XAPP482 (v2.0) 2005 年 6 月 27 日

MicroBlaze Platform Flash/PROM ブート ローダおよびユーザー データ ストレージ

著者 : Shalin Sheth

概要

このアプリケーション ノートでは、MicroBlaze™ システムが保存するソフトウェア コード、ユーザー データ、不揮発性 Platform Flash PROM でのコンフィギュレーション データ、システム デザインの簡略化およびコストの低減について説明します。また、ハードウェア デザイン、ソフトウェア デザイン、インプリメント フロー中に使用するスクリプト ユーティリティについて説明します。

はじめに

MicroBlaze および PowerPC™ プロセッサを使用したソフトウェア エンベデッド システムを組み込んだ FPGA 設計では、通常、外部の揮発性メモリを使用してソフトウェア コードを実行します。揮発性メモリを使用するシステムでは、電源が切れた状態でもソフトウェア コードを維持するために、不揮発性メモリ デバイスを組み込む必要があります。通常 FPGA システムには、PROM という Platform Flash PROM が組み込まれており、電源投入時に FPGA のコンフィギュレーション データをボード上で読み込みます。また、多くのアプリケーションで、それ以外の不揮発性デバイス (SPI Flash、Parallel Flash、PIC など) を使用して、MAC アドレスのような小規模ユーザー データを維持し、システム ボード上の多数の不揮発性デバイスへ接続する場合があります。

このアプリケーション ノートでは、システム ボード上で不揮発性デバイスの必要使用数を低減する方法、1 つの PROM でどのように FPGA コンフィギュレーション データ、ソフトウェア コードおよびユーザー データを扱うかを説明します。ここで説明する具体的な内容を次に示します。

- PROM からのアプリケーション ソフトウェアの読み込み
- PROM で複数のデータ ブロックを保存する方法 (図 1参照)
- ブート ローダなどのアプリケーション用に、最小限の MicroBlaze メモリ システムを作成
- C 言語での reset、interrupt、exception ベクタのダイナミックな書き換え方法
- ソフトウェアおよびユーザー データを使用した PROM ファイル変更のソフトウェア フロー

このアプリケーション ノートでは、低コストの MicroBlaze エンベデッド プロセッサ コア用に説明しますが、汎用入力/出力ポートのある 8 ビット、16 ビットおよび 32 ビットのマイクロコントローラに応用できます。

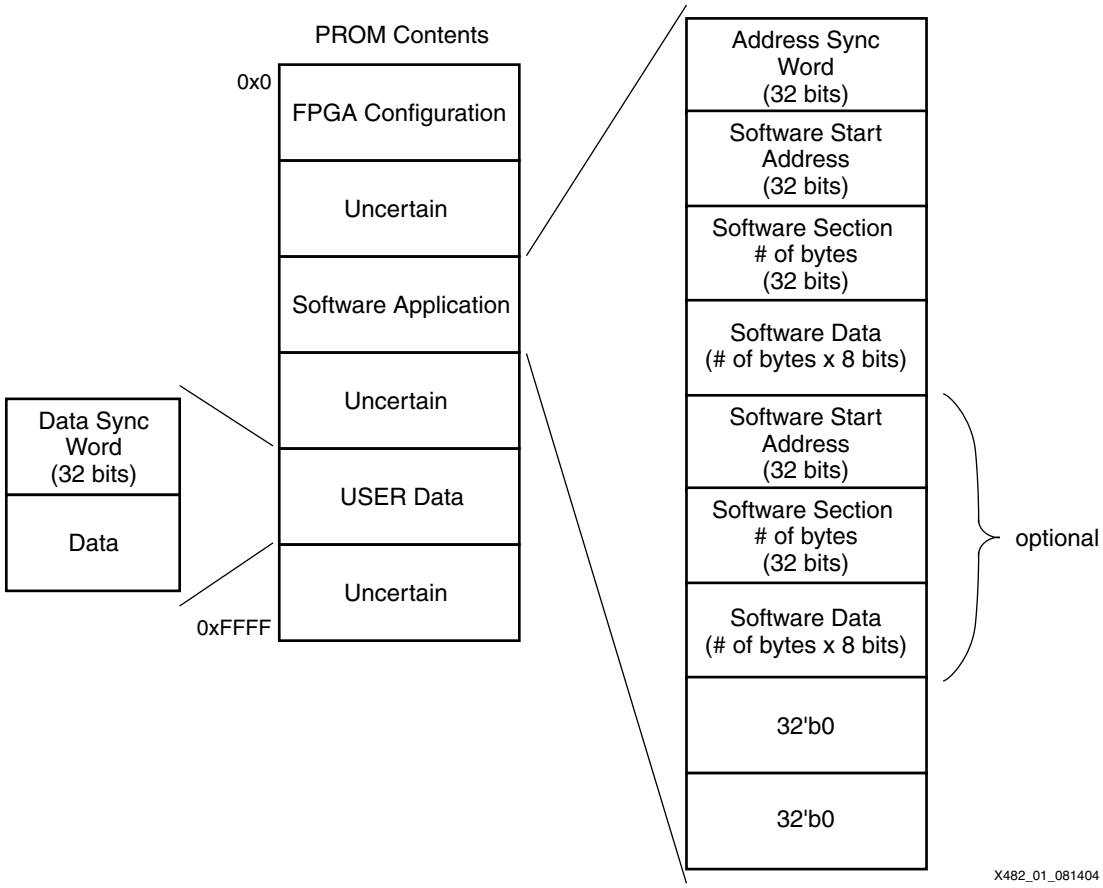

図 1： 1 つの PROM に複数のデータ セクションを保存する方法

図 1 に、複数のセクションを保存した PROM 構造を示します。ソフトウェア アプリケーションは、PROM 内の任意の場所に配置でき、Address Sync Word で識別されます。Address Sync Word の後に、32 ビット長のソフトウェア開始アドレス、32 ビット長のソフトウェア セクションのバイト数、そしてソフトウェア データが続きます。ソフトウェア開始アドレス、バイト数、ソフトウェア データは、同一ソフトウェア アプリケーションで複数回繰り返し配置できます。ソフトウェア アプリケーションの終了は、2 つの 32 ビット ワードを 0 にすることで指定できます。ユーザー データの領域は、データの前のユーザー Sync Word によって定義されます。FPGA コンフィギュレーション データ、ソフトウェア アプリケーション データおよびユーザー データ間のデータは不確定なため、Sync Word が使用されます。

ボード要件

FPGA のコンフィギュレーション後に PROM を読み出すには、システム ボードの設計上、考慮すべき要件があります。ここでは、マスタ シリアル コンフィギュレーション接続およびその接続の必要性について説明します。

図 2 に、マスタ シリアル コンフィギュレーションで必要なボード接続を示します。詳細については、XAPP694 を参照してください。リファレンス デザインと同様のボード要件が記載されています。

ザイリンクス Spartan™-3 スタータ キット ボードでは、これらのボード要件の実例を提供しています。このボードおよびその回路図は、UG130 に記載されています。

FPGA のコンフィギュレーションの詳細は、XAPP501 および XAPP138 を参照してください。

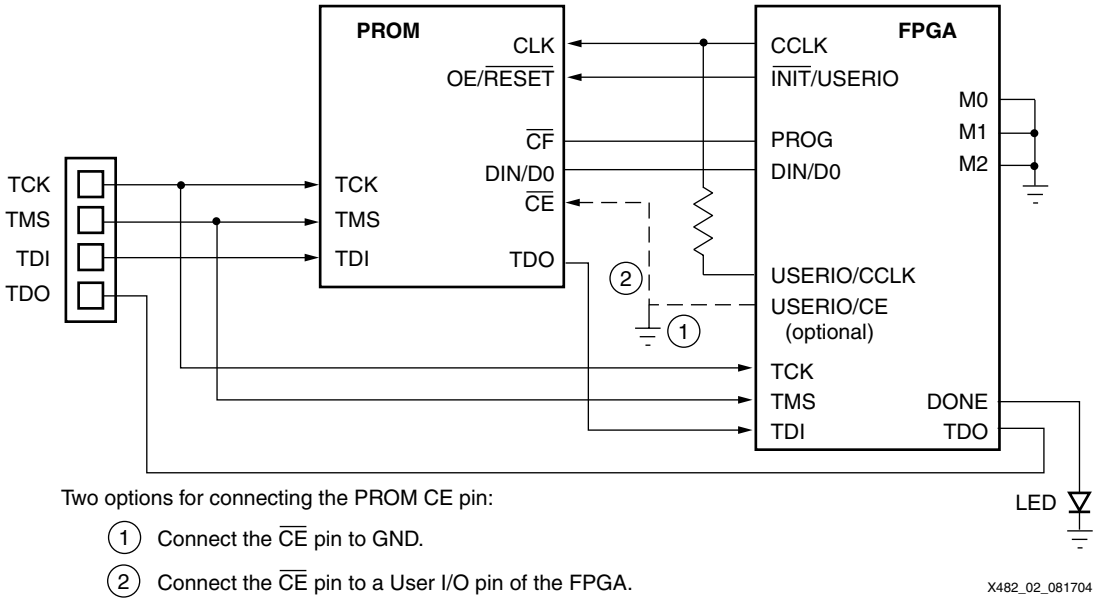

図 2： ボード要件

PROM の CE ピン

PROMの CE ピンは、通常 FPGA の DONE ピンに接続され、FPGA のコンフィギュレーション後に PROM をスタンバイ状態に保ちます。このピンを使用して PROM をイネーブルまたはディスエーブルにし、PROM の 非アクセス時の消費電力を抑えます。DONE ピンが PROM の CE ピンに接続されている場合は、FPGA のコンフィギュレーション後に PROM を読み出せません。

PROM の CE ピンを接続するには、2 種類の方法があります。

1. CE ピンを GND に接続する。
2. CE ピンを FPGA のユーザー I/O ピンに接続する。この方法では、I/O ピンが新たに必要ですが、PROM をスタンバイ モードにして消費電力を削減できます。このピンは、ソフトウェアでイネーブルまたはディスエーブルにされます。Platform Flash の最大スタンバイ電流は 1mAで、動作時の最大消費電流は 10mA (詳細は、DS123 を参照) です。

FPGA の DONE ピンを PROM のCE ピンに接続せずに、外部の LED に接続すると、FPGA のコンフィギュレーション終了を表示できます (図 2 参照)。

PROM の CLK ピン

FPGA から新たにユーザー I/O ピンを配線して、PROM の CLK 入力を駆動します。リファレンス デザインではこの接続が必須です。これは、いかなるマスタ コンフィギュレーション モードにおいても、FPGA が正常にコンフィギュレーションされると、FPGA から生成されるコンフィギュレーション クロック、CLK がトグルを止め、PROM に保存された FPGA デザインに反して PROM のアドレス カウンタがインクリメントされるのを防ぐためです。FPGA のコンフィギュレーション後に PROM を読み出す場合には、ユーザー I/O から PROM の CLK ピンにクロックを送ります。このトレース上には CLK 信号のドライバ同士が競合するのを防ぐ 390Ω の抵抗が 1 つあります。

PROM の OE / RESET ピン

PROM の OE/RESET ピンを FPGA の INIT ピンに接続すると、コンフィギュレーション中に CRC エラーが発生した場合に FPGA のコンフィギュレーションを再実行します。INIT ピンはコンフィギュレーション後にユーザー I/O ピンとなるため、PROM の出力をイネーブルに保つようロジック High の出力を設定できます。

PROM の DIN/D0 ピン

FPGA の DIN/D0 ピンを PROM の DIN/D0 ピンに接続し、PROM データを FPGA に読み出します。この接続方法は、このアプリケーションでは特別ではありません。なお、DIN ピンは、コンフィギュレーション後にユーザー I/O として使用できません。

ハードウェア デザイン

リファレンス デザインをインプリメントするには、エンベデッド開発キット (EDK) の MicroBlaze システムを使用します。ハードウェア コアはオンチップ ペリフェラル バス (OPB) の汎用の入力/出力 (GPIO) コアに組み込まれ、「ボード要件」で前述のように、$\overline{\text{INIT}}$、$\overline{\text{CE}}$、OE および DIN ピンを駆動します。図 3 に、ハードウェア デザインのブロック図を示します。

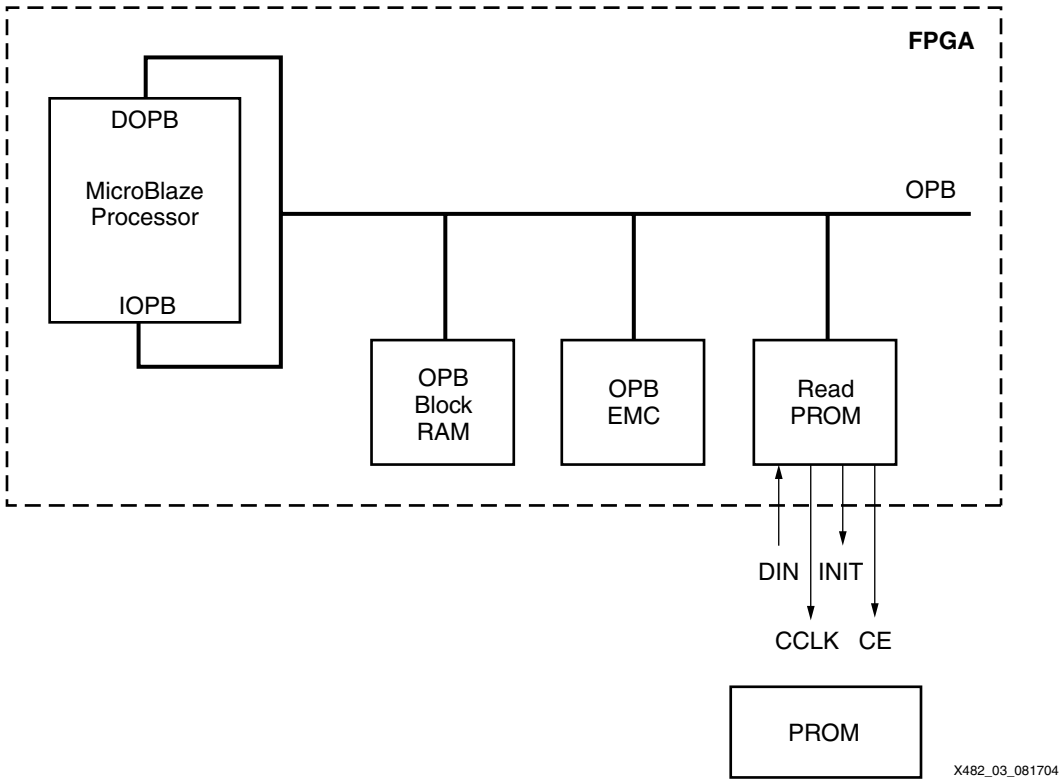

図 3 : MicroBlaze ハードウェア システムのブロック図

1 つの Spartan-3 デバイスで、promread GPIO コアには、26 個の 4 入力ルック アップ テーブル (LUT) と、61 個のフリップフロップが使用されます。リファレンス デザインでは、カスタム OPD ブロック RAM インターフェイス コントローラ コアを使用して、わずか 1 つのブロック RAM で、最少システムを構築しています。EDK 7.1i では、必ずブロック RAM を 4 つ使用します。ブート ローダのように、多くのブロック メモリを必要としない最適化されたシステムでは、カスタム ブロック RAM インターフェイス コントローラ コアを使用して、単一ブロック RAM を使用したシステムを実現します。

ファームウェア デザイン

このアプリケーションのファームウェア デザインは複雑です。PROM の制御には、C 言語のソフトウェア プログラムを使用します。コンフィギュレーション PROM にはアドレス指定ができないため、データはすべてシリアル送信され、ソフトウェア システムで読み出されます。

基本的な制御

promread 機能で PROM が使用できます。

```
Xuint32 promread (Xuint8 read)
```

表 1 に、PROM を読み出す 2 つのモードを示します。アドレス セクションの読み出しでは、promread 機能により、PROM のAddress Sync Word (デフォルトは 0x9F8FAFBF) 以降の PROM から指定されたメモリの位置にソフトウェア コードがコピーされます。データ ワードの読み出しでは、Data Sync Word (デフォルトは 0x8F9FAFBF) 直後の 32 ビットのデータ ワードを読み出します。

表 1 : promread 機能

機能	入力 (読み出し)	出力 (戻り値)
アドレス セクションの読み出し	0x1	常に 0x0
データ ワードの読み出し	0x0	Data Sync Word に続く 32 ビットのデータ ワード

図 4 に、この 2 つの機能の使用例を示します。

```
#define DATAREAD 0x0
#define ADDRREAD 0x1

//to copy the contents from the PROM to a memory location
  promread(ADDRREAD);

//to return a 32-bit word stored after the sync word.
  xil_printf("\n\rData %x\n\r", promread(DATAREAD));
```

図 4 : promread 機能の使用例

カスタム ユーティリティ (xapp482.exe および xapp694.exe) を使用して、MCS ファイル (ザイリンクスの Intel 拡張 16 進数形式の拡張子) をインストールします。Perl スクリプト言語およびユーティリティの更新についての詳細は、「使用方法および フロー」を参照してください。「ドライバについて」に、PROM データの構成を示します。

ドライバについて

ファームウェア デザインを理解するには、PROM からシリアル出力される内容についての理解が不可欠です。2 ページの図 1 に、PROM でどのようにデータが保存されるかを示しました。

アドレスの読み出し

アドレスの読み出しが指示されると (promread 入力 = 0x1)、ソフトウェアでは、PROM の読み出しが一度に 32 ビットずつ、32 ビット ワードが Address Sync Word と合致するまで実行されます。PROM からのデータの読み出し方法の詳細は、「PROM 内容の読み出し」を参照してください。デフォルトの Address Sync Word は 0x9F8FAFBF ですが、promread.h というヘッダ ファイル内で、MCS ファイルを生成する Perl スクリプトを使用して変更できます。PROM ファイルの生成時には、Sync Word が重複していないかが確認されます(「使用方法および フロー」参照)。

図 5 に、PROM でどのようにデータが検知されるかを示します。Address Sync Word が検知されると、その直後の 32 ビット ワードが、プロセッサ メモリ マップ上のソフトウェア コードが保存場所の開始アドレスを示します。このアドレスの後が、開始アドレス以降の保存領域のバイト数です。次に、ソフトウェアによって PROM からバイト数が読み出され、ブロック開始位置で指定されたアドレスにデータをコピーします。最初のアドレス セクションが完了すると、ソフトウェアによって 2 つの 32 ビット ワードが読み出されます。いずれかの値が 0 より大きい場合は、最初の 32 ビット ワードが次のソフトウェア データ セクションの開始アドレスとなり、2 つ目の 32 ビット ワードがアドレス セクションのバイト数となります。ソフトウェアによってアドレスの読み出しが継続され、2 ページの図 1に示すように、継続した 2 つの 32 ビット ワードが 0 を示すと、アドレス シーケンス終了となります。promread 機能は、PROM の読み出しを継続し、END_PROM ワード、0xFFFFFFFF が PROM から読み出され

るまで、前述のようにアドレス セクションにコピーします。そして、promread 機能が 0x0 を戻し、0xFFFFFFFF という値が、PROM のブランク データの開始アドレスに到達したことを示します。

```
9F8FAFBF  80180000  00007100  BA101056
--------  --------  --------  --------
address   memory    number    data
sync      mapped    of        …
word      starting  bytes
          address
```

図 5： PROM のソフトウェア セクションの内容例

データの読み出し

データの読み出しが指示されると (promread 入力 = 0x0)、ソフトウェアでは、PROM の読み出しが一度に 32 ビットずつ、32 ビット ワードが Data Sync Word と合致するまで実行されます。デフォルトの Data Sync Word は 0x8F9FAFBF ですが、`promread.h` というヘッダ ファイル内で、MCS ファイルを生成する Perl スクリプトで変更できます。PROM ファイルの生成時に、Sync Word が重複していないかが確認されます (「使用方法および フロー」参照)。Data Sync Word が検知されると、promread 機能によって、Data Sync Word 直後の最初の 32 ビット ワードが返されます。データをさらに必要とする場合は、promread 機能のデータの data retrieval セクションを変更してください。

PROM 内容の読み出し

ソフトウェアで PROM を読み出す場合、PROM の CLK ピンが MicroBlaze の GPIO にトグルされます。$\overline{\text{INIT}}$ ピンは、High に設定し、$\overline{\text{CE}}$ ピンは Low に設定して PROM の読み出しをイネーブルにします。PROM から出力されるすべてのバイトがビット スワップされます。

```
//clock the PROM to output data
XIo_Out32(XPAR_PROMREAD_BASEADDR, OE_HIGH | CCLK_HIGH | CE_LOW);
XIo_Out32(XPAR_PROMREAD_BASEADDR, OE_HIGH | CCLK_LOW | CE_LOW);
```

図 6 に、各バイトがどのようにビット スワップされるかを示します。Perl スクリプトによって、PROM にロードする前にビット スワップを実行し、データが PROM から読み出される時には、適切な読み出し順序になります。

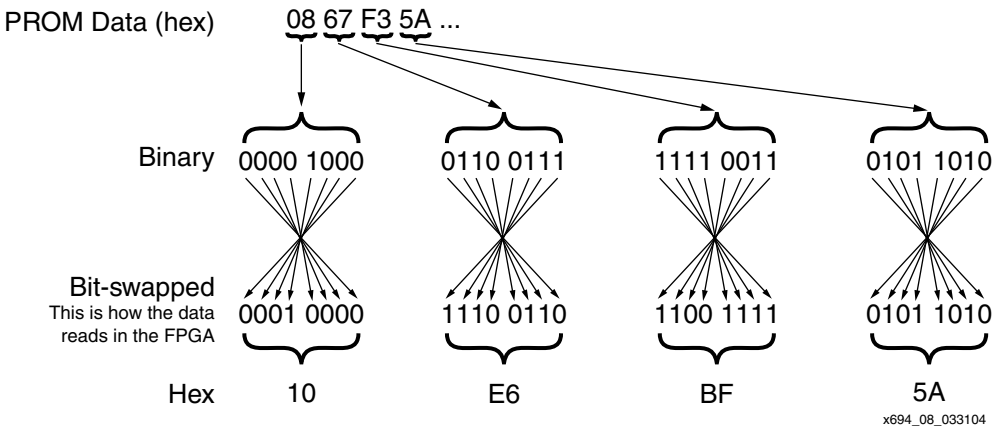

図 6： PROM のビット スワップ出力

PROM から読み出されたものと比較すると、MCS ファイルの内容がビット スワップされていることに注意してください。

表 2 の真理値表に、PROM 制御入力がどのように PROM からデータ出力されるかを示します。

表 2 : PROM 制御入力の真理値表

制御入力		内部アドレス	出力
OE/$\overline{\text{RESET}}$	$\overline{\text{CE}}$		データ
High	Low	アドレス<= TC の場合 : インクリメントするアドレス> TC の場合 : 変化しない	Active
Low	Low	リセットのまま	High-Z
High	High	リセットのまま	High-Z
Low	High	リセットのまま	High-Z

ファームウェアのパフォーマンス

ブート オペレーションの処理速度は、ソフトウェアで PROM クロックが生成され、データがシリアルで読み出されるため遅くなります。PROM へのアクセスにかかる時間は、PROM に保存されたビットストリームのサイズにも依存します。PROM 内のビットストリームのサイズが大きいと、promread 機能が、PROM を解析し、ソフトウェアや PROM 内のユーザー データを検索するのに長い時間を要します。

Spartan-3 スタータ キット ボードで、50MHz で使用した場合のリアル タイム パフォーマンスが評価されています。PROM で 1M ビットのビットストリームを解析するには、約 2 秒かかります。

Promread 機能のソフトウェア コード サイズは 0x344 (または 836) バイトです。

デュアル ソフトウェアのプロジェクト

多くのエンベデッド システムにおいて、設計者はリンカ スクリプトを使用して、ソフトウェア コードのセクションを別のメモリに分割したり、あるいは実行するコードに基づいた複数のソフトウェア プロジェクトを使用します。リファレンス デザインでは、デュアル ソフトウェア プロジェクト コンセプトを使用して、ブート ローダ ソフトウェアとアプリケーション ソフトウェアを分割します。ブート ローダ ソフトウェアはブロック RAM にあり、そこから命令を実行します。一方、アプリケーション ソフトウェアは SRAM にあり、そこから命令を実行します。したがって、SRAM が初期化され、ブロック RAM に流出しないように、このプログラムを設定しておく必要があります。起動時には、ブート ローダが PROM から SRAM へデータをコピーします。コピーが完了すると、ブート ローダが SRAM の開始アドレスに jump し、アプリケーションのソフトウェアを実行します。SRAM への jump には、ファンクション ポインタを使用します。機能ポインタについては、「C 言語プログラムでプログラム カウンタ (PC) を変更するには」で説明します。

図 7 に、EDK 7.1i で設定したデュアル ソフトウェアのプロジェクトを示します。

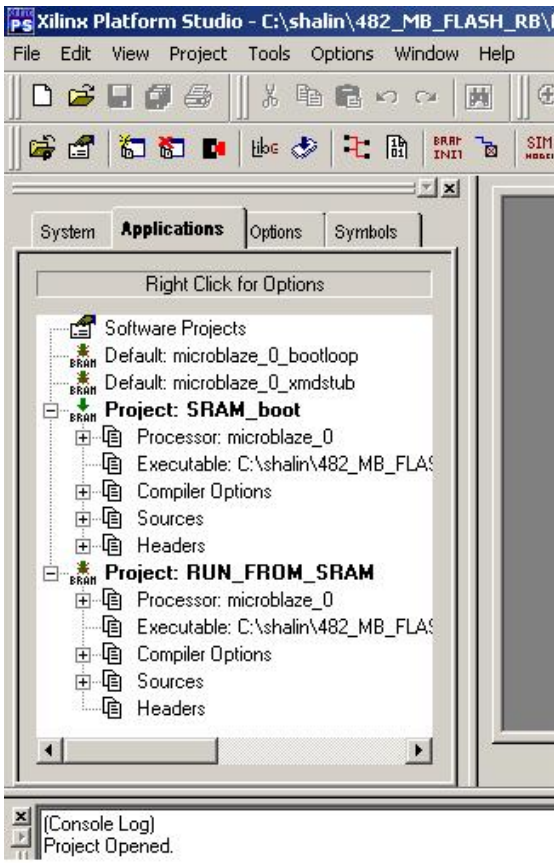

図 7 : デュアル ソフトウェアのプロジェクト設定

C 言語プログラムでプログラム カウンタ (PC) を変更するには

通常、ブート ローダの最後には、ブート プログラム スペースからほかのアドレスへ jump し、アプリケーション命令の実行が開始されます。このような場合、アセンブリ言語を使用して簡単に PC を変更できますが、C 言語で記述されたソフトウェアでは、変更結果が複数言語となります。この問題を回避するには、図 8 に示すように、ファンクション ポインタに C 言語で記述し、PROG_START_ADDR をブート ローダのアプリケーション ソフトウェアの開始アドレスに設定してください。

```
//declare before main()
// Function point that is used at the end of the program
// to jump to the address location stated by PROG_START_ADDR
#define PROG_START_ADDR 0x80180000
int (*func_ptr) ();


// declare after main()
// function point that is set to point to the address of
// PROG_START_ADDR
func_ptr = PROG_START_ADDR;
// jump to start execution code at the address
// PROG_START_ADDR
func_ptr();
```

図 8 : アセンブラでファンクション ポインタを C 言語で示し、jump 命令を実行する

図 9 に、図 8 の C 言語が MicroBlaze のアセンブリ言語でどのように変換されるかを示します。

```
    84     func_ptr = PROG_START_ADDR;
  - 0xe4 <main+16>:     imm    -32744
  - 0xe8 <main+20>:     addik  r3, r0, 0
  - 0xec <main+24>:     swi    r3, r0, 1808// 0x710 <func_ptr>
    86     func_ptr();
  - 0xf0 <main+28>:     brald  r15, r3
  - 0xf4 <main+32>:     or     r0, r0, r0
```

図 9： ファンクション ポインタの非アセンブリ

ブート コードを変更してコード サイズを縮小するには (オプション)

コード サイズを縮小するには、EDK ツールで挿入される SRAM_boot というブート ローダのソフトウェア プロジェクトのデフォルトである、C ランタイム ルーチン (CRT) ファイルを削除します。この最適化方法はオプションです。ソフトウェア コードを縮小する必要があり、ブート ローダの実行中に interrupt や exception を使用しない場合にのみ可能となります。最適化する場合は、interrupt および exception を RUN_FROM_SRAM というソフトウェア プロジェクト下に維持したまま、ブート ロード後に SRAM から実行します。「reset、exception および interrupt ハンドラの再設定」に示すように、プロセッサがこれらのハンドラにアクセスできるよう設定する手順が必要となります。ソフトウェアの初期化ファイルについての詳細は、エンベデッド システム ツール ガイドを参照してください。

`init.s` という初期化ファイルは、リファレンス デザインに含まれます。このファイルを変更するには、次の手順に従ってください。

1. `init.s` ファイルをソフトウェア プロジェクトに追加する
2. `bootlinker.scr` というリンカ スクリプトを加える
3. コンパイラのオプションを変更して、初期化ファイルの自動挿入をディスエーブルする

`init.s` ファイルをソフトウェア プロジェクトに追加した場合、リンカ スクリプトおよびコンパイラのオプションが設定されていると、アセンブラとリンカを使用して、ツールが自動的にコンパイルし、ファイルをリンクすることに注意してください。

リンカ スクリプトの例として、`bootlinker.scr` がリファレンス デザインに含まれており、ブートローダのソフトウェア プロジェクトに使用できます。そのリンカ スクリプトでは、初期化ファイルに 2KB (0x7FF) のメモリを割り当てます。この例では 2KB (0x0 to 0x7FF) のメモリ スペースが必要と想定され、このサイズは、設計者のマイクロプロセッサ ハードウェア スペック (MHS) ファイルに基づき、メモリ スペースに合わせて各デザインごとに変更する必要があります。このリンカ スクリプトでは、メモリ スペースの 0x0 ～ 0x7FF に .boot セクションを配置しています。

初期化ファイルの変更を反映させるには、コンパイラに 2 つの設定が必要です。

1. `-nostartfiles` 設定で、アセンブラがデフォルトの初期化ファイルを取り込まないよう設定します。このオプションが未設定の場合、初期化ファイルがソフトウェア プロジェクトに取り込まれる際に、リンカ内で複数のセクション宣言が発生します。
2. `-save-temps` では、リンカが `init.s` アセンブリ ファイルからハンドラを検出するよう設定します。

図 10 に、コンパイラのオプション設定方法を示します。

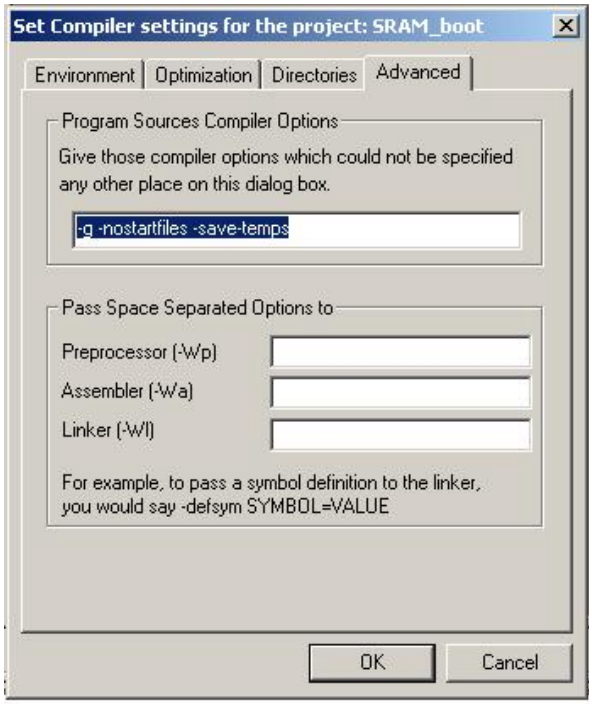

図 10： CRT ファイルを置き換えるためのコンパイラ変更

reset、exception および interrupt ハンドラの再設定

デュアル ソフトウェアのプロジェクト システムでは、ソフトウェア プロジェクトが SRAM に jump した後、SRAM に保存した interrupt および exception ハンドラにアクセスする場合があるかもしれません。一般に、MicroBlaze システムでは、ブロック RAM がメモリとして 0x0 にマップされ、SRAM はメモリ マップの別の位置に配置されています。しかしながら MicroBlaze システムのデフォルトでは、リセット時にアドレス 0x0 に jump するか、interrupt 時にアドレス 0x10 に jump してしまいます。デフォルトの MicroBlaze ハンドラのアドレスを次に示します。

- reset : 0x0
- exception : 0x8
- interrupt : 0x10

この問題を回避するには、MicroBlaze ハンドラのデフォルト位置にアセンブラの jump ルーチンを入れ、図 11 に示すように、SRAM のソフトウェア ハンドラの位置に jump するよう設定します。ソフトウェア ハンドラが処理されると、各 reset、exception、interrupt ハンドラに対して、それぞれ 0x0、0x8、0x10 に jump 命令が入ります。この方法はダイナミックに命令を変更するため、ソフトウェア設計者によっては懸念する場合がありますが、ブート ローダとソフトウェア コードを切り離して維持する 1 つの手段となります。このアプリケーション ノートで説明する方法は、ブート ローダにアプリケーション ソフトウェアの情報が不要であるという利点があります。しかし、ブート ローダが SRAM 内の interrupt および exception ハンドラの位置を認識できる場合には、アプリケーション ソフトウェアでダイナミックに変更する必要はありません。

```
//insert before main()
extern int _start;
extern int _exception_handler;
extern int __interrupt_handler;

//=====================================================

//insert after main() and after variable declarations
int x = &_start;
*(int*)(0x0) = 0xb0000000 | (((x-1) & 0xFFFF0000) >> 16 );
*(int*)(0x4) = 0xb8000000 | (((x-1) & 0xFFFF));

x = &_exception_handler;
*(int*)(0x8) = 0xb0000000 | (((x-1) & 0xFFFF0000) >> 16 );
*(int*)(0xB) = 0xb8000000 | (((x-1) & 0xFFFF));

x = &__interrupt_handler;
*(int*)(0x10) = 0xb0000000 | (((x-1) & 0xFFFF0000) >> 16 );
*(int*)(0x14) = 0xb8000000 | (((x-1) & 0xFFFF));
```

図 11 : ダイナミック ソフトウェアで MicroBlaze システムの reset、exception および interrupt ハンドラを書き変える

使用方法および フロー

Platform Flash/PROM ブート ローダを生成するには、カスタム スクリプトおよびフローを使用してください。フローおよびスクリプトの使用について、次に説明します。また、SRAM 内のユーザー データと共に、あるいは SRAM 外で、ソフトウェア実行用の Executable Linking Format (ELF) ファイルの内容を MCS ファイルに保存する方法についても説明します。表 3 に、リファレンス デザインで提供され、カスタマ スクリプトで使用できる機能を示します。

表 3 : 付属ユーティリティの機能

フォーマットの内容	MEM アドレスおよびデータ
サイズ/16 バイトのデータ → バイナリで保存	16 バイト
アドレス ブロックのオーバーヘッド	8 バイト
開始アドレスの保存	不可
アドレス位置の保存	可
チェックサム	不可
ブート コード	Medium
複数の ELF ファイルをサポート	可
フロー	3 つの手順 : gcc → Data2MEM → xapp482.exe

MCS ファイルの作成

フローはすべて MCS ファイルから始まります。MCS ファイルは iMPACT または promgen を使用して生成できます。MCS ファイルの生成方法については別途資料を参照してください。

ソフトウェア セクションを MCS ファイルに追加するには

図 12 に、PROM ファイルにコードを追加するフローを示します。

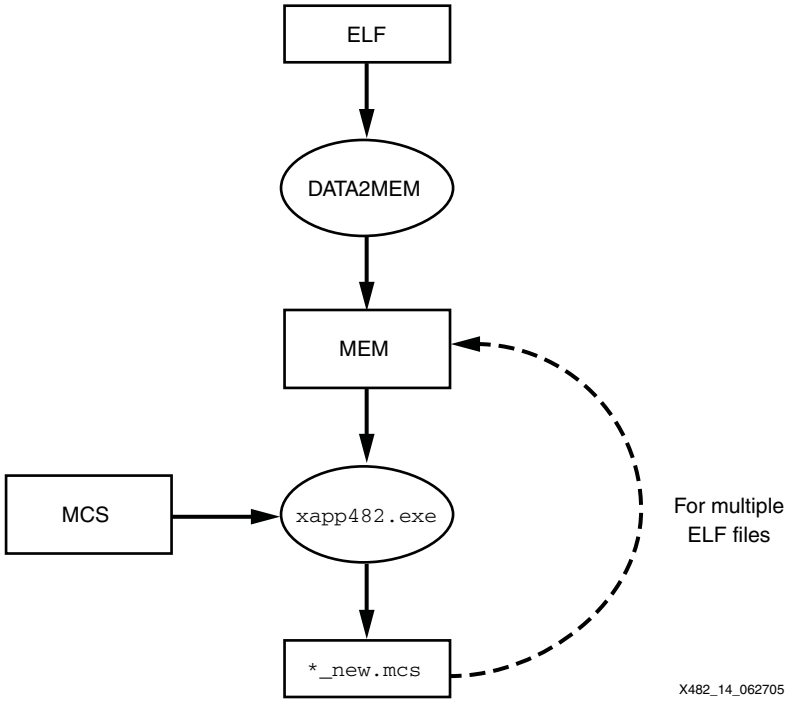

図 12 : PROM ファイルにアプリケーション ソフトウェアを追加するフロー

SRAM の実行用にコードをコンパイルした後、ELF ファイルを Data2MEM に読み込み、MEM ファイルとして出力します。Perl スクリプトを実行し、暗号化した ELF ファイル形式を 16 進数形式の MEM ファイルに変換します。ELF ファイルから MEM ファイルを生成するコマンド ラインを次に示します。

```
Data2MEM -bd *.elf -d -o m *.mem
```

Data2MEM の実行についての詳細は、『開発システム リファレンス ガイド』を参照してください。

次に、ユーティリティを使用して、MEM ファイルの内容を MCS ファイルと合成します。

```
xapp482 *.mem *.mcs new*.mcs [syncword]
```

このコマンド ラインによって、PROM のプログラムで使用する `new*.mcs` が出力されます。Sync Word が指定されていない場合には、デフォルトの Sync Word、0x9F8FAFBF が使用されます。この手順を繰り返し、MCS ファイルにアドレス セクションを追加します。生成される MCS ファイルに Sync Word のインスタンスが検知されると、ユーティリティは警告メッセージを表示します。

MCS ファイルにユーザー データ セクションを追加するには

MCS ファイルにユーザー データ セクションを追加するコマンド ラインを次に示します。

```
xapp694 user_data.txt input.mcs output.mcs [-noswap]
```

`userdata.txt` ファイルが次の条件を満たすようにして、合成してください。

1. 各データ ラインは 16 バイト長
2. 数字はすべて 16 進数を使用
3. コメントを追加する場合には、行の最初に「#」を使用

4. データ セクションの開始位置に Sync Word を入力。デフォルトの Sync Word は 0x8F9FAFBF

```
#This is data block 0
#The sync pattern is 8F9FAFBF
#The data is ASCII code for:
#XAPP 694 DATA BLOCK 0
#01234567890123456789012345678901
8F9FAFBF584150502036393420444154
4120424C4F434B203000000000000000
```

xapp694 のユーティリティでは Sync Word が検知されません。デフォルトでは、6 ページの図 6 に示すように、MCS ファイルを出力する前に、ユーザー データがスワップされます。スワップさせないためには、-noswap スイッチをイネーブルに設定してください。

MCS アップデート ユーティリティの注意点

MCS アップデート ユーティリティの使用例について上述しました。PROM に、ユーザー定義データを過剰に追加すると、コンフィギュレーション ツールで PROM ファイルが使用できない場合があります。FPGA コンフィギュレーションおよびユーザー定義データを共に保存する PROM を選択するには、FPGA のコンフィギュレーションに使用するビット数を、ユーザー定義データ ビット、ソフトウェア コード ビット、該当の同期パターンに追加します。FPGA コンフィギュレーションに使用するビット数は、FPGA のデータシートを参照してください。

おわりに

このアプリケーション ノートでは、FPGA コンフィギュレーション後、PROM を読み出すためのボード レベルの変更、PROM の複数のデータ経路、PROM からユーザー データを読み出すソフトウェア、ソフトウェア システムのブート ロード方法、ブート ローダのための MicroBlaze ハードウェア システムおよびソフトウェア システムの最適化、ソフトウェアおよびユーザー データを含む PROM ファイルを書き換えるためのソフトウェア フローについて説明しました。これらすべての事項を考慮して、MicroBlaze システム全体のシステム コストを抑えることができます。

デザイン リソース

このアプリケーション ノートで使用したリファレンス デザインは、次のリンクからダウンロードできます。

http://www.xilinx.co.jp/bvdocs/appnotes/xapp482.zip

参考資料

このアプリケーション ノートの参考資料を示します。

1. XAPP694 :『Reading User Data from Configuration PROMs』
2. XAPP501 :『Configuration Quick Start Guidelines』
3. XAPP138 :『Virtex FPGA Series Configuration and Readback』
4. UG130 :『Spartan-3 Starter Kit Board User Guide』
5. UG111 :『エンベデッド システム ツール リファレンス マニュアル』
6. 『開発システムリファレンスガイド』
7. DS099 :『Spartan-3 FPGA ファミリ：製品紹介および注文情報』
8. 『MicroBlaze Processor Reference Guide』
9. DS123 :『Platform Flash In-System Programmable Configuration PROMs』

改訂履歴

このドキュメントの改訂履歴を示します。

日付	バージョン	改訂内容
2004/08/19	1.0	初版リリース
2005/06/27	2.0	ブート初期化ファイルおよびリファレンス デザインの MCS ユーティリティの記載を追加。