

イントロダクション

プログラマブル・ロジック・デバイス (PLD) における技術の進展は、イン・システム・プログラマビリティ (ISP) とイン・サーキット・リコンフィギュラビリティ (ICR) という革新的な機能を実現しました。JEDECの標準規格、JESD-71で規定されているテストおよびプログラミング用の標準言語、Jam™ STAPL (Standard Test and Programming Language) は JTAG (Joint Test Action Group) ポートを使用したICRおよびISPをサポートしているすべてのPLDと互換性があります。このため、Jam STAPLを使用することによって、ソフトウェア・レベルで特定のベンダに依存しないイン・システムでのプログラミングとコンフィギュレーションが実現可能になります。Jam STAPLを使用してISPおよびICRを実現することで最終製品の品質や柔軟性が向上し、また製品の寿命をさらに延長させることができます。プログラムまたはコンフィギュレーションされる必要があるPLDの数とは関係なく、Jam STAPLはフィールドでのアップグレードを容易にし、PLDのプログラミングとコンフィギュレーションの方法を変革します。

このアプリケーション・ノートは、エンベデッド・システムにおけるJam STAPLを使用したプログラミングとコンフィギュレーションに対するアルテラのサポートについて解説したものです。

エンベデッド・システム

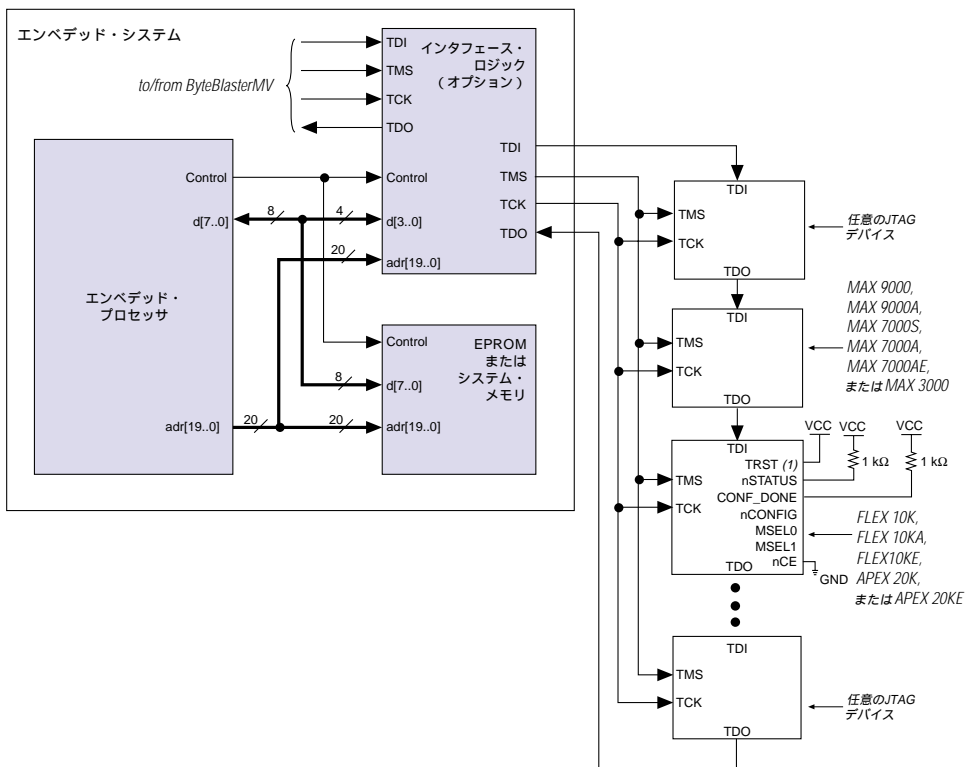
すべてのエンベデッド・システムはハードウェアとソフトウェアのコンポーネントによって構成されています。エンベデッド・システムを設計するとき、最初にプリント基板のレイアウトを行い、次にボードの機能をコントロールするファームウェアを開発します。

JTAGチェーンとエンベデッド・プロセッサの接続

JTAGチェーンとエンベデッド・プロセッサとの接続には、2種類の方法があります。もっとも簡単なのは、JTAGチェーンをエンベデッド・プロセッサにダイレクトに接続する方法です。この方法では、エンベデッド・プロセッサの4本のピンがJTAGインタフェース専用となるため、ボード・スペースが節減されますが、エンベデッド・プロセッサで使用できるピン数が減少します。

図1は、インタフェース用のPLDを介してJTAGチェーンを既存のバスに接続する方法を示したものです。この方法では、JTAGチェーンがバス上のひとつのアドレスとなります。プロセッサはJTAGチェーンを表すアドレスからのリード、またはそのアドレスへのライトを行います。

図 1 エンベデッド・システムのブロック図



前述の 2 種類の接続方法では、MasterBlaster™またはByteBlasterMV™用のヘッダを接続するためのスペースを設けておくことが重要です。このヘッダを用意しておくことによって、PLDの内容の検証や修正を簡単に行うことができるため、試作段階で便利なデバッグ機能が実現されます。また、量産時には、このヘッダを除去してコストの低減をはかることができます。

インタフェース用PLDのデザイン例

図 2 にインタフェース用PLDを使用したときのデザイン例が示されています。他のデザインも実現可能ですが、このデザインでの重要な点は、

- TMS、TCK、TDIは同期出力になっている必要がある。
- マルチプレクサのロジックを実現して、ボード上でMasterBlasterまたはByteBlasterMVダウンロード・ケーブルのアクセスを可能にする。



このデザイン例は参照用です。data[3..0]を除くすべての入力はオプションであり、インタフェース用のPLDがエンベデッド・データ・バス上のひとつのアドレスでどのように動作するかを示すため、この図の中に含まれています。

図 2 インタフェース・ロジックのデザイン例

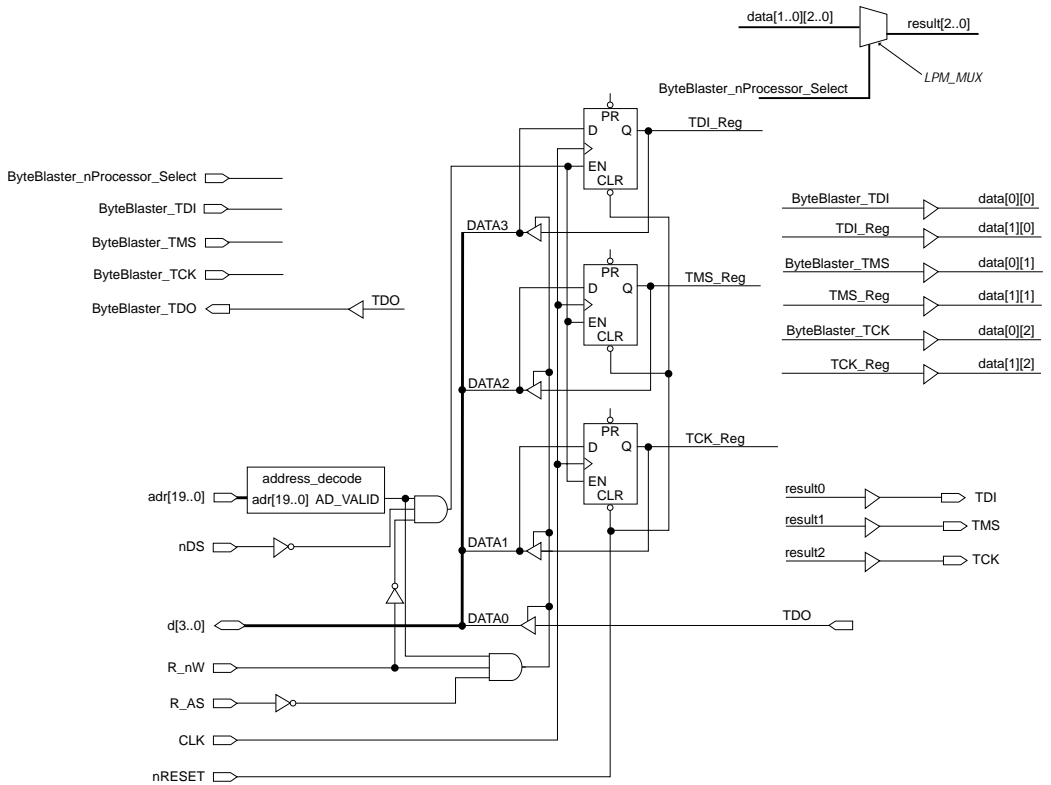


図 2 において、エンベデッド・プロセッサはJTAGチェーンのアドレスをアサートし、R_nWとR_ASの信号によってエンベデッド・プロセッサがチェーンのアクセスを要求していることをインタフェース用PLDに伝えます。ライト動作を行うことで、3個のDレジスタを通じてPLDのJTAG出力とデータ・バス、data[3..0]が接続されます。ここでDレジスタのクロックにはシステム・クロック (CLK) が使用されます。このクロックはプロセッサが使用するクロックと同じものにするのが可能です。同じように、リード動作を行うことでトライ・ステート・バッファがイネーブルとなり、TDO信号がプロセッサに戻されます。また、このデザインでは、TDI、TMSおよびTCKのレジスタの値をリード・バックするためのハードウェア接続も構成されています。このオプション機能を活用することによって開発段階でインタフェース用PLDの各レジスタが有効な値になっているかをソフトウェアでチェックすることができます。さらに、このデザインには、MasterBlasterまたはByteBlasterMVダウンロード・ケーブルを通じたデバイス・チェーンへのプログラムを可能にするためのマルチプレクサ・ロジックも含まれています。このオプション機能は、開発の試作段階において、プログラミングまたはコンフィギュレーションの結果を検証する必要があるときに便利です。



このインタフェース用PLDのデザインはアルテラのFTPサイト、<ftp://ftp.altera.com/pub/misc/intpld.zip>にMAX+PLUS® IIのグラフィック・デザイン・ファイル (.gdf) として提供されています。

ボード・レイアウト

IEEE Std. 1149.1のJTAGチェーンを通じてプログラムまたはコンフィギュレーションを行うボードを設計する場合は、以下の点が重要です。

- TCKの信号パターンはクロック・ツリーとして扱う。
- TCKにプルダウン抵抗を使用する。
- JTAG信号のパターンはできるだけ短くする。
- 出力が規定のロジック・レベルになるように外部抵抗を付加する。

TCK信号のパターンに対する保護と正常波形の維持

TCKはJTAGチェーン全体に対するクロック信号となっています。JTAGチェーンに接続されたデバイスはTCK信号のエッジでトリガされるため、TCK信号を高周波ノイズから保護し、TCKの波形を正常な状態に維持することが必要です。TCKは対応するデバイス・ファミリのデータシートで規定されている立ち上がり時間 (t_R) と立ち下り時間 (t_F) のパラメータ値を満足していなければなりません。また、TCK信号のオーバシュートやアンダシュートまたはリングングを防ぐために終端が必要になることもあります。この信号はソフトウェアで生成され、マイクロプロセッサの汎用I/Oピンから供給されるため、この点は見逃されがちです。

TCKのプルダウン抵抗

電源投入時に、JTAGのTest Access Port (TAP)をあらかじめ規定されたステートに維持するためには、プルダウン抵抗を使用してTCKをLowレベル

に保持しなければなりません。この抵抗が欠如していると、電源投入時に JTAG が BST ステートになり、ボード上で信号のコンフリクトが発生する原因となることがあります。この抵抗の標準的な値は $1k\Omega$ です。

JTAG 信号の配線パターン

JTAG 信号の配線パターンを短くすることが、ノイズやドライブの強さに関連した問題の解消に役立ちます。TCK と TMS のピンには特別な注意を払う必要があります。TCK と TMS は JTAG チェイン内の各デバイスと接続されるため、これらの信号の配線パターンに対する負荷は TDI や TDO よりも大きくなります。JTAG チェインの長さや負荷の状態によっては、追加のバッファを付加してこれらの信号が正常な波形を維持しながらプロセッサとの間で伝送されるようにする必要があります。

外部抵抗

各出力ピンには外部抵抗を付加し、プログラミングまたはコンフィギュレーションの実行時にこれらの出力が規定されたロジック・レベルになるようにする必要があります。出力ピンは、プログラミングまたはコンフィギュレーションの実行時にトライステートになります。MAX 7000、FLEX 10K、APEX 20K およびすべてのコンフィギュレーション・デバイスでは、弱い内部抵抗（例： $50k\Omega$ ）によって、ピンがプルアップされます。ただし、イン・システム・プログラミングやイン・サーキット・リコンフィギュレーションの実行時に、アルテラのすべてのデバイスがこの弱いプルアップ抵抗を持つわけではありません。どのデバイスがこのプルアップ抵抗を内蔵しているかについては、対応する各デバイス・ファミリのデータシートで確認してください。アルテラは、重要な入力をドライブしている出力については、 $1k\Omega$ オードの外部抵抗を接続して、これらの出力を適切なロジック・レベルに接続することを推奨します。

ボードに実装される各部品のレイアウトを進める前に、デザインの解析が必要になることもあります。特に、信号の波形を正常に維持するための解析は重要です。場合によっては、ディスクリートのバッファまたは終端が必要かどうかを判断するために、JTAG チェインの負荷とレイアウトを解析する必要があります。



詳細については、アプリケーション・ノート、AN 100「*In-System Programmability Guidelines*」（日本語版「ISP を使用するためのガイドライン」）を参照してください。

ソフトウェア 開発

アルテラのエンベデッド・プログラミングおよびコンフィギュレーションでは、MAX + PLUS II ソフトウェア・ツールから出力される Jam ファイルと標準化されたソフトウェア、Jam Player を使用します。Jam ファイルにはアルテラのデバイスをプログラミングまたはコンフィギュレーションするためのすべてのデータが含まれるため、これらのツールの開発時に設計者が行う作業は最小で済みます。開発に要する時間のほとんどは、Jam Player のホストとなるエンベデッド・プロセッサへのポーティングする作業に割り当てられます。



Jam Byte-Code Playerのポーティング方法に関する詳細については、9ページの「Jam STAPL Byte-Code Playerのポーティング方法」を参照してください。

Jamファイル (.jamおよび.jbc)

アルテラは下記の2種類のJamファイルをサポートしています。

- ASCIIテキスト・ファイル (.jam)
- Jam Byte-Codeファイル (.jbc)

ASCII テキスト・ファイル (.jam)

アルテラはJamファイルを以下の2種類のフォーマットでサポートしています。

- JEDEC Jam STAPLフォーマット
- Jamのバージョン1.1 (JEDECで制定される前のフォーマット)

JEDEC Jam STAPLフォーマットには、JEDECの標準規格、JESD-71Aで規定されているシンタックスが使用されます。アルテラは、すべての新しいプロジェクトには、JEDEC Jam STAPLファイルの使用を推奨します。ほとんどの場合、Jamファイルはテストの環境でも使用されます。

Jam Byte-Code ファイル (.jbc)

JBCファイルは、Jamファイルをコンパイルしたバイナリ・ファイルです。JBCファイルはバーチャルのプロセッサ・アーキテクチャに対してコンパイルされ、ASCIIのJamコマンドがバーチャル・プロセッサと互換性のあるバイト・コードの命令にマッピングされます。JBCファイルには以下の2種類があります。

- Jam STAPL Byte-Code (JEDEC Jam STAPL ファイルをコンパイルしたもの)
- Jam Byte-Code (Jamのバージョン1.1をコンパイルしたもの)

Jam STAPL Byte-Codeファイルで使用されるメモリは最小になるため、アルテラはエンベデッド・アプリケーションにはJam STAPL Byte-Codeファイルの使用を推奨します。

Jamファイルの生成方法

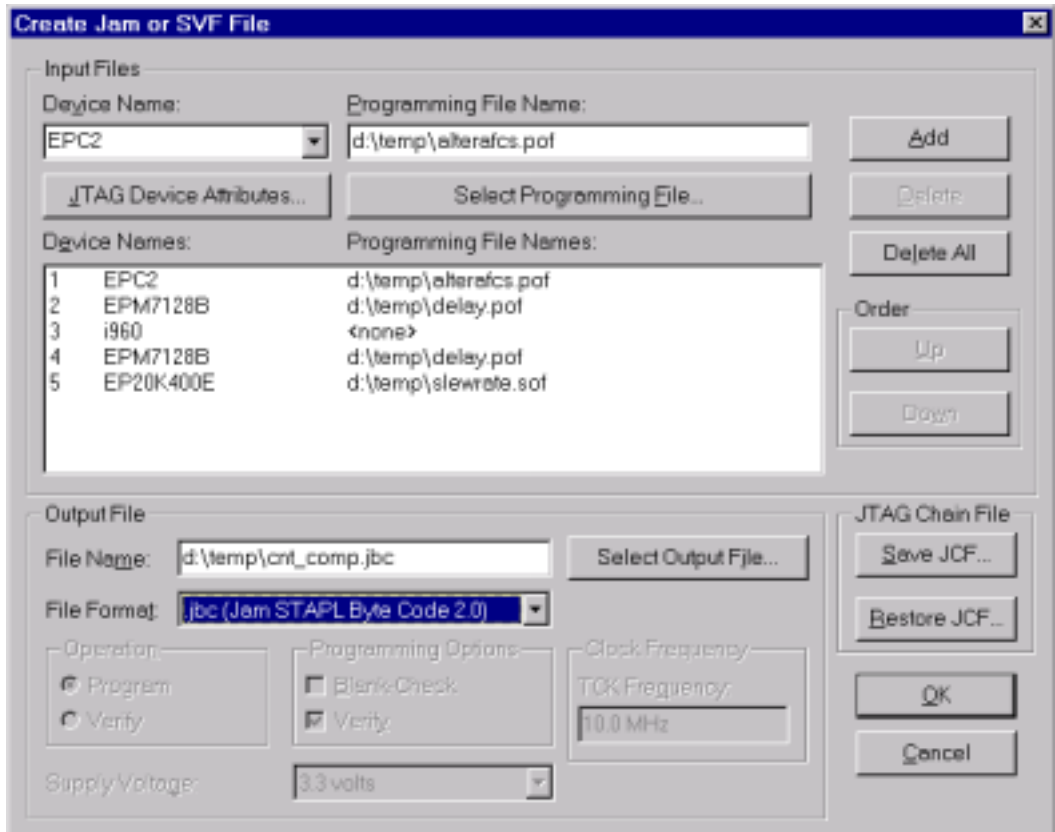
MAX+PLUS IIソフトウェアはJamファイルとJBCファイルの双方を生成することができます。また、スタンドアロン動作が可能なJam Byte-Codeコンパイラにより、JamファイルをJBCファイルにコンパイルすることもできます。このコンパイラは、機能的に等価なJBCファイルを生成します。




Jam Byte-Codeコンパイラは、Jamのwebサイト、<http://www.jamisp.com>からダウンロードすることができます。

MAX+PLUS IIからJBCファイルをダイレクトに生成する方法は簡単です。MAX+PLUS IIは、単独または複数のJBCファイルから複数のデバイスをプログラミングおよびコンフィギュレーションする機能もサポートしています。図3は、MAX+PLUS IIソフトウェアからどのJBCファイルを生成させるかを指定するダイアログ・ボックスを示しています。

図3 マルチ・デバイスJTAGチェーンに対するJBCファイルの生成画面



MAX+PLUS IIソフトウェアを使用したJBCファイルの生成は、以下の手順で行います。

 APEX™デバイス用のJBCファイルを生成するときは、Quartus™ソフトウェアによって生成されるSRAMオブジェクト・ファイル(.sof)を使用して、同じ手順で実行してください。

- 1 . MAX+PLUS II Programmerから、Create Jam or SVF File (Fileメニュー)を選択します。

- 2 . Create Jam or SVF Fileのダイアログ・ボックスで、JTAGチェーン内のデバイス名とシーケンス、各デバイスの関連するプログラミング・ファイルを指定します。
- 3 . *File Format*のドロップ・ダウン・リスト・ボックスからJam STAPL Byte-Codeファイルを指定します。
- 4 . OKをクリックします。

JTAGチェーンには、アルテラのデバイスとアルテラ以外のデバイスの双方を含めることができます。*Programming File Name*のフィールドでJTAGチェーン内のデバイスに対するプログラミング・ファイルが指定されていないと、そのデバイスはバイパスされます。

Jam Player

Jam PlayerはJamファイル内の記述情報を読み取り、ターゲットPLDをプログラムまたはコンフィギュレーションするデータに変換します。Jam Playerは特定のデバイス・アーキテクチャやベンダのデバイスをプログラムまたはコンフィギュレーションを行うのではなく、Jamファイルの仕様で規定されているシンタックスを読み込み、その内容を解読するだけです。フィールドで実施されるデザインの変更はJam Playerではなく、Jamファイル内で定義されます。このため、フィールドでのアップグレードを行うたびにJam Playerを変更する必要はありません。

Jam Playerには、2種類のJamファイルに対応したASCII Jam STAPL PlayerとJam STAPL Byte-Code Playerの2種類があります。このアプリケーション・ノートに記述されている一般的なコンセプトは双方のJam Playerに適用されますが、以下の情報はJam STAPL Byte-Code Playerを対象にしたものとなっています。

Jam Playerの互換性

エンベデッドJam Player (Jam STAPL Byte-Code Player) は、JEDECで規定された標準フォーマットに準拠したJamファイルを読み込むことができます。エンベデッドのJam STAPL Byte-Code Playerは以前のバージョン1.1のシンタックスを使用しているJamファイルとの互換性を持っています。双方のJam Playerは、バージョン1.1のJamファイルとJam STAPLファイルに対応できる互換性を持っています。



バージョン1.1のシンタックスに対するアルテラのサポートについては、アプリケーション・ノート、AN 88「*Using the Jam Language for ISP & ICR via an Embedded Processor*」を参照してください。

Jam STAPL Byte-Code Player

Jam STAPL Byte-Code Playerは、16ビットおよび32ビット・プロセッサ用のC言語にコーディングされています。8ビットのプロセッサもJamのwebサイト、<http://www.jamisp.com>に提供されている特定のソース・コードのサブセットによってサポートされています。



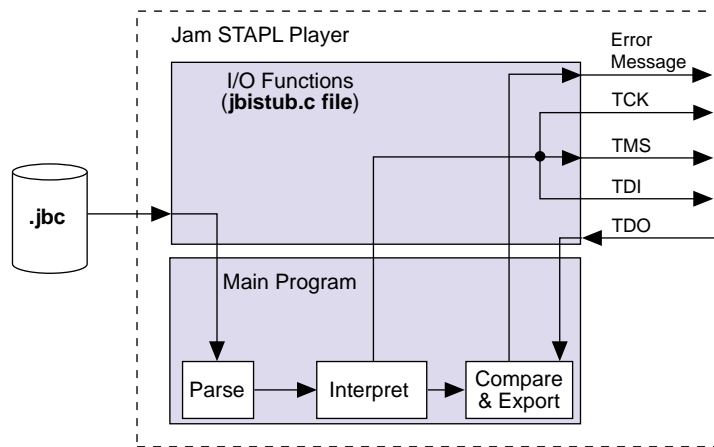
8ビットのプロセッサに対するアルテラをサポート状況については、アプリケーション・ノート、AN 111「*Embedded Programming Using the 8051 & Jam Byte-Code*」を参照してください。

16ビットおよび32ビットのコードは下記の2種類に分割されます。

- I/O機能を制御するためのコードで、特定のハードウェアに適用されるプラットフォームごとの固有コード (jbistub.c)
- Player内部の機能を実行するための汎用コード (他のすべてのC言語ファイル)

図4は、ソース・コード・ファイルが果たす機能を示したものです。jbistub.cファイルの内部を各プラットフォームに対応した固有のコードにカスタマイズすることによって、Jam STAPL Byte-Code Playerを特定のプロセッサにポーティングする作業が簡略化されます。

図4 Jam STAPL Byte-Code Playerのソース・コードの構造



Jam STAPL Byte-Code Playerのポーティング方法

jbistub.cファイルのデフォルト構成にはDOS、32bitのWindows、およびUNIX用のコードが含まれているため、これらのオペレーティング・システムにおけるソース・コードのコンパイル、機能の検証、デバッグが簡単に行えるようになっています。エンベデッド・システムの環境では、#defineのプロセッサ・ステートメントを使用するだけで、これらのコードを簡単に取り除くことができます。また、コードのポーティングは、jbistub.cファイル内の特定の部分のみを変更することだけで実現できます。

Jam Playerをポーティングするときは、表 1 に示されているjbstub.cファイル内のいくつかのファンクションをカスタマイズする必要があります。

ファンクション名	説明 / 機能
jbi_jtag_io()	IEEE 1149.1 のJTAG 信号、TDI、TMS、TCK および TDO をインタフェースする機能
jbi_export()	UES (User Electronic Signature) のような情報をコールしているプログラムに受け渡す機能
jbi_delay()	プログラミング・パルスの実現、または実行時に要求される遅延時間を実現する機能
jbi_vector_map()	IEEE 1149.1 JTAG 以外の信号のピンマッピング処理を実行する機能
jbi_vector_io()	IEEE 1149.1 JTAG 以外の信号を VECTOR MAP で規定されている通りにアサートする機能

以下の手順を実行することによって、要求されるコードのカスタマイズを確実に行うことができます。

1. 不必要なコードを排除するプリプロセッサのステートメントをセットする。
2. JTAG信号をハードウェア・ピンにマッピングする。
3. jbi_export() からのメッセージを処理する。
4. 遅延時間のキャリブレーションをカスタマイズする。

ステップ-1 不必要なコードを排除するプリプロセッサのステートメントをセットする

jbstub.cの先頭に設定されているデフォルトのパラメータ、PORTをEMBEDDEDに変更し、すべてのDOS、Windows、UNIXのソース・コードおよび含まれているライブラリを取り除きます。

```
#define PORT EMBEDDED
```

ステップ-2 JTAG信号をハードウェア・ピンにマッピングする

jbi_jtag_io() ファンクションには、バイナリのプログラミング・データを送信および受信するためのコードが含まれています。4本のJTAG信号は、それぞれエンベデッド・プロセッサのピンに再マッピングされる必要があります。デフォルトの設定では、PCの平行ル・ポートへの書き込みが行われるソース・コードとなっています。この場合、jbi_jtag_io() ファンクションは、図 5 に示すようにJTAGピンをPC平行ル・ポートのレジスタにマッピングします。

図5 デフォルト設定でのPCパラレル・ポートへの信号マッピング
注(1)

7	6	5	4	3	2	1	0	I/Oポート
0	TDI	0	0	0	0	TMS	TCK	出力データ- ベース・アドレス
TDO	X	X	X	X	---	---	---	入力データ- ベース・アドレス + 1

注:

- (1) PCパラレル・ポートでは、ハードウェアがMSBになっているTDOの極性を反転させます。

下記に示すjbi_jtag_io() のソース・コードに、マッピングが示されています。

```
int jbi_jtag_io(int tms, int tdi, int read_tdo)
{
    int data=0;
    int tdo=0;

    if (!jtag_hardware_initialized)
    {
        initialize_jtag_hardware();
        jtag_hardware_initialized=TRUE;
    }
    data = ((tdi?0x40:0)|(tms?0x2:0));          /*TDI,TMS*/
    write_byteblaster(0,data);
    if (read_tdo)
    {
        tdo=(read_byteblaster(1)&0x80)?0:1; /*TDO*/
    }
    write_blaster(0,data|0x01);                /*TCK*/
    write_blaster(0,data);
    return (tdo);
}
```

変更前のソース・コード（デフォルト）では、PCのパラレル・ポートでTDOの実際の値が反転されるようになっていました。上記のjbi_jtag_io() のソース・コードでは、TDOの極性を再度反転させて元のデータに戻しています。TDOの値を反転させるためのソース・コードは下記の通りです。

```
tdo=(read_byteblaster(1)&0x80)?0:1;
```

ターゲットとなるプロセッサがTDOの極性を反転させないときは、この部分のソース・コードを下記のようにします。

```
tdo=(read_byteblaster(1)&0x80)?1:0;
```

信号を正しいアドレスにマッピングするために、左シフト (<<)、または右シフト (>>) の演算子を使用してください。例えば、TMSとTDIをそれぞれポート-2とポート-3にマッピングする場合は、下記のコードを記述します。

```
data=((tdi?0x40:0)>>3)|((tms?0x02:0)<<1));
```

同じ処理をTCKとTDOにも適用してください。

read_byteblasterとwrite_byteblasterの信号はconio.hのライブラリからinp()とoutp()のファンクションを使用して、ポートに対するリードとライトの動作を実行します。これらのファンクションがない場合は、これらと等価のファンクションで同様の機能を実現する必要があります。

ステップ-3 jbi_export()からのメッセージを処理する

jbi_export()ファンクションは、printf()ファンクションを使用してテキスト・メッセージをstdioに送信します。Jam STAPL Byte-Code Playerはjbi_export()ファンクションを使用して情報(デバイスのUESコードやUSERCODEなど)をオペレーティング・システムまたはJam Playerをコールするプログラムに受け渡します。jbi_export()ファンクションはテキストをstringの形式で、また数値を10進法の形式で受け渡します。



これらの条件の詳細については、アプリケーション・ノート、AN 39「IEEE 1149.1(JTAG) Boundary-Scan Testing in Altera Devices」を参照してください。

stdoutに対応するデバイスがない場合は、情報をファイルまたはストレージ・デバイスに戻すか、Jam Playerをコールするプログラムに変数の戻りとして受け渡すことができます。

ステップ-4 遅延時間のキャリブレーションをカスタマイズする

calibrate_delay()ファンクションは、ホスト・プロセッサが1 ms内に何回のループ動作を実行するかを決定します。プログラミングとコンフィギュレーションには正確な遅延が要求されるため、このキャリブレーションが重要になります。デフォルトの設定では、これが1 msあたり1,000ループになっており、これは下記のように記述されています。

```
one_ms_delay = 1000
```

このパラメータがあらかじめ判明しているときは、その値に調整します。その値が事前に判明していないときは、WindowsおよびDOSのプラットフォーム用のコードと類似したものが使用できます。これらのプラットフォーム用のコードでは、1個のwhileループの実行に要するクロック・サイクル数をカウントするようになっています。そしてこのカウントが複数回実行され、その平均値を計算してどのような遅延を基準にすべきかを正確に算出します。この方法の利点は、ホスト・プロセッサのスピードに応じてキャリブレーションの値を変更できることです。

Jam STAPL Byte-Code Playerのポーティングが完了し、それが動作するようになったら、次にターゲット・デバイスでJTAGポートのタイミングとスピードを検証します。MAX[®]、FLEX[®]、およびAPEXデバイスのタイミング・パラメータは以下の表2と表3に示されている値を満足している必要があります。

シンボル	パラメータ	MAX 9000		MAX 7000A		MAX 7000AE		MAX 7000S		単位
		最小	最大	最小	最大	最小	最大	最小	最大	
t _{JCP}	TCKクロックの期間	100		100		100		100		ns
t _{JCH}	TCKクロックのHigh期間	50		50		50		50		ns
t _{JCL}	TCKクロックのLow期間	50		50		50		50		ns
t _{JPSU}	JTAGポートのセットアップ・タイム	20		20		20		20		ns
t _{JPH}	JTAGポートのホールド・タイム	45		45		45		45		ns
t _{JPCO}	JTAGポートの「Clock-to-Output」遅延		25		25		25		25	ns
t _{JPZX}	JTAGポートのハイ・インピーダンスから有効出力まで		25		25		25		25	ns
t _{JPXZ}	JTAGポートの有効出力からハイ・インピーダンスまで		25		25		25		25	ns
t _{JSSU}	キャプチャ・レジスタのセットアップ・タイム	20		20		20		20		ns
t _{JSH}	キャプチャ・レジスタのホールド・タイム	45		45		45		45		ns
t _{JSCO}	アップデート・レジスタの「Clock-to-Output」遅延		25		25		25		25	ns
t _{JSZX}	アップデート・レジスタのハイ・インピーダンスから有効出力まで		25		25		25		25	ns
t _{JSXZ}	アップデート・レジスタの有効出力からハイ・インピーダンスまで		25		25		25		25	ns

表3 IEEE Std. 1149.1のタイミング・パラメータ

シンボル	パラメータ	MAX 3000A		MAX 7000B		APEX 20K		FLEX 10K		EPC2		単位
		最小	最大	最小	最大	最小	最大	最小	最大	最小	最大	
t _{JCP}	TCKクロックの期間	100		100		100		100		100		ns
t _{JCH}	TCKクロックのHigh期間	50		50		50		50		50		ns
t _{JCL}	TCKクロックのLow期間	50		50		50		50		50		ns
t _{JPSU}	JTAGポートのセットアップ・タイム	20		20		20		20		20		ns
t _{JPH}	JTAGポートのホールド・タイム	45		45		45		45		45		ns
t _{JPCO}	JTAGポートの「Clock-to-Output」遅延		25		25		25		25		25	ns
t _{JPZX}	JTAGポートのハイ・インピーダンスから有効出力まで		25		25		25		25		25	ns
t _{JPXZ}	JTAGポートの有効出力からハイ・インピーダンスまで		25		25		25		25		25	ns
t _{JSSU}	キャプチャ・レジスタのセットアップ・タイム	20		20		20		20		20		ns
t _{JSH}	キャプチャ・レジスタのホールド・タイム	45		45		45		45		45		ns
t _{JSCO}	アップデート・レジスタの「Clock-to-Output」遅延		25		25		25		25		25	ns
t _{JSZX}	アップデート・レジスタのハイ・インピーダンスから有効出力まで		25		25		25		25		25	ns
t _{JSXZ}	アップデート・レジスタの有効出力からハイ・インピーダンスまで		25		25		25		25		25	ns

Jam STAPL Byte-Code Playerが規定されたタイミングの範囲内で動作しないときは、適切な遅延時間となるようにコードを最適化する必要があります。タイミング違反は、プロセッサの性能が非常に高く、TCKを10MHzよりも高速で生成することができる場合にのみ発生します。



アルテラは、jbistub.c以外のファイルについては、それらのソース・コードをデフォルト設定から変更せずに保存しておくことを推奨します。これらのファイルのコードを変更することによって、Jam Playerが予期しない動作を行うことがあります。

Jam STAPL Byte-Code Playerで使用されるメモリ・サイズ

Jam STAPL Byte-Code Playerに使用されるメモリ容量は予測可能となっています。このセクションでは、使用されるROMとRAMのサイズを推定する方法を解説します。

ROMサイズの推定

Jam PlayerとJBCファイルのストアに必要なROMサイズの推定には、下記の計算式を使用します。

$$\text{ROMサイズ} = \text{JBCファイルのサイズ} + \text{Jam Playerのサイズ}$$

ここで、JBCファイルのサイズは、プログラミング・データのストアに必要なメモリ容量とプログラミング・アルゴリズムのストアに必要な容量とに分割することができます。このため、JBCファイルのサイズは下記の式を使用して推定することができます。

$$\text{JBCファイルのサイズ} = Alg + \sum_{k=1}^N Data$$

ここで

- Alg = アルゴリズムのストアに使用されるスペース
- $Data$ = 圧縮されたプログラミング・データに使用されるスペース
- k = ターゲットとなるデバイスを表すインデックス
- N = チェイン内のターゲット・デバイスの数

この計算式で推定されるJBCファイルのサイズは、デバイス内部の使用率に応じて $\pm 10\%$ の範囲で変動します。リソースの使用率が低いときは、圧縮アルゴリズムが繰り返しになっているデータを見つけてファイル・サイズを最小に抑えるため、JBCファイルのサイズが縮小される傾向になります。

また、上記の計算式から、アルゴリズムのサイズはデバイス・ファミリーごとに一定になりますが、プログラミング・データのサイズはターゲットとなるデバイスの数が増加することによって増加していき、ことがわかります。同じデバイス・ファミリーにおいては、JBCファイルのサイズは、ターゲット・デバイスの数に応じてリニアに増加します（データの数に応じて）。

ターゲット・デバイスがアルテラと同じデバイス・ファミリーになっている場合、アルゴリズムに必要なファイル・サイズは表 4 に示されている通りとなります。また、Jam言語をサポートしているアルテラの複数のデバイス・ファミリーがターゲットになっている場合は、アルゴリズムに必要なファイル・サイズが表 5 のようになります。

表 4 単独のアルテラ・デバイス・ファミリがターゲットになっているときのアルゴリズム部分のファイル・サイズ	
デバイス名	JBC ファイル内のアルゴリズム部分の標準的なサイズ (Kバイト)
APEX 20K	14
APEX 20KE	14
FLEX 10K	15
FLEX 10KE	15
FLEX 10KA	15
FLEX 10KB	15
EPC2	19
MAX 7000AE	21
MAX 7000	21
MAX 3000A	21
MAX 9000	21
MAX 7000S	25
MAX 7000A	25
MAX 7000B	17

表 5 複数のアルテラ・デバイス・ファミリがターゲットになっているときのアルゴリズム部分のファイル・サイズ	
デバイス名	JBC ファイル内のアルゴリズム部分の標準的なサイズ (Kバイト)
FLEX 10K, MAX 7000A, MAX 7000S, MAX 7000AE (1)	31
FLEX 10K, MAX 9000, MAX 7000A, MAX 7000S, MAX 7000AE	45
MAX 7000S, MAX 7000A, MAX 7000AE	31
MAX 9000, MAX 7000A, MAX 7000S, MAX 7000AE	45

注：

- (1) FLEXまたはAPEXデバイスのコンフィギュレーションとMAXデバイスのプログラミングを行う場合、FLEXまたはAPEXデバイスのコンフィギュレーションのアルゴリズムに追加されるサイズは相対的に小さくなります。

表 6 はJam言語によるISPをサポートしているアルテラの各デバイスに使用されるデータ部分のサイズを示しています

表 6 データ部分のサイズ (1/2)		
デバイス名	Jam STAPL Byte-Codeにおけるデータ部分の標準的なサイズ (Kバイト)	
	圧縮	非圧縮 ⁽¹⁾
EPM7032S	8	8
EPM7032AE	6	6
EPM7064S	13	13
EPM7064AE	8	8
EPM7128S, EPM7128A	5	24
EPM7128AE	4	12
EPM7128B	4	12
EPM7160S	10	28
EPM7192S	11	35
EPM7256S, EPM7256A	15	51
EPM7256AE	11	18
EPM7512AE	18	37
EPM9320, EPM9320A	21	57
EPM9400	21	71
EPM9480	22	85
EPM9560, EPM9560A	23	98
EPF10K10, EPF10K10A	12	15
EPF10K20	21	29
EPF10K30	33	47
EPF10K30A	36	51
EPF10K30E	36	59
EPF10K40	37	62
EPF10K50, EPF10K50V	50	78
EPF10K50E	52	98
EPF10K70	76	112
EPF10K100, EPF10K100A, EPF10K100B	95	149
EPF10K100E	102	167
EPF10K130E	140	230
EPF10K130V	136	199
EPF10K200E	205	345
EPF10K250A	235	413

デバイス名	Jam STAPL Byte-Code におけるデータ部分の標準的なサイズ (Kバイト)	
	圧縮	非圧縮 (1)
EP20K100	128	244
EP20K200	249	475
EP20K400	619	1,180
EPC2	136	212

注：

- (1) プログラミング・データを非圧縮形式のJBCファイルで生成する方法については、販売代理店または日本アルテラの応用技術部にお問い合わせください。

JBCファイルのサイズの推定が完了したら、次に表7に示されている情報をベースにしてJam Playerのサイズを推定します。

プロセッサ	説明	サイズ (Kバイト)
16 ビット	MasterBlasterまたはByteBlasterMV ダウンロード・ケーブルを使用する 286/68K プロセッサ	80
32 ビット	MasterBlasterまたはByteBlasterMV ダウンロード・ケーブルを使用するペンティアム /486 プロセッサ	85

ダイナミック・メモリ領域の推定

下記の計算式で、Jam Playerに必要なDRAMの最大容量を推定することができます。

$$\text{RAMサイズ} = \text{JBCファイルのサイズ} + \sum_{k=1}^N \text{Data (非圧縮のデータ・サイズ)}_k$$

ここで、JBCファイルのサイズは、1個または複数のデバイスをターゲットにしたときの計算式で決定されます (15ページの「ROMサイズの推定」を参照)。

Jam Playerによって使用されるRAMの容量は、JBCファイルのサイズにターゲットとなる各デバイスに必要な全データ量を加えたものとなります。JBCファイルが圧縮されたデータを使用して生成されている場合は、Jam PlayerがRAMを使用してデータを解凍し、これを暫定的にストアします。非圧縮データのサイズは表5と表6に示されています。非圧縮のJBCファイルが使用される場合は、下記の式が適用できます。

$$\text{RAMサイズ} = \text{JBCファイルのサイズ}$$



スタックやヒープに対して要求されるメモリ容量は、Jam STAPL Byte-Code Playerによって使用されるメモリ容量に比較して非常に小さくなるため、無視することができます。スタックの深さの最大値は、jbimain.cファイル内のJBI_STACK_SIZEのパラメータによって設定されます。

メモリ・サイズの推定例

モトローラ社の16ビット・プロセッサ、68000を使用して、圧縮されたデータで構成されたJBCファイルで、IEEE Std. 1149.1 JTAGチェーン内のEPM7128AEとEPM7064AEをプログラムするときに必要なメモリ・サイズを推定した例を以下に示します。使用されるメモリ・サイズを推定するときは、最初に必要なROMのサイズを推定し、次に使用されるRAMの容量を推定します。Jam STAPL Byte-Code Playerに要求されるDRAMの容量は、以下の手順で推定します。

1. JBCファイルのサイズを計算します。ここでは、下記に示す複数のデバイスを対象にした計算式を使用して、JBCファイルのサイズを算出します。この例では、JBCファイルは圧縮されたデータで構成されているため、表5と表6に示されている圧縮されたデータの場合のファイル・サイズを使用して、Data部分のサイズを計算します。

$$\text{JBCファイルのサイズ} = Alg + \sum_{k=1}^N \text{Data}$$

ここで、

$$Alg = 21\text{Kバイト}$$

$$Data = \text{EPM7064AEのデータ} + \text{EPM7128AEのデータ} = 8 + 4 = 12\text{Kバイト}$$

したがって、JBCファイルのサイズは33Kバイトとなります。

2. 次に、JBC Playerのサイズを推定します。この例では、プロセッサが16ビットの68000になっているため、JBC Playerのサイズが80Kバイトになります。必要になるROMサイズは下記の式で計算できます。

$$\text{ROMのサイズ} = \text{JBCファイルのサイズ} + \text{Jam Playerのサイズ}$$

$$\text{ROMのサイズ} = 113\text{Kバイト}$$

3. 最後に下記の計算式を使用してRAMのサイズを推定します。

$$\text{RAMのサイズ} = 33\text{Kバイト} + \sum_{k=1}^N \text{Data (非圧縮データのサイズ)}_k$$

この例では、JBCファイルに圧縮されたデータが使用されているため、各デバイスの非圧縮データのサイズを合計して必要なトータルのRAM容量を算出する必要があります。この例での非圧縮データのサイズは、次の通りです。

EPM7064AE = 8Kバイト

EPM7128AE = 12Kバイト

トータルのDRAMのサイズは下記のように計算されます。

RAMのサイズ = 33Kバイト + (8Kバイト + 12Kバイト) = 53Kバイト

一般的にJamファイルに対してはROMよりもRAMに大きな容量が必要になりますが、RAMはROMよりも安価であり、多数のデバイスがプログラムされるほど簡単なアップグレードを実現するために要求される全体的なコストが低下するため、これは好ましい傾向です。ほとんどのアプリケーションでは、簡単なアップグレードを実現することのほうが、メモリのコストよりも重要です。

Jamを使用したデバイスのアップデート

フィールドにあるデバイスをアップデートすることは、ほとんどの場合、新しいJBCファイルをダウンロードし、Jam STAPL Byte-Code Playerを動作させて、デバイスをプログラムすることを意味します。

Playerを実行させる主要なエントリ・ポイントは、`jbi_execute()`になります。このルーティンは特定の情報をPlayerに受け渡します。Playerの動作が完了すると、`exit_code`のコードに戻り、動作中のエラーの詳細が表示されます。このインタフェース部分は、以下のルーティンのプロトタイプで規定されます。

```

JBI_RETURN_TYPE jbi_execute
(
    PROGRAM_PTR program
    long program_size,
    char *workspace,
    long workspace_size,
    *action,
    char **init_list,
    long *error_line,
    init *exit_code
)

```

`jbistub.c`の`main()`内のコードで`jbi_execute()`に受け渡される変数が決定されます。ほとんどの場合、このコードはエンベデッド・システムの環境には適用できません。したがって、このコードは除去し、`jbi_execute()`のルーティンをそれぞれのエンベデッド・システムの環境に適合するように設定します。表 8 はこのときの各パラメータを、表 9 は各アクション名を解説したものです。

表8 パラメータ 注(1)		
パラメータ	必須/オプション	説明
program	必須	JBCファイルに対するポインタ。ほとんどのエンベデッド・システムでは、このパラメータをjbi_execute()をコールする前のポインタのアドレスに指定する。
program_size	必須	JBCファイルが占有するメモリの容量(バイト単位)
workspace	オプション	JBC Playerが要求されたファンクションを実行するときに使用できるダイナミック・メモリに対するポインタ。このパラメータを設定する目的は、Playerのメモリ使用領域を事前に指定したメモリ空間に制限することにある。このメモリ空間は、jbi_execute()をコールする前に割り当てられる必要がある。使用されるダイナミック・メモリの容量に十分な余裕があるときは、このパラメータをnullに指定する。これをnullに設定すると、Playerは指定されたファンクションの実行に必要なメモリ空間をダイナミックに割り当てることができる。
workspace_size	オプション	workspaceで指定されたメモリの容量(バイト単位)を表すスケーラ。
action	必須	文字列(Playerにアクションを指示するテキスト列)に対するポインタ。例としては、PROGRAMやVERIFYなどがある。ほとんどの場合、このポインタはPROGRAMの文字列に対して設定される。Playerは大文字と小文字の双方に対応できるため、このテキストは大文字でも小文字でも良い。Playerは <i>Jam Standard Test and Programming Language Specification</i> で規定されているすべてのアクションをサポートする。表9を参照。文字列はnullコードで終了していなければならない。
init_list	オプション	複数の文字列に対するポインタの配列。このパラメータは、バージョン-1.1のJam Playerに適用するときに使用される。(2)
error_line	—	倍長整数列に対するポインタ。実行時にエラーが発生したときは、Playerがエラーの発生しているJBCファイルの行を記録する。
exit_code	—	倍長整数列に対するポインタ。JBCファイルのシンタックスまたは構造に関連したエラーが発生した場合は、このコードにリターンする。このようなエラーが発生したときは、サポート・ベンダに連絡して、exitコードが発生した詳しい状況を説明する。

注:

- (1) 必須のパラメータは、Playerが動作する時に受け渡される必要があります。
- (2) 詳細については、アプリケーション・ノート、AN 88「*Using the Jam Language for ISP & ICR via an Embedded Processor*」を参照してください。

表9 サポートされるアクション 注(1)

アクション名	説明
CHECKCHAIN	IEEE Std. 1149.1 のスキャン・チェーンの連続性を検証する。
READ_IDCODE	IEEE Std. 1149.1 のIDCODEをリードし、これをEXPORTする。
READ_USERCODE	IEEE Std. 1149.1 のUSERCODEをリードし、これをEXPORTする。
READ_UES	UESCODEをリードし、これをEXPORTする。
ERASE	デバイスのバルク・イレーズを実行する。
BLANK_CHECK	デバイスがイレーズされているかをチェックする。
PROGRAM	デバイスをプログラムする。
VERIFY	デバイスにプログラムされたデータとプログラミング・データの一致を検証する。
READ	デバイスからプログラミング・データをリードする。
CHECKSUM	デバイスのプログラミング・データからチェックサムを計算する。
SECURE	デバイスのセキュリティ・ビットをセットする。
QUERY_SECURITY	セキュリティ・ビットがセットされているかをチェックする。
TEST	テストを実行する。このテストには、バウンダリ・スキャン・テスト、内部テスト、ベクタ・テスト、内蔵の自己診断テストなどが含まれる。

注：

- (1) アルテラの全デバイスに対して、ここにリストされたすべてのアクションを適用できるわけではありません。どのアクションが適用できるかについては、各デバイス・ファミリのデータシートで確認してください。

Playerはtype JBI_RETURN_TYPEまたは整数のステータス・コードを返します。この値はアクションが問題なく実行されたかどうかを示します（問題がないときは、"0"を返す）。jbi_execute()は*Jam Standard Test and Programming Language Specification*で規定されているように、表10に示されている任意のexitコードに戻ることができます。

表10 Exitコード

Exitコード	説明
0	成功
1	チェーンの不良が発生
2	IDCODE リードで不良を確認中
3	USERCODE リードで不良を確認中
4	UESCODE リードで不良を確認中
5	ISP 不良が発生中
6	デバイスID が認識不能
7	サポートされていないデバイス・バージョン
8	イレーズ不良
9	ブランク・チェック不良
10	プログラミング不良
11	バイファイ不良
12	リード不良
13	チェックサム不良
14	セキュリティ・ビットをセットしたときの不良
15	セキュリティ・ビットの状態をチェックしたときの不良
16	ISP 不良から抜ける
17	システム・テスト不良をチェック中

Jam STAPL Byte-Code Playerを動作させる方法

Jam STAPL Byte-Code Playerのコールは、他のサブルーティンをコールするのと同じように実行することができます。この場合、サブルーティンは指定されたアクションとファイル名となり、それぞれのファンクションが実行されます。

場合によっては、デバイスのデザインがフィールドで最新のものにアップデートされることがあります。この場合、JTAGのUSERCODEが、PLDのデザインのレジジョンを示す電気的な「スタンプ」の役割を果たす目的に使用されることがあります。USERCODEが古い値にセットされている場合は、エンベデッド・ファームウェアによりデバイスをアップデートします。下記の擬似コードはターゲットPLDをアップデートするためにJam Byte-Code Playerを複数回にわたってコールする方法を示しています。

```
result = jbi_execute(jbc_file_pointer, jbc_file_size, 0, 0,
"READ_USERCODE", 0, error_line, exit_code);
```

Jam STAPL Byte-Code PlayerはJTAGのUSERCODEをリードし、これをjbi_export()のルーティンを使用して取り出します。そして、その結果に応じたブランチ動作を設定することができます。

図6は、Jam Playerを使用してブランチの設定を行うコードの記述例です。

図6 Jam Playerコードの記述例

```
switch (USERCODE)
{
    case "0001": /*Rev 1 is old - update to new Rev*/
        result = jbi_execute (rev3_file, file_size_3, 0, 0, "PROGRAM", 0,
            error_line, exit_code);
    case "0002": /*Rev 2 is old - update to new Rev*/
        result = jbi_exeecute(rev3_file, file_size_3, 0, 0, "PROGRAM", 0,
            error_line, exit_code);
    case "0003":
        ; /*Do nothing - this is the current Rev*/
    default: /*Issue warning and update to current Rev*/
        Warning - unexpected design revision; /*Program device with newest rev
            anyway*/
        result = jbi_execute(rev3_file, file_size_3, 0, 0, "PROGRAM", 0,
            error_line, exit_code);
}
```

switchのステートメントを使用することによって、どのデバイスのアップデートが必要か、どのデザイン・リビジョンを使用すべきかを判断させることができます。Jam STAPL Byte-Codeソフトウェアのサポートにより、PLDのアップデートは数行のコードを追加するだけで簡単に実現できるようになっています。

まとめ

Jam STAPLの採用によって、ISPとICRを簡単に実行できる方法が実現されます。Jamは、最小のファイル・サイズ、使いやすさ、プラットフォームに対する非依存性など、エンベデッド・システムで要求されるすべての項目を満足しています。アップデートのためのデータをJam STAPL Byte-Codeファイルで供給することによって、フィールドでのアップデートが簡略化されます。Jam Playerの実行は、使用されるリソースの計算のように簡単に実現することができます。最新のアップデート状況および最新の情報については、Jamのwebサイト、<http://www.jamisp.com>で確認してください。

Altera, APEX, ByteBlaster, ByteBlasterMV, EPF10K10, EPF10K10A, EPF10K20, EPF10K30, EPF10K30A, EPF10K30E, EPF10K40, EPF10K50, EPF10K50E, EPF10K50V, EPF10K70, EPF10K100, EPF10K100A, EPF10K100B, EPF10K100E, EPF10K130E, EPF10K130V, EPF10K200E, EPF10K250A, EPF10K250E, EPM7032S, EPM7064AE, EPM7064S, EPM7128A, EPM7128AE, EPM7128S, EPM7160S, EPM7192S, EPM7256A, EPM7256S, EPM9400, EPM9480, EPM9560, EPM9560A, FLEX 10K, FLEX 10KA, Jam, MasterBlaster, MAX, MAX+PLUS, Quartus, MAX 7000A, MAX 7000AE, MAX 7000S, MAX 9000, MAX 9000Aは、Altera Corporationの米国および該当各国におけるtrademarkまたはservice markです。この資料に記載されているその他の製品名などは該当各社のtrademarkです。Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Copyright ©2000 Altera Corporation. All rights reserved.



I.S. EN ISO 9001

ALTERA

日本アルテラ株式会社

〒163-1332

東京都新宿区西新宿6-5-1

新宿アイランドタワー32F 私書箱1594号

TEL. 03-3340-9480 FAX. 03-3340-9487

<http://www.altera.com/japan>

E-mail: japan@altera.com

本社 Altera Corporation

101 Innovation Drive,

San Jose, CA 95134

TEL : (408) 544-7000

<http://www.altera.com>

この資料に記載された内容は予告なく変更されることがあります。最新の情報は、アルテラのwebサイト (<http://www.altera.com>) でご確認ください。この資料はアルテラが発行した英文のアプリケーション・ノートを日本語化したものであり、アルテラが保証する規格、仕様は英文オリジナルのものです。