AN0003：UART ブートローダ

このアプリケーション・ノートは、EFM32 UART ブートローダのユーザを対象としています。このブートローダを使用すると、UART を通じて EFM32 シリーズ 0、EZR32 シリーズ 0、および EFM32 シリーズ 1 デバイスをプログラムできるので、デバッガは必要ありません。

EFM32/EZR32 ブートローダには、ユーザ・アプリケーションの起動に加え、フラッシュ領域全体をユーザ・アプリケーションに使用できるようにブートローダを上書きできる、破壊的書き込みモードが用意されています。EFM32 シリーズ 1 デバイスの場合、ブートローダは、フラッシュ・メモリの予約済み領域に常駐するので、これらのデバイスのブートローダには破壊的書き込みモードは用意されていません。ただし、ロック・ビット・ページでビットを設定して、EFM32 シリーズ 1 デバイスでブートローダを無効にすることができます。

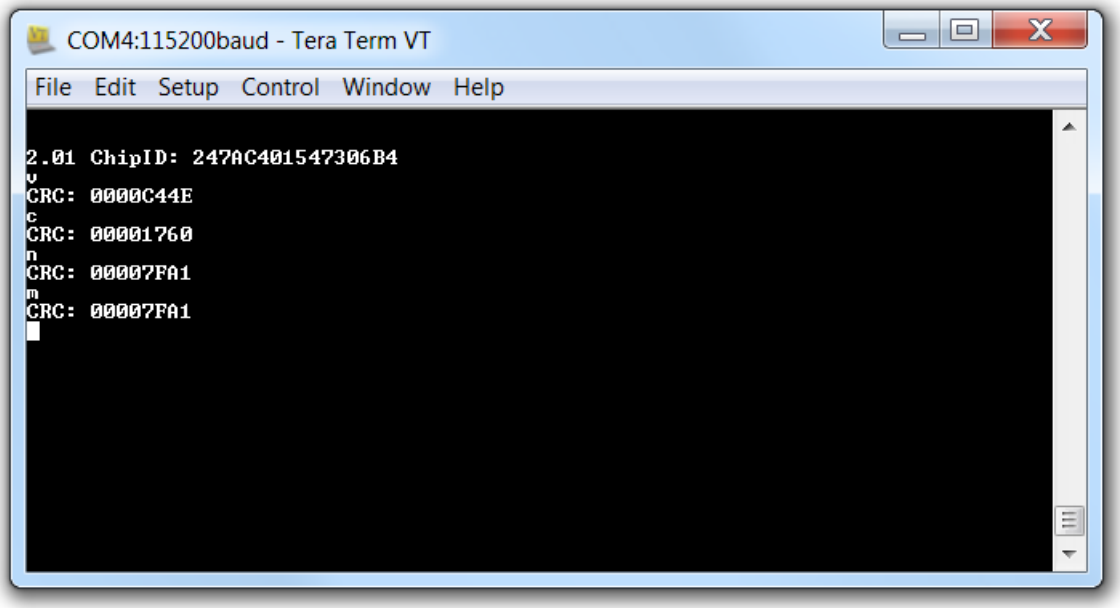
どのデバイスでも、CRC チェックサムを通じてフラッシュの内容を検証でき、IP を保護するためのデバッグ・ロックを有効にすることができます。このブートローダは既成の XMODEM-CRC プロトコルを使用してデータ・アップロードを行うので、ブートローダとの通信には、どのシリアル・ターミナル・プログラムでも使用できます。

このアプリケーションノートには、以下が含まれています。

- この PDF 文書
- ソースファイル（zip）
  - 完全なブートローダ・ソースコード
  - IAR EW プロジェクトファイル
- アプリケーション用 IAR リンカファイル
- ブートローダのバイナリイメージ

要点

- EFM32 デバイスにはすべて、以下のように動作するブートローダが事前にプログラムされています。
  - ユーザ・アプリケーションと共存可能で、フィールド・アップグレードに対応。
  - 利用可能なフラッシュ領域を最大限活用できるように上書きが可能（EFM32 シリーズ 0 および EZR32 シリーズ 0 のみ）、またはロック・ビット・ページでロック・ビットを設定して無効化が可能（EFM32 シリーズ 1 のみ）。
  - UART インターフェイスを使用して通信（部品別のペリフェラルの構成とサポートされているボーレートについては、セクション 1.2 を参照）。
  - EFM32 デバイスのプログラム（アップロード/上書き）、検証（チェックサムの計算）、保護（書き込み禁止、デバッグ・ポートのロック）などを実行する 1 文字コマンドをサポート。
  - XMODEM-CRC プロトコルを使用したファイル転送が可能。
  - DBG_SWCLK を High に設定するとリセット後に起動。



第 1 章 デバイスの互換性

このアプリケーション・ノートは複数のデバイス・ファミリを対象にしています。一部の機能はデバイスによって異なります。

UART ブートローダは、すべての EFM32 シリーズ 0、すべての EZR32 シリーズ 0、一部の EFM32 シリーズ 1 デバイスに事前にプログラムされています。USB 対応の EFM32 シリーズ 0 および EZR32 シリーズ 0 デバイスには、USB ブートローダも含まれています。USB 対応デバイスのブートローダ については、アプリケーションノートに記載されています。*AN0042：USB ブートローダ*、Simplicity Studio（www.silabs.com/simplicity）、または Silicon Labs のウェブサイト www.silabs.com からご利用いただけます.

以下は、サポートされているデバイスのリストです。

EFM32 シリーズ 0：

- EFM32 Gecko (EFM32G)
- EFM32 Giant Gecko (EFM32GG)
- EFM32 Wonder Gecko (EFM32WG)
- EFM32 Leopard Gecko (EFM32LG)
- EFM32 Tiny Gecko (EFM32TG)
- EFM32 Zero Gecko (EFM32ZG)
- EFM32 Happy Gecko (EFM32HG)

EZR32 シリーズ 0：

- EZR32 Wonder Gecko (EZR32WG)
- EZR32 Leopard Gecko (EZR32LG)
- EZR32 Happy Gecko (EZR32HG)

EFM32 シリーズ 1:

- EFM32 Pearl Gecko（EFM32PG1/EFM32PG12 Rev C 以降）
- EFM32 Jade Gecko (EFM32JG1/EFM32JG12 Rev C 以降)
- EFM32 Giant Gecko GG11 (EFM32GG11)
- EFM32 Giant Gecko GG12 (EFM32GG12)
- EFM32 Tiny Gecko 11 (EFM32TG11)

Note: IOVDD へは、EFM32xG11/12 デバイス上の DC-DC コンバータから給電しないでください。リセット時に DC-DC コンバータはデフォルトで 安全状態が未構成となり、出力がフロートされるため、 ファームウェアで必要な構成が行われるまで接続回路の給電が行われない状態が続きます。 工場出荷時のブランク・デバイスはブートローダを実行しますが、 IOVDD 電力がないため、BOOT_RX ピンと BOOT_TX ピンを介したホストとの通信に失敗します。 この場合、ファームウェア の初回ダウンロードにデバッグ・インターフェイス（DBG_SWCLKTCK および DBG_SWDIOTMS）を使用しても、同様に失敗します。

Note: このブートローダ・ソフトウェアは、EFM32 シリーズ 1

EFR32 シリーズ 1 (EFR32xG1/EFR32xG12/EFR32xG13/EFR32xG14) デバイスは、この ブートローダではサポートされていません。これらのデバイス向けのブートローダは、プロトコル・スタック・ソフトウェアに含まれています。 これらのデバイスには、以下が含まれます。

- EFR32 Blue Gecko (EFR32BGxx)
- EFR32 Flex Gecko (EFR32FGxx)
- EFR32 Mighty Gecko (EFR32MGxx)

第 2 章 UART ブートローダの起動

2.1 ブートローダ・モードへの切り替え

ブートローダ・モードに切り替えるには、DBG_SWCLK を High に設定して EFM32 シリーズ 0、EZR32 シリーズ 0、または EFM32 シリーズ 1 デバイスをリセットする必要があります。DBG_SWCLK がリセットの際に Low になっていると、ブートローダはフラッシュ内のアプリケーション領域をチェックします。アプリケーション領域に有効なアプリケーションがあり、DBG_SWCLK が Low である場合、ブートローダはそのアプリケーションを実行します。有効なアプリケーションがない場合、ブートローダは電力を節約するために EM2 のスリープ・モードになり、ブートローダ・ピンを定期的にチェックします。

Note: DBG_SWCLK には内部プルダウンがあります。このピンを接続しないと、リセット時にブートローダ・モードが起動しません。

Note: 旧バージョンのブートローダでは、ブートローダへの切り替えに DBG_SWDIO ピンと DBG_SWCLK ピンをどちらも使用していました。現在も DBG_SWDIO ラインを Low に設定するとブートローダに切り替えることができますが、デバッグ・ロックは機能しなくなります。

2.2 UART ブートローダとの通信の初期化

EFM32 シリーズ 0 および EZR32 シリーズ 0 デバイスの場合、UART ブートローダは一般的に、UART 通信に GPIO ピン PE11 (RX) と PE10 (TX) を使用します（例外については、下記の「注」を参照）。ピンの位置については、デバイスのデータシートを参照してください。

Note: EFM32G、EFM32GG、EFM32LG デバイスとほとんどの EFM32TG デバイスでは、UART ブートローダは、位置 0 の USART0 を使用して通信します。ただし、デバイスによっては USART0 ペリフェラルのないものや、USART0 ペリファラルがあっても位置 0 オプションのないものがあります。このような理由から、EFM32ZG、EFM32HG、および一部の EFM32TG（具体的には EFM32TG108Fxx と EFM32TG110Fxx）デバイスのブートローダはいずれも、位置 3 の LEUART0 を使用します。この位置は、標準 SWD ポートと共有され、前述のようにブートローダへの切り替えに使用されます。したがって、これらのデバイスを使用する場合は、DBG_SWCLK で 4 kΩ プルアップを使用してください。EFM32 シリーズ 1 デバイスの場合、ブートローダは F0 (DBG_SWCLK) ポートと F1 (DBG_SWDIO) ポートで USART 接続を使用します。また、これらのデバイスについても、DBG_SWCLK で 4 kΩ プルアップを使用してください。

UART では、1 ストップ・ビット、パリティなし、8 データ・ビットを使用します。また、この ブートローダは、オートボー機能を使用してさまざまな種類の端末に対応します。オートボー 機能は、ターミナル・プログラムが使用しているボーレートを検出し、必要に応じて調整します。この 初期化処理は、デバイスとの シリアル接続が確立された直後に大文字 "U" 1 文字をブートローダに送信することで行われます。 ブートローダは、ビット間のタイミングを検出し、検出されたボーレートに合せて 自己のプリスケーラを調整します。AN0003 ソフトウェアパッケージで [**an0003_efm32_uart_bootloader**]>[**バイナリ**]>[**notes.txt**] に移動し（AN003 ソフトウェアは https://www.silabs.com/support/resources から入手可能）、自動ボーアルゴリズムと 互換性のあるボーレートの部品固有範囲を入手できます。

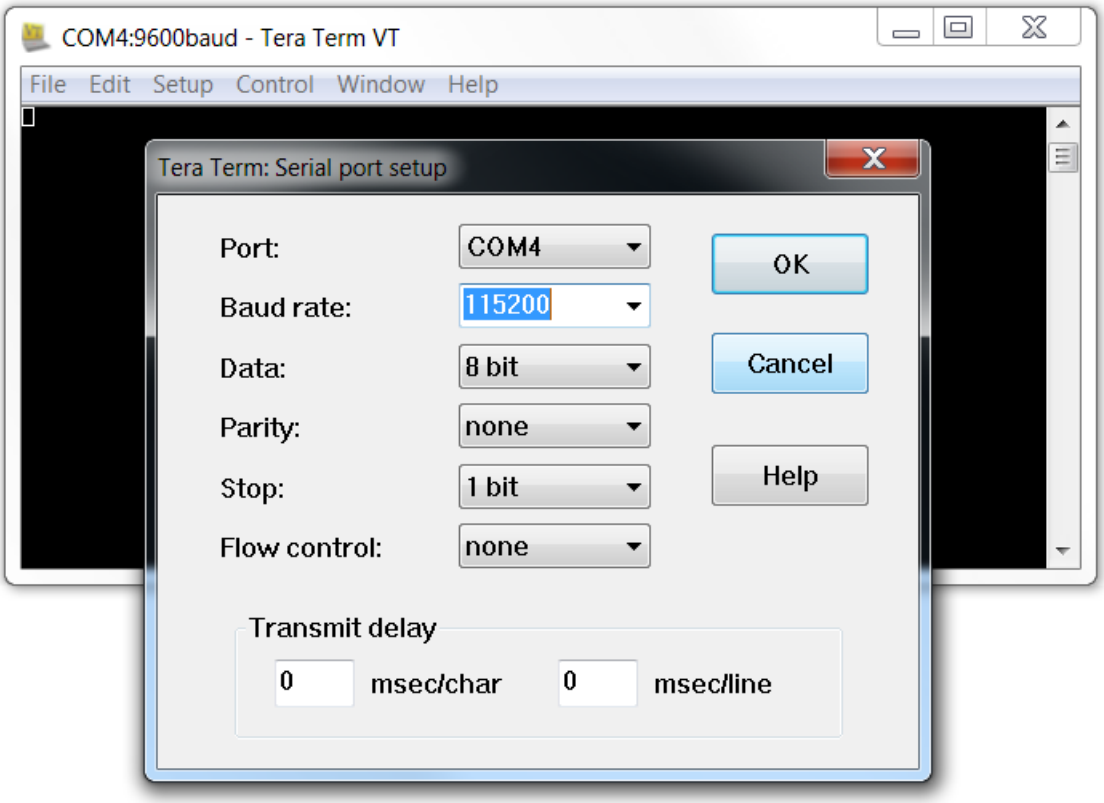


図 2.1. Windows 7 で Tera Term を使用して UART ブートローダのシリアル・ポートを構成

ブートローダの初期化が成功すると、ブートローダのバージョンとチップの固有 ID が出力されます。

```
1.40 ChipID: F08AB6000B153525
```

Note: 1.40 より前のバージョンのブートローダでは、ブートローダのバージョンが出力されず、チップの固有 ID のみ出力されます。

Note: AN0003 ソフトウェアパッケージで [**an0003_efm32_uart_bootloader**]>[**バイナリ**]>[**bootloader-matrix.txt**] に移動し （AN003 ソフトウェアは https://www.silabs.com/support/resources から入手可能）、ファミリーごとの最新バージョンのブートローダを入手できます。

2.3 コマンドライン・インターフェイス

コマンドライン・インターフェイスでは、1 文字のコマンドを使用します。コマンドライン インターフェイスでサポート される小文字 1 文字の文字コマンドは次のとおりです。

u

アプリケーションのアップロード。

EFM32 シリーズ 0 および EZR32 シリーズ 0：このコマンドを使用すると、ブートローダをそのままにして、フラッシュにアプリケーションをアップロードできます。アプリケーションが正しく機能するには、EFM32G、EFM32TG、EFM32ZG、および EFM32HG デバイスでは 0x800 に、EFM32GG、EFM32LG、および EFM32WG デバイスでは 0x1000 にアプリケーションの開始アドレスを配置するリンカ・ファイルを使用する必要があります。アプリケーションは、XMODEM-CRC プロトコルを使用して転送されます。

EFM32 シリーズ 1： このコマンドを使用すると、フラッシュにアプリケーションをアップロードできます。ブートローダは予約済みのフラッシュ領域に常駐し、上書きできないため、ユーザ・アプリケーションを特に修正する必要はありません。アプリケーションは、XMODEM-CRC プロトコルを使用して転送されます。

d

破壊的アップロード（EFM32 シリーズ 0 および EZR32 シリーズ 0 のみ）。このコマンドを実行すると、ブートローダを上書きして、フラッシュにアプリケーションをアップロードできます。ユーザ・アプリケーションの開始アドレスの変更は不要です。アプリケーションは、XMODEM-CRC プロトコルを使用して転送されます。EFM32 シリーズ 1 デバイスでは、破壊的アップロードは実行できません。

t

ユーザ・ページへのアップロード。このコマンドを実行すると、ユーザ情報ページに書き込むことができます。データは、XMODEM-CRC プロトコルを使用してアップロードされます。

p

ロック・ページへのアップロード。このコマンドを実行すると、ロック・ビット情報ページに書き込むことができます。データは、XMODEM-CRC プロトコルを使用してアップロードされます。

b

アプリケーションの起動。このコマンドは、アップロードしたアプリケーションを起動します。

l

デバッグ・ロック。このコマンドは、ロック・ページでデバッグ・ロック・ビットを設定します。EFM32 シリーズ 0、EZR32 シリーズ 0、または EFM32 シリーズ 1 はデバッグのためにロックされます。

v

フラッシュ・チェックサムの検証。このコマンドは、フラッシュ全体の CRC-16 チェックサムを計算して出力します。このコマンドは、[**d**] コマンド（EFM32 シリーズ 0 および EZR32 シリーズ 0）または [**u**] コマンド（EFM32 シリーズ 1）との併用に適しています。EFM32 シリーズ 1 デバイスでは、[**v**] コマンドと [**c**] コマンドで同じ結果が生成されることに注意してください。

c

アプリケーション・チェックサムの検証。このコマンドは、アプリケーションの CRC-16 チェックサムを計算して出力します。[**u**] コマンドとの併用に適しています。
EFM32 シリーズ 1 デバイスでは、[**c**] コマンドと [**v**] コマンドで同じ結果が生成されることに注意してください。

n

ユーザ・ページ・チェックサムの検証。このコマンドは、ユーザ・ページの CRC-16 チェックサムを計算して出力します。[**t**] コマンドとの併用に適しています。

m

ロック・ページ・チェックサムの検証。このコマンドは、ロック・ページの CRC-16 チェックサムを計算して出力します。[**p**] コマンドとの併用に適しています。

r

EFM32 シリーズ 0、EZR32 シリーズ 0、または EFM32 シリーズ 1 デバイスのリセット。

第 3 章 アプリケーションのアップロード

アプリケーションをアップロードするには EFM32 シリーズ 0、EZR32 シリーズ 0、および EFM32 シリーズ 1、[**u**] （すべてのデバイス）または [**d**] （EFM32 シリーズ 0 および EZR32 シリーズ 0 のみ) コマンドを使用する必要があります。ファイルを転送するには、キーを押した後、ターミナル・ソフトウェアの 組み込み XMODEM-CRC サポートを使用します。ターミナル・ソフトウェアは、XMODEM-CRC 転送をサポートしていれば、 どれでも使用できます。

Tera Term の XMODEM-CRC を介してファイルを送信するには、[**ファイル**]>[**転送**]>[**XMODEM**]>[**送信…**]に移動します。以下の図に、Tera Term で 組み込み転送サポートを使用したファイル転送の例を示します。

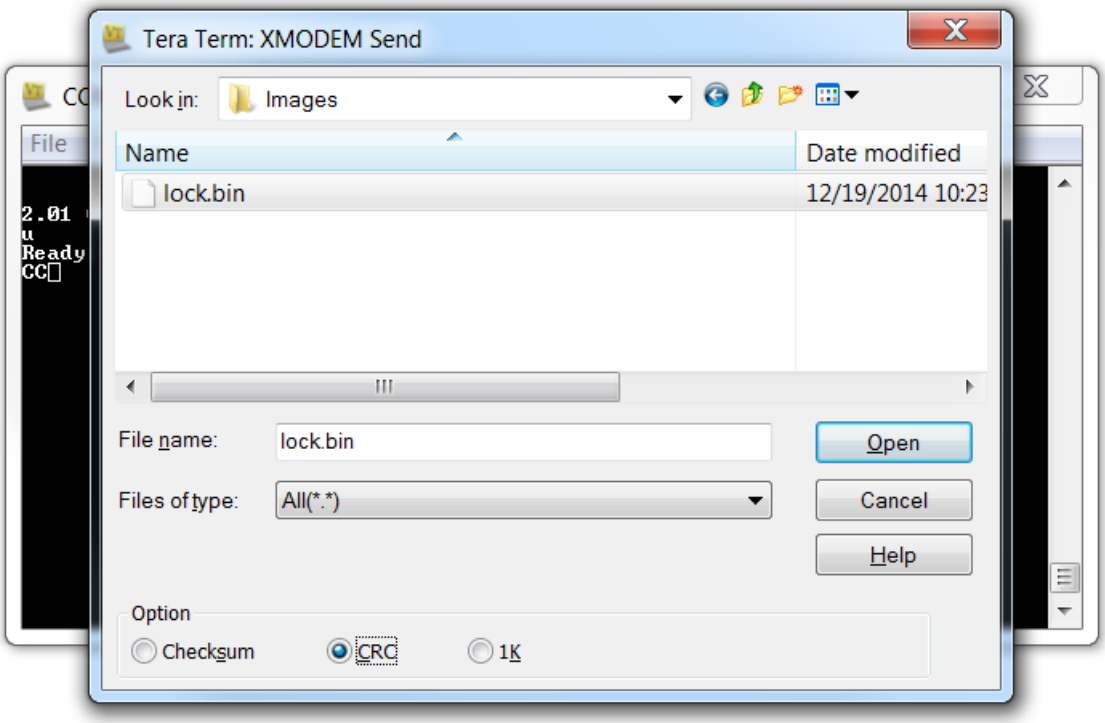


図 3.1. XMODEM-CRC と Tera Term を使用したファイルの転送

3.1 ブートローダで使用するアプリケーションの作成 - EFM32 シリーズ 0 および EZR32 シリーズ 0

Note: EFM32 シリーズ 1 デバイスで使用するアプリケーションの作成方法については、「3.2 ブートローダで使用するアプリケーションの作成 - EFM32 シリーズ 1」を参照してください。

EFM32 シリーズ 0 および EZR32 シリーズ 0 デバイス用のブートローダを使用してアプリケーションをアップロードする方法には、破壊的アップロードと標準アップロードの 2 つがあります。破壊的アップロードではブートローダが上書きされます。この場合、アプリケーションを作成する追加手順は必要ありません。標準アップロードではブートローダがそのまま維持されます。そのため今後もこのブートローダを使用してアップグレードできます。ただし、そのためにはアプリケーションの準備が必要になります。アプリケーションをブートローダと共に使用するには、EFM32G、EFM32TG、EFM32ZG、および EFM32HG デバイスでは 0x800 に、EFM32GG、EFM32LG、および EFM32WG デバイスでは 0x1000 に開始アドレスを設定してアプリケーションを作成する必要があります。その理由は、ブートローダ自体が 0x0 ～ 0x7FF 間または 0x0 ～ 0x0FFF 間のフラッシュ領域を占有するためです。アプリケーションとブートローダを共存させるには、アプリケーション・リンカ・ファイルのデフォルトのフラッシュ開始アドレス 0x0 を変更する必要があります。

Note:

これらのリンカ・ファイルのいずれかを使用していてアプリケーションをデバッグする場合は、コード内でベクタ・テーブルの位置を明示的に設定する必要があります。例を以下に示します。

```
SCB->VTOR=0x800; // EFM32G, EFM32TG, EFM32ZG and EFM32HG parts
```

または

```
SCB->VTOR=0x1000; // EFM32GG, EFM32LG and EFM32WG parts
```

リリースした後のアプリケーションについては、アプリケーションを起動する前にブートローダによって VTOR が設定されるので、この作業は不要です。詳細については、「Boot.c」を参照してください。

3.1.1 IAR を使用したアプリケーションの作成

IAR を使用してアプリケーションを作成するには、AN0003 ソフトウェアパッケージで [**an0003_efm32_usb_uart_bootloader**]>[**iar_linker_files**] に移動し（AN0003 ソフトウェアパッケージは https://www.silabs.com/support/resources で入手可能）、プロジェクト用の部品固有リンカファイルを使用します。これによって、バイナリの正しい開始アドレスが設定されます。もう 1 つのオプションは、IAR プロジェクトを右クリックして、[**オプション…**]を選択します。[**リンカー**] タブに移動して [**デフォルトをオーバーライド**] オプションがチェックされていることを確認します。

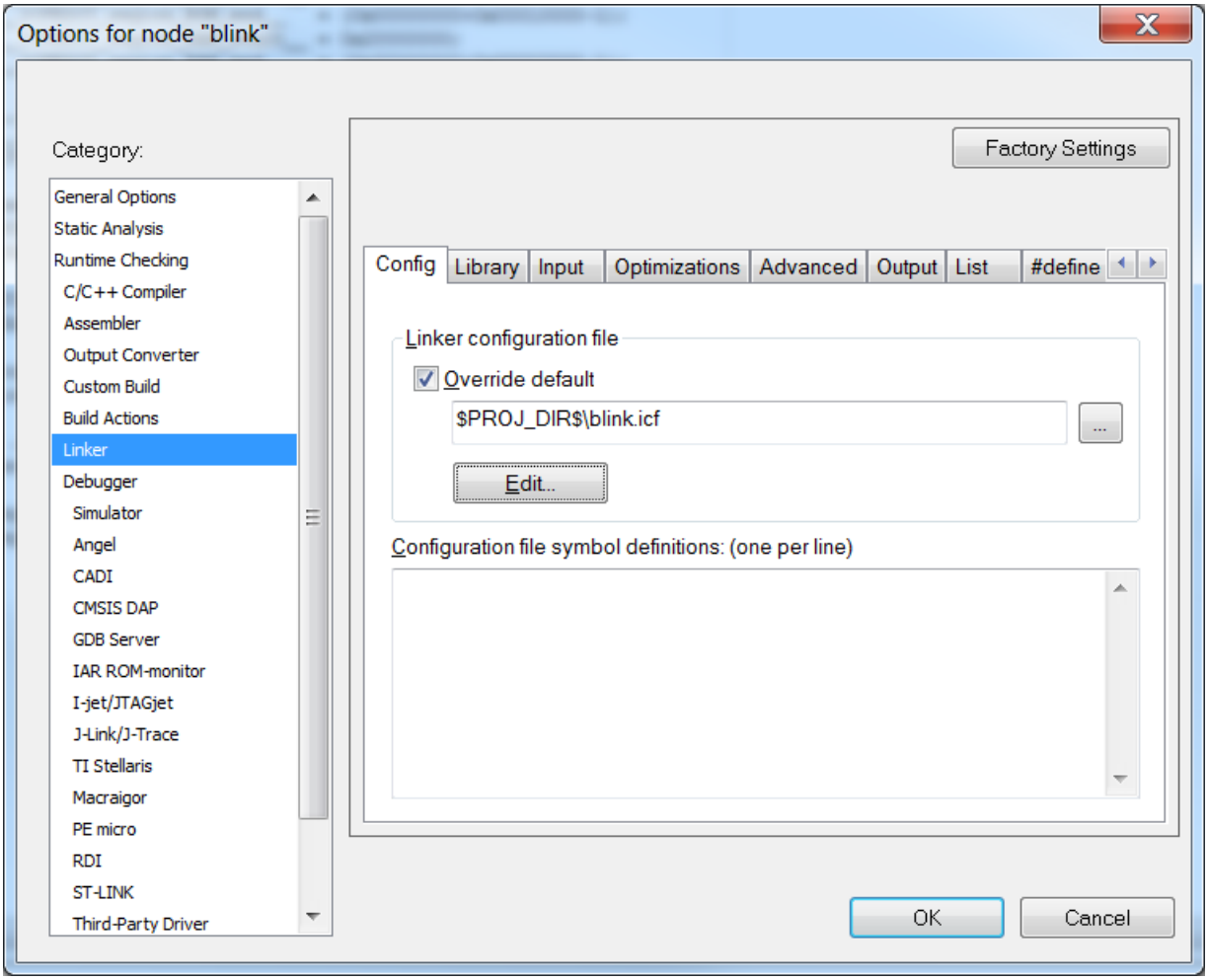


図 3.2. IAR プロジェクトオプション

プロジェクトディレクトリ内に.icf ファイルを作成し、[**編集**]をクリックします。ベクタ テーブルの開始アドレスを [**ベクタテーブル**] タブで [**0x0800**] または [**0x1000**]（部品によって異なる）に設定します。ROM の開始アドレスを [**メモリ領域**] タブで [**0x0800**] または [**0x1000**]（部品によって異なる）に設定します。終了したら [**保存**] をクリックします。終了したら [**OK**] を [**オプション**] ウィンドウでクリックします。

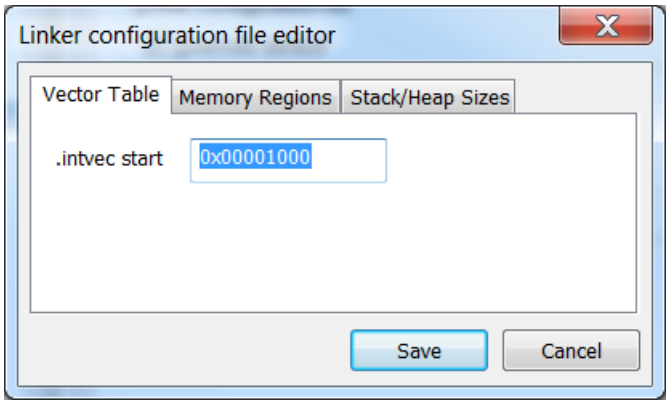


図 3.3. ベクタテーブルアドレスの変更

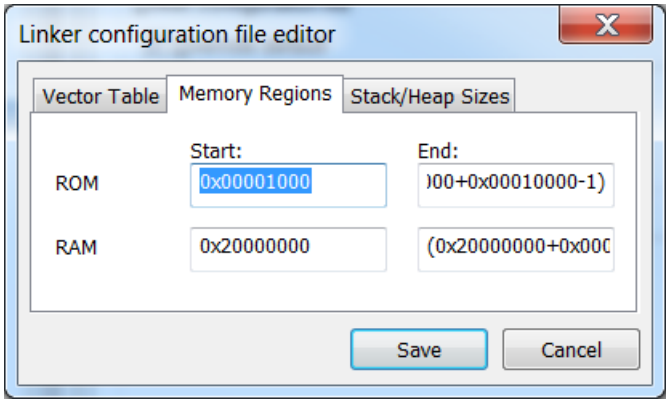


図 3.4. ROM 開始アドレスの変更

プロジェクトオプションメニューで、[**出力コンバータ**] および [**追加出力を生成**]を選択します。 [**バイナリ**] 出力形式を選択します。 生成されたバイナリは UART ブートローダで使用することができます。

3.1.2 Keil uVision 4/MDK-ARM を使用したアプリケーションの作成

Keil uVision 4/MDK-ARM を使用してアプリケーションを作成するには、まず、プロジェクトのターゲット設定を変更する必要があります。[Options（オプション）] ダイアログで、IROM1 の [Start（開始）] フィールドを 0x800 または 0x1000 に変更し、[Size（サイズ）] フィールドから 0x800 または 0x1000 を差し引きます （0x800 と 0x1000 のどちらに設定するかは、使用するデバイスによって異なります）。

バイナリ出力ファイルを生成するには、コマンドライン・ユーティリティ「fromelf.exe」を使用できます。このユーティリティは、通常、C:¥Keil¥ARM¥BIN40¥fromelf.exe にインストールされています。詳細については、uVision ヘルプの『*Realview Utilities Guide*』を参照してください。

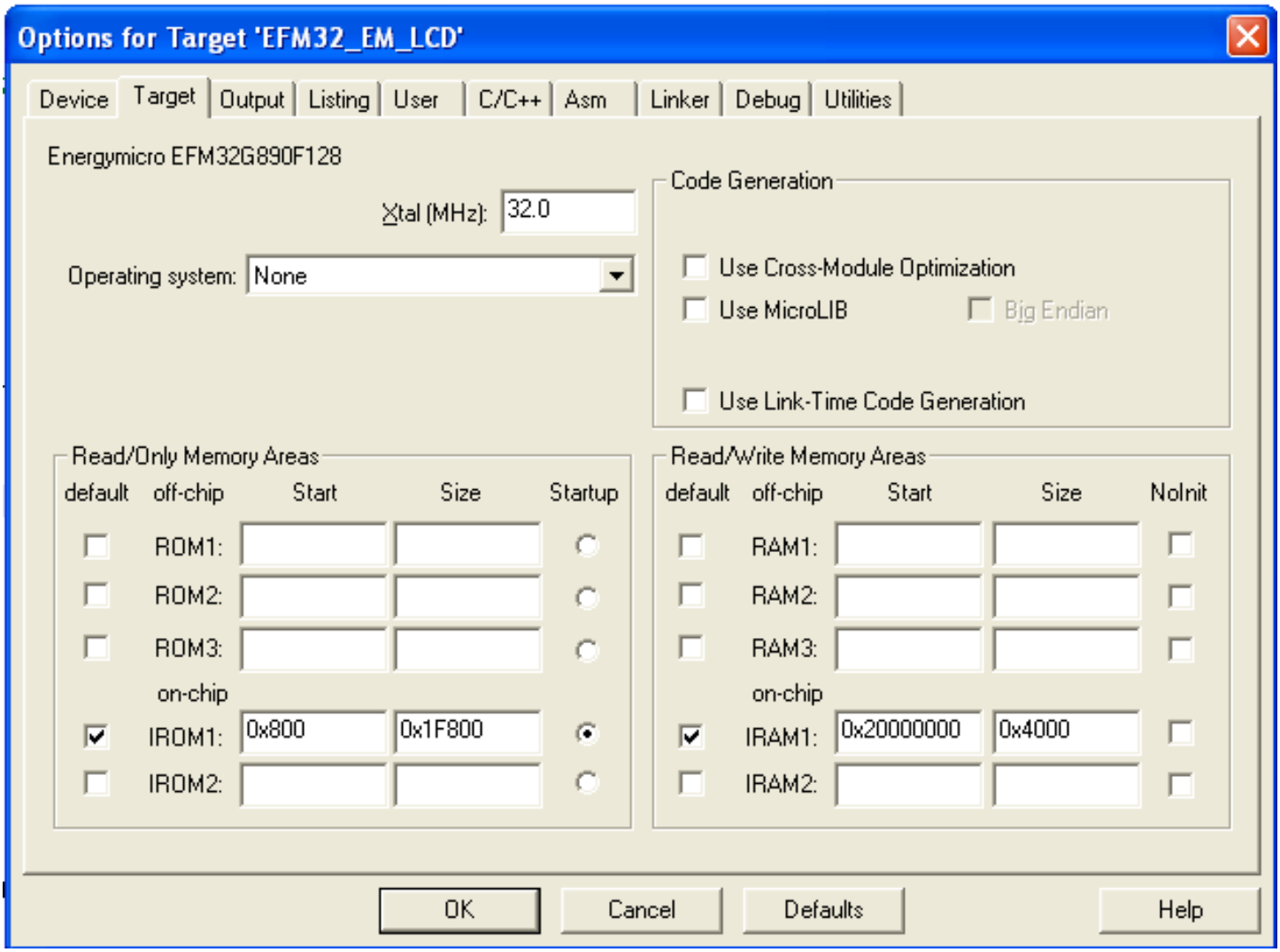


図 3.5. Keil uVision 4/MDK-ARM の設定

3.1.3 Eclipse/GCC/Sourcery CodeBench を使用したアプリケーションの作成

Eclipse、GCC、または Sourcery CodeBench を使用して、ブートローダと連動するアプリケーションを作成するには、リンカファイルを修正する必要があります。アプリケーション・ノートとサンプル・プロジェクトのリンカファイルの場所は、ソフトウェア・プロジェクトに付属のメイクファイル内で指定されています。リンカファイル MEMORY コマンドで、ROM ORIGIN を 0x00000800 または 0x00001000 に変更します。長さも以下の図のように変更します。

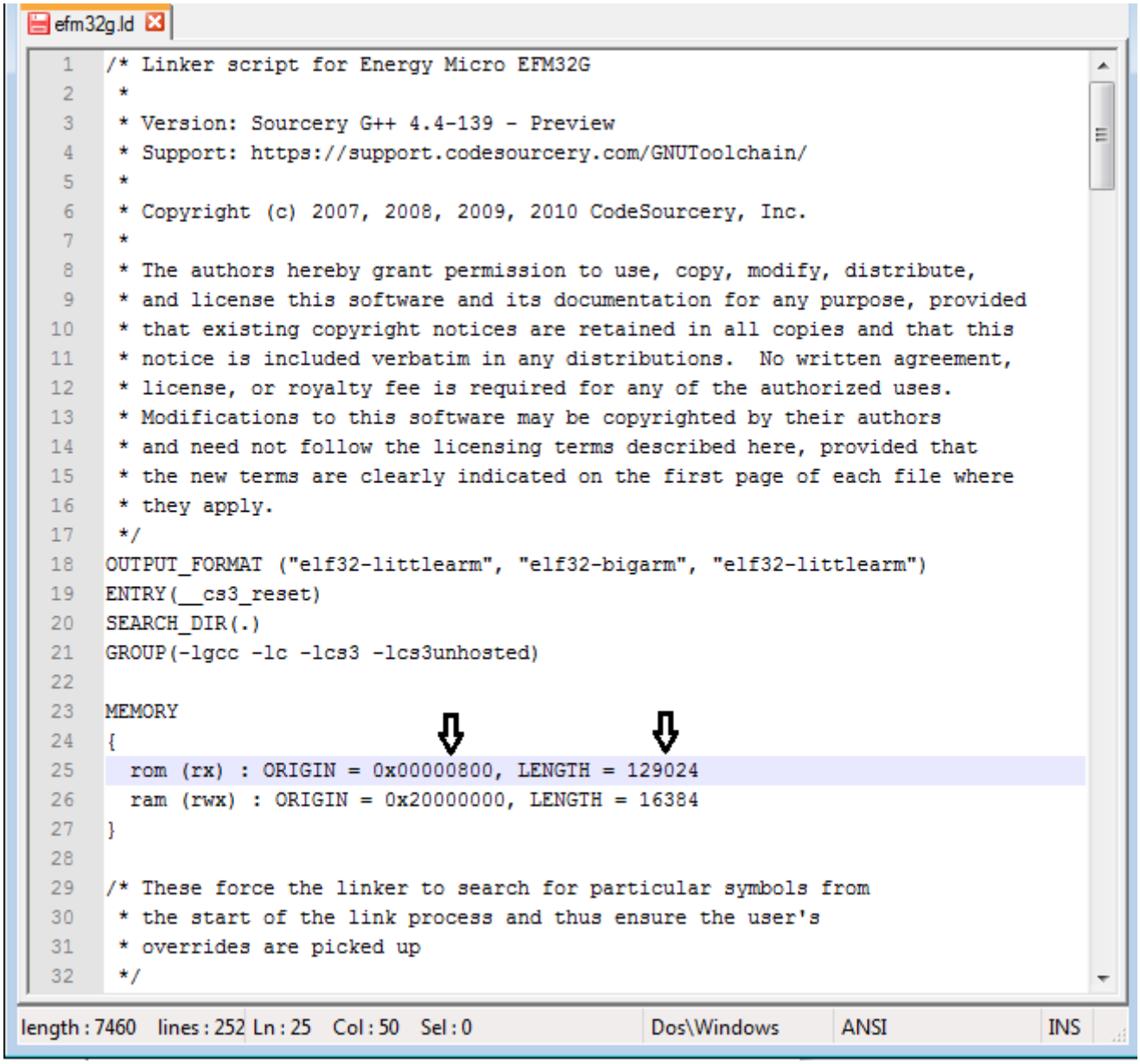


図 3.6. Eclipse/gcc/cs リンカ・ファイルのアプリケーション開始アドレス

Note: これらのリンカ・ファイルのいずれかを使用していてアプリケーションをデバッグする場合は、コード内でベクタ・テーブルの位置を明示的に設定する必要があります。例を以下に示します。

```
SCB->VTOR=0x800; // EFM32G, EFM32TG, EFM32ZG and EFM32HG parts
```

または

```
SCB->VTOR=0x1000; // EFM32GG, EFM32LG and EFM32WG parts
```

リリースした後のアプリケーションについては、アプリケーションを起動する前にブートローダによって VTOR が設定されるので、この作業は不要です（詳細については、「 Boot.c」 を参照）。

3.1.4 Simplicity Studio を使用したアプリケーションの作成

Simplicity Studio を使用して EFM32 シリーズ 0 または EZR32 シリーズ 0 ブートローダと連動するアプリケーションを作成するには、プロジェクト・プロパティを変更して、メモリ内でアプリケーションを探す場所をリンカに指示する必要があります。 Simplicity Studio IDE から この作業を行うには、[**プロジェクト**] > [**プロパティ**]をクリックします。[**プロパティ**] ウィンドウで [**ツール設定**] タブを選択し、[**メモリレイアウト**]を選択します。[**デフォルトのフラッシュオプションをオーバーライド**] を選択して、 [**オリジン**] フィールドに、0x800 （EFM32G、EFM32TG, EFM32ZG、EFM32HG）または 0x1000（EFM32GG、EFM32LG、EFM32WG）を入力し、 [**長さ**] の値を図 3.7 Simplicity Studio におけるアプリケーション開始アドレスとメモリ長の変更(13 ページ)に示すように調整します（ 元の長さからそれぞれ 0 x 800 または 0 x 1000 を減算）。リンカ・ファイルは Simplicity Studio によって自動的に作成されるため、後続のビルドでは変更内容が 上書きされるので、リンカ・ファイルのこれらのメモリ設定は手動で変更しないでください。

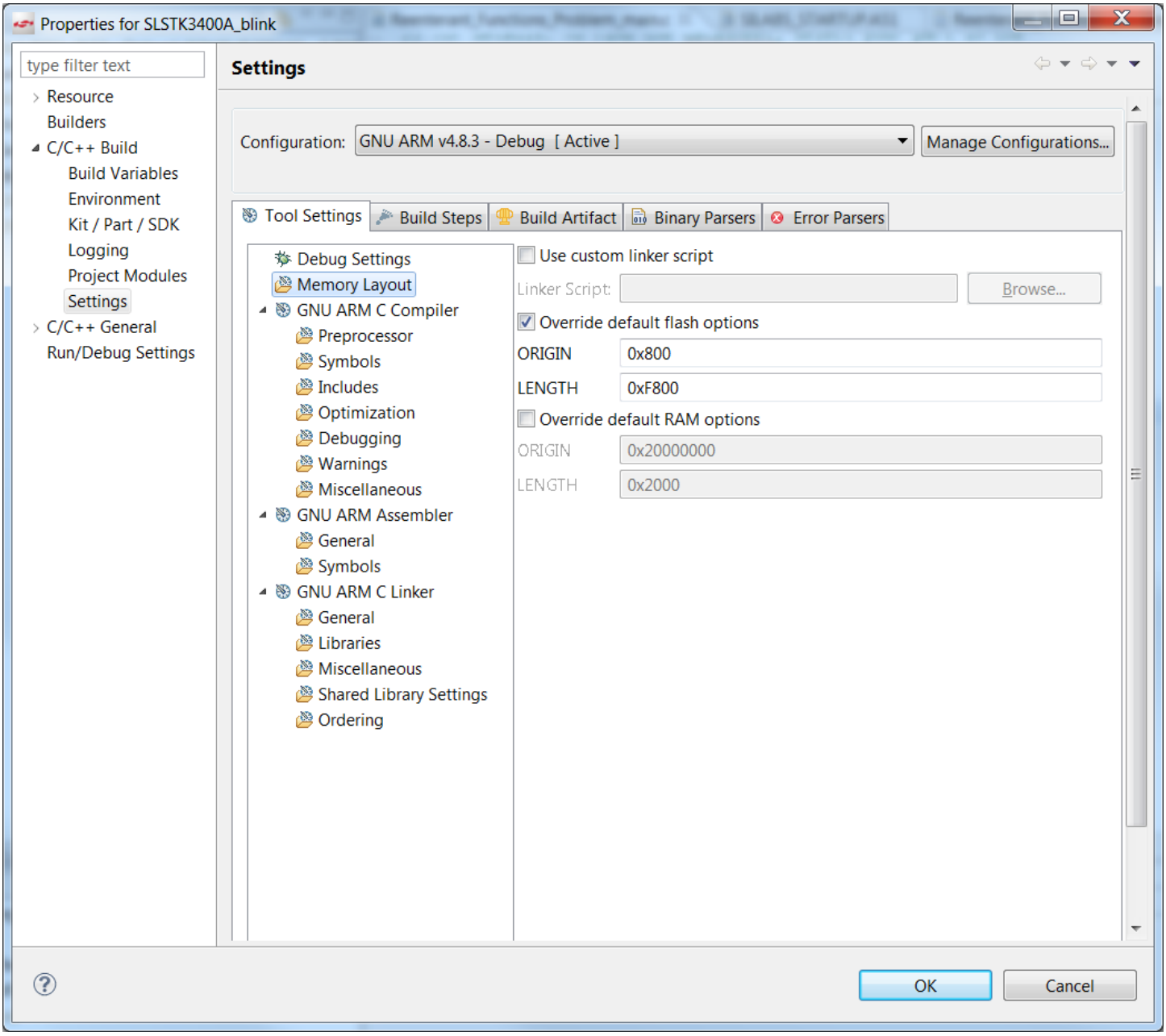


図 3.7. Simplicity Studio におけるアプリケーション開始アドレスとメモリ長の変更

Note: これらの変更を行った後にアプリケーションをデバッグする場合は、コード内でベクタ・テーブルの位置を明示的に設定する必要があります。例を以下に示します。

```
SCB->VTOR=0x800; // EFM32G, EFM32TG, EFM32ZG and EFM32HG parts
```

または

```
SCB->VTOR=0x1000; // EFM32GG, EFM32LG and EFM32WG parts
```

リリースした後のアプリケーションについては、アプリケーションを起動する前にブートローダによって VTOR が設定されるので、この作業は不要です（詳細については、「 Boot.c」を参照）。

3.2 ブートローダで使用するアプリケーションの作成 - EFM32 シリーズ 1

ブートローダは専用ブートローダ領域に存在するため、EFM32 シリーズ 1 EFx32xG1 デバイスを除く すべてのデバイスのブートローダはアプリケーションによって上書きされることはありません。したがって、専用のブートローダを持つこれらのデバイスのユーザーアプリケーションを作成する追加の手順は必要ありません。 これらのデバイスのブートローダは、ロック・ビット ・ページでビット 122 (CLW0) をクリアして無効にすることができます。ロック・ビット・ページ 構成の詳細については、これらのデバイスのリファレンス・マニュアルを参照してください。EFx32xG1 デバイスについては、 3.1 ブートローダで使用するアプリケーションの作成 - EFM32 シリーズ 0 および EZR32 シリーズ 0 セクションを参照してください。

3.3 アプリケーションのアップロード

[**u**] コマンドは、アプリケーションをアップロードします。ターミナル・ソフトウェアを使用してアプリケーション・バイナリをチップに転送します。アップロードが完了した後は、アップロードしたバイナリに対して CRC-16 を計算して正確性を検証できます。この検証を行うには、「アプリケーション・チェックサムの検証」で説明しているコマンドを実行します（「4.1 アプリケーション・チェックサムの検証」を参照）。ブートローダからアプリケーションを起動するには、「起動」コマンド（[**b**–]「5.1 アプリケーションの起動」を参照）を使用します。

3.4 破壊的アップロード - EFM32 シリーズ 0 および EZR32 シリーズ 0 のみ

[**d**] コマンドは、EFM32 シリーズ 0 および EZR32 シリーズ 0 デバイスに対して破壊的アップロードを開始します。このオプションは、EFM32 シリーズ 1 デバイスでは使用できません。バイナリをチップに転送するには、ターミナル・ソフトウェアを使用してください。破壊的アップロードが標準アップロードと異なるのは、ブートローダが上書きされる点です。そのため、別のブートローダをアップロードできます。また、ブートローダが必要ない場合は、ブートローダが占有しているフラッシュを解放できます。アップロードが完了した後は、CRC-16 チェックサムを計算して正確性を検証できます。この検証を行うには、「フラッシュの内容の検証」で説明しているコマンドを実行します（「4.2 フラッシュの内容の検証」を参照）。アプリケーションを起動するには、「リセット」コマンド（[**r**]–「5.2 デバイスのリセット」を参照）を使用します。

3.5 ユーザ情報ページへの書き込み

[**t**] コマンドを使用すると、ユーザ情報ページにデータを書き込むことができます。ユーザ情報ページにユーザ・データを転送するには、ターミナル・ソフトウェアを使用します。

3.6 ロック・ビット情報ページへの書き込み

[**p**] コマンドを使用すると、ロック・ビット情報ページにデータを書き込むことができます。ユーザ情報ページにユーザ・データを転送するには、ターミナル・ソフトウェアを使用します。このコマンドを使用すると、書き込みや消去に対してフラッシュのページをロックできますが、内容は保護されません。ロック・ビットの詳細については、リファレンス・マニュアルを参照してください。

第 4 章 アップロードの検証

Note: XMODEM-CRC は、128 バイトのブロックでデータを転送します。バイナリのサイズが 128 バイトの倍数でない場合、ターミナル・プログラムは残りのバイトをパディングします。詳細については、ターミナル・プログラムのドキュメントを参照してください。

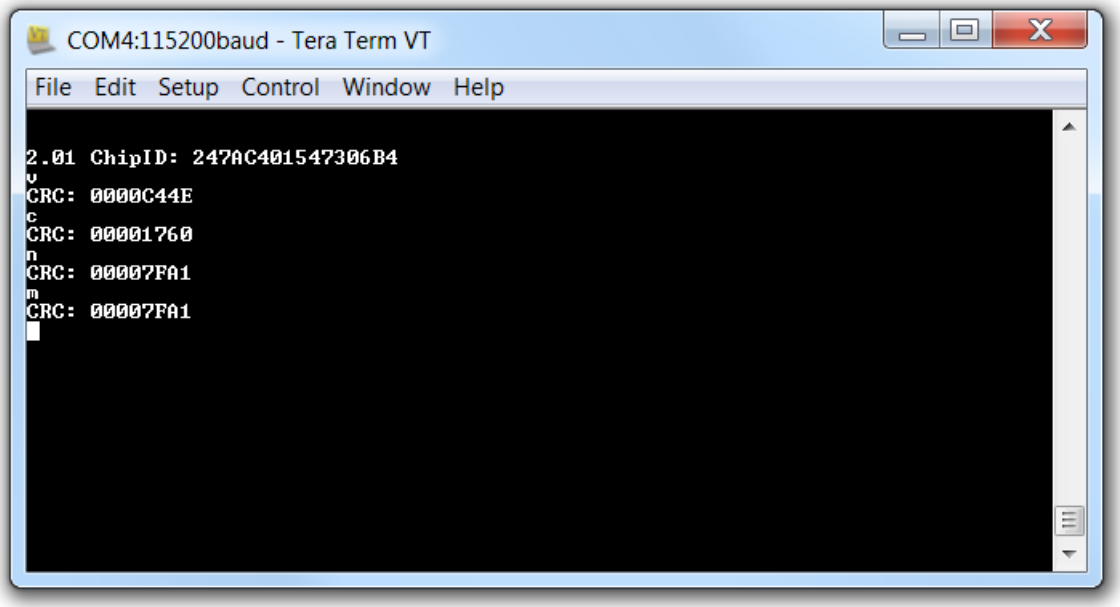


図 4.1. ブートローダ・チェックサム・コマンドを使用した Tera Term セッションの例（初期化の文字コマンド「U」は表示されていない）。

4.1 アプリケーション・チェックサムの検証

[**c**] コマンドは、ベース・アドレス 0x800 または 0x1000（アプリケーションの先頭）からフラッシュ領域の終わりまでフラッシュの CRC-16 チェックサムを計算し、出力します。

4.2 フラッシュの内容の検証

[**v**] コマンドは、ベース・アドレス 0x0（フラッシュ領域の先頭）からフラッシュ領域の終わりまでフラッシュの CRC-16 チェックサムを計算し、出力します。

4.3 ユーザ・ページ・チェックサムの検証

[**n**] コマンドは、ユーザ・データ (UD) ページ（情報フラッシュ・ブロックのページ 0）の CRC-16 チェックサムを計算して出力します。

4.4 ロック・ページ・チェックサムの検証

[**m**] コマンドは、ロック・ビット (LB) ページ（情報フラッシュ・ブロックのページ 1）の CRC-16 チェックサムを計算して出力します。

第 5 章 その他のコマンド

5.1 アプリケーションの起動

コマンド [**b**] は、デバッグ・ピンを High に設定してブートローダを無効にした場合と同じ方法で、アップロードしたアプリケーションを起動します。ブートローダはこの操作を行うために、まず プロセッサのベクタ・テーブルをアプリケーションのベース・アドレスに設定します。次に、ブートローダは、 新しいベクタ・テーブルの最初のワードを読み出し、それに応じて SP を設定します。最後に、リセット・ベクタで定義される値に PC を設定してベクタ・リセットを実行します。

Note: ブートローダが正常に動作すると、TIMER、USART、CMU、および GPIO を構成します。このコマンドを使用してアプリケーションを起動する際、これらの設定は維持されます。ただし、ブートローダ・ピンをアサートしてブートローダに切り替えない場合、これらのレジスタは修正されません。この状態が一般的です。

Note: UART ブートローダを使用する場合、文字 [**b**] はシリアルターミナルに印刷しませんが、 ユーザーはアプリケーションの起動を [**b**] コマンドを押した後に見ることができます。

5.2 デバイスのリセット

[**r**] コマンドはデバイスをリセットします。破壊的アップロードの後にこのコマンドを実行すると、新しいバイナリが起動します。標準アップロードの後にこのコマンドを実行し、デバッグ・ピンを High に設定していない場合、アプリケーションが起動します。それ以外の場合、ブートローダが再起動します。

5.3 デバッグ・ロック

[**l**] コマンドはデバッグ・インターフェイスをロックします。ロックした後、標準デバッグ機能にはアクセスできなくなり、デバッグ・インターフェイスから可能な操作はデバイスの消去だけになります。

Note: デバイスは、デバッグ・インターフェイスがロックされる前に一度リセットする必要があります。このコマンドは、ロックが成功すると「OK」を、それ以外の場合は「Fail」を返します。デバッグ・ロックが失敗した場合は、DBG_SWDIO が接続されていないことと、DBG_SWCLK が High に接続されていることを確認してください。

第 6 章 ブートローダのコンパイル

このアプリケーション・ノートには、ブートローダ自体のソース・コードが添付されています。このソース・コードを使用すると、独自のブートローダをコンパイルできます。このソース・コードの使用について、いくつかの注意点があります。

- コンパイルしたブートローダは、フラッシュの構成済みの領域に収まる必要があります。Gecko、Tiny Gecko、Zero Gecko、および Happy Gecko の場合、これは 0x800 バイトです。Leopard Gecko、Giant Gecko、および Wonder Gecko の場合は 0x1000 バイトです。
- ソース・プロジェクトが用意されているのは IAR のみです。他の IDE でもソース・ファイルを使用できますが、割り当てられているサイズ内にコンパイルしたブートローダが収まることを確認する必要があります。
- IAR ではリリース構成を使用してソース・コードをコンパイルする必要があります。デバッグ構成でコンパイルすると、コンパイル後のプログラムのサイズが大きくなります。[High optimization for size（サイズを高度に最適化）] をオンにして、[Multi-file compilation（マルチファイル・コンパイル）] を有効にしてください。[Discard unused publics（未使用のパブリックを破棄）] も役に立つ機能できますが、この機能は、リリース構成でコンパイルした場合はすでに有効になっています。
- Tiny Gecko の場合、プロジェクトは 2 つあります。EFM32TG108Fxx および EFM32TG110Fxx デバイスでは、bootloader-tg-small を使用してください。その他の Tiny Gecko デバイスでは、bootloader-tg を使用してください。

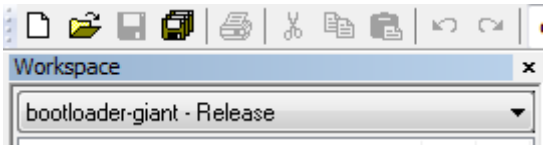


図 6.1. リリース・モードでのコンパイル

第 7 章 修正履歴

改訂 1.77

2022 年 3 月

- 第 1 章 デバイスの互換性 に EFM32GG12 を追加しました。

改訂 1.76

2020 年 7 月

- アプリケーションノートに含まれる ファイルの情報を 表紙に追加しました。.
- 2.3 コマンドライン・インターフェイス および 3.2 ブートローダで使用するアプリケーションの作成 - EFM32 シリーズ 1 セクションの表現を明確化しました。
- 文書全体にわたり諸々の変更がなされています。

改訂 1.75

2020 年 5 月

- セクション 3.2 ブートローダで使用するアプリケーションの作成 - EFM32 シリーズ 1 の表現を明確化しました。

改訂 1.74

2018 年 3 月

- EFM32GG11 および EFM32TG11 デバイスを含むようにデバイスの互換性を変更しました。
- EFM32PG13 と EFM32JG13 を EFM32 シリーズ 1 リストから削除しました。
- EFR32 シリーズ 1 リストに EFR32XG14 を追加しました。
- EFM32GG11-X エンジニアリング・ステータス・デバイスのブートローダのサポートに関する注記を削除しました。

改訂 1.73

2017 年 10 月

- シリーズ 1 デバイスを含むようにデバイスの互換性を変更しました。
- オートボーアルゴリズムと互換性のあるボーレートを更新しました。
- Teraterm での XMODEM 転送に関する表現を更新しました。
- UART ブートローダを使用する際のシリアル端子ディスプレイ の違いに関する注記を追加しました。
- ブートローダの最新バージョンを示す テキストファイルの場所に関する注記を追加しました。
- IAR を使用したアプリケーションの作成を詳細に説明するスクリーンショットを追加しました。

改訂 1.72

2017 年 6 月

- 「互換性リスト」の名前を「第 1 章 デバイスの互換性」に変更し、EFM32GG11 を追加。

改訂 1.71

2017 年 2 月

- シリーズ 0 とシリーズ 1 の命名方法に合せてデバイス・リファレンスを修正。

改訂 1.70

2015 年 11 月

- EFM32PG および EFM32JG デバイスを追加。
- Simplicity Studio でメモリ設定を変更する手順を追加。
- EFM32 シリーズ 0 および EZR32 シリーズ 0 デバイスと EFM32 シリーズ 1 デバイスに関する手順を区別。

改訂 1.69

2015 年 3 月

- EFM32HG デバイスを追加。
- 形式を更新
- 各種ファミリのブートローダ UART 構成を明記。
- IAR コンパイル・ガイダンスを追記。

改訂 1.68

2014 年 5 月

- EFM32 Wonder Gecko のバイナリとリンカ・ファイルを追加
- 「CodeSourcery」と「Eclipse」の記述を一般的な「GCC」に置き換え
- Silicon Labs ライセンスに変更

改訂 1.67

2013 年 10 月

- 新しいカバー・レイアウト

改訂 1.66

2013 年 7 月

- LEUART0 を使用するデバイスを明記。

改訂 1.65

2013 年 5 月

- ブートローダのコンパイルに関するセクションを追加。

改訂 1.64

2012 年 11 月

- 「EFM32TG110 には USART0 がない」という記述の誤りを修正。

改訂 1.63

2012 年 4 月

- アセンブリ・ファイル内のコメント部分の「;」の欠落を修正。

改訂 1.62

2012 年 4 月

- ビルド済みのバイナリを新たに追加。これまでのバイナリは、32K 未満のフラッシュを装備したデバイスでは動作しませんでした。
- 新しいペリフェラル・ライブラリの命名規則と CMSIS_V3 に合わせてソフトウェア・プロジェクトを変更。
- Eclipse/GCC/Sourcery CodeBench 用のアプリケーション・コードのリンカ・ファイルの修正方法に関するセクションを追加。

改訂 1.61

2011 年 11 月

- 欠落していた EFM32LG バイナリを追加。
- アプリケーション・ノートの名前を変更。

改訂 1.60

2011 年 9 月

- EFM32GG および EFM32LG デバイスのサポートを追加。
- 他のブートローダについての機能的な違いはなし。

改訂 1.50

2011 年 6 月

- TG110 および TG108 デバイスでブートローダ USART を F0、F1（ブートローダ・ピンと共有）に移行。
- 他のブートローダについての機能的な違いはなし。

改訂 1.40

2011 年 4 月

- ブートローダ・ピンを SWDIO と SWDCLK から SWDCLK のみへ変更。
- デバッグ・ロックを修正。
- Tiny デバイスのサポートを追加。
- バージョン番号とともにチップ ID を出力。
- フラッシュと RAM の使用量を自動的に抑えるためのリンカ・ファイルを追加。
- 部品別の構成を config.h に移行。
- 一部の機能をフラッシュに移動して SRAM フットプリントを縮小。

改訂 1.30

2010 年 9 月

- DMA を使用してダウンロード速度を向上。
- 高ビットレートの処理を改善。
- テスト用バイナリを追加。

改訂 1.22

2010 年 9 月

- 最初のページの主見出しを更新。コードの変更はありません。
- このブートローダ・アプリケーション・ノートが適用されるデバイスの記述を更新。

改訂 1.21

2010 年 9 月

- 最初のページの主見出しを更新。コードの変更はありません。

改訂 1.20

2010 年 8 月

- プログラミングの速度を向上。RTC を使用したアプリケーションがクラッシュするバグを修正。このバグはバージョン 1.10 で発生していました。

改訂 1.10

2010 年 4 月

- ブートローダがアイドリング時に EM2 を使用。

改訂 1.00

2010 年 1 月

・ 初期改定。

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com