ADSP-BF533 Blackfin®のブート・プロセス

著者: Hiren Desai

Rev 4 –2008 年 9 月 29 日

はじめに

この EE ノートでは、ADSP-BF531、ADSP-BF532、ADSP-BF533 Blackfin®プロセッサのブート・プロセスについて説明します。シリコン・レビジョン・レベル間の相違を示してあります。
この EE ノートでは次について説明します。

- ブート・モード
- ローダ・ファイルのヘッダー情報
- 初期化コード(Init コード)
- マルチアプリケーション(multi-DXE)管理

ブート・プロセス

ブートとは、外部メモリ・デバイス(または外部ホスト)に格納されているアプリケーション・コード/データをBlackfinプロセッサの種々の内部メモリおよび外部メモリにロードするプロセスを意味します。このプロセスは、Blackfinメモリのアドレス 0xEF00 0000～0xEF00 03FFに格納されている内蔵ブートROMにより制御されます。図 1に、ソース・コードから最終ターゲット・スタンドアロン・システムまでの操作シーケンスを示します。

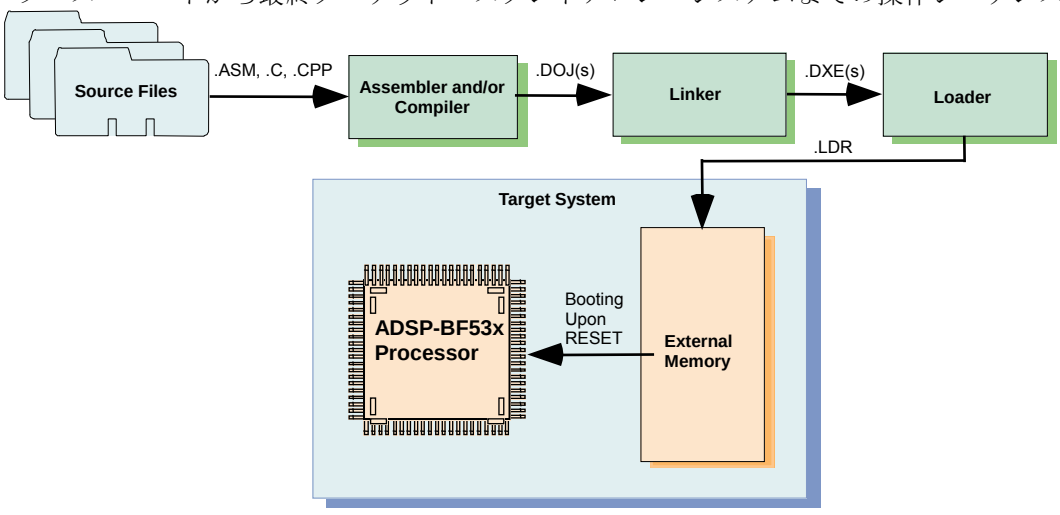


図1.ADSP-BF531/BF532/BF533 スタンドアロン・システム

ブート・モード(シリコン・レビジョン0.3)*

Blackfinプロセッサは、EBIUの非同期バンク 0 を経由したフラッシュ/PROMまたはSPIインターフェースを介したSPIデバイス(メモリまたはホスト)からブートすることができます。表 1に、RESET信号の非アサート時のBMODE[1:0]ピンの状態により選択されるADSP-BF531/BF532/BF533 プロセッサのブート・モードを示します。

BMODE[1:0]	Description (See also Specific Blackfin Boot Modes)
00	Executes from external 16-bit memory connected to ASYNC Bank0 (bypass Boot ROM)
01	Boots from 8/16-bit flash/PROM
10	Boots from a SPI host in SPI Slave mode
11	Boots from a 8/16/24-bit addressable SPI memory in SPI Master mode with support for Atmel AT45DB041B, AT45DB081B, and AT45DB161B **DataFlash**® devices

表1.Blackfin ADSP-BF531/BF532/BF533 のブート・モード

*シリコンの旧レビジョンでサポートされていたブート・モードについては、アペンディックス:ブート・モード対シリコン・レビジョンを参照してください。

図 1に示すように、ローダ・ユーティリティ(`elfloader.exe`)が入力実行可能ファイル(`.DXE`)を解釈して、先頭にヘッダーが付いた複数のブロックから構成されるローダ・ファイル(`.LDR`)*を生成します。このローダ・ファイルは、外部メモリ/デバイスに書き込まれます。このヘッダーは、ブート時に内蔵ブートROMにより読み出され、解釈されます。

*スイッチ・ローダ・ファイルについては「*VisualDSP++® Loader Manual for 16-Bit Processors [1]*」を参照してください。

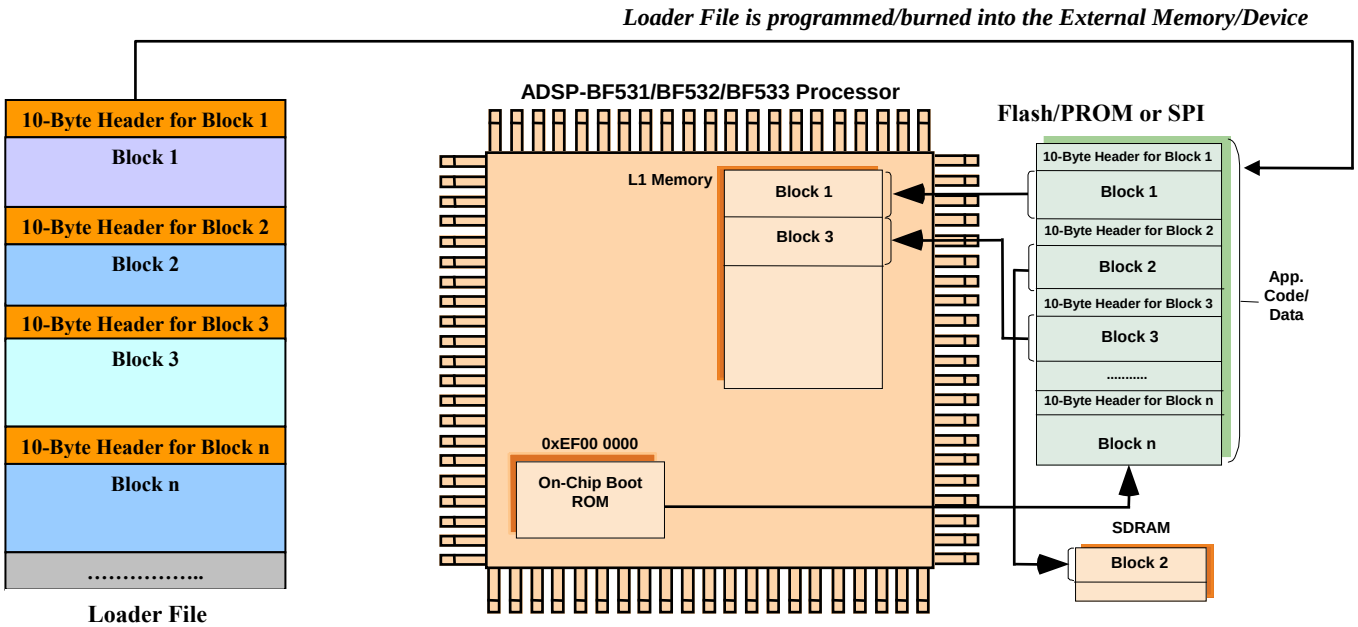


図2. ADSP-BF531/BF532/BF533 のブート・プロセス

スクラッチパッド・メモリ(`0xFFB0 0000`～`0xFFB0 0FFF`)へのブートはサポートされていません。スクラッチパッド・メモリへのブートを行うと、プロセッサは内蔵ブート ROM 内でハングアップします。

ヘッダー情報

図 3に示すように、ローダ・ファイル内の各 10 バイト・ヘッダーは、4 バイトADDRESSフィールド、4 バイトCOUNTフィールド、2バイトFLAGフィールドから構成されています。

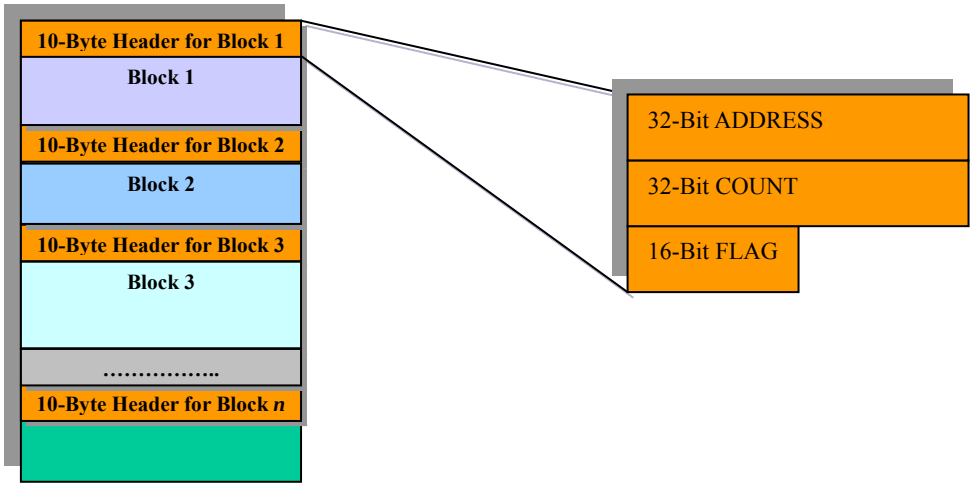


図3.10 バイト・ヘッダーの内容

ローダ・ファイル内で各ブロックの先頭に付く 10 バイト・ヘッダーには、ブート・プロセス時に内蔵ブート ROM により使用される次の情報が含まれています。

- **ADDRESS (4 バイト)**—メモリ内でブロックがブートされるターゲット・アドレス
- COUNT (4 バイト)—ブロック内のバイト数
- **FLAG (2 バイト)**—ブロック・タイプとコントロール・コマンド:

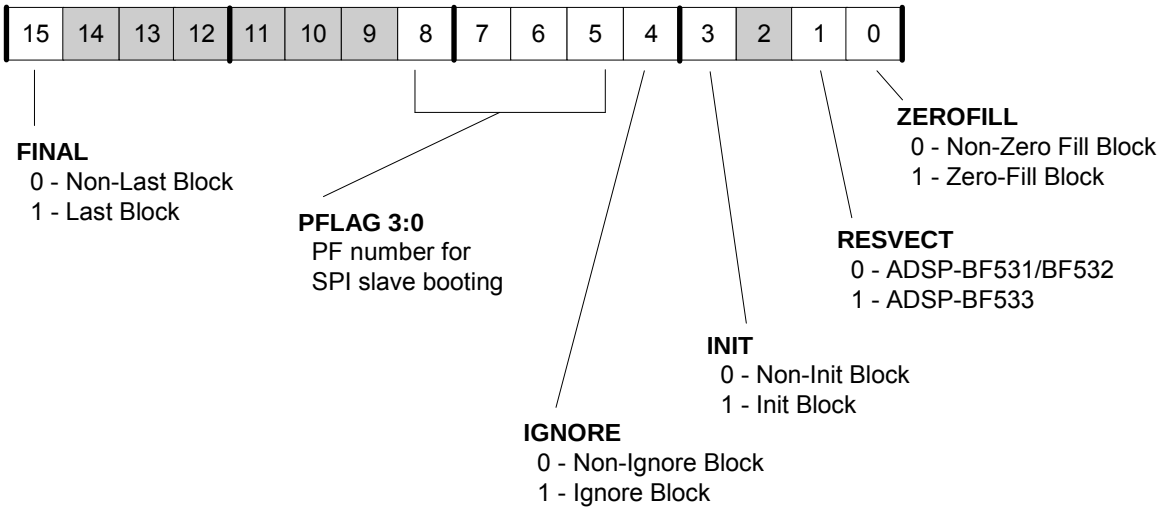


図4.FLAG ワードの各コントロール・ビット

FLAG ビットには次が含まれます。

- **ビット 0: ZEROFILL**—ブロックがゼロを持つバッファであることを表示します。ZEROFILL ブロックにはペイロード・データはありません。これらは、メモリ内の ADDRESS から開始される COUNT バイトをゼロにすることを内蔵ブート ROM に指示します。これにより、大きなゼロ・バッファを持つアプリケーションに対して圧縮したローダ・ファイルが生成されます。これは、ブート時に大きなバッファをゼロにすることが要求されることがある ANSI-C 準拠のプロジェクトにも非常に役立ちます。
- **ビット 1: RESVECT**—ブート後のリセット・ベクタを表示します。すべての ADSP-BF531/BF532/BF533 派生品は同じブート ROM を使います。ADSP-BF531/BF532 ではこのビットに 0 を、ADSP-BF533 では 1 を、それぞれ設定します。ブートが完了すると、内蔵ブート ROM はこのビットを使って、ADSP-BF533 ではアドレス 0xFFA0 0000 へ、ADSP-BF531/BF532 ではアドレス 0xFFA0 8000 へ、それぞれジャンプします。
-

  ハードウェア・リセットの後、RESVECT ビットに応じてリセット・ベクタ(EVT1 レジスタ)に 0xFFA0 0000 または 0xFFA0 8000 が設定されます。SYSCR レジスタビット 4 (ソフトウェア・リセット時にブートなし)がセットされ、かつソフトウェア・リセットが発行されると、プロセッサは EVT1 レジスタ内に設定されたアドレスに分岐します。このリセット・ベクタはランタイム時に別のアドレスに再設定できるため、ソフトウェア・リセットの後にアプリケーションは 0xFFA0 0000 または 0xFFA0 8000 以外のアドレスへ分岐することができます。ランタイム時にリセット・ベクタを変更する場合には、EVT1 レジスタ内のリセット・ベクタ・アドレスが有効な命令アドレスであることを確認してください。このアドレスとしては、内部命令メモリ、SDRAM メモリ、非同期メモリが可能です。EVT1 レジスタにはデフォルト値はありません。このレジスタの値は、リセットが発行された後も保持されます。BMODE = 00 の場合、内蔵ブート ROM がバイパスされるため、EVT1 レジスタを初期化した後にソフトウェア・リセットを発行する必要があります。

- **ビット 3: INIT**—初期化ブロック(Init Block)は、実際のアプリケーション・コードがブートする前に実行されるコード・ブロックです。内蔵ブートROMがInit Blockを検出すると、そのブロックを内部メモリ内でブートさせ、そのブロックに対してCALLを実行します(Initコードの終わりにはRTSが必要)。Initコードを実行した後、一般にアプリケーション・コードで上書きされます。図 5を参照してください。
- **ビット 4: IGNORE**—メモリ内でブートしないブロックを表示します。ブートROMに、ブート・ストリームのCOUNTバイトをスキップするように指示します。マスター・ブート・モードでは、ブートROMはソース・アドレス・ポインタを変更することだけが可能です。スレーブ・ブート・モードでは、ブートROMはペイロード・データを廃棄する必要があります。現在のVisualDSP++®ツールは、グローバル・ヘッダーのIGNOREブロックのみをサポートしています(現在は 4 バイトのDXE Count、マルチアプリケーション(multi-DXE)の管理のセクション参照)。
- **ビット 8:5: PFLAG**—これらのビットは、SPIスレーブ・モード・ブート(BMODE = 10)に使用されます。PFLAGは、Blackfinプロセッサからマスタ SPIホストへのホスト・ウエイト(HWAIT)信号に使用されるPFx番号を表示します。この値としては、ADSP-BF531/BF532/BF533 プロセッサに対して 1～15 (0x1～0xF)を使用することができます。このPFストローブの使い方の詳細については、マスタ・ホスト経由のSPIスレーブ・モード・ブート(BMODE = 10)のセクションを参照してください。
- **ビット 15: FINAL**—このブロックの後でブート・プロセスが完了することを表示します。FINAL ブロックを処理した後、内蔵ブート ROM は EVT1 レジスタに格納されているリセット・ベクタ・アドレスへジャンプします。プロセッサはまだスーパーバイザ・モードにあるため、コード実行のために L1 メモリにジャンプしたとき最下位の割り込み順位(IVG15)にあります。

ADSP-BF531/BF532/BF533 プロセッサは ADSP-BF535 プロセッサとは異なり、セカンド・ステージ・ローダを必要としません。10 バイト・ヘッダーの FLAG フィールドは、セカンド・ステージ・ローダなしでシングル・ステージ・ブート・シーケンスを実行するために必要なすべての情報を ADSP-BF531/BF532/BF533 プロセッサに提供します。

初期化コード(Initコード)

Init コードは、実際のアプリケーションがブートされる前にコードの実行を可能にする機能です。このコードは、SDRAM コントローラの初期化、または PLL 設定の変更、SPI ボー・レートの変更、高速なブート時間への EBIU ウエイト状態の変更などの多くの目的に使用することができます。Init コードは、elfloader の`-Init Init_Code.DXE` コマンドライン・スイッチを使ってローダ・ファイル・ストリームの先頭に追加されます。ここで、`Init_Code.DXE` はユーザ定義のカスタム Init コード実行可能形式です。

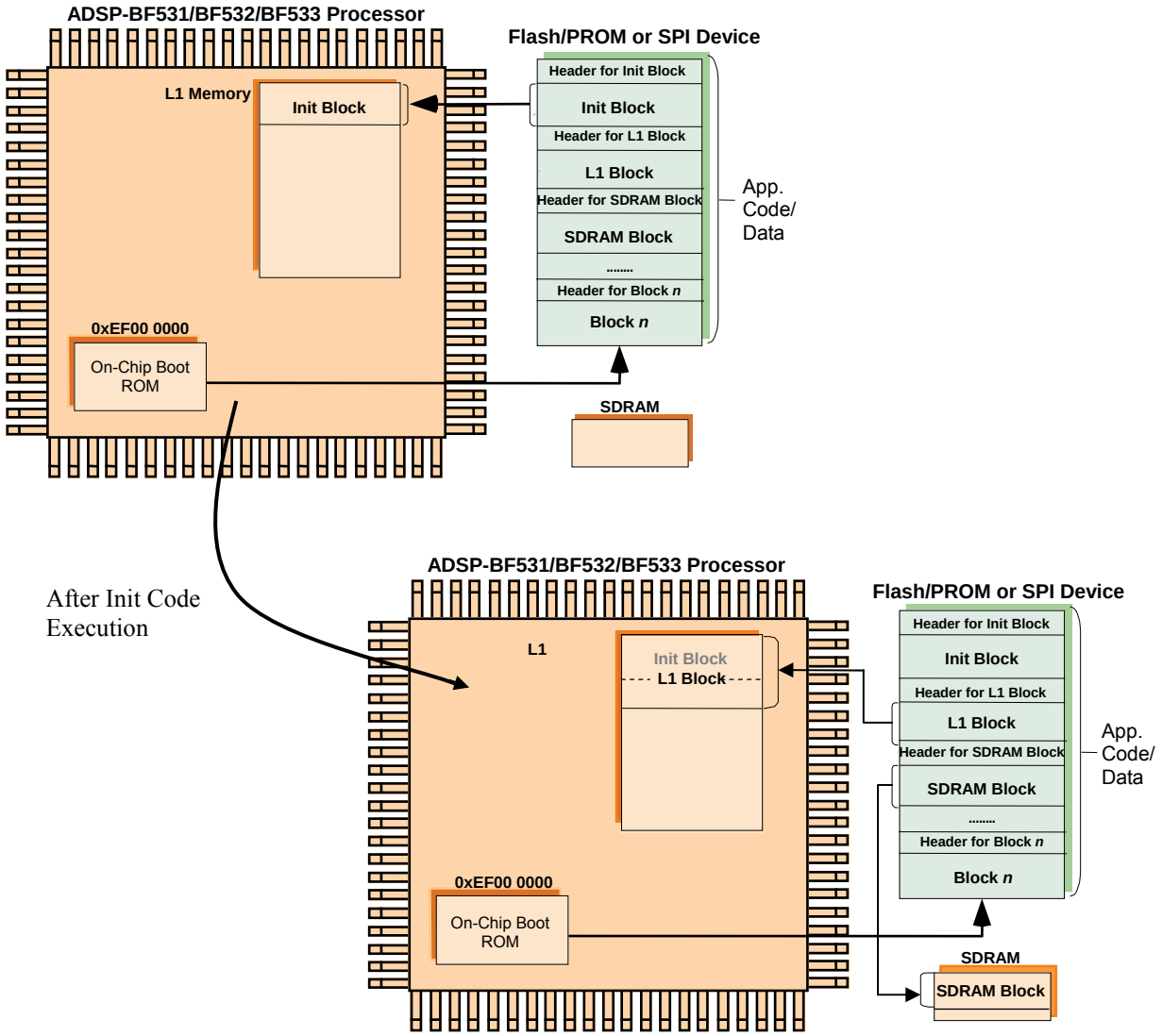


図5.Init コードの実行/ブート

INIT ビットがセットされたブロックを内蔵ブート ROM が検出すると、まず Blackfin メモリでブートし、次にターゲット・アドレスへ CALL を発行することによりそれを実行します。このため、RTS 命令を使って Init コードを終了させて、プロセッサがブート・プロセスの残りを実行するために内蔵ブート ROM へ確実に戻るようにする必要があります。

Initコードにより変更されるすべてのプロセッサ・レジスタを待避させ、Initコードのリターンの前にこれらのレジスタを回復させるのはユーザの責任で行う必要があります。少なくとも、すべてのInitコードで、ASTATレジスタ、RETSレジスタ、すべてのRxとPxレジスタを待避させることが推奨されます。Blackfinプロセッサでは、スクラッチパッド・メモリに十分なスタック・スペースを確保しています(0xFFB0 0000～0xFFB0 0FFF)。Initコードでは、スタック・ポインタSPを使ってプッシュ命令とポップ命令を実行することができます。リスト 1に、SDRAMコントローラのセットアップを行うInitコード・ファイルの例を示します。

```
#include <defBF532.h>
.section program;
/*********************************************************************/
    [--SP] = ASTAT;           // Save registers onto Stack
    [--SP] = RETS;
    [--SP] = (R7:0);
    [--SP] = (P5:0);
/*********************************************************************/
/*******Init Code Section*********************************************/
/*******SDRAM Setup************/
Setup_SDRAM:
    P0.L = lo(EBIU_SDRRC);
    P0.H = hi(EBIU_SDRRC);    // SDRAM Refresh Rate Control Register
    R0 = 0x074A(Z);
    W[P0] = R0;
    SSYNC;

    P0.L = lo(EBIU_SDBCTL);
    P0.H = hi(EBIU_SDBCTL);   // SDRAM Memory Bank Control Register
    R0 = 0x0001(Z);
    W[P0] = R0;
    SSYNC;

    P0.L = lo(EBIU_SDGCTL);
    P0.H = hi(EBIU_SDGCTL);   // SDRAM Memory Global Control Register
    R0.H = 0x0091;
    R0.L = 0x998D;
    [P0] = R0;
    SSYNC;
/*********************************************************************/
    (P5:0) = [SP++];          // Restore registers from Stack
    (R7:0) = [SP++];
    RETS = [SP++];
    ASTAT = [SP++];
/*********************************************************************/
    RTS;
```

リスト 1. Init コードの例

一般に、Init コードはシングル・セクションで構成され、ブート・ストリーム内のシングル・ブロックで表されます。このブロックではもちろん、INIT ビットがセットされています。それでも、Init ブロックは複数のセクションから構成することもできます。その場合、複数のブロックがブート・ストリーム内で Init コードを表します。最後のブロックでのみ INIT ビットがセットされます。elfloader ユーティリティで、これらのブロックの最後が Init コードのエントリ・アドレスへ分岐することが保証されます。これが elfloader にとって厳し過ぎる場合は、最終ブロックの INIT ビットをクリアしたままにして、後でブロックを追加発行してください。この追加ブロックで INIT ビットはセットしますが、ペイロード・データは与えません(COUNT = 0)。内蔵ブート ROM に、指定したアドレスへの CALL 命令の実行を指示します。

Init コード.DXE ファイルはそれらの固有 VisualDSP++プロジェクトで構築されますが、標準プロジェクトとは異なります。Init コードは、呼出し可能なサブ関数のみを提供するため、アプリケーションというよりはライブラリに似ています。Init コードは常に、通常のアプリケーション・コードの先頭になります。したがって、Init コードの構成が 1 個かまたは複数のブロックであるかに無関係に、ブート ROM にブート・プロセスを終了させる FINAL ビット・インジケータにより終了させられることはありません。

マルチアプリケーション(multi-DXE)の管理

プリブート初期化の他に、Initコード機能はブート管理にも使用することができます。elfloaderコマンドラインに複数の実行可能形式(.DXEファイル)が指定されると、ローダ・ファイル(.LDR)は複数のアプリケーションを格納することができます。elfloaderは、複数のブート・ストリームを生成し、各実行可能形式が順次続き、InitコードDXEが先頭にロードされます(図 6参照)。

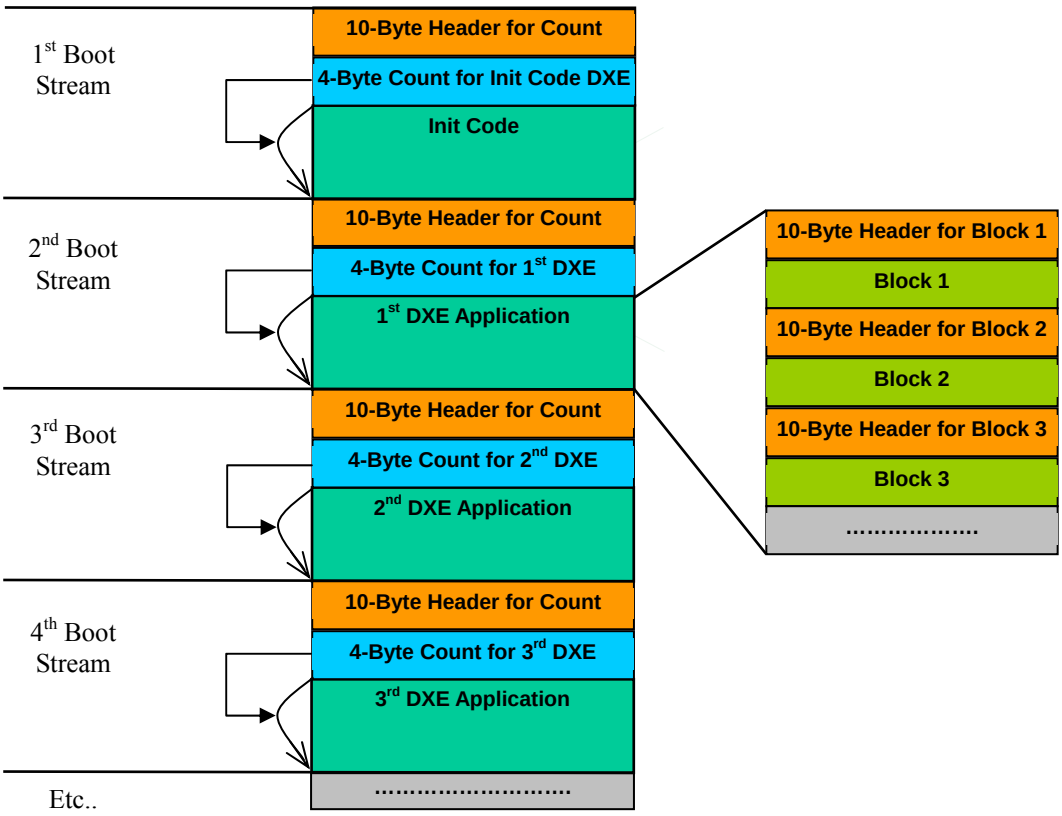


図6.multi-DXE ローダ・ファイルの内容

ADSP-BF531/BF532/BF533 ローダ・ファイル(.LDR)構造を使うと、外部メモリ内に格納されている複数の実行可能形式(DXE アプリケーション)の境界を指定することができるため、特定の DXE アプリケーションでブートすることができます。.LDR ファイル内で解析され配置された各.DXE ファイルの前には、IGNORE ブロックがあります。現在、この IGNORE ブロックは 4 バイト・カウント値で構成されており、DXE アプリケーション内に含まれるバイト数(ヘッダーも含む)を表しています。すなわち、次の DXE アプリケーションまでのオフセットです。将来、IGNORE ブロックは 4 バイト・カウント値の他の情報も保持するために使われます。各 IGNORE ブロックの前は 10 バイト・ヘッダーであることに注意してください。
このDXEカウント情報を使うと、.LDRファイル内で全DXEアプリケーションに分岐することができるため、ブート用に選択したDXEアプリケーションに到達することができます。リスト 2に、8 ビット・フラッシュ内の複数のDXEアプリケーションで分岐する方法を示すInitコードの例を示します。.LDRファイルに 1 つのInitコードDXEと 2 つのDXEアプリケーションが含まれるとすると、Initコードは.LDRファイル内でジャンプして、2 番目のDXEアプリケーションでブートします。

```
#include <defbf533.h>
.section program;
    [--SP] = ASTAT;          // save registers onto Stack
    [--SP] = RETS;
    [--SP] = (r7:0);
    [--SP] = (p5:0);
    [--SP] = LC0;
    [--SP] = LT0;
    [--SP] = LB0;
/*****************************/
BOOT_DXE:
    R0.H = 0x2000;           // R0 = start of ASYNC Bank 0
    R0.L = 0x0000;
    P1 = 2;                  // Number of DXEs to jump over (CANNOT BE ZERO!!)
                             // After first iteration, R0 will point to DXE1
                             // After second iteration, R0 will point to DXE2
    LSETUP(ADD_DXE_COUNT_BEGIN, ADD_DXE_COUNT_END) LC0 = P1;
ADD_DXE_COUNT_BEGIN:
    R1 = 0xA;                // Skip over 10 bytes for the 1st 10-byte header
    R1 = R1 << 1;            // Multiply by 2 since we are booting from a 16-bit
                             // flash (compensate for zero padding)
    R0 = R0 + R1;
    P0 = R0;                 // P0 points to 4-Byte DXE COUNT
    R0 = W[P0++](Z);         // R0 = xx | Bits[7:0] of DXE COUNT
    R1 = W[P0++](Z);         // R1 = xx | Bits[15:8] of DXE COUNT
    R1 = R1 << 8;
    R2 = W[P0++](Z);         // R2 = xx | Bits[23:16] of DXE COUNT
    R2 = R2 << 16;
    R3 = W[P0++](Z);         // R3 = xx | Bits[31:24] of DXE COUNT
    R3 = R3 << 24;
    R0 = R0 | R1;            // R0 = Bits[15:0] of DXE COUNT
    R2 = R2 | R3;            // R2 = Bits[31:16] of DXE COUNT
    R3 = R0 | R2;            // R3 = DXE COUNT
    R0 = P0;
    R0 = R0 + R3;            // Modify pointer by the DXE COUNT so now R0 points
    P0 = R0;                 // to next DXE
ADD_DXE_COUNT_END:
    NOP;
/*****************************/
DONE:
    LB0 = [SP++];            // Restore Regs
```

```
    LT0 = [SP++];
    LC0 = [SP++];
    (p5:0) = [SP++];
    (r7:1) = [SP++];          //----->DO NOT RESTORE R0<-------
    RETS = [SP++];            // Modify SP by one for R0 case
    RETS = [SP++];            // Pop off the real value of RETS
    ASTAT = [SP++];
    RTS;
```

リスト 2.multi-DXE ブートの Init コード例

データ・レジスタ R0 は Init コードの終わりで回復されていないことに注意してください。これは、R0 がフラッシュ/PROM ブートの外部ポインタであるためです(BMODE = 01)。プロセッサが RTS 命令の後で内蔵ブート ROM に戻ると、内蔵ブート ROM は R0 に格納されているロケーションからブートを継続します。同様に、ブート・モードが SPI ブートに設定された場合(BMODE = 11)、外部ポインタが R3 に格納されます。このため、SPI ブートの場合、multi-DXE アプリケーションの Init コード内で R3 は回復されません。
この EE ノートに添付されているのは、ADSP-BF533 EZ-KIT Lite®ボードを使用する multi-DXE ブート例(BF533 Ez Kit Multiple DXE Boot.zip)です。ZIP ファイルには、2 つのブリンク・アプリケーション・プロジェクトと 1 つの Init コード・プロジェクトが含まれています。RESET 後に、内蔵ブート ROM は Init コードでブートします。Init コードは、その後 PF8 (SW4 プッシュ・ボタン)または PF9 (SW5 プッシュ・ボタン)がアサートされるのを待ちます。PF8 がアサートされると、DXE1 でブートして実行されます。PF9 がアサートされると、DXE2 でブートして実行されます。DXE1 は、ボード上の LED を交互にブリンクさせるアプリケーションで、DXE2 はボード上のすべての LED をブリンクさせるアプリケーションです。

特定のBlackfinブート・モード

ここまでは、ADSP-BF531、ADSP-BF532、ADSP-BF533プロセッサのブート・プロセスの概要を説明してきました。このEEノートの残りの部分では、ハードウェア・インターフェース、ローダ・ファイル構造、予測されるピン動作のような各ブート・モードに関連する情報について説明します。外部16ビット・メモリ(BMODE = 00)からの実行については、「*Running Programs from Flash on ADSP-BF533 Blackfin Processors (EE-239)* [3]」で説明しています。
次のセクションでは、ADSP-BF533 EZ-KIT Lite ボードの ASM ブリンク例と Init コードを使用します。この例はこの EE ノートに添付されています。

下記の各ブート・モードでは、アドレス 0xFF80 7FF0～0xFF80 7FFF (L1 データ・バンク A の最後の 16 バイト)は予約されています。このメモリ範囲は、ローダ・ファイル内で各ブロックのヘッダー情報を格納するために内蔵ブート ROM が使います。ブート後には、このメモリ範囲はランタイム時にアプリケーションから使うことができます。シリコン・レビジョン 0.1 と 0.2 についてはアペンディックスを参照してください。

8 ビット・フラッシュ/PROMブート(BMODE = 01)

BlackfinプロセッサのEBIUは 16 ビット幅であるため(ADDR[0]はありません)、8 ビット・フラッシュ/PROMはデータ・バス(D[7:0])の下位 8 ビットのみを使用します。図 7に、Blackfinプロセッサと 8 ビット・フラッシュ/PROMのピン間接続を示します。

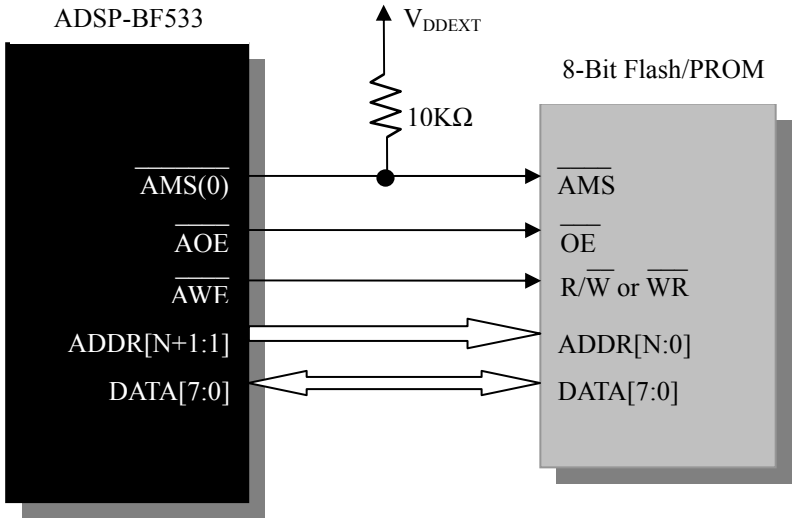
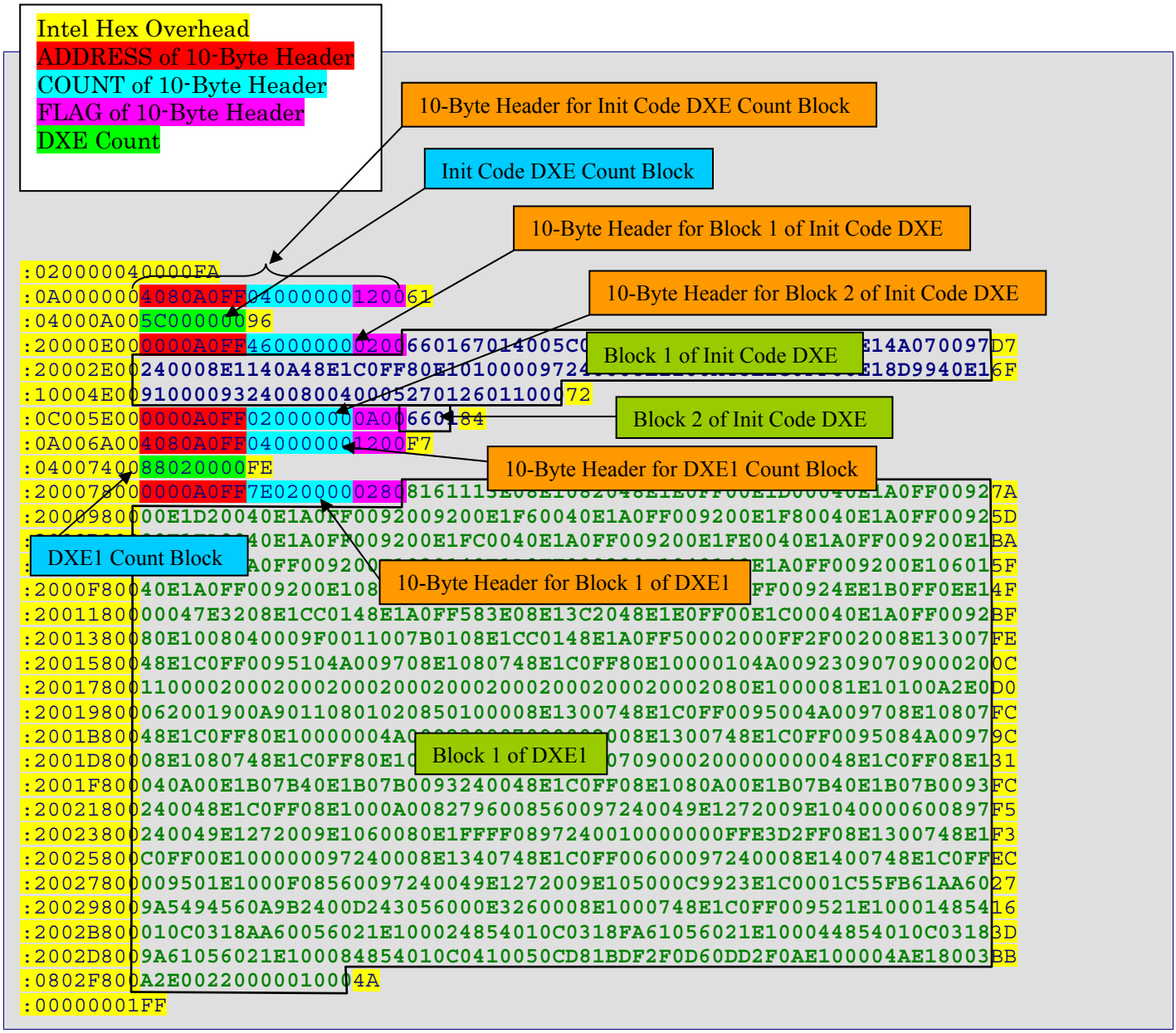


図7.Blackfin プロセッサと 8 ビット・フラッシュ/PROM との間の接続

リスト 3 に、IntelHexフォーマットで生成された 8 ビット・フラッシュ/PROMのローダ・ファイルを示します(このEEノートに添付されているexample.zip)。ローダ・ファイルの構造を説明するために、複数のセクションに分割しています。



リスト 3.IntelHex ローダ・ファイルの例

リスト 3のローダ・ファイルは、シリコン・レビジョン 0.3 に対してビルドしたものです。シリコン・レビジョン 0.2 以下に対してビルドされたローダ・ファイルは少しローダ・ファイル構造が違っています。

このローダ・ファイルをBlackfinプロセッサのASYNCバンク 0 に接続された 8 ビット・フラッシュに書き込むと、Blackfinから見たメモリの値(ロケーション 0x2000 0000 から開始)は 図 8の様に見えます。

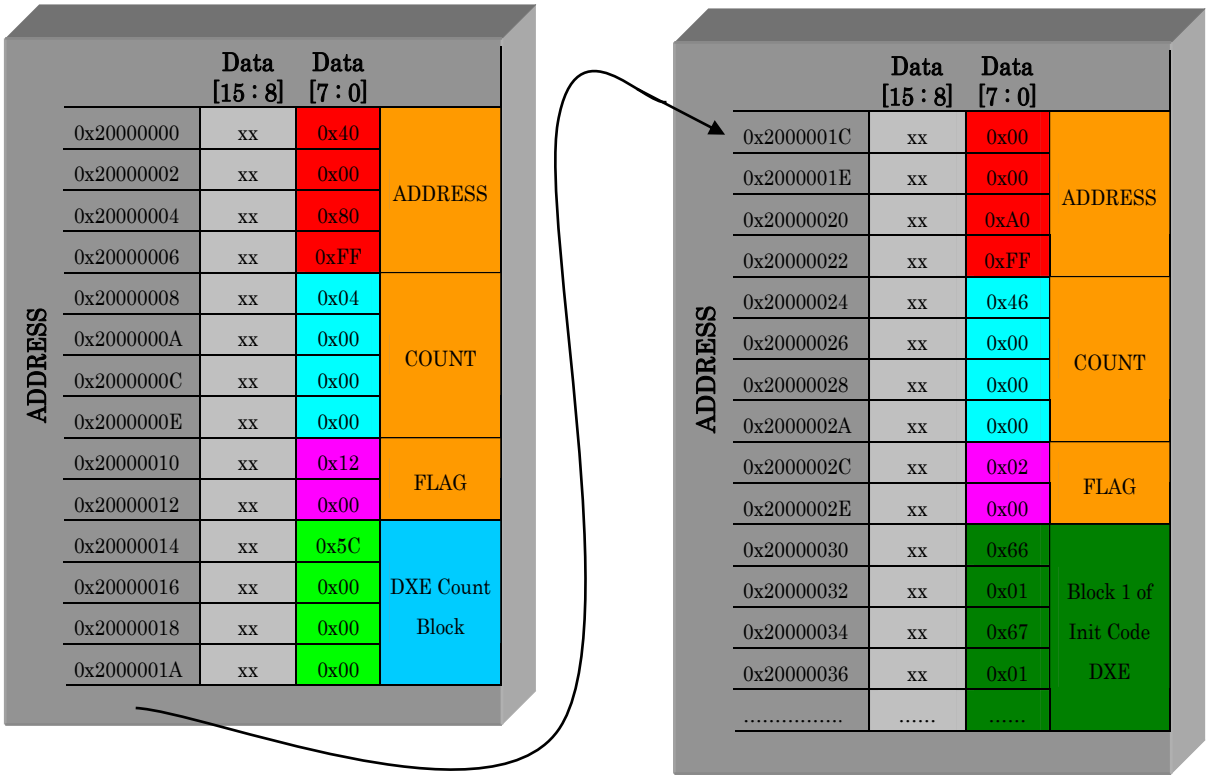


図8.Blackfin メモリ・ウインドウから見た 8 ビット・フラッシュ/PROM メモリの値

図 9 に、8 ビット・フラッシュ/PROMブートのブート・シーケンスの開始を示します。

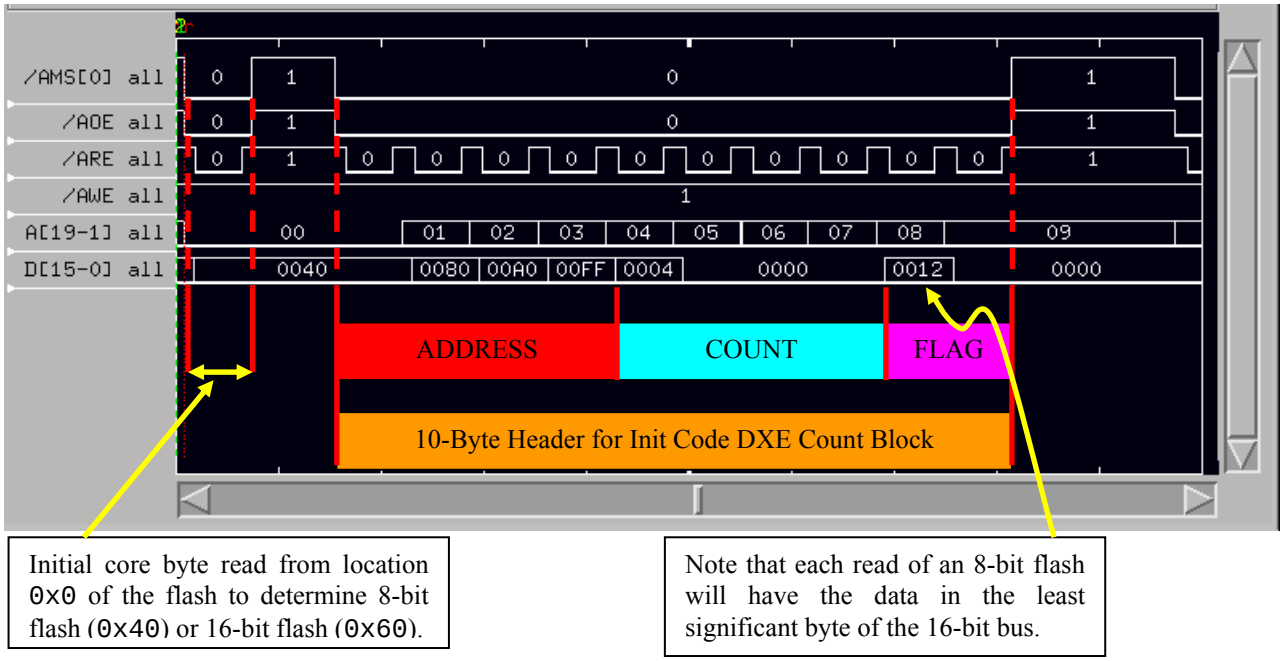


図9. 8 ビット・フラッシュ・ブート・シーケンスのタイミング図

16 ビット・フラッシュ/PROMブート(BMODE = 01)

図 10 に、Blackfinプロセッサと 16 ビット・フラッシュ/PROMとの間のピン間接続を示します。

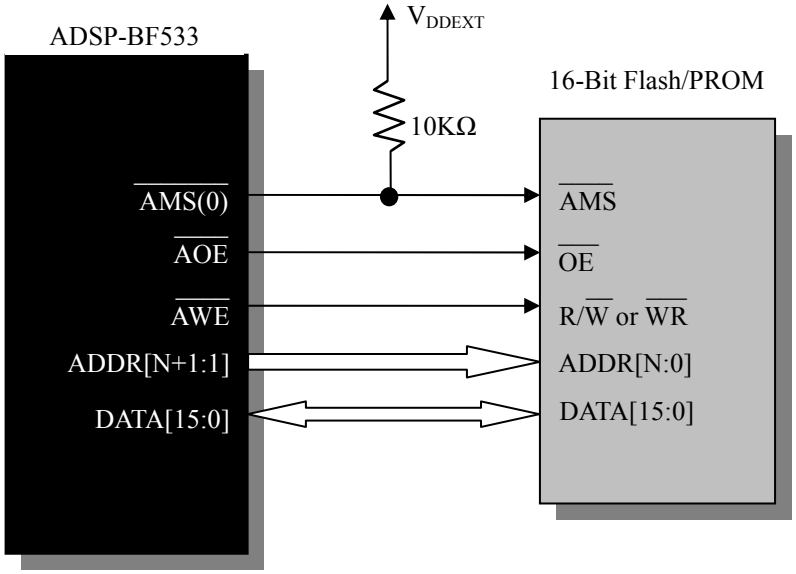


図10.Blackfin プロセッサと 16 ビット・フラッシュ/PROM との間の接続

このEEノートに添付されている例から 16 ビット・フラッシュ/PROMのローダ・ファイルを生成する場合、リスト 3 に示す内容と全く同じですが、DXEカウント・ブロックの 10 バイト・ヘッダーのアドレスが、8 ビット・フラッシュ/PROMの場合の `0xFF80 0040` の代わりに `0xFF80 0060` となる点が異なります。このため、ローダ・ファイルの先頭バイトは `0x40` の代わりに `0x60` になります。内蔵ブートROMは、この先頭バイトを使って、8 または 16 ビットのいずれのフラッシュ/PROMが接続されているかを判断します。先頭バイトが `0x60` の場合 16 ビット・フラッシュ/PROMと見なし、先頭バイトが `0x40` の場合 8 ビット・フラッシュ/PROMと見なします。
このローダ・ファイルをBlackfinプロセッサのASYNCバンク 0 に接続された 16 ビット・フラッシュに書き込むと、Blackfinプロセッサから見たメモリの値(ロケーション `0x2000 0000` から開始)は 図 11の様に見えます。

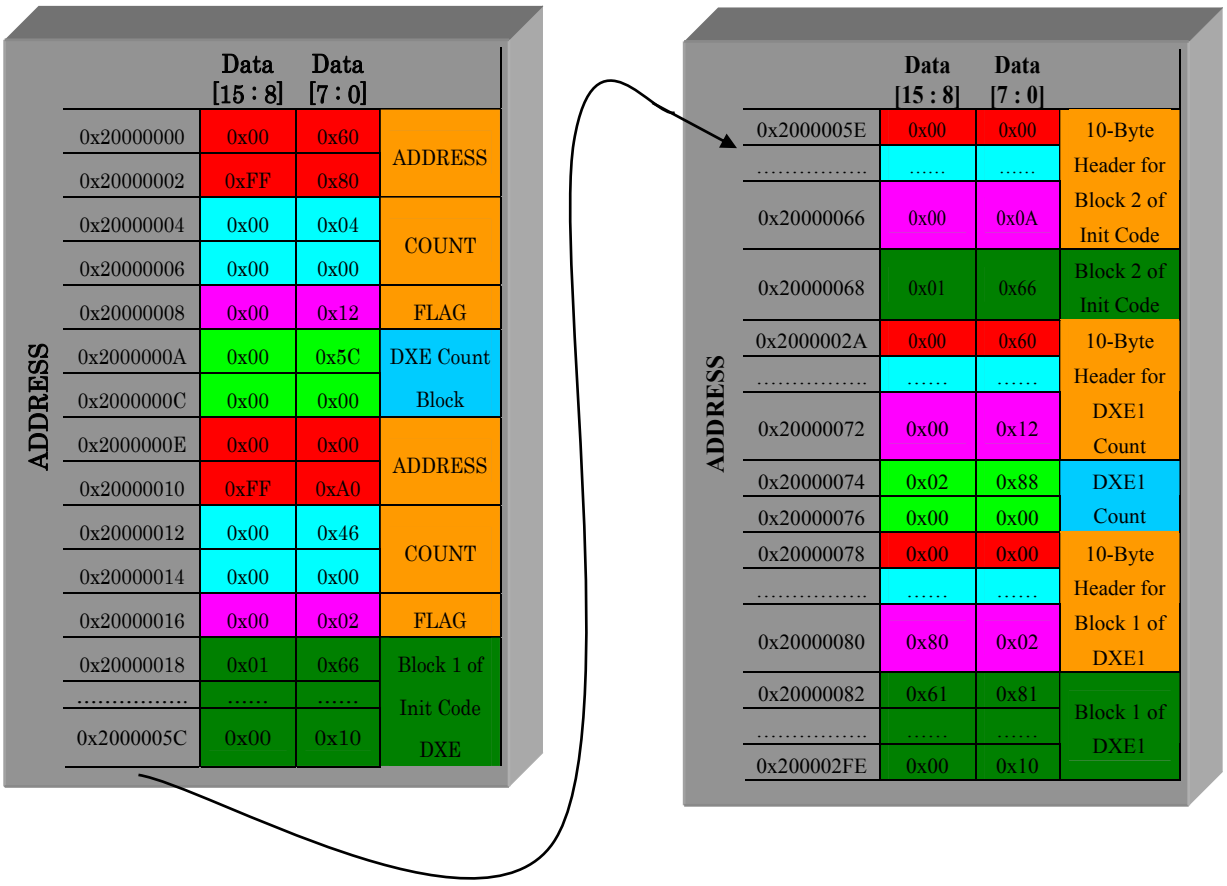


図11.Blackfin メモリ・ウインドウから見た 16 ビット・フラッシュ/PROM メモリの値

16 ビット・フラッシュ/PROMのローダ・ファイル構造(図 11)は、シリコン・レビジョン 0.3 以降でのみサポートされています。ADSP-BF531/BF532/BF533 のシリコン・レビジョン 0.2 以前では、8 ビット・ブートのみをサポートしています。したがって、シリコン・レビジョン 0.2 以前で 16 ビット幅を選択すると、出力ローダ・ファイルは、16 ビット・フラッシュ/PROMからの 8 ビット・ブートを"シミュレート"するためゼロを埋め込んだファイルになります。このローダ・ファイルをフラッシュ/PROMメモリに書き込むと、データ・バスの上位 8 ビット(DATA[15:8])にゼロを埋め込んだ図 8のように見えます。

図 12 に、16 ビット・フラッシュ/PROMブートのブート・シーケンスの開始を示します。

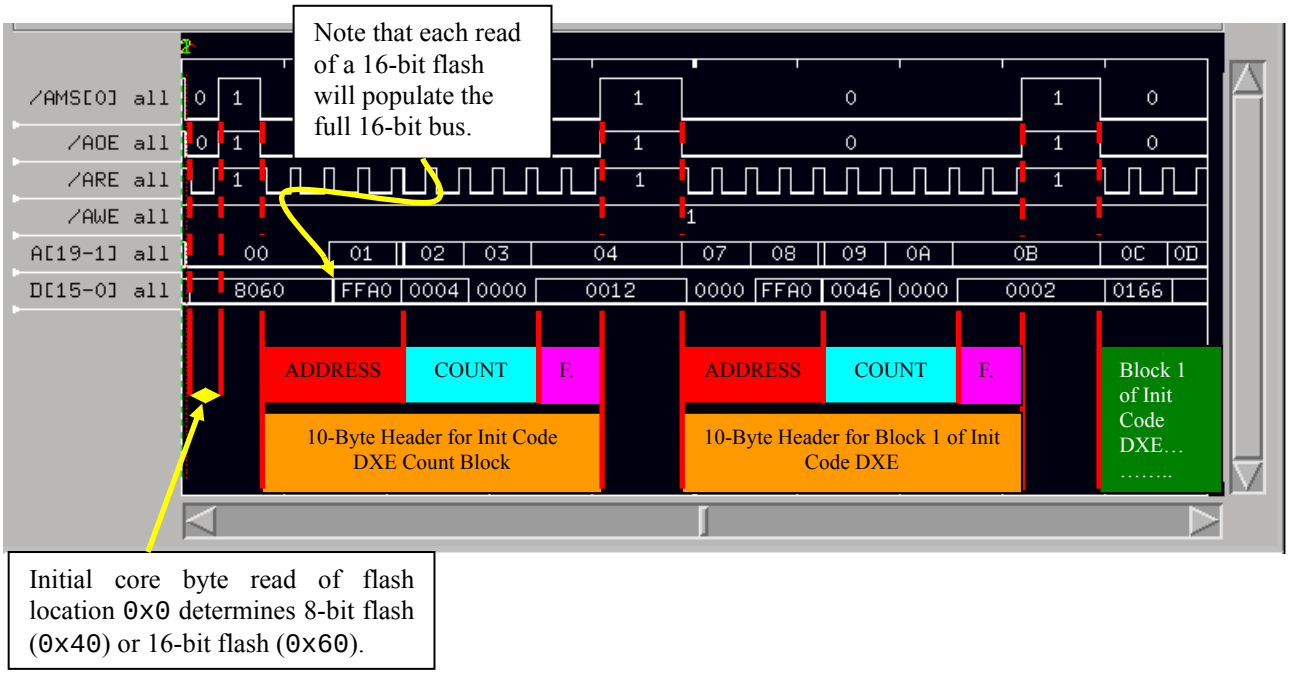


図12.16 ビット・フラッシュ・ブート・シーケンスのタイミング図

プロセッサはフラッシュのロケーション 0x0 から初期コア・バイト読み出しを行って、フラッシュのメモリ幅を判断します。FIFO からブートするときは、先頭バイト(ローダ・ファイル内の先頭の 10 バイト・ヘッダーの一部)は 2 回送信される必要があります。1 回はこの初期コア読み出し、もう 1 回は実際のブート・シーケンスです。

マスタ・ホスト経由のSPIスレーブ・モード・ブート(BMODE = 10)

SPI スレーブ・モード・ブートの場合、ADSP-BF531/BF532/BF533 は SPI スレーブ・デバイスに設定され、ホストを使ってプロセッサをブートさせます。

このブート・モードは、ADSP-BF531/BF532/BF533 プロセッサのシリコン・レビジョン 0.2 以前ではサポートされていません。

図 13 に、このモードに必要なピン間接続を示します。ホストはローダ・ファイル・ストリームのアクノリッジを必要とせずに、Blackfinプロセッサをブートさせます。ホストは、ローダ・ファイル(ASCIIフォーマット)から 1 バイトずつ送信するように設定する必要があります。このセットアップでは、PFxがBlackfinプロセッサからマスター・ホスト・デバイスへのホスト・ウエイト(HWAIT)信号になります。この信号は、ブート・プロセスの所定区間(特にinitコード実行時とゼロ・フィル・ブロック時)にホストを“ホールド・オフ”させるためにBlackfinプロセッサが使用する信号になります。 PFxがアサートされると(ハイ・レベル)、マスター・ホスト・デバイスはBlackfinプロセッサへのバイトの送信を中断する必要があります。PFxがデアサートされたら(ロー・レベル)、マスター・ホスト・デバイスはバイトの送信を中断されたところから再開します。スレーブは先頭ブロックの処理を完了するまでPFxピンを駆動しないため、HWAIT信号をプルダウンする抵抗を使用してください。

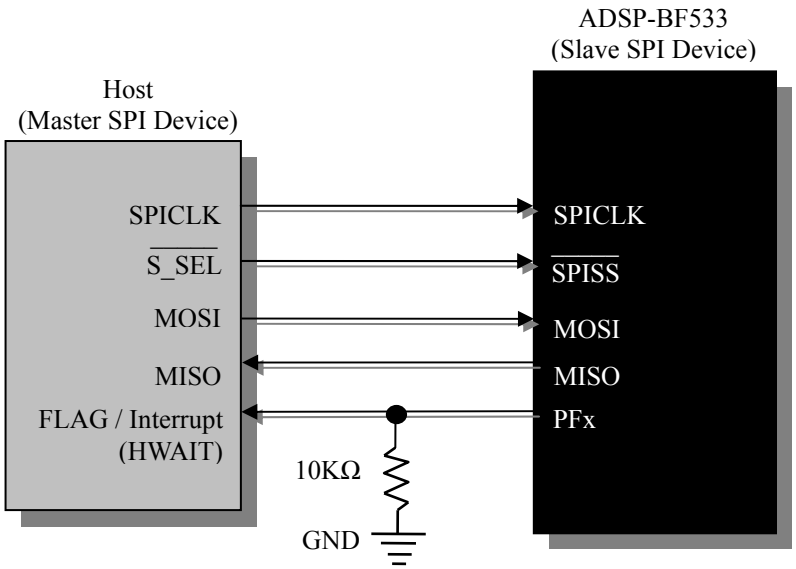


図13.ホスト(SPI マスタ)と Blackfin プロセッサ(SPI スレーブ)との間の接続

ホストは、バイトを送信する前に Blackfin がリセットから抜け出ていることを確認する必要があります。リセットの前に Blackfin プロセッサへ送信されたバイトは失われるため、ブート・シーケンスは失敗します。

この PFx 番号はユーザにより定義されて、ローダ・ファイル内に組み込まれます。elfloader ユーティリティが全 10 バイト・ヘッダーごとにこの番号を PFLAG ビット・フィールドに組み込みます(FLAG ワードのビット[8:5])。これは-pflag number コマンドライン・スイッチを使って行います。ここで、number は Blackfin スレーブが使用するための PFx フラグで、値は 1～15 です。

-pflag number スイッチを使用しない場合、FLAG のビット 8:5 内のデフォルト値は 0 になり、PF0 がホストに対する HWAIT 信号と見なされます。PF0 は/SPISS ピン(これは SPI スレーブ・ブートに必須なピン)と共用されるため、必ず-pflag スイッチを使用して 0 以外の値を指定してください。

Rev 0.3 ブート Rom に不具合があるため、SPI コントロール・レジスタと DMA5 コンフィギュレーション・レジスタにはブート後にデフォルト値が設定されません。

CCLK = 333 MHz、SCLK = 66.6 MHz を使用するスレーブを持つシステムで、ホストが最大 SPI ボーレート= 1 MHz で動作することがテストされました。
この EE ノートに添付されているホスト・コード例(Host_Code.zip)は、ADSP-BF532 プロセッサをホストとして使用した場合のものです。
ADSP-BF532 プロセッサをホストとして使用し、ADSP-BF533 プロセッサをスレーブ SPI デバイスとして使用した SPI スレーブ・モード・ブートのタイミング図を下図に示します。ホスト側では、PF4 を/CS として使用し、この信号はスレーブ ADSP-BF533 の/SPISS に接続されています。SPI スレーブ(ADSP-BF533 プロセッサ)の PF13 は、ホスト(ADSP-BF532 プロセッサ)の PF15 に接続されています。この接続は、ホストをホールド・オフさせる HWAIT 信号になります。すべてのタイミング図は、SPI スレーブ側から見たものです。

使用したローダ・ファイル(SPI_Slave_HostFile.ldr)は、リスト 3と同じですが、このブート・モードの機能を示すため 2 個のブロックが追加されている点が異なります。この追加された 2 ブロックは、ロケーション 0xFFA0 0300 でバイト数 0x4000 のゼロ・フィル・ブロックと、ロケーション 0xFFA0 4300 で 0x11、0x22、0x33、0x44、0x55、0x66、0x77、0x88、0x99、0xAA、0xBB、0xCC、0xDD、0xEE、0xFF、0x19 の値を持つデータ・ブロックです。
図14 に、SPIスレーブ・モードのブート・シーケンスを示します。

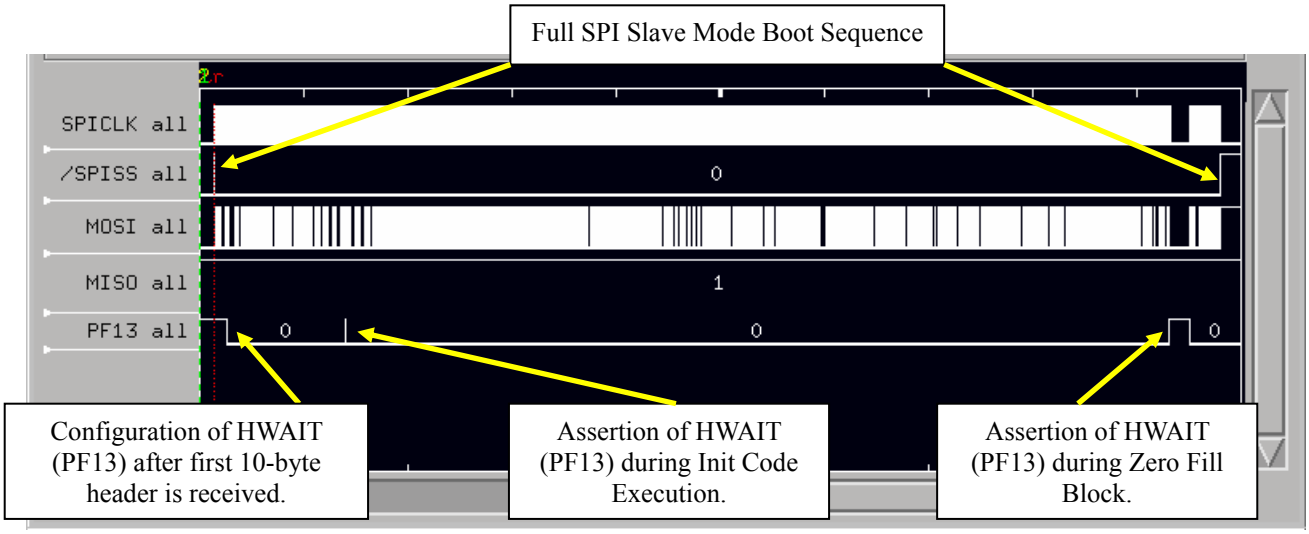


図14.SPI スレーブ・モード・ブート・シーケンスのタイミング図

SPIスレーブはホストから先頭の 10 バイト・ヘッダーを受信すると、HWAIT信号として設定されたPFxフラグを知ることができます。このケースでは、PF13 が使用されています。 図 15で、FLAGフィールドのビット 8:5 が処理された後にPF13 のアサーションが解除されていることに注意してください。

図 15は、PF13 にプルアップ抵抗を使用して、デバッグ目的のために取得しました。通常動作には、プルダウン抵抗の使用が推奨されます。この場合はPF13 は常にロー・レベルです。

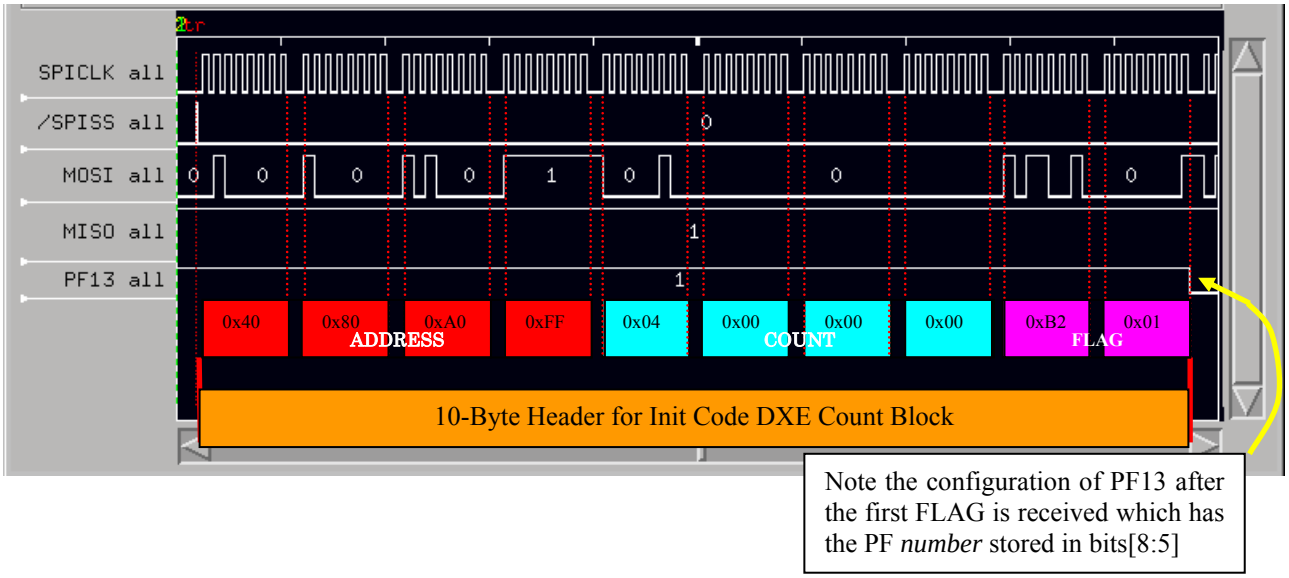


図15.SPI スレーブ・モード・ブート・シーケンス:ブート・シーケンスの開始

その後、ホストは 4 バイトの Init コード DXE カウント・ブロック、Init コード DXE ブロック 1 の 10 バイトのヘッダー、ブロック 1 自体を送信します。

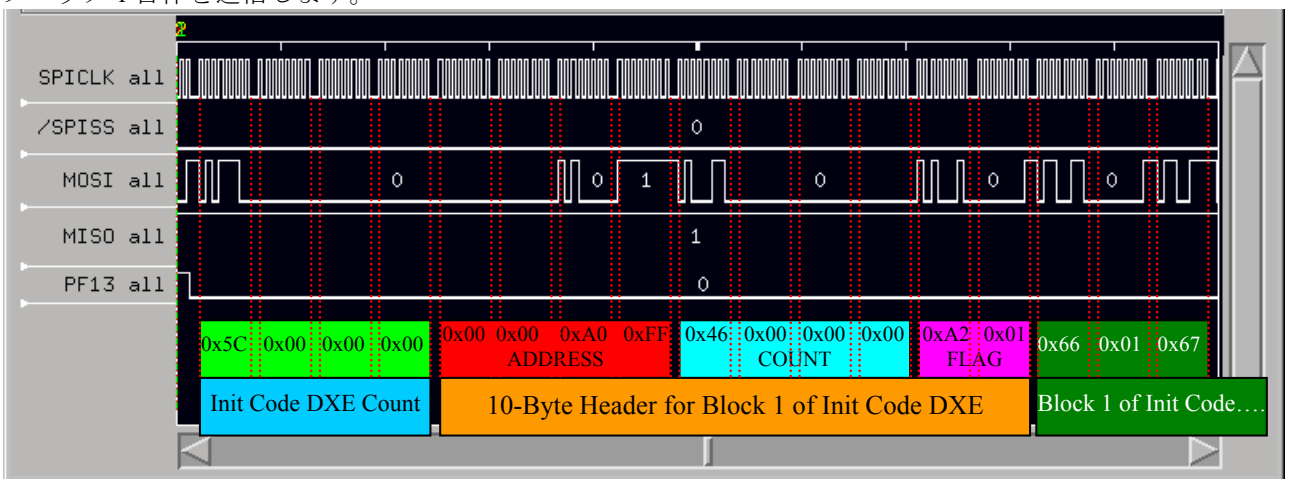


図16.SPI スレーブ・モード・ブート・シーケンス: Init コード DXE のブート・ブロック 1

Init コード DXE を Blackfin メモリへ書き込んだ後、スレーブ SPI Blackfin プロセッサは HWAIT 信号 PF13 をアサートして、Init コードを実行する間、バイトを送信しないようホストに通知します。Blackfin プロセッサ・コアは SPI インターフェースより高速動作しているため、ホストからバイトが送信される速度より高速に Init コードが実行されます。

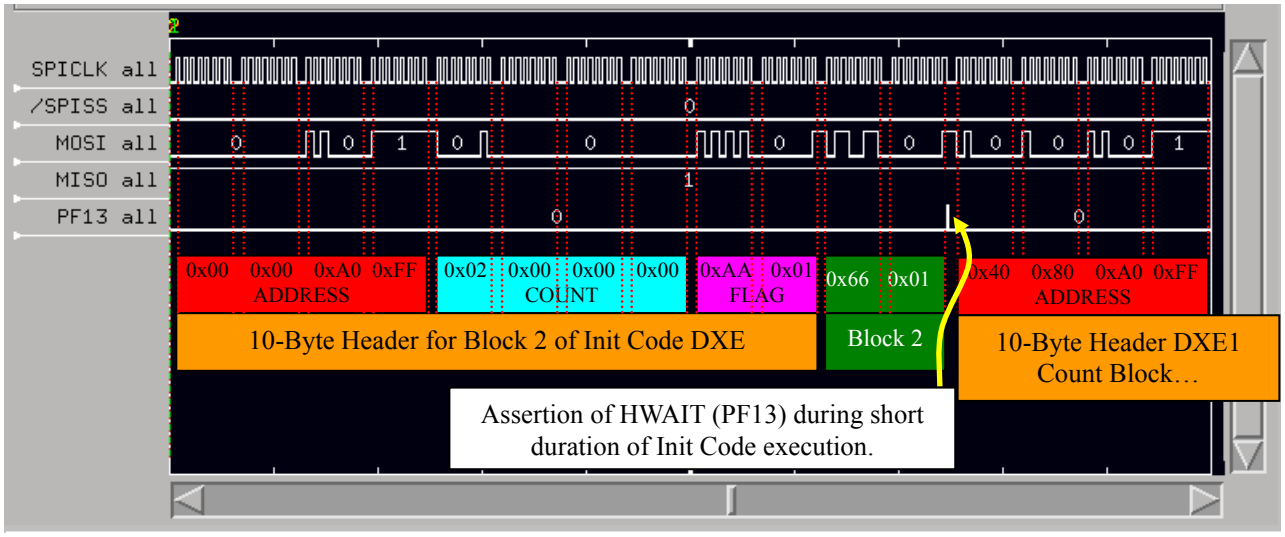


図17.SPI スレーブ・モード・ブート・シーケンス: Init コード DXE のブート・ブロック 2

図 18 に、このブート・モードのゼロ・フィル・ブロックの処理を示します。内蔵ブート ROM がゼロ・フィル・ブロックに遭遇すると、HWAIT(PF13)をアサートして、ホストのバイト送信を中断させます。この間、MemDMA を実行して 0x4000 のゼロをロケーション 0xFFA0 0300～0xFFA0 4300 に転送します。終了すると、内蔵ブート ROM は HWAIT のアサートを解除し、ホストはブート・プロセスの残りのバイト(ブロック 3 の 10 バイト・ヘッダーとブロック 3 自体)の送信を再開します。

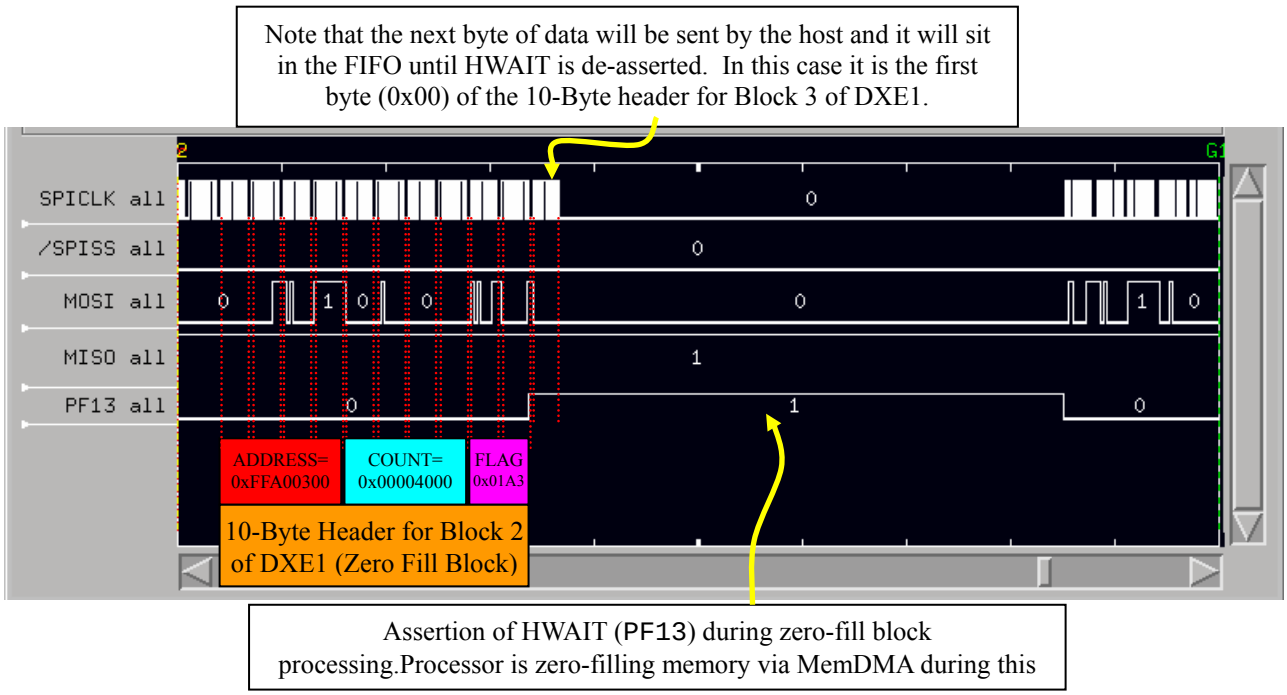


図18.SPI スレーブ・モード・ブート・シーケンス:ゼロ・フィル・ブロックの転送(DXE1 のブロック 2)

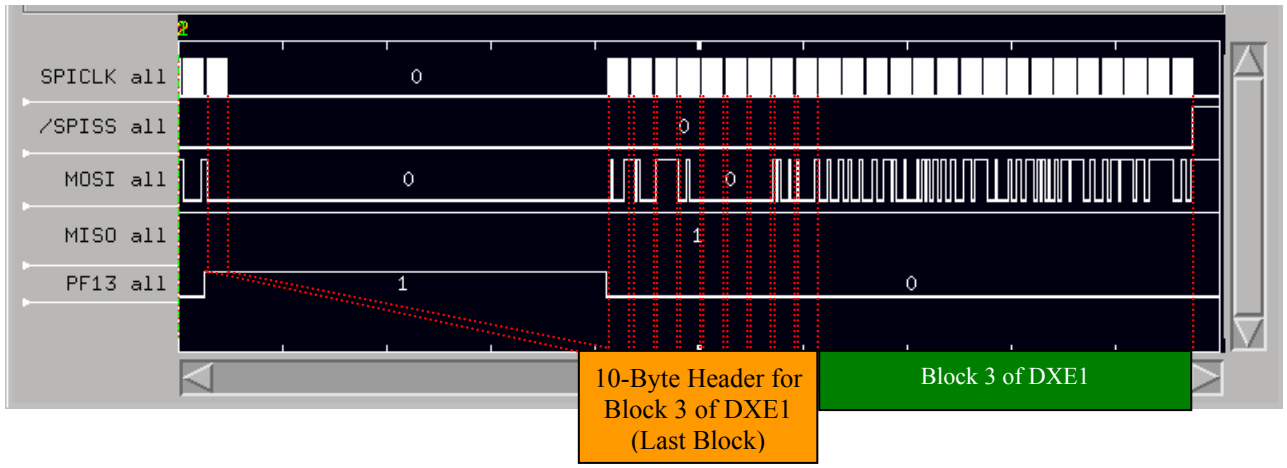


図19.SPI スレーブ・モード・ブート・シーケンス: DXE1 のブート・ブロック 3 (最終ブロック)

SPIメモリを使用するSPIマスタ・モード・ブート (BMODE = 11)

SPI マスター・モード・ブートの場合、ADSP-BF531/BF532/BF533 プロセッサは SPI マスターとして設定され、SPI メモリに接続されます。このモードに必要なピン間接続を下図に示します。

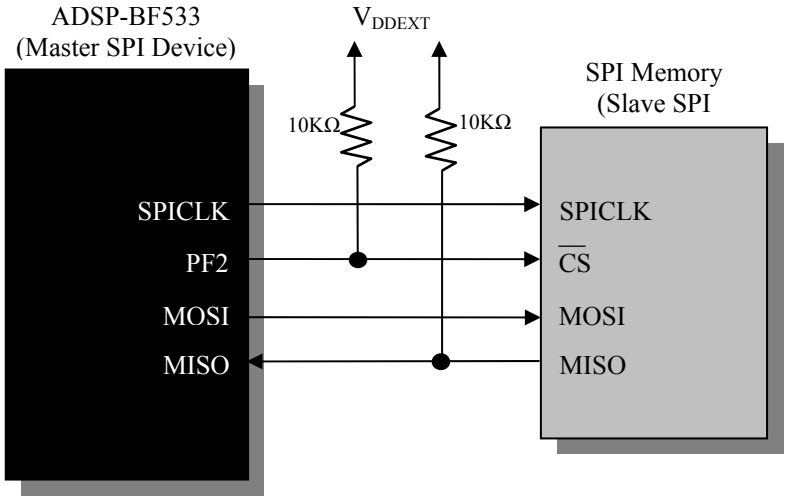


図20.Blackfin—SPI メモリ・ピン間接続

このブート・モードが動作するためには MISO のプルアップ抵抗*が必ず必要*です。このため、SPI メモリが応答しない場合(すなわち SPI メモリから MISO ピンにデータが書き込まれない場合)、ADSP-BF531/BF532/BF533 プロセッサは MISO ピンから 0xFF を読み込みます

MISO ラインにはプルアップ抵抗が必須ですが、プルアップ/プルダウン抵抗の追加も次の場合には意味があります。1) Blackfin プロセッサのリセット中、SPI メモリをアクティブにしないようにするため PF2 信号をプルアップする場合。2)オシロスコープ像を明瞭にするため SPICLK をプルダウンする場合。

シリコン・レビジョン 0.2 以前では、SPI コントロール・レジスタ(SPICTL)の CPHA ビットと CPOL ビットに 1 が設定されていました(これらのビットについては、「*Hardware Reference Manual* [2]」を参照してください)。このため、SPI メモリがスリー・ステートから戻るときクロック信号の誤った立ち上がりエッジを検出してしまうことがあります。このためにブート・プロセスが失敗する場合には、SPICLK 信号にプルアップ抵抗を接続するとこの問題を軽減することができます。シリコン・レビジョン 0.3 では、SPI コントロール・レジスタで CPHA = CPOL = 0 とすることによりこの問題は解決されています。シリコン・レビジョン 0.3 は、SPICLK のプルアップ抵抗に対して強化されています。したがって、いずれのシリコン・レビジョンを使うボードでも SPICLK を安全にプルアップすることができます。ただし、シリコン・レビジョン 0.3 では、オシロスコープで見た場合、PF2 のアサート解除中に SPICLK が予想外に高いレベルになることがあることに注意してください。

このインターフェースでサポートしているSPIメモリは、標準の8/16/24ビット・アドレサブルSPIメモリ(読み出しシーケンスを下記に説明)とAtmel社のSPI DataFlashデバイスAT45DB041B、AT45DB081B、AT45DB161B*です。

* この EE ノートに添付されているのは、ADSP-BF532 Blackfin プロセッサを使用して Atmeld 社の *DataFlash* デバイスに書き込みを行うコード例です(Program_Atmel.zip 参照)。

標準の8/16/24ビット・アドレサブルSPIメモリとは、0x03の読み出しコマンド・バイト、それに続いて1アドレス・バイト(8ビット・アドレサブルSPIメモリ)、2アドレス・バイト(16ビット・アドレサブルSPIメモリ)、または3アドレス・バイト(24ビット・アドレサブルSPIメモリ)を入力するメモリです。

読み出しコマンドとアドレスが送信されると、選択されたアドレスに格納されているデータがMISOピンからシフト出力されます。データは、そのアドレスから連続クロック・パルスによりシーケンシャルに出力されます。アナログ・デバイセズは、次の標準SPIメモリ・デバイスをテスト済みです。

- 8 ビット・アドレサブル SPI メモリ: Microchip 社の 25LC040
- 16 ビット・アドレサブル SPI メモリ: Microchip 社の 25LC640
- 24 ビット・アドレサブル SPI メモリ: STMicroelectronics 社の M25P80

SPIメモリ検出ルーチン

BMODE = 11では、種々のSPIメモリからのブートをサポートしているため、内蔵ブートROMは接続されているメモリ・タイプを検出します。プロセッサに接続されているメモリ・タイプ(8、16、または24ビット・アドレサブル)を調べるため、内蔵ブートROMはSPIメモリが応答するまで、メモリへ次のバイト・シーケンスを送信します。SPIメモリは、正しいアドレスを受け取るまで応答しません。内蔵ブートROMは次のステップを実行します。

1. MOSI ピンに読み出しコマンド 0x03 を送信して、MISO ピンのダミー読み出しを行います。

2. MOSI ピンにアドレス・バイト 0x00 を送信して、MISO ピンのダミー読み出しを行います。

3. MOSI ピンにさらにバイト 0x00 を送信して、MISO ピンの着信バイトが 0xFF (プルアップ抵抗からの値、次の注参照)以外であることをチェックします。着信バイトが 0xFF 以外の場合は、1 アドレス・バイトの後に SPI メモリが応答してきたことを意味し、8 ビット・アドレサブル SPI メモリ・デバイスが接続されていると見なします。

4. 着信バイトが 0xFF の場合は、内蔵ブート ROM が MOSI ピンにさらにバイト 0x00 を送信して、MISO ピンの着信バイトが 0xFF 以外であることをチェックします。着信バイトが 0xFF 以外の場合は、2 アドレス・バイトの後に SPI メモリが応答してきたことを意味し、16 ビット・アドレサブル SPI メモリ・デバイスが接続されていると見なします。

5. 着信バイトが 0xFF の場合は、内蔵ブート ROM が MOSI ピンにさらにバイト 0x00 を送信して、MISO ピンの着信バイトが 0xFF 以外であることをチェックします。着信バイトが 0xFF 以外の場合は、3 アドレス・バイトの後に SPI メモリが応答してきたことを意味し、24 ビット・アドレサブル SPI メモリ・デバイスが接続されていると見なします。

6. 着信バイトが 0xFF の場合(応答するデバイスがない場合)、内蔵ブート ROM は AT45DB041B、AT45DB081B、または AT45DB161B のいずれかの Atmel 社の DataFlash デバイスが接続されているものと見なします。これらの DataFlash デバイスは、前述の標準 SPI メモリの読み出しシーケンスと異なる読み出しシーケンスを持っています。さらに詳しい情報については、これらのデバイスのデータ・シート[4]、[5]、[6]を参照してください。内蔵ブート ROM は、ステータス・レジスタを読み出すことにより、上記の内のどの Atmel DataFlash メモリが接続されているかを調べることができます。上記 DataFlashes 間の主な違いは、ページあたりのバイト数です。AT45DB041B と AT45DB081B は 264 バイト/ページで、AT45DB161B は 528 バイト/ページです。これらのいずれが Blackfin に接続されているかを調べるために、内蔵ブート ROM はデバイス集積度ビットが配置されている DataFlash のステータス・レジスタを読み出します。デバイス集積度ビット= 1011 (バイナリ)の場合、内蔵ブート ROM は AT45DB161B が接続しているものと見なし、それに従ってデバイスをアドレス指定します。その他の場合は、AT45DB041B または AT45DB081B が接続されているものと見なし、適切にアドレス指定します。シリコン・レビジョン 0.3 の内蔵ブート ROM コードが生成された後、Atmel 社はさらに DataFlashe の派生品を導入しました。これら派生品を使う場合は、デバイスが 264 バイト/ページであることを確認してください。そうしないと、ブート・シーケンスは失敗します。

前述の SPI メモリ検出ルーチンの場合、シリコン・レビジョン 0.2 以前の内蔵ブート ROM は、`MISO` ピンの着信データが `0x00`(ローダ・ファイルの先頭バイト)であるか否かチェックします。シリコン・レビジョン 0.3 の内蔵ブート ROM は、`MISO` ピンの着信データが `0xFF` 以外であることをチェックします。このため、シリコン・レビジョン 0.2 以前用にビルドされた SPI ローダ・ファイルは先頭バイトが `0x00` である必要があります。シリコン・レビジョン 0.3 の場合、ローダ・ファイルの先頭バイトは `0x40` に設定されています。

SPIボーレート・レジスタは133に設定されます。これは、システム・クロックが54 MHzの場合、54 MHz/(2*133) = 203 kHzボー・レートになることを意味します。ADSP-BF533 EZ-KIT Liteボードでは、デフォルト・システム・クロック周波数は54 MHzです。

16ビット・アドレサブルSPIメモリ(Microchip社の25LC640)を使用したSPIマスタ・モード・ブートのブート・シーケンスを次の図に示します。使用したローダ・ファイルは、リスト 3と同じです。すべての図はシリコン・レビジョン 0.3を使って取得しました。
図21 に、SPIマスタ・モード・ブート・シーケンスを示します。

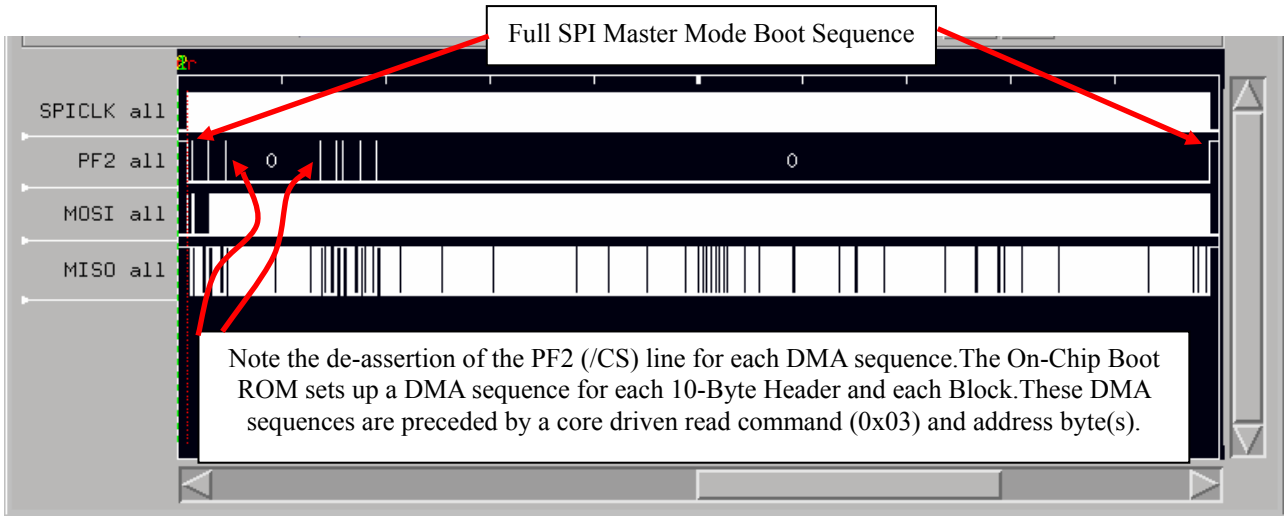


図21.SPI マスタ・モード・ブート・シーケンスのタイミング図

最初に、内蔵ブート ROM は接続されている SPI メモリ・タイプ(8/16/24 ビット・アドレサブルまたは Atmel 社の DataFlash)を調べます。

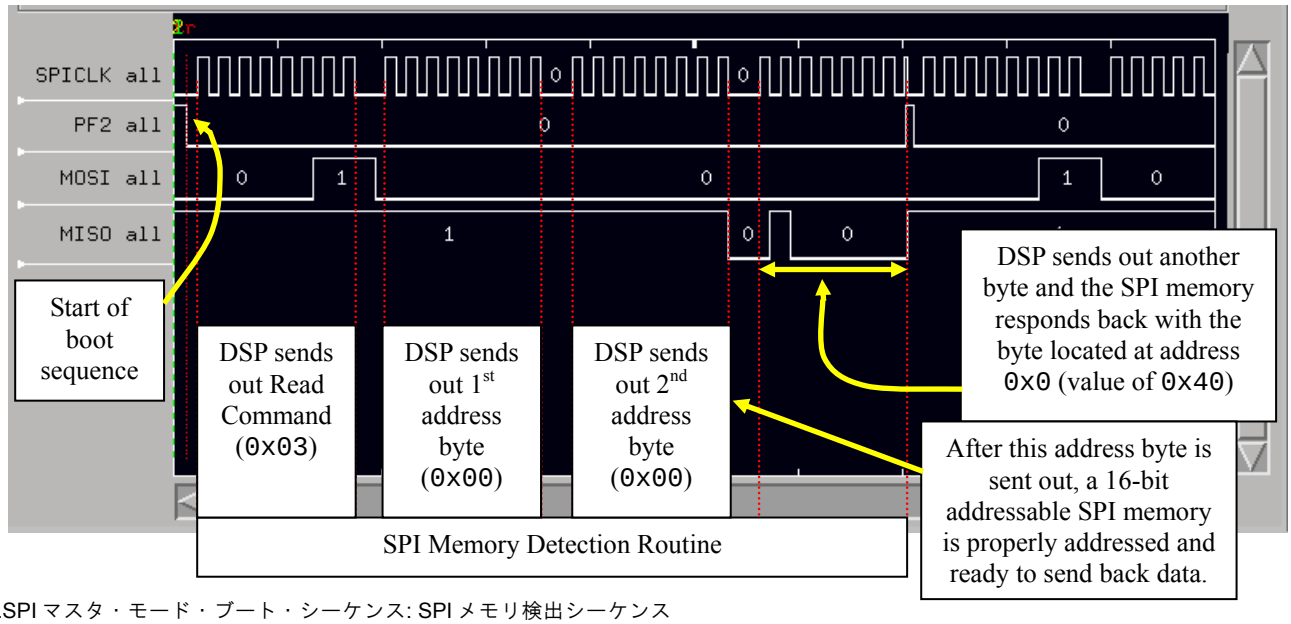


図22.SPI マスタ・モード・ブート・シーケンス: SPI メモリ検出シーケンス

図22は、シリコン・レビジョン0.3の製品を使って取得したものであることに注意してください。シリコン・レビジョン0.2では、SPICLKは、転送前と転送中にハイ・レベルになります。
内蔵ブートROMは、この時点で16ビット・アドレサブルSPIメモリが接続されていることを検出しています。次に、読み出しコマンドを発行して、InitコードDXEカウント・ブロックの先頭の10バイト・ヘッダーで読み出すアドレス0x0000を送信します。

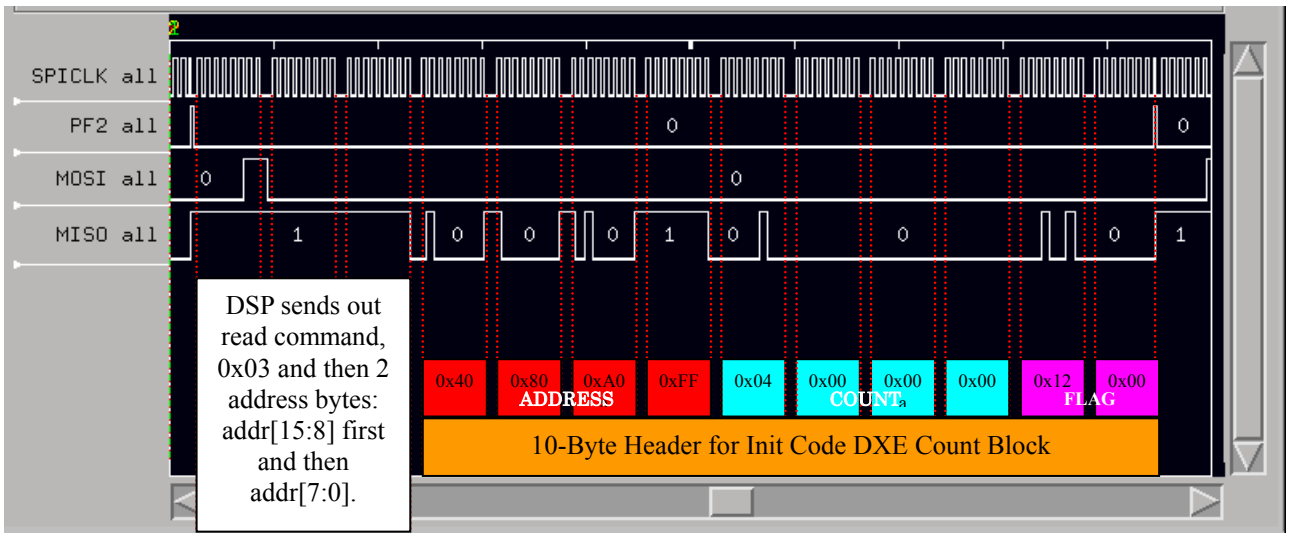


図23.SPI マスタ・モード・ブート・シーケンス: Init コード DXE カウント・ブロックのブート 10 バイト・ヘッダー

Init コード DXE カウント・ブロックは 4 バイトの無視ブロックであるため、内蔵ブート ROM は読み出しコマンドを発行して、Init コード DXE のブロック 1 の 10 バイト・ヘッダーでアドレス 0x000E を送信します。このヘッダーを読み出した後、内蔵ブート ROM はメモリ内のブロック 1 のロケーションとそのロケーションへ書き込むバイト数を知ります。

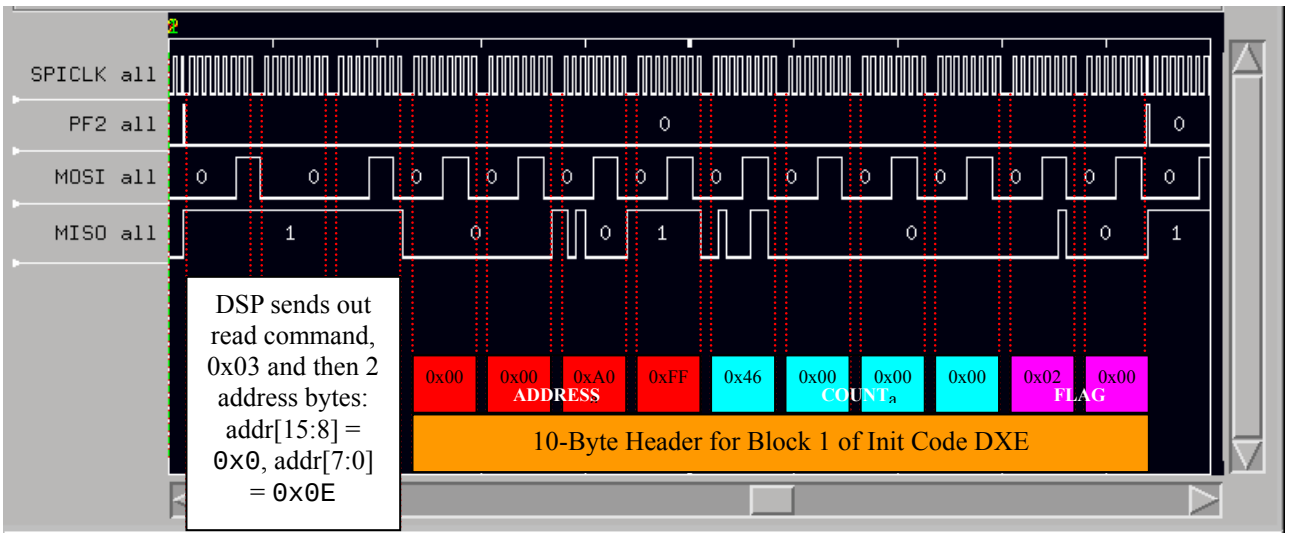


図24.SPI マスタ・モード・ブート・シーケンス: Init コード DXE のブロック 1 のブート 10 バイト・ヘッダー

この情報を処理した後、内蔵ブート ROM は再度読み出しコマンドを発行して、Init コード DXE のブロック 1 でブートするアドレス 0x0018 を送信します。

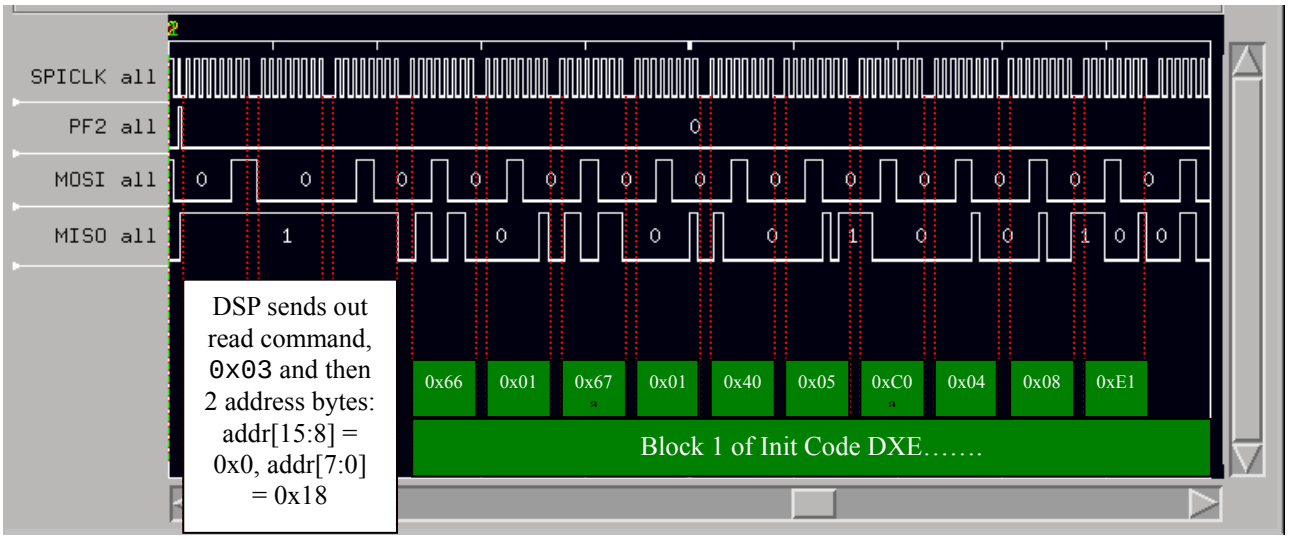


図25.SPI マスタ・モード・ブート・シーケンス: Init コード DXE のブート・ブロック 1

アペンディックス:ブート・モード対シリコン・レビジョン

すべての ADSP-BF531/BF532/BF533 派生品に対して次のブート・モード・オプションが適用されます。

シリコン・レビジョン 0.1

BMODE[1:0]	Description
00	Executes from external 16-bit memory connected to ASYNC Bank0 (bypass Boot ROM)
01	Boots from 8/16-bit flash/PROM. (Only 8-bit boot supported)[1]
10	Boots from an 8-bit addressable SPI memory in SPI Master mode
11	Boots from a 16-bit addressable SPI memory in SPI Master mode

[1] 8 ビット・ブートのみをサポート。16 ビット・フラッシュには、 8 ビット・ブートを”シミュレート”するためゼロがパディングされ ます。

表2.シリコン・レビジョン 0.1 でのブート・モード

シリコン・レビジョン 0.1 では、`IGNORE` ブロックと `Init` ブロックはサポートされていません。

シリコン・レビジョン 0.1 の場合、アドレス `0xFF90 0000`～`0xFF90 000F` (L1 データ・バンク B の先頭の 16 バイト)は予約済みであることに注意してください。このメモリ範囲は、ローダ・ファイル内で各ブロックのヘッダー情報を格納するために内蔵ブート ROM が使います。ブート後には、このメモリ範囲はランタイム時にアプリケーションから使うことができます。

シリコン・レビジョン 0.2

シリコン・レビジョン 0.2 で 24 ビット SPI モードが導入されました。SPI デバイス自動検出機能を使うと、1 つの `BMODE` 構成で 8/16/24 ビット・メモリ・デバイスをカバーすることができます。さらに、シリコン・レビジョン 0.2 では、`INIT` コード機能が導入されました。

BMODE[1:0]	Description
00	Executes from external 16-bit memory connected to ASYNC Bank0 (bypass Boot ROM)
01	Boots from 8/16-bit flash/PROM. (Only 8-bit boot supported). [1]
10	Reserved
11	Boots from a 8/16/24-bit addressable SPI memory in SPI Master mode (pull-up required on `MISO`). [2, 3]

[1] 8 ビット・ブートのみをサポート。16 ビット・フラッシュは、8 ビット・ブートを"シミュレート"するためゼロが詰め込まれます。
[2] 内蔵ブート ROM にエラーがあるため、シリコン・レビジョン 0.2 では、SPI マスタ・モード・ブートに対するゼロ・フィル・ブロックはサポートされていません。このモードのローダ・ファイル内でゼロを詰め込む必要があります。
[3] SPI ブータブル・ローダ・ファイルの先頭バイトは `0x00` である必要があります。

表3.シリコン・レビジョン 0.2 でのブート・モード

シリコン・レビジョン 0.2 の場合、アドレス `0xFF80 7FE0`～`0xFF80 7FFF` (L1 データ・バンク A の最後の 32 バイト)は予約済みであることに注意してください。このメモリ範囲は、ローダ・ファイル内で各ブロックのヘッダー情報を格納するために内蔵ブート ROM が使います。ブート後には、このメモリ範囲はランタイム時にアプリケーションから使うことができます。

上記の注 [1、2、3] は、0.2 モードで起動する場合にのみ elfloader ユーティリティにより処理されます(`-si-revision 0.2` コマンドライン・スイッチを使用)。これは現在の VisualDSP++ 3.5 ツールではデフォルトのケースになっていますが、これはシリコン・レビジョン 0.3 が使用可能になったとき将来のツール・リリースで変更されます。

シリコン・レビジョン 0.3

シリコン・レビジョン 0.3 で SPI スレーブ・ブートが導入されました。SPI マスタ・モードでは、標準の 8/16/24 ビット SPI メモリの他に Atmel 社の DataFlash デバイスがサポートされています。また、真の 16 ビット・フラッシュ/PROM モードもサポートしています。ソフトウェア・リセットのサポートも改善されています。

BMODE[1:0]	Description
00	Executes from external 16-bit memory connected to ASYNC Bank0 (bypass Boot ROM)
01	Boots from 8/16-bit flash/PROM
10	Boots from an SPI host in SPI Slave mode
11	Boots from an 8/16/24-bit addressable SPI memory in SPI Master mode with support for the following Atmel DataFlash devices: AT45DB041B, AT45DB081B, and AT45DB161B

表4.シリコン・レビジョン 0.3 でのブート・モード

シリコン・レビジョン 0.3 の場合、アドレス `0xFF80 7FF0`～`0xFF80 7FFF` (L1 データ・バンク A の最後の 16 バイト)は予約済みであることに注意してください。このメモリ範囲は、ローダ・ファイル内で各ブロックのヘッダー情報を格納するために内蔵ブート ROM が使います。ブート後には、このメモリ範囲はランタイム時にアプリケーションから使うことができます。

シリコン・レビジョン 0.3 機能をイネーブルするときは、elfloader ユーティリティを 0.3 モードで起動する必要があります(`-si-revision 0.3` コマンドライン・スイッチを使用)。シリコン・レビジョン 0.3 が使用可能になり次第、VisualDSP++ 3.5 ツールの将来のアップデートでこれがデフォルト・ケースになります。elfloader ユーティリティの現在のバージョン(プリシリコン・レビジョン 0.3)は、デフォルトで 0.2 モードを使用しているため、`-si-revision 0.3` コマンドライン・スイッチを使って、シリコン・レビジョン 0.3 の機能を呼び出す必要があります。

シリコン・レビジョン 0.3 の内蔵ブート ROM は、シリコン・レビジョン 0.2 と完全な後方互換性を持っています。シリコン・レビジョン 0.2 と 0.3 では、`-si-revision 0.2` コマンドライン・スイッチを使って elfloader ユーティリティを起動する必要があります。

アペンディックス: Blackfinローダ・ファイル・ビューア

http://www.blackfin.org/toolsから提供している Blackfin Loader File Viewer (LdrViewer)は、非常に便利なユーティリティであり、ローダ・ファイルを入力して、個々の.DXEファイルに分解/分類してヘッダー付きの個々のブロック(ADDRESS、COUNT、FLAG)として表示します。この便利なユーティリティは、ローダ・ファイル内容の表示に役立ちます。リスト 3のファイルを LdrViewerにロードすると、GUIにより次のように表示されます。

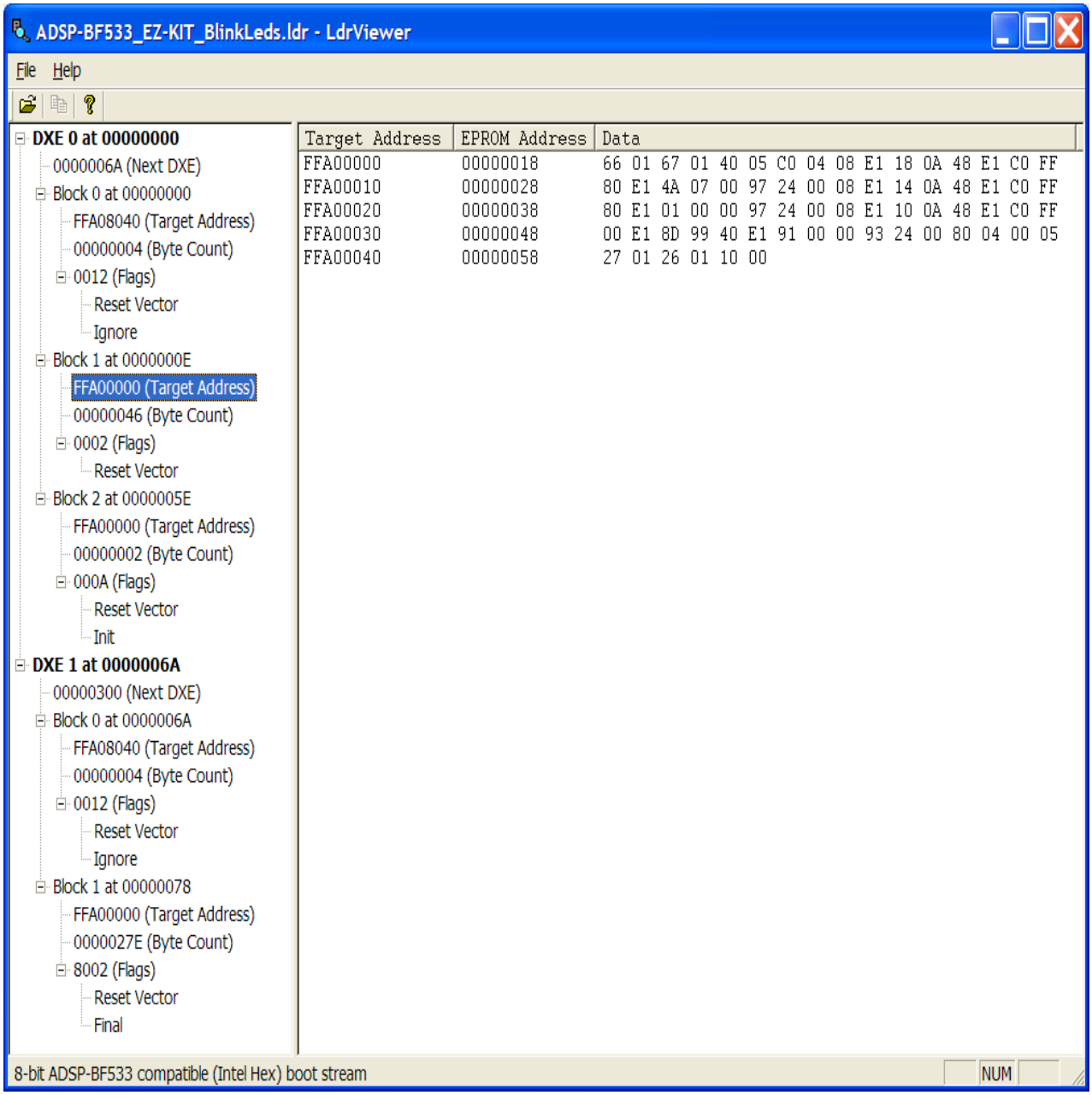


図 26. Blackfin ローダ・ファイル・ビューア・ユーティリティ

LdrViewer ユーティリティは、標準 VisualDSP++ソフトウェア・ツール・セットに含まれないことに注意してください。

参考

[1] *VisualDSP++ 3.5 Loader Manual for 16-bit Processors.* Rev 1.0, October 2003. Analog Devices, Inc.
[2] *ASDP-BF533 Blackfin Processor Hardware Reference.* Rev 3.3, September 2008. Analog Devices, Inc.
[3] *Running Programs from Flash on ADSP-BF533 Blackfin Processors (EE-239).* Rev 1, May 2004. Analog Devices Inc.
[4] *AT45DB041B DataFlash Datasheet.* April 2004. Atmel Inc.
[5] *AT45DB081B DataFlash Datasheet.* November 2003. Atmel Inc.
[6] *AT45DB161B DataFlash Datasheet.* November 2003. Atmel Inc.

ドキュメント改訂履歴

Revision	Description
Rev 4 – September 29, 2008 by M. Kokaly-Bannourah	• Added informational bullet for Figure 15.
Rev 3 – January 11, 2005 by H. Desai	• Added informational bullets for reserved memory regions
Rev 2 – December 17, 2004 by H. Desai	• Added pull-ups to the /AMS0 for all flash boot figures • Added a pull-up to PF2 in Figure 20 • Described pull-up or pull-down on SPICLK in Figure 20 • Specified maximum SPI Baud Rate for SPI Slave Mode Boot • Indicated that rev. 0.1 does not support IGNORE and INIT Blocks • Changed the term “feedback strobe” to host wait (HWAIT) signal • Added a discussion on Atmel DataFlash derivatives
Rev 1 – June 03, 2004 by H. Desai	Initial Release