.BYTE.H = TWO_USEC>>8;
   P_TMRF.OCRF.BYTE.L = TWO_USEC;

   // Clear compare match flag
   P_TMRF.TCSRF.BIT.CMFH = 0;
```

```
        // Clear counter and start the timer F
        P_TMRF.TCF.BYTE.H = 0;
        P_TMRF.TCF.BYTE.L = 0;


        // Loop until we have a compare match
        while (P_TMRF.TCSRF.BIT.CMFH == 0);


        if (*ul_v_read != BLANK_VALUE)
        {
           // this word is not erased yet
           erase = 0;
        }
        else
        {
           // advance to the next byte write address
           for (ax=0; ax<sizeof(read_datum); ax++)
              uc_v_write++;

           // advance to the next verify read address
           ul_v_read++;
        }
     }

     // exit erase verify mode

     FLASH_EV = 0;

     // wait tCEV (4 us)
     delay (FOUR_USEC);
  }  // end of outer while loop

  // end either of erase attempts or block has been erased ok
  //
  // disable flash writes
  FLASH_SWE = 0;

  // wait tCSWE (100 us)
  delay (ONE_HUNDRED_USEC);

  // check if block has been erased ok
  if (erase == 1)
  {
     // successfully erased
     // disable access to the flash registers
     FLASH_FENR = 0;
     return ERASE_PASS;
  }
  else
  {
     // failed to erase this block
     // disable access to the flash registers
     FLASH_FENR = 0;
     return ERASE_FAIL;
  }
```

```
    }
}
// erase block functions end here

/************************************************************
/*
/*  FUNCTION    :delay
/*  DESCRIPTION :Timer F delay function
/*  INPUT       :d = time in us
/*  OUTPUT      :None
/************************************************************/
// delay functions start here
void delay (unsigned short d)
{
    // load compare match value into the output compare register

    P_TMRF.OCRF.BYTE.H = d>>8;
    P_TMRF.OCRF.BYTE.L = d;

    // Clear compare match flag
    P_TMRF.TCSRF.BIT.CMFH = 0;

    // Clear counter and start the timer F
    P_TMRF.TCF.BYTE.H = 0;
    P_TMRF.TCF.BYTE.L = 0;

    // Loop until we have a compare match
    while (P_TMRF.TCSRF.BIT.CMFH == 0);

    P_TMRF.TCSRF.BIT.CMFH = 0;
}
// delay functions start here

#pragma section
//end of kernel section
```

9.3.2 ROM から RAM へのマッピングのプログラム

ROMtoRAM.c の ROM から RAM へのマッピングセクション宣言のプログラムリストを以下に示します。

このプログラムは ROM に格納されますが，RAM で実行されます。このため，異なる扱いが必要です。コンパイラが正しい実行コードを生成するためには，セクションを正確にマッピングされていなければなりません。

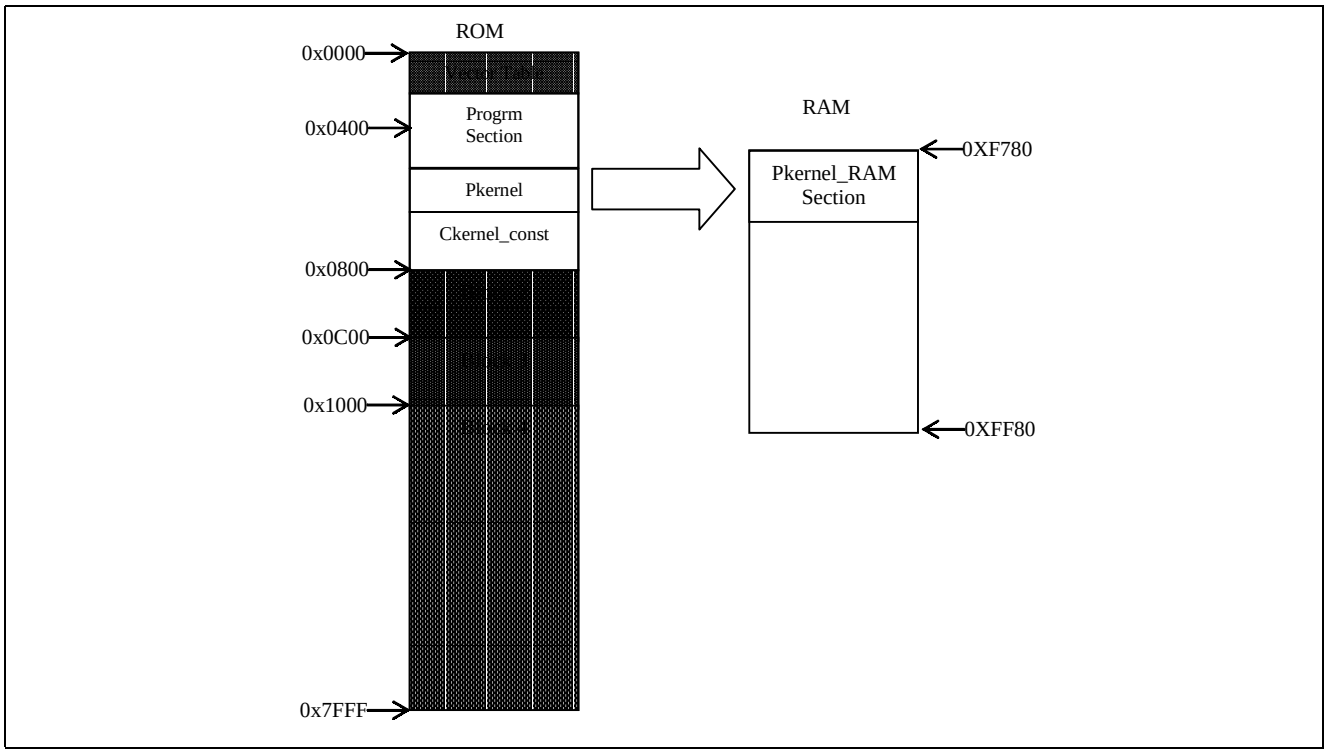

図32　カーネルセクションのメモリマップ

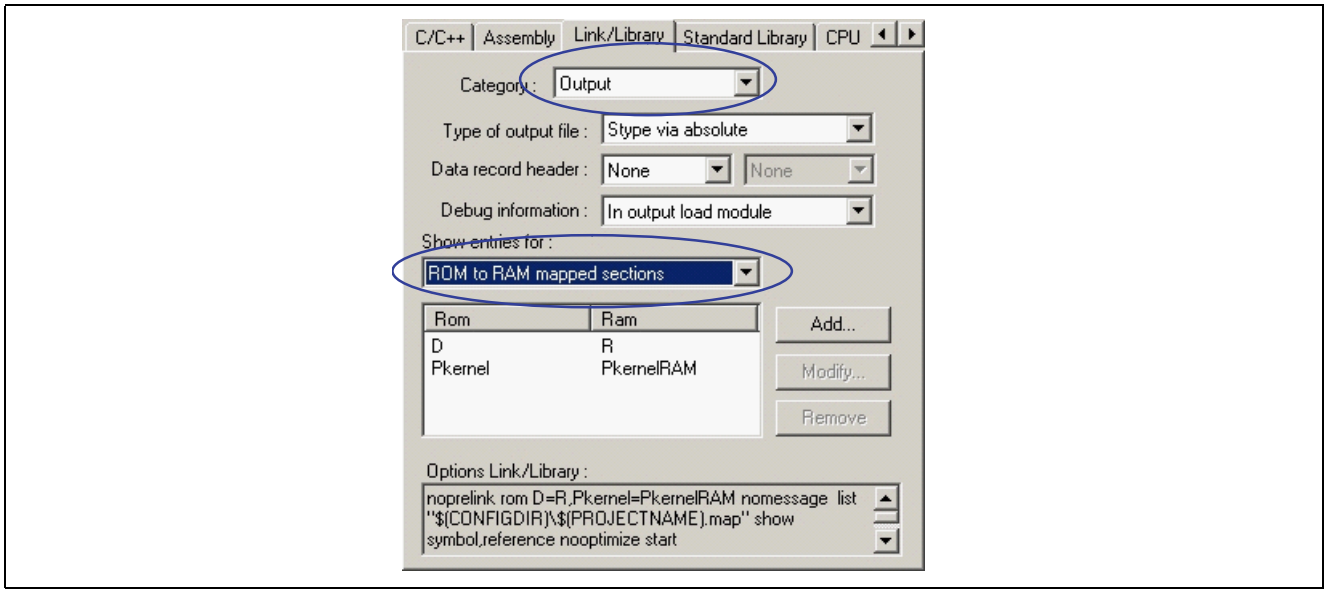

図33　ROM から RAM へのセクションマッピングの構成

```
#pragma asm
   .SECTION PkernelRAM,CODE,ALIGN=2

   .SECTION Pkernel,CODE,ALIGN=2

   .SECTION Ckernel_const,DATA,ALIGN=2

;Start Address of Section ROMCODE - kernel
__PkernelBegin .DATA.W (STARTOF Pkernel)

;End Address of Section ROMCODE - kernel
__PkernelEnd .DATA.W (STARTOF Pkernel) + (SIZEOF Pkernel)

;Start Address of Section RAMCODE - kernel
__Pkernel_RAMBegin .DATA.W (STARTOF PkernelRAM)

   .EXPORT __PkernelBegin
   .EXPORT __PkernelEnd
   .EXPORT __Pkernel_RAMBegin
#pragma endasm
```

注意：上記コードはアセンブリ言語で記述されています。したがって，*Options* メニューの *Renesas H8 Tiny/SLP Toolchain* で Assembly source code (*.src)**を構成してください。**

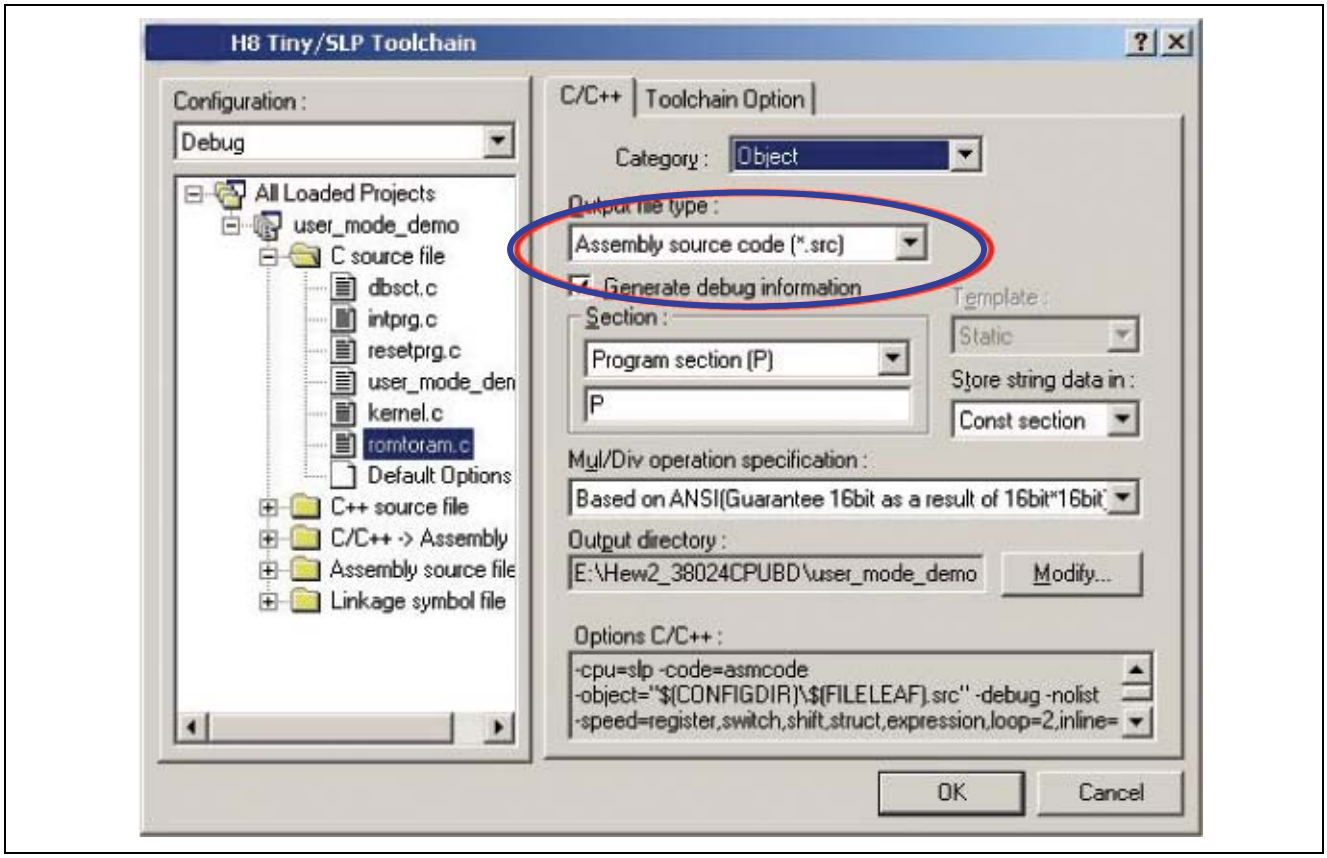

図34　ROM to RAM .c **ファイルの構成**

10. シリアル通信を介したデバッグのテクニック

インタフェースプロトコルを変更する場合，プログラマは以下のテクニックを用いてデバッグすることができます。PC と SLP MCU との間の送受信をモニタリングするための簡単なシリアル通信ツールを構築することができます。

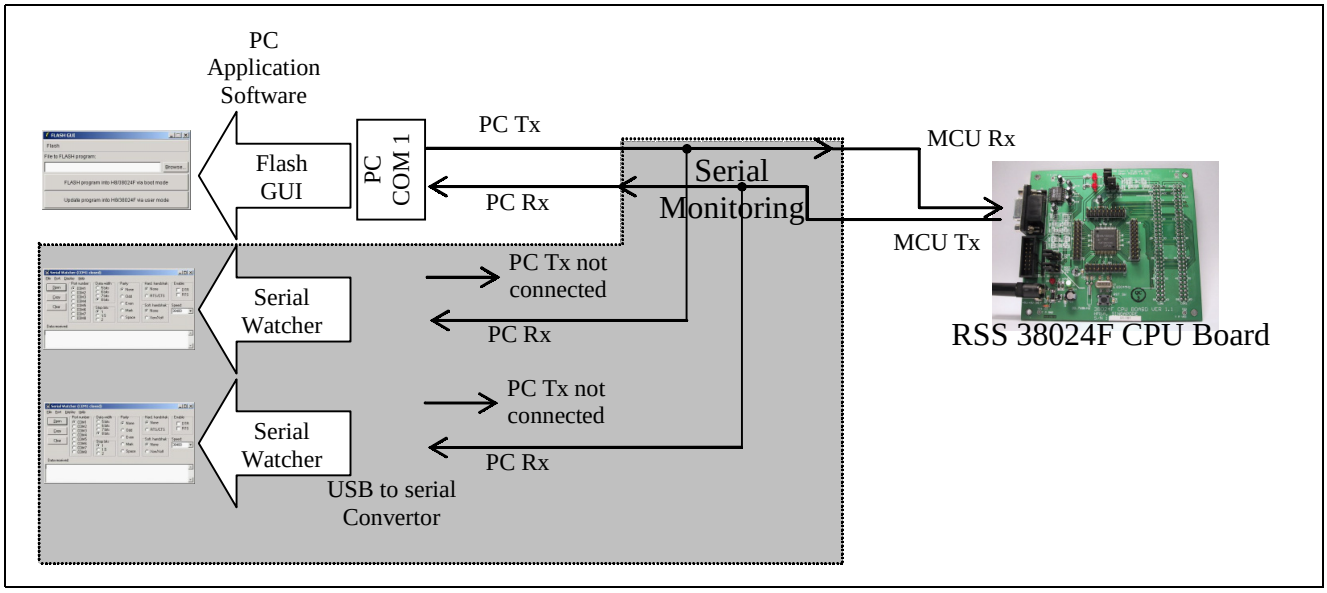

図35　シリアル通信モニタリングツール

この場合，PC 側には 3 つのシリアルポートが必要です。

1. SLP 制御用フラッシュ GUI
2. PC の送信側のモニタリング
3. PC の受信側のモニタリング

“SerialWatcher.exe は COM ポートの動作をモニタリングするためのソフトウェアです。データを 16 進数や ASCII で表示することができ，一度に 8 つの COM ポートをサポートできます。

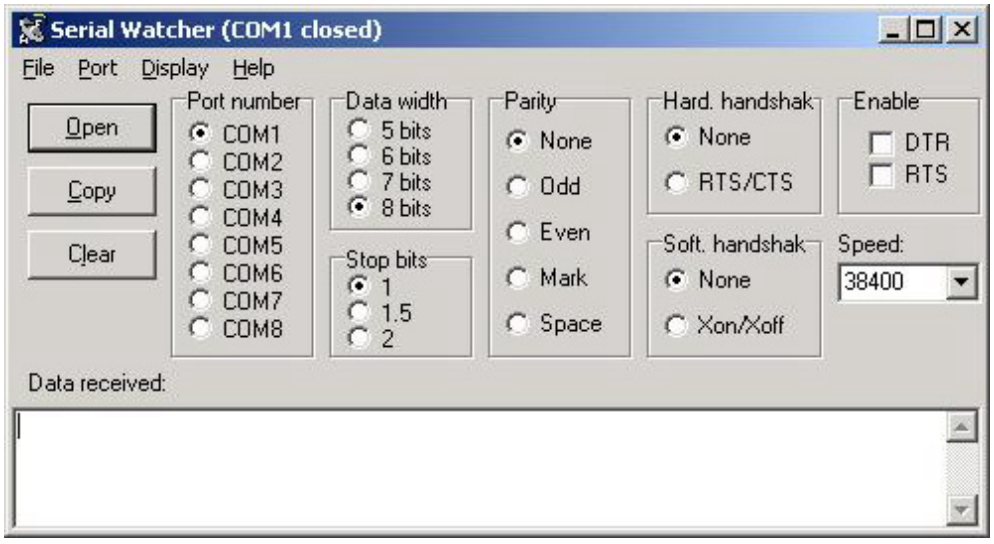

図36　Windows 用シリアルウォッチャ 2.0.4

ユーザはシリアルウォッチャソフトウェアを以下の URL からダウンロードできます。

http://www.pcremotecontrol.com/serialwatcher.zip .

11. 参考文献

TCL関連：

1. http://www.activestate.com/Products/ActiveTcl/
2. http://freewrap.sourceforge.net/

他の関連アプリケーションノート：

1. F-ZTAT™ Microcomputer On-Board Programming (Application Note ref. no: ADE-502-042, http://renesas.com,)
2. F-ZTAT™ Microcomputer Single Power Supply F-ZTAT™ On-Board Programming, (Application Note ref. no: ADE-502-055, http://renesas.com)

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2004.08.06	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。