ANALOG DEVICES

AN-2058
アプリケーション・ノート

ADuCM355 ユーザ・ブートローダ

はじめに

カーネル・ブートローダがないため、消去された ADuCM355 のフラッシュ・メモリは、最初にシリアル・ワイヤを介してユーザ・アプリケーションをプログラムする必要があります。

ユーザ・アプリケーションでは、フィールド内で自己更新するメカニズムを備えた独自のブートローダを実装することができます。独自のブートローダを実装するには、ユーザ・アプリケーションがユーザ・ブートローダに適した形で構成されている必要があります。

このアプリケーション・ノートでは、以後ユーザ・ブートローダと呼ぶ 1 つのアプローチについて説明します。このユーザ・ブートローダは、ユーザ・フラッシュを 2 つの別々の領域にパーティショニングし、1 つの領域では CrossCore®シリアル・フラッシュ・プログラマに対応できるブートローダを実装することで実現できます。

Rev. 0

アナログ・デバイセズ株式会社

本　　社／〒105-6891　東京都港区海岸 1-16-1　ニューピア竹芝サウスタワービル 10F
電話 03（5402）8200
大　阪営業所／〒532-0003　大阪府大阪市淀川区宮原 3-5-36　新大阪トラストタワー 10F
電話 06（6350）6868
名古屋営業所／〒451-6038　愛知県名古屋市西区牛島町 6-1　名古屋ルーセントタワー 38F
電話 052（569）6300

目次

改訂履歴

10/2020— Revision 0: Initial Version

ADuCM355 の概略

ADuCM355 の次の機能が、ユーザ・ブートローダを実装するために使用されます。

ADuCM355には128kBのユーザ・フラッシュがあり、これは消去と書込み保護のために8kBのブロックごとにパーティショニングされています。

リセット後、内蔵カーネルによって、以下が直ちに実行されます。

- 製造時のデータを使用してデバイスをブート。
- ユーザ・フラッシュ・メタデータを調べ、ユーザ・アプリケーションを実行するよう切り替えるか、内蔵カーネルにとどまるかを判定。
- ユーザ・フラッシュ・メタデータを調べ、ユーザ・フラッシュの容量内のフラッシュ・ブロックを書込み保護するかどうかを判定。ユーザ・アプリケーションが有効な場合、ブロックの書込み保護を適用して終了し、ユーザ・アプリケーションに移行します。ユーザ・アプリケーションが有効でないか、BM/P1.1 ピンがアサートされている場合、ブロックの書込み保護は適用されず、内蔵カーネルにとどまります（カーネル実行のフローチャートについては図 7 を参照）。

内蔵カーネルにとどまる場合、次の利点があります。

- 復元。デバイスへの必要なアクセスを妨げる可能性がある動作をユーザ・コードが実行することがなくなります。
- 書込み保護の回避。ブロックが書込み保護されると消去できず、これは一括消去によっても不可能です。そのため、ユーザ・フラッシュへの書込み保護の適用は、それがユーザ・フラッシュ・メタデータ経由の間接的な方法であっても、ユーザ・アプリケーションの実行による直接的な方法であっても、回避されます。コード実行がカーネル内にとどまるため、ユーザ・フラッシュの一括消去が可能となります。

ユーザ・ブートローダの実装

ユーザ・ブートローダはユーザ・フラッシュの最初の 8kB に配置されます。残りの 120kB は、ユーザ・アプリケーションの開発用に使用できます（図 1 を参照）。

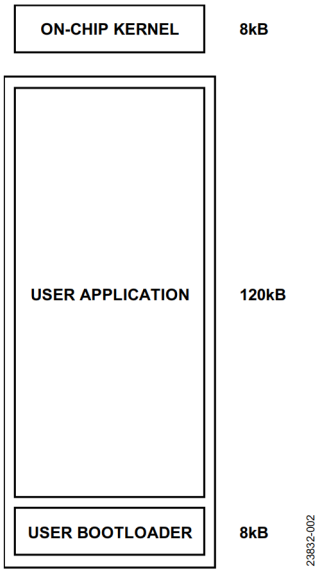

図 1. メモリ・レイアウト

ユーザ・ブートローダは、ユーザ・フラッシュの最初の 8kB にある、スタンドアロンのアプリケーションです。ユーザ・アプリケーションは、0x2000（つまり 8kB）以降で実行するよう構築する必要があります。そのため、スタンドアロン・アプリケーションに多少の修正を加える必要があります。

この修正を行うため、カスタム・リンカ設定ファイルを使用してユーザ・アプリケーションをユーザ・フラッシュ内の適切な場所に配置することが必要です。

カスタム・リンカ・ファイルの読込みは、IAR Embedded Workbench®環境で、［**Linker**］>［**Config**］タブにアクセスして実行できます（図 5 を参照）。手順の全体については変換手順のセクションを参照してください。

.icf リンカ・スクリプト・ファイルを ADuCM355 用に GitHub で提供されている標準的なスクリプト・ファイルと比較すると、いくつかの違いが浮かび上がります。（図 2 を参照）。

```
/* The user bootloader resides in the first 8K flash block */
define symbol BOOTLOADER_SIZE = 8K;   Bootloader size is 8K

/* The user code (and user code checksum) is pushed up in the memory space
   by the size of the bootloader.
*/
define symbol CHECKSUM_ADDR  = BOOTLOADER_SIZE + 0x000007FC;   The location of the checkum is offset by 8K

/* The bootloader is provided to this project as an array in BootloaderConstantArray.c
   Place that at 0x0
*/
define block BOOTLOADER    with alignment = 4, size = BOOTLOADER_SIZE { readonly object BootloaderConstantArray.o };
place at address 0x0 { block BOOTLOADER };   Can optionally include the a bootloader in the first 8K


/* The flash available for the user application starts after the bootloader and is reduced by the size of the
   bootloader.
*/                                                  User flash is adjusted to use flash after bootloader
define symbol FLASH                                 = BOOTLOADER_SIZE;  // "flash" starts after the booloader
define symbol FLASH_SIZE                            = 128K - BOOTLOADER_SIZE;       // 128k flash size
define symbol FLASH_PAGE_SIZE                       = 2K;           // 2k flash page size
define symbol PAGE0_ROM_START                       = 0x1A0;
define symbol UF_START_PAGE                         = BOOTLOADER_SIZE/FLASH_PAGE_SIZE;
export symbol UF_START_PAGE;                       // C file needs this information
export symbol FLASH_PAGE_SIZE;                      // C file needs this information
```

23832-003

図 2. リンカ設定ファイルの修正

ブートローダの配置

ユーザ・ブートローダはユーザ・フラッシュの最初の 8kB に配置されます。これは、例外ベクタ・テーブル、チェックサムの配置、ユーザ・フラッシュ・メタデータを備える他の ADuCM355 アプリケーションと同様に構築されています。

ユーザ・アプリケーションの配置

ユーザ・アプリケーションは 0x2000 以降に配置され、使用可能なアプリケーション容量は、0x2000 だけ減少します。Cortex®-M3 割込みベクタ・テーブルは 0x2000 に配置されます。ユーザ・ブートローダは、この配置に一致するよう、Cortex-M3 ベクタ・テーブル・オフセット・レジスタ（VTOR）を更新します。

ユーザ・フラッシュ・メタデータは 0x2000 だけオフセットされます。

ユーザ・ブートローダ・メタデータ

ユーザ・ブートローダ・メタデータは、内蔵カーネルによってチェックと処理が行われます。内蔵カーネルは、ユーザ・ブートローダ領域の有効性を検証してから、ユーザ・フラッシュのユーザ・ブートローダ領域に制御を移し、任意のユーザ・フラッシュ書込み保護を適用します。

ユーザ・アプリケーション・メタデータ

ユーザ・アプリケーション・メタデータは、ユーザ・ブートローダによってチェックと処理が行われます。ユーザ・ブートローダは、ユーザ・アプリケーション領域の有効性を検証してから、ユーザ・フラッシュのユーザ・アプリケーション領域に制御を移し、任意のユーザ・フラッシュ書込み保護を適用します。

ユーザ・ブートローダは、内蔵カーネルがユーザ・ブートローダ・メタデータ（または標準的なアプリケーション・メタデータ）に対し実行するのと同じタイプのチェックと動作をユーザ・アプリケーション・メタデータに対し行いますが、使用するアドレスと範囲だけが異なります。したがって、標準アプリケーションは 0x2000 だけオフセットすることで、ユーザ・ブートローダで動作するよう容易に変換できます。

デスクトップ・アプリケーション

プロトコルは、www.analog.com/jp/crosscore-utilities でダウンロード可能な、アナログ・デバイセズの **CrossCore Serial Flash Programmer** ツールで使用できます。

この **CrossCore Serial Flash Programmer** ツールは、いくつかの異なるプロトコルの変種に対応しています。ユーザ・ブートローダのこの実装は、ADuCM3027 および ADuCM3029 によって実装されるバージョンをサポートしています。図 3 に示すように、［**Target**］で［**ADuCM302x**］オプションを選択してください。

［**Action**］がサポートするオプションは、［**Program**］と［**Erase**］です。

［**Browse**］ボタンをクリックして［**File to download**］からユーザ・アプリケーションの Intel Hexadecimal ファイルをロードし、［**Start**］をクリックします。

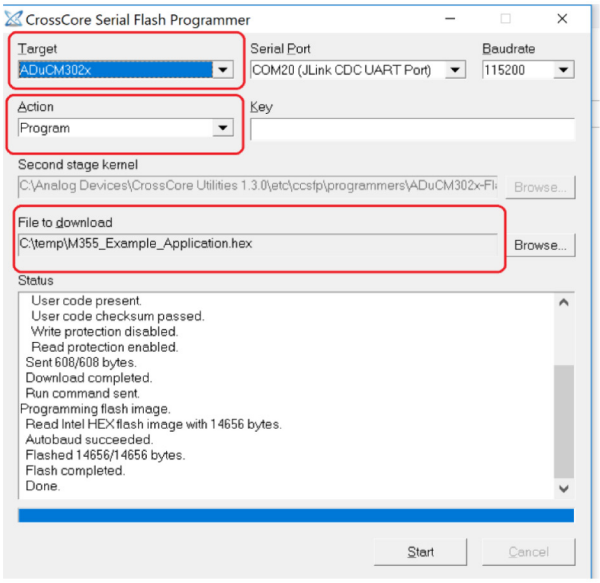

図 3. **CrossCore Serial Flash Programmer** ウィンドウ

変換手順

ユーザ・ブートローダまたはユーザ・アプリケーションのモデルで使用できるように既存のアプリケーションを変換するには、次の手順が必要です。

1. ADuCM355 ソフトウェア開発キット（SDK）を GitHub からダウンロードまたは複製します。GitHub で ADuCM355 を検索し、**aducm355-examples** SDK ファイルを見つけてください。
2. SDK で、［**examples**］>［**DigitalDie**］>［**M355_Bootloader**］の順に移動します。**BootloaderConstantArray.c** ファイルと、**iar** フォルダの **user-bootloader-sample-application.icf** ファイルを、ブートローダの実装先のアクティブなプロジェクト・ディレクトリにコピー&ペーストします。
3. IAR Embedded Workbench で目的のプロジェクトを開き、プロジェクト・エクスプローラのアプリケーション・フォルダを右クリックして **BootloaderConstantArray.c** ファイルをプロジェクトに追加します。次に、［**Add**］>［**Add Files**］の順に移動して、**BootloaderConstantArray.c** ファイルを選択します。ユーザ・ブートローダは C ファイルで提供されます。これには、ユーザ・ブートローダを実装するための命令コードの C 配列が含まれています。カスタム・リンカ設定ファイルが備わっているため、この配列は確実に 0x0000 0000 の場所に正しく配置されます。

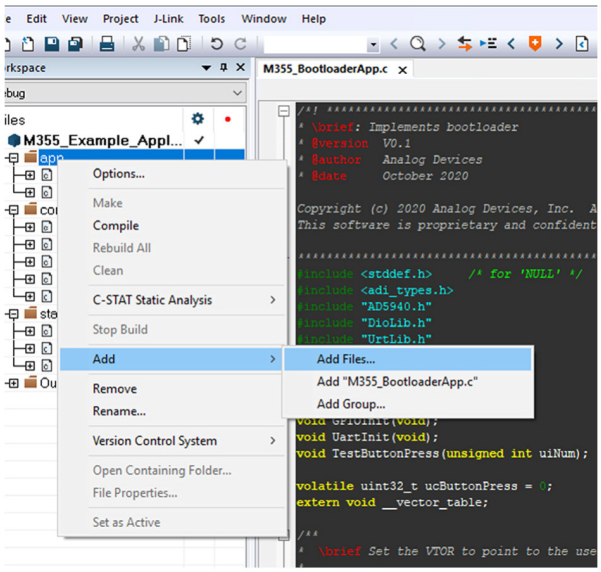

図 4. ブートローダの追加

4. 図 5 に示すように、IAR Embedded Workbench で［**Linker**］>［**Config**］タブにアクセスして、カスタム・リンカ設定ファイルを選択します。［**Override default**］ボックスを選択して、**user-bootloader-sample-application.icf** ファイルを探します。
この変更されたリンカ設定ファイルによって、ユーザ・ブートローダとユーザ・アプリケーションはユーザ・ブートローダまたはユーザ・アプリケーションのモデルに従って適切に配置されるようになります。

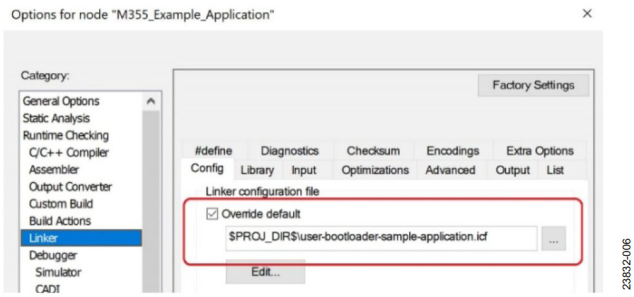

図 5. カスタム・リンカ設定ファイルの指定

5. ユーザ・アプリケーションのチェックサム計算の範囲は、変更されたメモリ・レイアウトに合わせて調整する必要があります。IAR Embedded Workbench でチェックサム計算を調整するには、［**Linker**］>［**Checksum**］タブに移動します（図 6 を参照）。［**Start address**］ボックスに **0x2000** を、［**End address**］ボックスに **0x27FB** を入力します。
標準のアプリケーションでは、この計算は内蔵カーネルを使用し、ユーザ・フラッシュ内のアプリケーションの整合性をチェックします。
ユーザ・ブートローダまたはユーザ・アプリケーションのモデルでは、内蔵カーネルがユーザ・ブートローダの整合性をチェックした後にユーザ・ブートローダに切り替え、ユーザ・ブートローダがユーザ・アプリケーションの整合性をチェックした後にユーザ・アプリケーションに切り替えます。

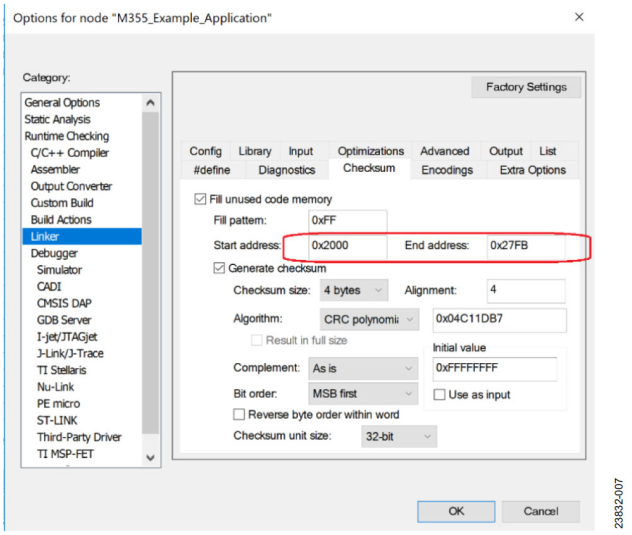

図 6. チェックサム計算

6. **startup_ADuCM355.c** ファイルを変更します。
   a. 次のコード行を追加します。
   ```
   /* Linker provided symbols */
   extern uint32_t FINAL_CRC_PAGE;
   ```
   b. **NumOfCRCPages** を検索し、次の行
   ```
   __root const uint32_t
   NumOfCRCPages=0;
   ```
   を次の行に置き換えます。
   ```
   __root const uint32_t
   NumOfCRCPages=(uint32_t)&FINAL_CR
   C_PAGE;
   ```
7. 最終手順はメイン・プログラムに関数を追加することです。
```
void_iar_init_core (void)
{
SCB->VTOR = (uint32_t)
&_vector_table;
}
```

この関数の目的は、VTOR がユーザ・ファームウェアの例外テーブルをポイントするようにすることです。ブートローダは既にこの手順を実行していますが、デバッグ・モードの場合、IAR Embedded Workbench はブートローダの実行をバイパスします。IAR Embedded Workbench のデバッガは、PC にタイプ 0 のリセット（ハードウェア・リセット）を実行してから、リセット・ベクタであると考えられる状態にリセットします。そのため、デバッグ・モードで割込みが機能するためには、この関数が必要になります。

以上で、このアプリケーションは通常どおりにダウンロードとデバッグができます。

ブート・モード・ピン

内蔵カーネルとユーザ・ブートローダには、それぞれの実行フローを変更する個別のブート・モード・ピンがあります。

内蔵カーネルのブート・モード・ピンの優先度の方がユーザ・ブートローダのブート・モード・ピンより高くなっています。

内蔵カーネル BM/P1.1 ピン

BM/P1.1 は、ユーザ・フラッシュ内のすべてのソフトウェアの実行をバイパスします。

BM/P1.1 ピンは、EVAL-ADucM355QSPZ評価用ボードのスイッチ S3 に接続します。スイッチ S3 を押し続けながらリセットを実行する（スイッチ S1 を押して放す）と、内蔵カーネルがブート・モードに設定されます。

ユーザ・ブートローダ・ブート・モード P1.0/SYS_WAKE ピン

P1.0/SYS_WAKE ピンは、ユーザ・フラッシュにあるユーザ・アプリケーションの実行をバイパスします。これがアサートされている場合、ユーザ・ブートローダは有効なユーザ・アプリケーションの存在をチェックせずに、直ちにダウンロード・モードに入ります。

P1.0/SYS_WAKE ピンは、EVAL-ADucM355QSPZ 評価用ボードのスイッチ S2 に接続します。スイッチ S2 を押し続けながらリセットを実行する（スイッチ S1 を押して放す）と、ユーザ・ブートローダがブート・モードに設定されます。

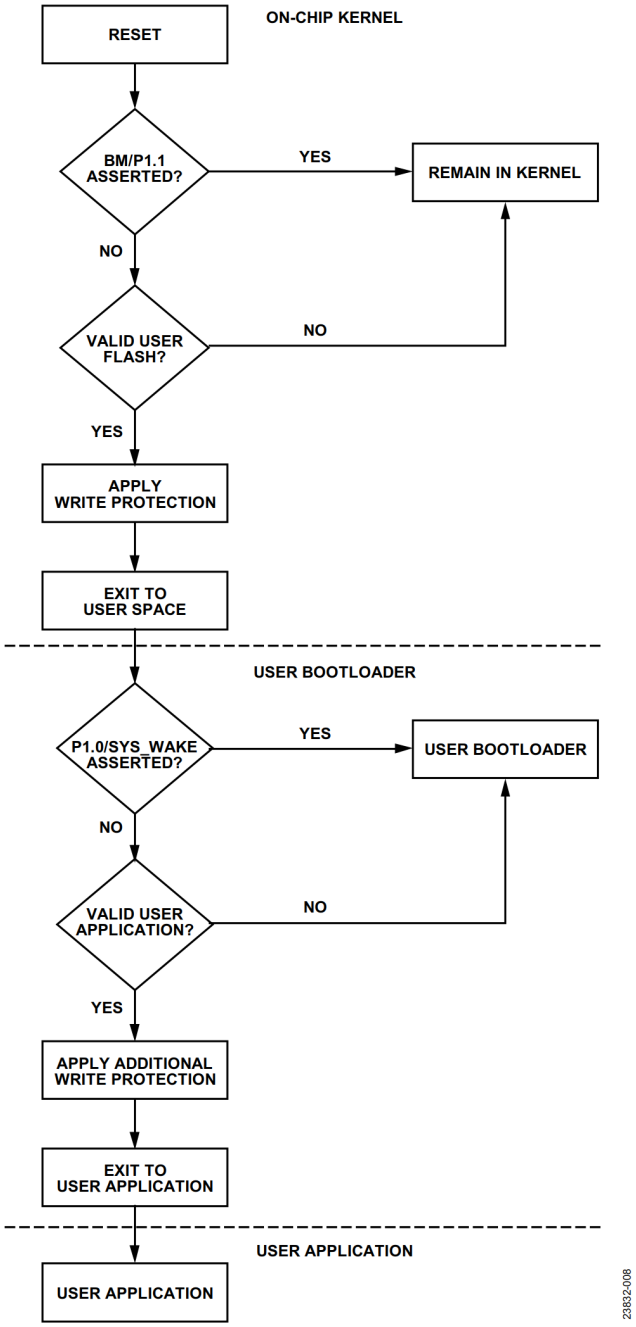

図 7. ブートローダのフロー図