EPSON
EXCEED YOUR VISION

S1C31 Family Application Note

S1C31 Family
ブートローダ マニュアル

arm

セイコーエプソン株式会社

Rev.1.00

評価ボード・キット、開発ツールご使用上の注意事項

1. 本評価ボード・キット、開発ツールは、お客様での技術的評価、動作の確認および開発のみに用いられることを想定し設計されています。それらの技術評価・開発等の目的以外には使用しないで下さい。本品は、完成品に対する設計品質に適合していません。

2. 本評価ボード・キット、開発ツールは、電子エンジニア向けであり、消費者向け製品ではありません。お客様において、適切な使用と安全に配慮願います。弊社は、本品を用いることで発生する損害や火災に対し、いかなる責も負いかねます。通常の使用においても、異常がある場合は使用を中止して下さい。

3. 本評価ボード・キット、開発ツールに用いられる部品は、予告無く変更されることがあります。

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告無く変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りいたします。

2. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これらに起因する第三者の知的財産権およびその他の権利侵害あるいは損害の発生に対し、弊社はいかなる保証を行うものではありません。また、本資料によって第三者または弊社の知的財産権およびその他の権利の実施権の許諾を行うものではありません。

3. 特性値の数値の大小は、数直線上の大小関係で表しています。

4. 製品および弊社が提供する技術を輸出等するにあたっては「外国為替および外国貿易法」を遵守し、当該法令の定める手続きが必要です。大量破壊兵器の開発等およびその他の軍事用途に使用する目的をもって製品および弊社が提供する技術を費消、再販売または輸出等しないでください。

5. 本資料に掲載されている製品は、生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本（当該）製品をこれらの用途に用いた場合のいかなる責任についても負いかねます。

6. 本資料に掲載されている会社名、商品名は、各社の商標または登録商標です。

目　次

1. 概要

S1C31 ブートローダは、PC 上のターミナルソフトウェアを使用し、対象機種の内蔵フラッシュメモリを新しいユーザプログラムに書き換えるサンプルソフトウェアです。

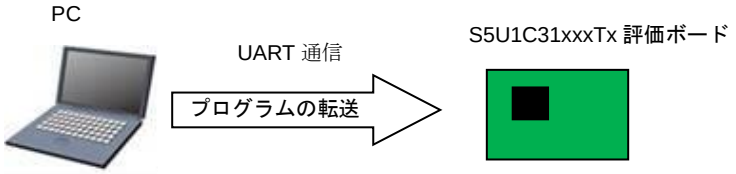

図 1.1 構成図

本ドキュメントは、以下の 2 種類のサンプルソフトウェアについて説明します。

- ブートローダ（bootloader）

  UART 通信によりユーザプログラムを受け取り、対象機種の内蔵フラッシュメモリを書き換えます。内蔵フラッシュメモリの書き換えには、S1C31 自己書き換えライブラリ（seFlashLibrary）を使用します。

- ユーザプログラム（loadsample）

  ブートローダを経由し、対象機種の内蔵フラッシュメモリに書き込むプログラムです。

本ソフトウェアは、S1C31xxx 周辺回路サンプルソフトウェアパッケージに同梱されています。S1C31xxx 周辺回路サンプルソフトウェアパッケージは、弊社 Web より入手可能です。
また、本マニュアルに加えて、“S1C31xxx テクニカルマニュアル”も併せてご参照ください。

1.1 動作環境

本サンプルソフトウェアの書き込み時およびデバッグ時には、以下が必要です。

- 評価ボード
  - S1C31 シリーズ搭載 S5U1C31xxxTx 評価ボード
- デバッグプローブ *1*2
  - IAR Systems 社製 I-jet または SEGGER 社製 J-Link
- 統合開発環境
  - IAR Embedded Workbench for ARM®（IAR EWARM）または MDK-ARM®（µVision）
- S1C31SetupTool パッケージ
  - フラッシュローダおよび構成ファイル（.svd など）
- S1C31xxx 周辺回路サンプルソフトウェアパッケージ
- PC 用ターミナルソフト
- その他のデバイス（オプション）
  - UART 用 USB アダプタ

*1:サンプルソフトウェアからの関数コールによるフラッシュメモリ消去、書き込み処理時には、デバッグプローブは不要となります。

*2: I-jet は IAR EWARM でのみ使用可能です。J-Link は IAR EWARM と MDK-ARM の両方で使用可能です。

上記の詳細については、それぞれに付属するマニュアルをご参照ください。

1.2 使用上の注意事項

本サンプルソフトウェアは、参考資料です。これらに起因する不具合が発生した場合、弊社は如何なる責任についても負いません。製品上でお使いになる場合には、十分な動作検証を実施してください。

本マニュアルは、S1C31 シリーズの各機種に用意しているブートローダ共通の資料です。機種ごとに異なる仕様（使用端子等）については、S1C31xxx 周辺回路サンプルソフトウェアパッケージに同梱しているリードミーをご参照ください。また、S131 自己書き換えライブラリについては、“S1C31 Family 自己書き換えライブラリマニュアル”をご参照ください。

2. ソフトウェアの構成

2.1 フォルダ構成

S1C31xxx 周辺回路サンプルソフトウェアパッケージに含まれる S1C31 ブートローダおよび関連プログラムの構成は以下の通りです。

```
S1C31xxxSamplePKG_very_yy.zip
[S1C31xxxSamplePKG_very_yy]
      |- [Licenses]
      |- [Drivers] : ドライバ群
      |      |- [board] : 評価ボード関連ドライバ
      |      |- [CMSIS] : CMSIS ドライバ
      |      |      |- [Device]
      |      |      |      |- [S1C31xxx]
      |      |      |             |- [Include]
      |      |      |             |      |-S1C31xxx.h : CMSIS 周辺回路アクセスレイヤヘッダファイル
      |      |      |             |      |- …
      |      |      |             |- [Source]
      |      |      |                    |- [ARM]
      |      |      |                    |- [IAR]
      |      |      |                    |      |- startup_S1C31xxx.s : CMSIS スタートアッププログラム
      |      |      |                    |- system_S1C31xxx.c : CMSIS 周辺回路アクセスレイヤプログラム
      |      |      |- [Driver]
      |      |      |      |- [Include]
      |      |      |      |      |- Driver_Common.h :共通ドライバ定義
      |      |      |      |      |- Driver_Flash.h : CMSIS 自己書き換えライブラリドライバ定義
      |      |      |      |      |- …
      |      |      |      |- [Source]
      |      |      |- [SVD]
      |      |- [sePeripheralLibrary] : 周辺回路ライブラリ
      |
      |- [Middlewares] : ミドルウェア群
      |      |- [seFlashLibrary] : 自己書き換えライブラリ
      |      |      |- [Device]
      |      |      |      |- [S1C31xxx]
      |      |      |             |- seFlashLibraryS1C31xxx.a : IAR EAWRM 用ライブラリ
      |      |      |             |- seFlashLibraryS1C31xxx.lib : MDK-ARM 用ライブラリ
      |      |      |- flashLibraryForS1c31xxx_readme_e.txt : リードミー
      |      |      |- flashLibraryForS1c31xxx_readme_j.txt
      |      |- …
      |- [Projects] : サンプルソフトウェア群
      |      |- [Applications] : 各種アプリケーションソフトウェア
      |      |      |- [BOOTLOADER] :ブートローダ用サンプルソフトウェア
      |      |      |      |- [bootloader] : ブートローダ
      |      |      |      |      |- [ARM] : MDK-ARM 用プロジェクト
      |      |      |      |      |- [IAR] : IAR EAWRM 用プロジェクト
      |      |      |      |      |- main.c
      |      |      |      |      |- …
      |      |      |      |- [loadsample] :ユーザプログラム
      |      |      |      |      |- [ARM] : MDK-ARM 用プロジェクト
      |      |      |      |      |- [IAR] : IAR EAWRM 用プロジェクト
      |      |      |      |      |- main.c
      |      |      |      |- bootLoaderForS1c31xxx_readme_e.txt : リードミー
      |      |      |      |- bootLoaderForS1c31xxx_readme_j.txt
      |      |      |- …
      |      |- …
      |
      README_e.txt
      README_j.txt
```

図 2.1.1 S1C31xxx サンプルソフトウェアパッケージの構成

2.2 ユーザプログラム形式

新しいユーザプログラムとして書き換え可能なデータ形式は、モトローラ S レコードです。また、データの終端は、S7/S8/S9 レコードのいずれかである必要があります。これにより、ブートローダは内蔵フラッシュメモリの書き換えが完了したことを認識します。

3. ブートローダの機能

ブートローダは、シリアルポートをサポートするターミナルソフトおよび、内蔵フラッシュメモリ（Flash ROM）の書き換えを行う S1C31 自己書き換えライブラリ（seFlashLibrary）を使用することで、機能を実現します。

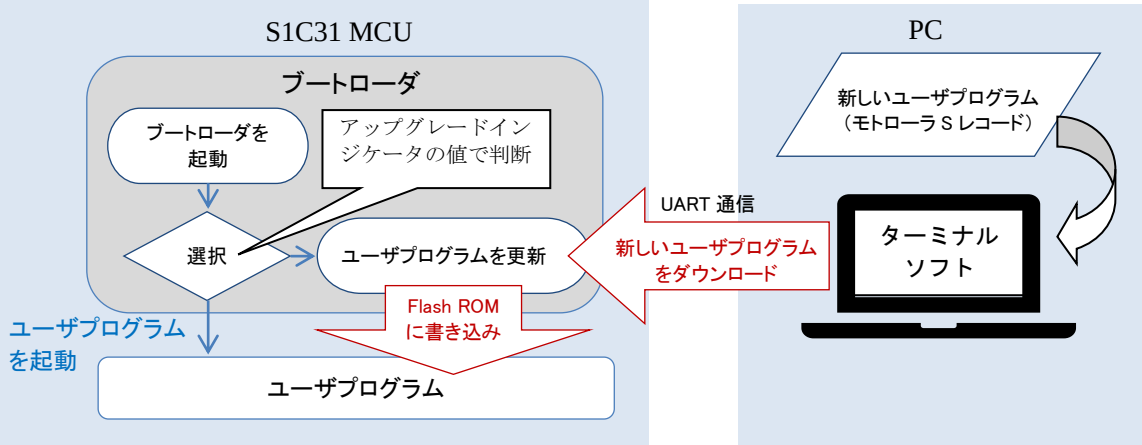

図 3.1 ブートローダの機能概要

3.1 ブートローダ

ブートローダ（bootloader）のプロジェクト構成、使用するライブラリ、動作エリアは以下の通りです。

- bootloader プロジェクト構成 ：
  - board
  - CMSIS
  - sePeripheralLibrary
  - main.c　　ブートローダ本体
  - indicator.c　　アップグレードインジケータ定義
  - srec.c　　モトローラ S レコードのデコード処理
  - srec.h　　モトローラ S レコードのデコード処理用定義

- 使用するライブラリ:
  S1C31 自己書き換えライブラリ　　seFlashLibrary
  ※S1C31 自己書き換えライブラリについては、”S1C31 Family 自己書き換えライブラリマニュアル”を参照してください。

- 動作エリア ：
  Flash ROM　　0x0000-0x3FFF(16KB)

ブートローダは、MCU のリセット後に動作を開始し、アップグレードインジケータの値により以下の機能のいずれかを実行します。
- ユーザプログラムの起動
- ユーザプログラムを更新

3.1.1 使用する周辺回路

ブートローダでは以下の周辺回路を使用しています。

- UART CH0：PC とのシリアル通信に使用

表 3.1.1.1 シリアル通信フォーマット

通信速度	230400bps
データ長	8 ビット
パリティ	なし
ストップビット	1 ビット

UART チャンネル 0 が使用する端子については、S1C31xxx 周辺回路サンプルソフトウェアパッケージに同梱しているリードミーをご参照ください。

- ウォッチドッグタイマ：データの終端を受信した場合、システムを再起動用として使用
- 16 ビットタイマ CH0：S1C31 自己書き換えライブラリ内で使用
- 16 ビットタイマ CH1：通信タイムアウト用タイマとして使用

3.1.2 アップグレードインジケータ

アップグレードインジケータは、アドレス 0x3000 番地に配置され、ユーザプログラムの更新状態を管理します。アップグレードインジケータのサイズは 4KB です。ただし、実際に使用されるのは 4 バイトのみで、4 バイトの値は以下の状態を示します。

表 3.1.2.1 アップグレードインジケータの状態

状態	値
“upgrade completed” 更新が完了し、ユーザプログラムを実行できます。	0xAA, 0xAA, 0xAA, 0xAA
“do upgrade” ユーザプログラムが更新を決定しましたが、更新が開始されていません。	0x00, 0x00, 0x00, 0x00
“now upgrading” ブートローダがユーザプログラムを更新しています。更新は完了していません。	0xFF, 0xFF, 0xFF, 0xFF

■ **indicator.c について**

indicator.c にアップグレードインジケータの初期値が定義されています。
使用するワークベンチにより定義が異なります。
通常時は“0xaa, 0xaa, 0xaa, 0xaa”を指定してください。(デフォルト)
ブートローダ自身をデバッグしたい場合は、”0x00, 0x00, 0x00, 0x00”としてください。

```
#ifdef __IAR_SYSTEMS_ICC__
#pragma location = ".ConstSection1"
const unsigned char upgrade_indicator[]    = {
                        //0xFF, 0xFF, 0xFF, 0xFF,        /// now upgrading
                        0xaa, 0xaa, 0xaa, 0xaa,        /// upgrade_completed
                        //0x00, 0x00, 0x00, 0x00,        /// do_upgrade
};
#else
const unsigned char upgrade_indicator[] __attribute__((section(".ConstSection1"))) =
                        //0xFF, 0xFF, 0xFF, 0xFF,        /// now upgrading
                        0xaa, 0xaa, 0xaa, 0xaa,        /// upgrade_completed
                        //0x00, 0x00, 0x00, 0x00,        /// do_upgrade
};
#endif
```

IAR EWARM 使用時

µVision 使用時

3.1.3 ブートローダの動作

ブートローダは、以下の動作をします。

■ ユーザプログラムの起動

アップグレードインジケータが“**upgrade completed**”（0xAA,0xAA,0xAA,0xAA）の場合、ブートローダはアドレス 0x4000 番地以降に存在するユーザプログラムへジャンプします。その際ベクタテーブルオフセットアドレス(VTOR)を 0x4000 に変更します。

■ ユーザプログラムの更新

アップグレードインジケータが“**do upgrade**”（0x00,0x00,0x00,0x00）または“**now upgrading**”（0xFF,0xFF,0xFF,0xFF）の場合、ブートローダは UART を使用して新しいユーザプログラムを受信し、アドレス 0x4000 番地以降の領域に書き込みます。また、“**do upgrade**”（0x00,0x00,0x00,0x00）の場合、既存のユーザプログラムを消去します。

ブートローダは、データの終端を受信するまで、ユーザプログラムの更新を継続します。データの終端を受信すると、ブートローダは停止し、WDT により MCU をリセットします。リセット後は新しいユーザプログラムが起動します。

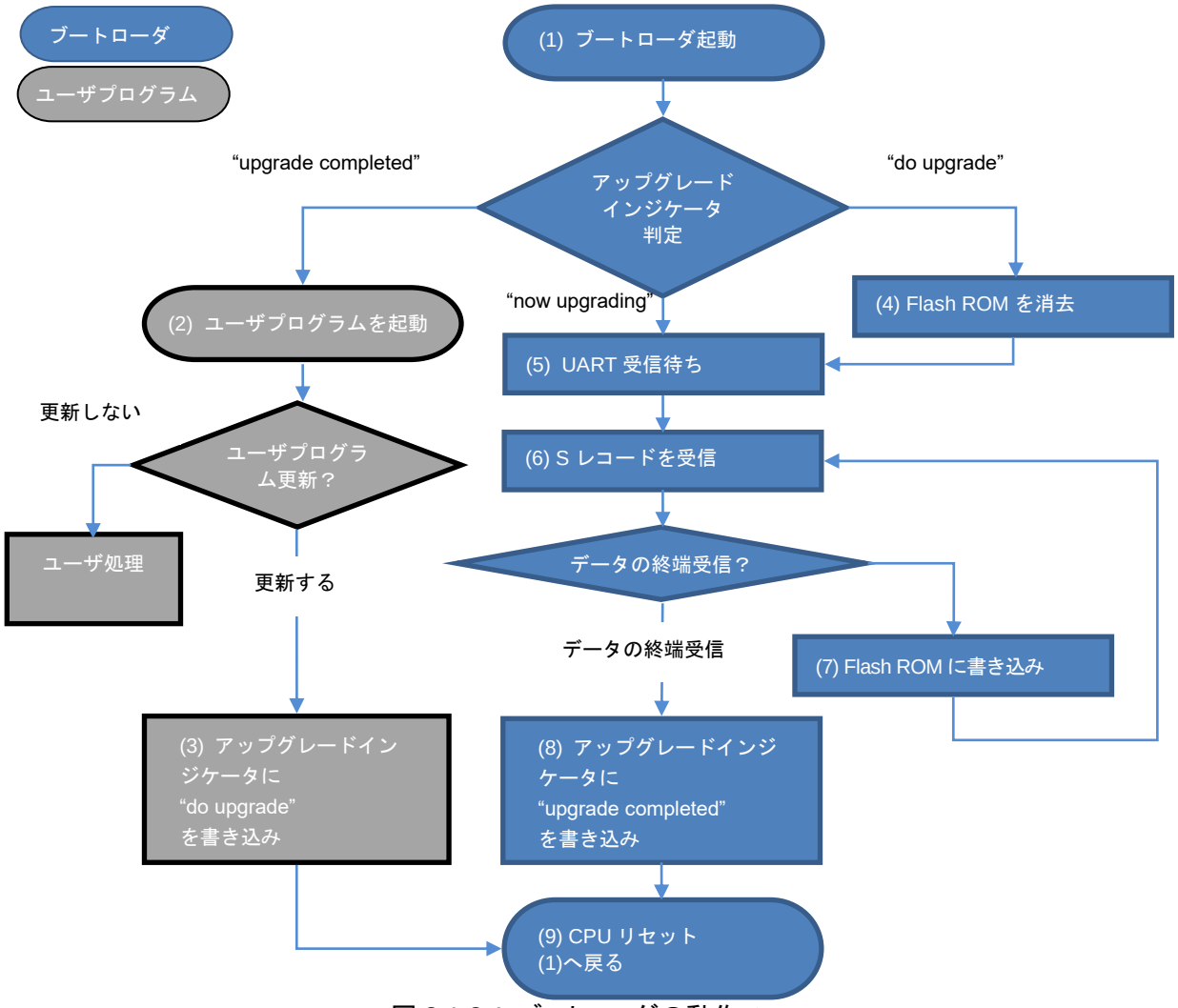

図 3.1.3.1 ブートローダの動作フロー

3. ブートローダの機能

ブートローダをデバイスに実装するときは、以下の点を検討してください。

A) PC と UART 通信する端子はハードウェア構成に合わせて変更してください。

B) 最適なボーレートの設定
ハードウェア構成に合わせて通信エラーのでないボーレートに変更してください。
UART の源振を内蔵発振ではなく水晶振動子に変更することも検討してください。

C) 問題を検出したときにブートローダがどのように動作すべきか。
   i. 通信タイムアウト
   このブートローダの(5)(6)は UART の受信またはタイムアウトをモニタします。しかし、タイムアウトが発生しても、(5)(6)を再実行します。ブートローダは、XOFF を送信した後も、タイムアウト期間（3 ミリ秒）待機します。このタイムアウトは短縮可能です。
   ii. 通信エラー
   UART 通信でエラーが発生した場合、レコードの再送を要求する必要があります。このブートローダの(5)(6)は UART エラーをモニタしませんが、これは、PC 上で動作しているターミナルソフトが再送要求を受け付けないためです。
   iii. データの誤り
   このブートローダは、S レコードのチェックサムを確認し(6)、エラー時には書き込みを行わずに停止します(9)。レコードの再送を要求したほうがよいです。
   iv. Flash ROM の書き込みエラー
   書き込み済みのアドレスに、データを重ね書きする可能性があります。同じ値は再書き込みできます。1 から 0 になる値は再書き込みできますが、0 から 1 になる書き込みはエラーとなります。
   このエラーは、ダウンロードされたデータによって発生します。このエラーが発生した場合、Flash ROM を消去し、もう一度最初からダウンロードする必要があります。

D) ブートローダはリセットされても正常に機能するか。
更新中にリセットされると、このブートローダは(1)から(5)へ遷移し、Flash ROM を消去しなおしません。消去しなおせば正常に動作するはずですが、最初から書き直す必要があります。
ブートローダが再度消去しない場合、以下の確認が必要です。これらの条件は S レコードを受信しただけでは判断できません。これらの条件に当てはまらない場合は、再度消去してください。
   i. 書き込み済みのデータが正しいか。
   ii. 新たに受信したデータと、かつて受信したデータは、同じプログラムか。

E) いつブートローダが Flash ROM を消去すべきか。
アップグレードインジケータが"do upgrade"のとき、このブートローダ(4)はユーザプログラムとアップグレードインジケータを消去します。消去により、アップグレードインジケータの値は、"now upgrading"を意味する 0xFF になります。この後、アップグレードが完了するまで、ブートローダが消去することはありません。
結果として、ブートローダがリセットのたびに Flash ROM を消去することはありません。ただし、間違ったダウンロードなどによりユーザプログラム自体が機能しない場合、アップグレードできない状態になります。
ブートローダが Flash ROM を消去する条件を追加したほうがよいです。

F) ブートローダがロード後(8) にどのように動作すべきか。
このブートローダは(9)でウォッチドッグタイマを起動して、MCU をリセットします。しかし、このとき新しいプログラムを実行することも可能です。

G) PC 上のローダアプリケーション
PC 上に専用のローダアプリケーションを用意すれば、新しいユーザプログラムをスムーズにダウンロードできます。このようなアプリケーションには以下の機能が必要です。
   i. プログラムファイルが正しいか検査する。
   ii. アップグレードプロセスを一時停止する。
   iii. アップグレードプロセスを再開する。
   iv. レコードを再送する。
   v. アップグレードの完了を確認する。

3.2 ユーザプログラム

ユーザプログラム（loadsample）のプロジェクト構成および、動作エリアは以下の通りです。

- loadsample プロジェクト構成 ：
  - board
  - CMSIS
  - sePeripheralLibrary
  - main.c　　ユーザのプログラム

- 動作エリア ：
  Flash ROM　　0x4000 以降に配置してください。

ユーザプログラムを更新するためには、ユーザプログラムにてアップグレードインジケータを“do upgrade”（0x00,0x00,0x00,0x00）に書き換える必要があります。

3.2.1 アップグレードインジケータの書き換え方法

ユーザプログラムは、ユーザプログラムからブートローダで用意しているアップグレードインジケータ更新処理ルーチン（ブートローダのベクタテーブル 47（0x00BC））を呼び出すことで、アップグレードインジケータを書き換え出来ます。そのため、ユーザプログラム内に start_upgrade 関数を用意し、start_upgrade 関数を実行することで、アップグレードインジケータを書き換えます。

start_upgrade 関数の仕様については、Appendix をご参照ください。

■ ユーザプログラムから start_upgrade 関数の呼び出し方法

```
/// call "start_upgrade" function of the boot loader
start_upgrade(0x00BC);
```

ユーザプログラムからブートローダへの切り替えは、start_upgrade 関数を実行後、MCU をリセットしてください。

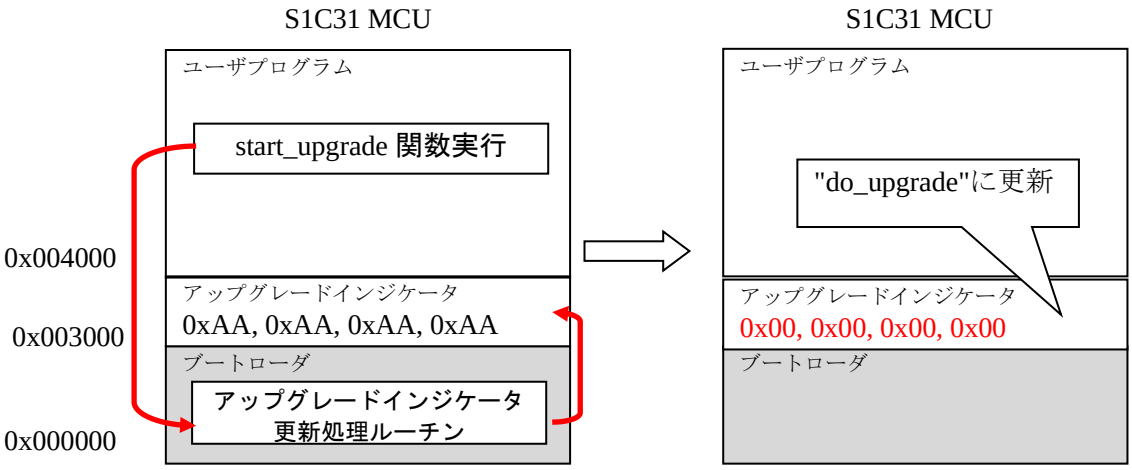

図 3.1.2.1 アップグレードインジケータの書き換え

3.3 ターミナルソフトウェアの設定

新しいユーザプログラムをダウンロードするには、シリアルポートをサポートしたターミナルソフトウェアを使用してください。ここでは、「TeraTerm」を使用した例を示します。

TeraTerm : http://ttssh2.osdn.jp/index.html.en

PC と S5U1C31xxxTx 評価ボードの UART を USB-Serial ケーブル等で接続し、以下の手順に従って TeraTerm を操作してください。

(1) [File]メニューから[New connection]を選択し、[Serial]と使用する[Port]を指定した後、[OK]ボタンをクリックします。
(2) [Setup]メニューから[Serial Port...]を選択します。[Serial port setup]ダイアログボックスが表示されます。
(3) 以下のパラメータを入力します。

Port:	COMx（x：使用するポート番号）
Baud Rate:	230400bps
Data:	8bit
Parity:	none
Stop:	1bit
Flow control:	Xon/Xoff
Transmit delay:	0msec/char, 0msec/line

(4) [OK]ボタンをクリックします。
(5) [File]メニューから[Send File]を選択し、モトローラ S レコード形式のファイルを指定します。「TeraTerm」は新しいプログラムのダウンロードを開始します。

4. メモリの使用

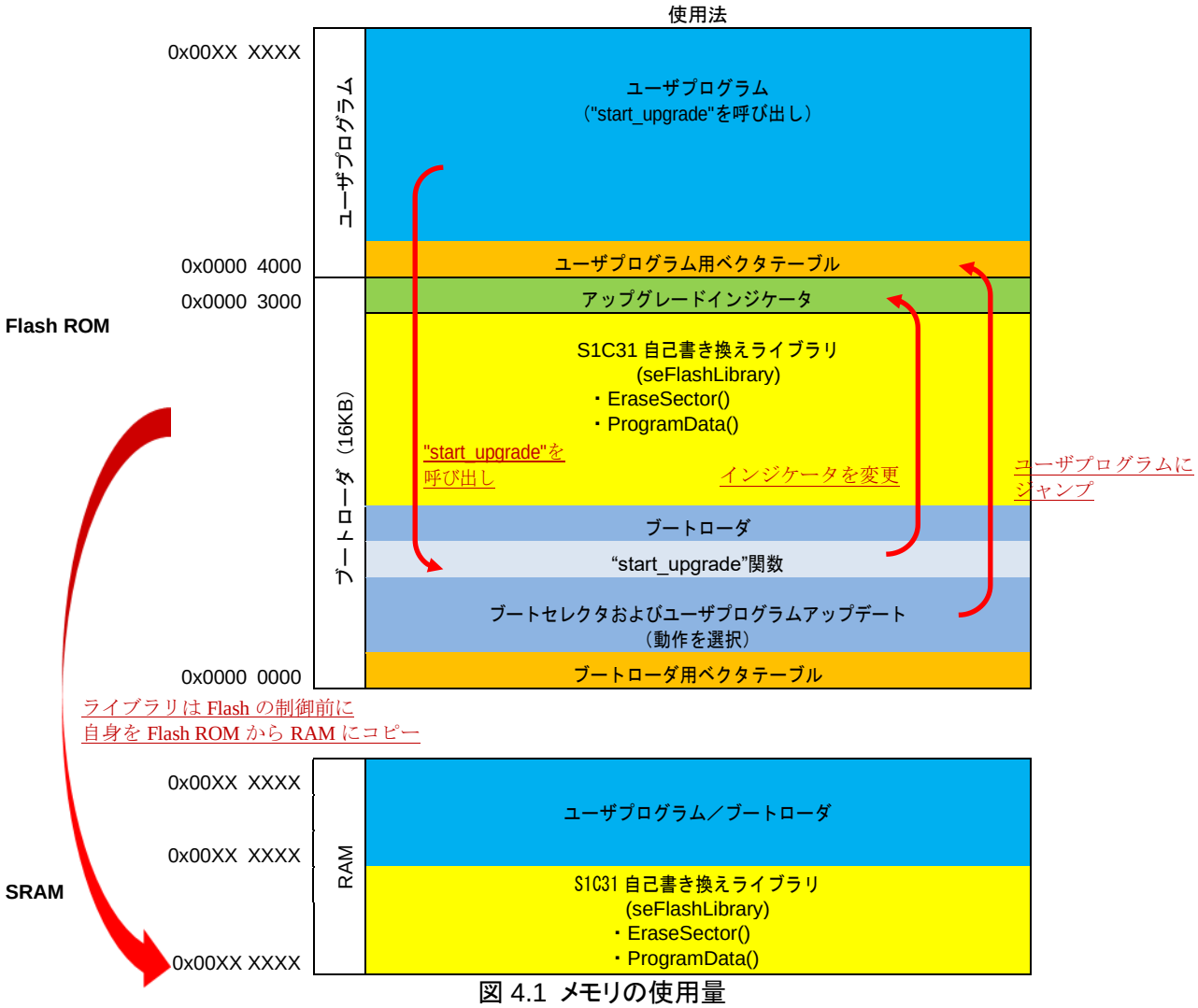

図 4.1 メモリの使用量

上記、0x00XX XXXX については、対象機種のメモリサイズに依存します。メモリサイズについては、“S1C31xxx テクニカルマニュアル”をご参照ください。また、S1C31 自己書き換えライブラリが使用するメモリサイズについては、、S1C31xxx 周辺回路サンプルソフトウェアパッケージに同梱の flashLibraryForS1c31xxx_readme_j.txt をご参照ください。

4.1 S1C31 自己書き換えライブラリの予約 RAM

S1C31 自己書き換えライブラリはブートローダ側に含まれており、ライブラリが使用する RAM はブートローダの一部として定義されます。

ユーザプログラム(ロードされたプログラム)がブートローダの start_upgrade 関数を使用する場合、ブートローダに含まれるフラッシュプログラミングライブラリが動作します。このとき、S1C31 自己書き換えライブラリは作業領域として RAM 領域を使用します。よって、ユーザプログラムの使用する RAM 領域と重ならないように配置してください。

5. デバッグについて

ブートローダおよびユーザプログラムをデバッグする方法について説明します。

indicator.c にアップグレードインジケータの初期値が定義されています。アップグレードインジケータの値によりブートローダの動作が異なります。

■ ブートローダをデバッグする場合

アップグレードインジケータの初期値を“do upgrade”（0x00,0x00,0x00,0x00）に変更してブートローダ（bootloader プロジェクト）をデバッガで書き込んでください。

ブートローダはユーザプログラムを呼び出すことなく、更新モードに入りますので、ソースレベルでのブートローダのデバッグが可能となります。

■ ユーザプログラムをデバッグする場合

(1) アップグレードインジケータの初期値を“upgrade_completed”（0xaa,0xaa,0xaa,0xaa）に変更してbootloader プロジェクトをデバッガで書き込んでください。
(2) ユーザプログラム（loadsampleプロジェクト）をデバッガで書き込みます。この状態でユーザプログラムのソースレベルでのデバッグが可能です。

bootloader プロジェクトは Flash ROM を全消去しない限り、1 度だけ書き込めばよいです。
以降必要に応じて、ユーザプログラムの修正、デバッガ書き込みを行ってください。

Appendix. 関数仕様

ブートローダおよびユーザプログラムが提供する関数の詳細について記します。

■ ユーザプログラム

関数名		
void start_upgrade(uint32_t app_link_location)		
引数		
uint32_t	app_link_location	通常は 0x00bc を設定
戻り値		
none		
機能		
ブートローダのベクタテーブル 47（0x00BC）に格納されたアップグレードインジケータ更新処理ルーチンを呼び出し、アップグレードインジケータを(0x00,0x00,0x00,0x00)に書き換えます。 ① 割り込みの禁止 ② ベクタテーブル 47（0x00BC）に定義されたアドレスの関数を呼び出します。		
備考		

■ ブートローダ

関数名
void Bootloader_IRQHandler(void)
機能
アップグレードインジケータ更新処理ルーチン アップグレードインジケータを“do upgrade”（0x00,0x00,0x00,0x00）へ書き換えます。 ① 割り込みを禁止します。 ② 自己書き換えライブラリにて 0x3000 番地に 0x00,0x00,0x00,0x00 を書き込みます。
備考
本関数のアドレスはベクタテーブル 47（0x00BC）に定義されます。 ユーザプログラムより呼び出されます。

関数名
int main(void)
機能
ブートローダのメインルーチンです。 ① WDT 停止 ② アップグレードインジケータの判定により以下のいずれかの処理を実行します。 ユーザープログラム（0x4000）へジャンプ Flash のユーザ領域の消去 UART 受信処理・Flash のデータ書き込み処理

関数名		
void start_application(uint32_t app_link_location)		
引数		
uint32_t	app_link_location	ユーザプログラムアドレス(0x4000)
機能		
ユーザプログラム（0x4000）へジャンプします。 ① オフセットアドレス(VTOR)を 0x4000 ② ユーザプログラム（0x4000）へジャンプ		

関数名
int update(void)
機能
UART 受信待ちをし、受信データを解析し、Flash への書き込みを実施します。 ① UART の設定 ② UART 受信待ち ③ 受信データの解析 ④ Flash へのデータ書き込み

関数名		
void ConfigureDebugUART3_0(seUART_InitTypeDef* InitStruct)		
引数		
seUART_InitTypeDef*	InitStruct	UART の設定値の構造体
戻り値		
none		
機能		
UART の初期設定、使用ポートの設定を行います。 ① UART で使用するポート設定 ② UART 初期設定		
備考		
UART のポートを変更する場合に修正してください。		

関数名		
int get_record(void)		
戻り値		
EXIT_SUCCESS	改行コード受信あり（S レコードの 1 行分の取得した場合）	
EXIT_FAILURE	改行コード受信なし（S レコードの 1 行分の取得しない場合）	
機能		

UART の受信を監視し、S レコードの 1 行分のデータを取得します。

① XON コードを送信して PC ターミナルへ送信を許可します。

② UART 受信データをバッファへ保存し、改行コードを受信したところで、XOFF コードを送信して PC ターミナルへ送信を禁止します。

③ 改行コードを受信した場合は EXIT_SUCCESS を返します。

関数名
void data_copy(void)
機能

UART の受信バッファの管理を行います。

① S レコードの 1 行分に満たない半端データサイズを取得

② 半端データをバッファの先頭へコピー

関数名		
int memory_write(struct srec_decode_t * srec)		
引数		
struct srec_decode_t *	srec	S レコードデータ
戻り値		
ARM_DRIVER_OK	Flash／SRAM への書き込み成功	
ARM_DRIVER_ERROR	Flash／SRAM への書き込み失敗	
機能		

受信した S レコードデータを Flash／SRAM へ書き込みをします。

① S レコードのアドレスより、書き込み先が Flash/SRAM かの判定を行います。

② Flash の場合、自己書き換えライブラリを使用し Flash へデータを書き込みます。

③ SRAM の場合、データを SRAM へ書き込みます。

関数名
void start_wdt(void)
機能
リセット用に WDT を起動させます。

関数名
void stop_wdt(void)
機能
リセット用に WDT を停止させます。

関数名		
enum srec_type_t srec_decode_record(struct srec_decode_t * srec, char * record)		
引数		
struct srec_decode_t * char *	srec record	デコードデータ保存用バッファアドレス S レコードデータ
戻り値		
srec_type_t	S レコードのタイプ SREC_TYPE_S0~S9、SREC_TYPE_UNKNOWN, SREC_TYPE_ERROR_CHECKSUM, SREC_TYPE_ERROR_LENGTH	
機能		
S レコードのデコードを行い、タイプ、アドレス、データ、チェックサムに変換します。 ① S レコードのタイプの取得 ② S レコードのアドレスの取得 ③ S レコードのデータの取得 ④ S レコードのチェックサムの取得および判定		
備考		
PC より受信したデータの正当性のチェックを行い、Flash へ書き込み可能なデータ形式へ変換します。		

関数名		
void sbytes_decode_schars(unsigned char * sbytes, char * schars, int length)		
引数		
unsigned char * char * int	sbyte schars length	デコードデータ保存用バッファアドレス ASCII データ データ長
機能		
ASCII データ列を数値変換し、デコードデータ保存用バッファに保存します。		

関数名		
unsigned char sbytes_checksum(unsigned char * sbytes, int length, unsigned char sum)		
引数		
unsigned char * int unsigned char	sbyte length sum	デコードデータ保存用バッファアドレス データ長 チェックサム
戻り値		
unsigned char	チェックサム	
機能		
デコードデータのチェックサムを計算し返します。		

関数名		
unsigned long sbytes_get_XXbit(unsigned char * sbytes)		
引数		
unsigned long	sbyte	アドレスデータ保存用バッファアドレス
戻り値		
unsigned long	S レコードのアドレス	
機能		
ASCII 形式のアドレス(32Bit,24Bit,16Bit)を数値に変化し返します。 関数名の XX：32,24,16		

関数名		
unsigned char schars_to_sbyte(char * schars)		
引数		
char *	schars	ASCII コード
戻り値		
unsigned char	数値	
機能		
ASCII 形式のデータを数値に変化し返します。		

改訂履歴表

付－1

Rev. No.	日付	ページ	種別	改訂内容（旧内容を含む） および改訂理由
Rev.1.00	2021/04/12	全ページ	新規	新規制定

セイコーエプソン株式会社

営業本部　デバイス営業部

東京　〒160-8801 東京都新宿区新宿 4-1-6 JR 新宿ミライナタワー29F

大阪　〒530-6122　大阪市北区中之島 3-3-23　中之島ダイビル 22F

ドキュメントコード：414177700

2021 年 4 月 作成