



この翻訳版ドキュメントのメンテナンスは終了しております。

この文書には、古いコンテンツや商標が含まれている場合があります。

最新情報につきましては、次のリンクから英語版の最新資料をご確認ください。

<https://www.intel.com/content/www/us/en/programmable/documentation/lit-index.html>

Please take note that this document is no longer being maintained. It may contain legacy content and trademarks which may be outdated.

Please refer to English version for latest update at

<https://www.intel.com/content/www/us/en/programmable/documentation/lit-index.html>

このアプリケーション・ノートでは、PC またはエンベデッド・プロセッサを搭載したイン・システム・プログラミング (ISP) に、STAPL (Jam™ Standard Test and Programming Language) でアルテラのプログラミングとコンフィギュレーションに関するサポートを説明します。それは、Jam STAPL Player と実行可能な quartus\_jli コマンド・ラインの ISP のためのガイドラインを提供します。

Jam STAPL を使用して ISP を実装することにより、最終製品の品質、柔軟性、および製品ライフ・サイクルを向上させることができます。プログラムと設定しなければならない PLD の数が関係なく、イン・フィールドのアップグレードが簡素化されます。

Jam STAPL JEDEC 規格 JESD71 は、ISP プログラミング用のソフトウェア・レベルとベンダ依存しない規格を提供することにより、プログラマブル・ロジック・デバイス (PLD) のプログラミングおよびコンフィギュレーションを変革させます。この規格は、JTAG を使用して ISP をサポートしているすべての現在の PLD と互換性があります。それはそのような小さなファイル・サイズ、使いやすさ、プラットフォームの独立性などのエンベデッド・システムに必要なすべての要件を満たします。

このアプリケーション・ノートでは、以下のトピックについて説明します。

- 1-1 ページの「Jam STAPL Players」
- 1-3 ページの「Jam STAPL ファイル」
- 1-9 ページの「Jam STAPL Player の使用」
- 1-9 ページの「quartus\_jli コマンドライン実行コマンドの使用」
- 1-12 ページの「エンベデッド・プロセッサで ISP 用の Jam STAPL の使用」
- 1-15 ページの「ボード・レイアウト」
- 1-16 ページの「エンベデッド Jam STAPL Players」
- 1-26 ページの「Jam を使用したデバイスのアップデート」

## Jam STAPL Players

アルテラがサポートする 2 種類の Jam ファイルに対応するために、2 種類の Jam STAPL があります。

- Jam STAPL Player— テキスト形式の Jam STAPL ファイル (.jam) 用
- Jam STAPL Byte-Code Player— バイトコードの Jam STAPL ファイル (.jbc) 用

Jam STAPL Player は、.jam または .jbc 内の記述的情報を読み出し、これらの情報をターゲット PLD をプログラムするデータに変換します。Player は特定のデバイス・アーキテクチャやベンダをプログラムするのではなく、Jam STAPL 仕様で定義された構文の読み出しを解釈のみを行います。

また、Altera® デバイスには、**.jam** や **.jbc** を使用して Quartus® II ソフトウェア・バージョン 6.0 以降で提供された `quartus_jli` コマンド・ライン実行コマンドでプログラムおよびテストすることもできます。

## Jam STAPL Players および `quartus_jli` の相違点

Jam STAPL Player は **.jam** および **.jbc** ファイルを読み出しおよび実行インタプリタのプログラムです。シングル **.jam** および **.jbc** には、プログラミング、コンフィギュレーション、確認、消去、および PLD のブランク・チェックなどのいくつかの機能が含まれています。Jam STAPL Player は、IEEE 1149.1 インタフェースに基づくすべての命令で使用される IEEE 1149.1 の信号にアクセスできます。Player はまた **.jam** および **.jbc** にユーザーが指定したアクションと手順を処理することができます。



アルテラのウェブサイト上の [Altera Jam STAPL Software](#) のページからアルテラ Jam STAPL Player をダウンロードすることができます。

`quartus_jli` コマンド・ライン実行コマンドは、Jam STAPL Player と同じ機能があります。しかし、それは 2 つの追加機能があります。

- UNIX または DOS プロンプトから Quartus II ソフトウェアのコマンド・ライン・コントロールを提供します。
- Quartus II ソフトウェア・バージョン 6.0 およびそれ以降で利用可能なすべてのプログラミング・ハードウェアをサポートしています。

<Quartus II system directory>\bin ディレクトリに `quartus_jli` コマンド・ライン実行コマンドを見つけることができます。Quartus II ソフトウェアをインストールするときに、このディレクトリは、デフォルトで作成されます。

表 1 には、Jam STAPL Player および `quartus_jli` コマンド・ライン実行コマンドの相違点を示します。

表 1. Jam STAPL Players および `quartus_jli` Command-Line Executable の差

機能	Jam STAPL Players	<code>quartus_jli</code>
サポートされるダウンロード・ケーブル	ByteBlaster™ II、ByteBlasterMV、および ByteBlaster パラレル・ポートのダウンロード・ケーブル	全てのプログラミング・ケーブルは、USB-Blaster™、ByteBlaster II、ByteBlasterMV、ByteBlaster, MasterBlaster™、および EthernetBlaster などの JTAG サーバーでサポートされます。
エンベデッド・プロセッサへのソース・コードのポート	あり	なし
サポートされるプラットフォーム・フォーム	<ul style="list-style-type: none"> <li>■ 16 ビットおよび 32 ビットのエンベデッド・プロセッサ</li> <li>■ 32 ビット Windows</li> <li>■ DOS</li> <li>■ UNIX</li> </ul>	<ul style="list-style-type: none"> <li>■ 32 ビット Windows</li> <li>■ 64 ビット Windows</li> <li>■ DOS</li> <li>■ UNIX</li> </ul>
コマンド・ライン Syntax から手順をイネーブルまたはディセーブルにする	<ul style="list-style-type: none"> <li>■ オプションのプロシージャをイネーブルするには、<code>-d&lt;procedure&gt;=1</code> のオプションを使用してください。</li> <li>■ 推奨のプロシージャをディセーブルするには、<code>-d&lt;procedure&gt;=0</code> のオプションを使用してください。</li> </ul>	<ul style="list-style-type: none"> <li>■ オプションのプロシージャをイネーブルするには、<code>-e&lt;procedure&gt;</code> のオプションを使用してください。</li> <li>■ 推奨のプロシージャをディセーブルするには、<code>-d&lt;procedure&gt;</code> のオプションを使用してください。</li> </ul>

## Jam STAPL ファイル

この項では、アルテラがサポートする Jam STAPL ファイルのタイプとフォーマットについて説明します。

### ASCII テキスト・ファイル

アルテラは、ASCII テキスト・ベースの **.jam** の 2 つのフォーマットをサポートしています。

- JEDEC JESD71 STAPL フォーマット
- Jam バージョン 1.1 フォーマット (pre-JEDEC)



アルテラは、新しいプロジェクトのための JEDEC JESD71 STAPL **.jam** のファイルを使用することを推奨しています。ほとんどのケースでは、テスト環境での **.jam** ファイルを使用してください。

### Byte-Code ファイル

**.jbc** ファイルは、**.jam** ファイルのコンパイルされたバージョンのバイナリ・ファイルです。**.jbc** は、ASCII テキスト・ベースの Jam STAPL コマンドが仮想プロセッサと互換性のあるバイト・コード命令にマップされ、仮想プロセッサ・アーキテクチャにコンパイルされます。

**.jbc** には、次の 2 つのタイプがあります。

- Jam STAPL Byte-Code—JEDEC JESD71 STAPL ファイルのコンパイルされたバージョン
- Jam Byte-Code—Jam バージョン 1.1 フォーマット・ファイルのコンパイルされたバージョン



アルテラは、メモリ使用量を最小化するにはエンベデッド・アプリケーションでの Jam STAPL Byte-Code **.jbc** を使用することを推奨します。

## Jam STAPL ファイルの生成

Quartus II ソフトウェアは、**.jam** および **.jbc** ファイルを生成できます。また、スタンドアロンの Jam STAPL Byte-Code Compiler と **.jbc** に **.jam** をコンパイルすることができます。コンパイラは、**.jam** との機能等価性の **.jbc** を生成します。



アルテラのウェブサイトの [Altera Jam STAPL Software](#) のページから Jam STAPL Byte-Code コンパイラをダウンロードできます。

Quartus II ソフトウェア・ツールは、単一または複数の **.jbc** ファイルから複数のデバイスのプログラミングやコンフィギュレーションをサポートしています。



JTAG チェイン・ファイルを **.jam** に変換する場合、JTAG チェインの中の他のデバイスに選択する Quartus II プログラマ・オプションは、新しい **.jam** にプログラムされません。


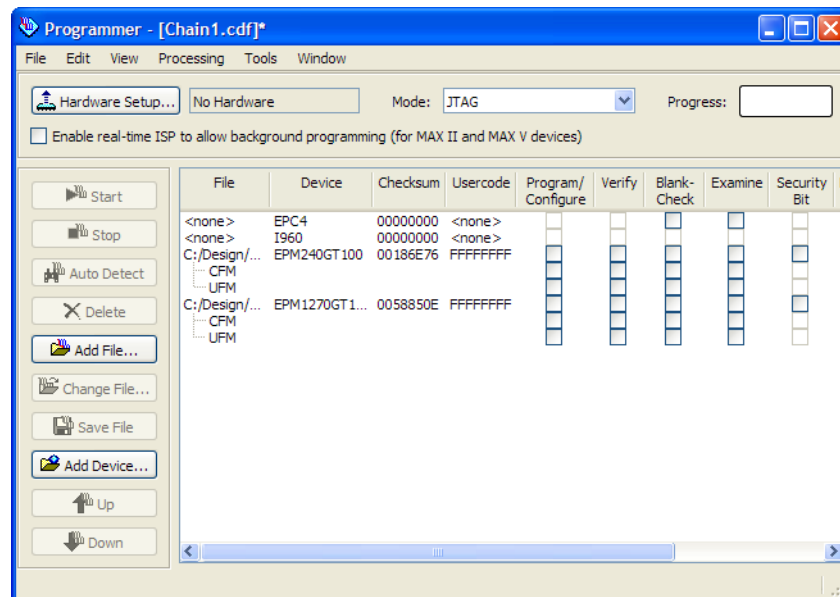
 1 に、Quartus II Programmer ウィンドウのマルチ・デバイス JTAG のチェインとシーケンスの設定を示します。

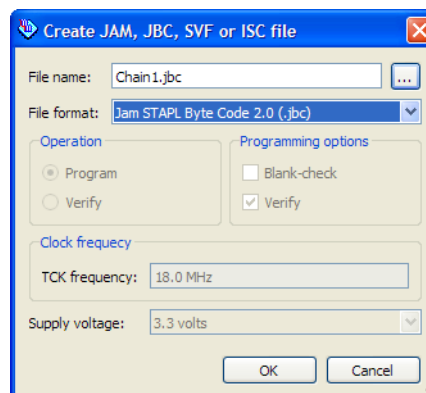
図 1. Quartus II ソフトウェアでのマルチ・デバイス JTAG チェインおよびシーケンスのダイアログ・ボックス



Quartus II ソフトウェアで **.jbc** ファイルを生成する、次の手順を実行します。

1. Tools メニューの **Programmer** をクリックします。
2. **Add File** をクリックし、それぞれのデバイス用のプログラミング・ファイルを選択します。
3. File メニューの **Create/Update** をポイントして、**Create JamSVF**、または **ISC File** をクリックします。
4. **File Format** で、図 22 に示すように、**.jbc** のフォーマットを選択します。
5. **OK** をクリックします。

図 2. Quartus II ソフトウェアでマルチ・デバイス JTAG チェイン用の .jbc の生成



JTAG チェインには、アルテラと非アルテラの JTAG 準拠デバイスを含めることができます。**Programming File Names** のフィールドでプログラミング・ファイルを指定しない場合、JTAG チェイン内のデバイスがバイパスされます。



Quartus II Programmer は、複数のデバイスの **.jam** または JTAG インダイレクト・コンフィギュレーション (**.jic**) ファイルを作成している間、プログラミングのオプションを無視します。しかし、生成した **.jam** で Jam STAPL Player を使用するときにはデバイスに適用するためのプログラミングのオプションを選択することができます。マルチ・デバイス **.jam** の場合、選択したプログラミングのオプションは、JTAG チェイン内のデータ・ファイルを持つ各デバイスに適用されます。

## サポートされている .jam と .jbc アクションおよびプロシージャのリスト

**.jam** または **.jbc** には、以下の 2 つのタイプの文で構成されています。

- **Action**— 完全な動作を実装するための必要なステップのシーケンス。
- **Procedure**— Action 文に含まれるステップの一つ。



Action 文には、1 つ以上の Procedure 文、または Procedure 文を含めないこともできます。Procedure 文が含まれている Action 文の場合は、Procedure 文が関連付けられている動作を完了するために、指定された順序で呼び出されます。Action 文の実行で含めるか除外するかを「推奨」または「オプション」のような Procedure 文の一部を指定することができます。

表 2 に、別のアルテラのデバイス・ファミリで実行可能な各アクションでサポートされている Action 文およびオプションのプロシージャを示します。

表 2. アルテラ・デバイスのサポートされる .jam または .jbc のアクションおよびプロシージャ (その 1)

デバイス	(.jam) / (.jbc) 文	オプションの手順 (オフに設定される)
MAX® 3000A MAX 7000B MAX 7000AE	Program	do_blank_check do_secure do_low_temp_programming do_disable_isp_clamp do_read_usercode
	Blankcheck	do_disable_isp_clamp
	Verify	do_disable_isp_clamp do_read_usercode
	Erase	do_disable_isp_clamp
	Read_usercode	—

表 2. アルテラ・デバイスのサポートされる .jam または .jbc のアクションおよびプロシージャ (その 2)

デバイス	(.jam) / (.jbc) 文	オプションの手順 (オフに設定される)
MAX II MAX V	Program	do_blank_check do_secure do_disable_isp_clamp do_bypass_cfm do_bypass_ufm do_real_time_isp do_read_usercode
	Blankcheck	do_disable_isp_clamp do_bypass_cfm do_bypass_ufm do_real_time_isp
	Verify	do_disable_isp_clamp do_bypass_cfm do_bypass_ufm do_real_time_isp do_read_usercode
	Erase	do_disable_isp_clamp do_bypass_cfm do_bypass_ufm do_real_time_isp
	Read_usercode	—
Stratix® デバイス・ファミリ Arria® デバイス・ファミリ Cyclone® デバイス・ファミリ	Configure	do_read_usercode do_halt_on_chip_cc do_ignore_idcode_errors
	Read_usercode	—
エンハンスド・コンフィギュレーション・デバイス	Program	do_blank_check do_secure do_read_usercode do_init_configuration
	Blankcheck	—
	Verify	do_read_usercode
	Erase	—
	Read_usercode	—
	Init_configuration	—
シリアル・コンフィギュレーション・デバイス	Configure	do_read_usercode do_halt_on_chip_cc do_ignore_idcode_errors
	Program	do_blank_check
	Blankcheck	—
	Verify	—
	Erase	—
	Read_usercode	—

表 3 に、それぞれのアクションとプロシーダの説明を示します。

表 3. .jam および .jbc のアクションとプロシーダ文の定義

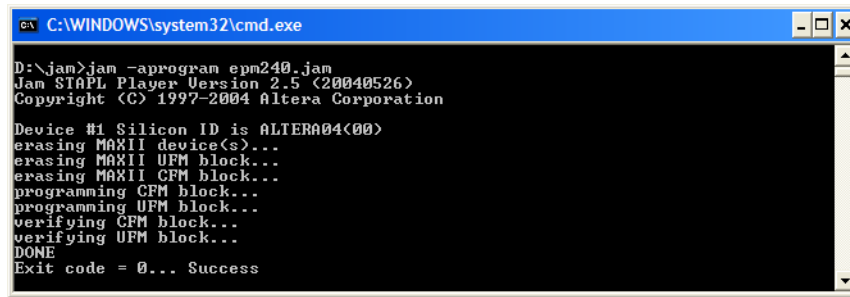
アクション / プロシーダ	説明
<b>動作</b>	
Program	デバイスをプログラムする。
Blankcheck	デバイスの消去ステータスをチェックする。
Verify	.jam または .jbc のプログラミング・データに対してデバイス全体を検証します。
Erase	デバイスのバルク消去を実行する。
Read_usercode	デバイスからの JTAG USERCODE レジスタ情報を返す。
Configure	デバイスをコンフィギュレーションする。
Init_configuration	コンフィギュレーション・デバイスは直ちにプログラミング後に接続されたデバイスを設定することを指定する。
Check_idcode	.jam と .jbc で生成された期待される IDCODE と実際のデバイスの IDCODE を比較する。
<b>プロシーダ</b>	
do_blank_check	イネーブルされると、デバイスはブランク・チェックされる。
do_secure	イネーブルされると、デバイスのセキュリティ・ビットは設定される。
do_read_usercode	イネーブルされると、プレイヤは、デバイスの JTAG USERCODE を読み込み、それを画面に出力する。
do_disable_isp_clamp	イネーブルされると、デバイスの ISP クランプ・モードは無視される。
do_low_temp_programming	イネーブルされると、プロシーダは、MAX3000A、7000B、および 7000AE デバイス用のインダストリアル低温 ISP が可能になる。
do_bypass_cfm	イネーブルされると、プロシーダは、ユーザー・フラッシュ・メモリ (UFM) で指定されたアクションを実行する。
do_bypass_ufm	イネーブルされると、プロシーダは、コンフィギュレーションフラッシュメモリ (CFM) で指定されたアクションを実行する。
do_real_time_isp	イネーブルされると、リアル・タイム ISP 機能は、ISP のアクションが実行されるためにオンになる。
do_init_configuration	イネーブルされると、コンフィギュレーション・デバイスは直ちにプログラミング後に接続されたデバイスを設定する。
do_halt_on_chip_cc	イネーブルされると、プロシーダは、自動コンフィギュレーション・コントローラを停止して、JTAG インタフェースを使用してプログラミングを可能にする。nSTATUS ピンは、デバイスが正常に設定されている後でも Low ままになる。
do_ignore_idcode_errors	イネーブルされると、プロシーダは、IDCODE のエラーが存在する場合でも、デバイスをコンフィギュレーションすることができます。
do_erase_all_cfi	イネーブルされると、プロシーダは、MAX V または MAX II デバイスの平行・フラッシュ・ローダ (PFL) に接続されている共通フラッシュ・メモリ・インタフェース (CFI) フラッシュ・メモリを消去する。

## 終了コード

終了コードは、.jam や .jbc の実行結果を示す整数値です。ゼロの終了コードの値は成功を示します。ゼロ以外の値は失敗を示します。発生した失敗の一般的なタイプを識別します。図 3 には、ゼロの終了コードの値を持つ正常な実行の例を示しています。



図 3. Jam STAPL Player で EPM240 デバイスのプログラミング



```

C:\WINDOWS\system32\cmd.exe
D:\jam>jam -aprogram epm240.jam
Jam STAPL Player Version 2.5 (20040526)
Copyright (C) 1997-2004 Altera Corporation

Device #1 Silicon ID is ALTERA04(00)
erasing MAXII device(s)...
erasing MAXII UFM block...
erasing MAXII CFM block...
programming CFM block...
programming UFM block...
verifying CFM block...
verifying UFM block...
DONE
Exit code = 0... Success

```

Jam STAPL は Jam STAPL Specification (JESD71) で定義されている Jam STAPL Player と quartus\_jli コマンド・ライン実行コマンドの両方が表 4 で終了コードを返すことができます。

表 4. 終了コード

終了コード	説明
0	成功
1	チェインのチェックの失敗
2	IDCODE の読み出しの失敗
3	USERCODE の読み出しの失敗
4	UESCODE の読み出しの失敗
5	ISP への移行の失敗
6	認識されないデバイス ID
7	デバイスのバージョンがサポートされていない
8	消去の失敗
9	ブランク・チェックの失敗
10	プログラミングの失敗
11	検証の失敗
12	読み出しの失敗
13	チェックサム計算の失敗
14	セキュリティ・ビット設定の失敗
15	セキュリティ・ビット照会の失敗
16	ISP 終了の失敗
17	システム・テスト実行の失敗

## Jam STAPL Player の使用

Jam STAPL Player のコマンドとパラメータは、大文字と小文字は区別されません。任意のシーケンスでオプション・フラグを書き込むことができます。

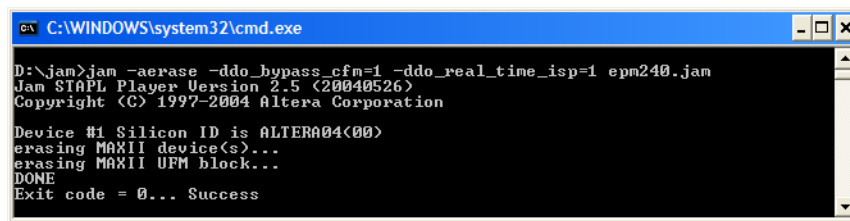
Jam STAPL Player のコマンドでアクションを指定するには、**-a** オプションを使用して、スペースなしの **Action** 文ですぐに続けてください。[8 ページの図 3](#) に示すように、以下のコマンドは、指定された **.jam** を使用してデバイス全体をプログラムします。


```
jam -aprogram <filename>.jam
```


スペースなしの **Procedure** 文の直後に **-d** オプションを使用して各アクションに関連付けられた任意のプロシージャを実行することができます。[図 4](#) に示されるように、以下のコマンドはリアル・タイム ISP を使用して、デバイスの UFM ブロックだけを消去します。

```
jam -aerase -ddo_bypass_cfm=1 -ddo_real_time_isp=1 <filename>.jam
```


図 4. イネーブルされるリアル・タイム ISP 機能でデバイスの UFM ブロックのみを消去



 **.jbc** を実行するには、Jam STAPL Player と同じコマンドとパラメータで Jam STAPL Byte-Code Player は実行コマンド名 (JBI) を使用してください。

 Jam STAPL Player でシリアル・コンフィギュレーション・デバイスをプログラムするには、まず **Serial FlashLoader** イメージで FPGA をコンフィギュレーションする必要があります。以下のコマンドが必要です。

```
jam -aconfigure <filename>.jam ↵
jam -aprogram <filename>.jam
```

 シリアル・コンフィギュレーション・デバイス用の **.jam** を生成する情報について詳しくは、[AN370: Using the Serial FlashLoader with the Quartus II Software](#) を参照してください。

## quartus\_jli コマンドライン実行コマンドの使用

quartus\_jli コマンドライン実行コマンドファイルは、ByteBlaster、ByteBlasterMV、ByteBlaster II、USB-Blaster、MasterBlaster、および Ethernet Blaster などのアルテラのすべてのダウンロード・ケーブルをサポートしています。

quartus\_jli コマンドとパラメータは、大文字と小文字は区別されません。任意のシーケンスでオプション・フラグを書き込むことができます。[表 5](#) に、quartus\_jli コマンド・ラインのオプションをリストします。

表 5. コマンド・ライン実行コマンドの quartus\_jli のためのコマンド・ライン実行オプション

オプション	説明
-a	実行するアクションを指定する。
-c	JTAG サーバのケーブルの数を指定する。
-d	推奨プロシージャをディセーブルする。
-e	オプションの手順をイネーブルする。
-i	特定のオプションやトピックに関する情報を表示する。
-l	ファイルがアクション文で実行されたとき .jam でのヘッダファイルの情報または .jbc ファイルでサポートされるアクションとプロシージャのリストを表示する。
-n	使用可能なハードウェアのリストを表示する。
-f	追加のコマンド・ライン引数を含むファイルを指定する。

以下の例は、quartus\_jli コマンドライン形式の実行コマンド・ファイルを実行するためのコマンド・ライン構文を示しています。

図 5 に示すように、コマンド・プロンプトでマシン上で使用可能なダウンロード・ケーブルのリストを表示するには、このコマンドを入力します。

```
quartus_jli -n ←
```

ダウンロード・ケーブルについて詳しくは、2 ページの表 1 を参照してください。

図 5. 使用可能なダウンロード・ケーブルのディスプレイ (注 1)

```

C:\WINDOWS\system32\cmd.exe
C:\altera\91\quartus\bin>quartus_jli -n
Info: *****
Info: Running Quartus II Jam Tools
Info: Version 9.1 Build 222 10/21/2009 SJ Full Version
Info: Copyright (C) 1991-2009 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Altera Program License
Info: Subscription Agreement, Altera MegaCore Function License
Info: Agreement, or other applicable license agreement, including,
Info: without limitation, that your use is for the sole purpose of
Info: programming logic devices manufactured by Altera and sold by
Info: Altera or its authorized distributors. Please refer to the
Info: applicable agreement for further details.
Info: Processing started: Sun Jan 24 22:54:22 2010
Info: Command: quartus_jli -n
1) USB-Blaster [USB-0]
2) EthernetBlaster on aceb009e [/dev/ebhwipl]
Info: Quartus II Jam Tools was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 68 megabytes
Info: Processing ended: Sun Jan 24 22:54:23 2010
Info: Elapsed time: 00:00:01
Info: Total CPU time (on all processors): 00:00:00
C:\altera\91\quartus\bin>

```

図 5 の注:

(1) 図での番号 1) および 2) はケーブルのインデックス番号です。コマンドでは、< cable index > を関連するケーブルのインデックス番号に置き換えます。

図 6 に示されるように、Action 文を実行する場合に .jam 内のヘッダ・ファイル情報を表示するためには、このコマンド構文を使用してください：

```
quartus_jli -a<action name> <filename>.jam -l
```

図 6. Action 文の実行時の Jam ファイルのヘッダ・ファイル情報

```
C:\WINDOWS\system32\cmd.exe
D:\altera\design1>quartus_jli -aconfigure ep3c10.jam -l -c2
Info: *****
Info: Running Quartus II Jam Tools
Info: Version 9.1 Build 222 10/21/2009 SJ Full Version
Info: Copyright (C) 1991-2009 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Altera Program License
Info: Subscription Agreement, Altera MegaCore Function License
Info: Agreement, or other applicable license agreement, including,
Info: without limitation, that your use is for the sole purpose of
Info: programming logic devices manufactured by Altera and sold by
Info: Altera or its authorized distributors. Please refer to the
Info: applicable agreement for further details.
Info: Processing started: Sun Jan 24 23:14:42 2010
Info: Command: quartus_jli -a configure ep3c10.jam -l -c 2
CRC matched: CRC value = 6870
Export: key = "JAM_STATEMENT_BUFFER_SIZE", value = 3088
NOTE "CREATOR" = "QUARTUS II JAM COMPOSER 9.1"
NOTE "DATE" = "2010/01/24"
NOTE "DEVICE" = "EP3C10"
NOTE "FILE" = "ciii_running_led.sof"
NOTE "TARGET" = "1"
NOTE "IDCODE" = "020F10DD"
NOTE "USERCODE" = "FFFFFFF"
NOTE "CHECKSUM" = "0008BD22"
NOTE "SAVE_DATA" = "DEVICE_DATA"
NOTE "SAVE_DATA_VARIABLES" = "U0, A12, A13, A25, A42, A93, A43, A92, A94, A95, A
105, A109, A111"
NOTE "STAPL_VERSION" = "JESD71"
NOTE "JAM_VERSION" = "2.0"
NOTE "ALG_VERSION" = "51"
Device #1 IDCODE is 020F10DD
configuring SRAM device(s)...
DONE
Exit code = 0... Success
Elapsed time = 00:00:05
Info: Quartus II Jam Tools was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 70 megabytes
Info: Processing ended: Sun Jan 24 23:14:48 2010
Info: Elapsed time: 00:00:06
Info: Total CPU time (on all processors): 00:00:00
```

Action 文を実行するときに使用するプログラミング・ハードウェアまたはケーブルは、このコマンド構文を使用することを指定するにはこのコマンドの構文を使用します。

```
quartus_jli -a<action name> -c<cable index> <filename>.jam
```

Action 文に関連付けられているプロシージャをイネーブルするには、このコマンドの構文を使用します。

```
quartus_jli -a<action name> -e<procedure to enable> -c<cable index>
<filename>.jam
```

Action 文に関連付けられているプロシージャをイネーブルするには、このコマンドの構文を使用します。

```
quartus_jli -a<action name> -d<procedure to disable> -c<cable index>
<filename>.jam
```

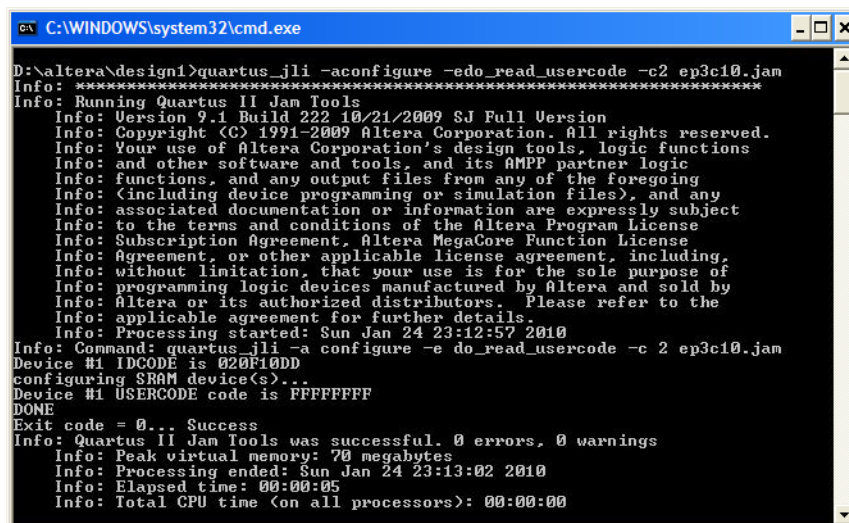
オプションの詳細情報を入手するには、このコマンド構文を使用します。

```
quartus_jli --help=<option/topic>
```

コマンド・プロンプトで、図 7 に示すように、特定の .jam 付きのマシンで 2 番目のダウンロード・ケーブルで FPGA デバイスの JTAG USERCODE を設定とコンフィギュレーションするには、このコマンドを入力します。

```
quartus_jli -aconfigure -edo_read_usercode -c2 <filename>.jam ←
```


図 7. USB-Blaster ケーブルで EP2C70 デバイスの JTAG USERCODE のコンフィギュレーションおよび読み出し



```

C:\WINDOWS\system32\cmd.exe
D:\altera\design1>quartus_jli -aconfigure -edc_read_usercode -c2 ep3c10.jam
Info: ****
Info: Running Quartus II Jam Tools
Info: Version 9.1 Build 222 10/21/2009 SJ Full Version
Info: Copyright (C) 1991-2009 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Altera Program License
Info: Subscription Agreement, Altera MegaCore Function License
Info: Agreement, or other applicable license agreement, including,
Info: without limitation, that your use is for the sole purpose of
Info: programming logic devices manufactured by Altera and sold by
Info: Altera or its authorized distributors. Please refer to the
Info: applicable agreement for further details.
Info: Processing started: Sun Jan 24 23:12:57 2010
Info: Command: quartus_jli -a configure -e do_read_usercode -c 2 ep3c10.jam
Device #1 IDCODE is 020F10DD
Configuring SRAM device(s)...
Device #1 USERCODE code is FFFFFFFF
DONE
Exit code = 0... Success
Info: Quartus II Jam Tools was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 70 megabytes
Info: Processing ended: Sun Jan 24 23:13:02 2010
Info: Elapsed time: 00:00:05
Info: Total CPU time (on all processors): 00:00:00

```

 quartus\_jli コマンド・ライン実行コマンドでシリアル・コンフィギュレーション・デバイスをプログラムするには、以下のコマンドが必要です。

```

quartus_jli -aconfigure <filename>.jam ←
quartus_jli -aprogram <filename>.jam

```

## エンベデッド・プロセッサで ISP 用の Jam STAPL の使用

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の両方で構成されます。エンベデッド・システムをデザインする場合、最初に PCB をレイアウトしてください。そして、ボードの機能性を管理するファームウェアを開発してください。

### エンベデッド・プロセッサへの JTAG チェインの接続

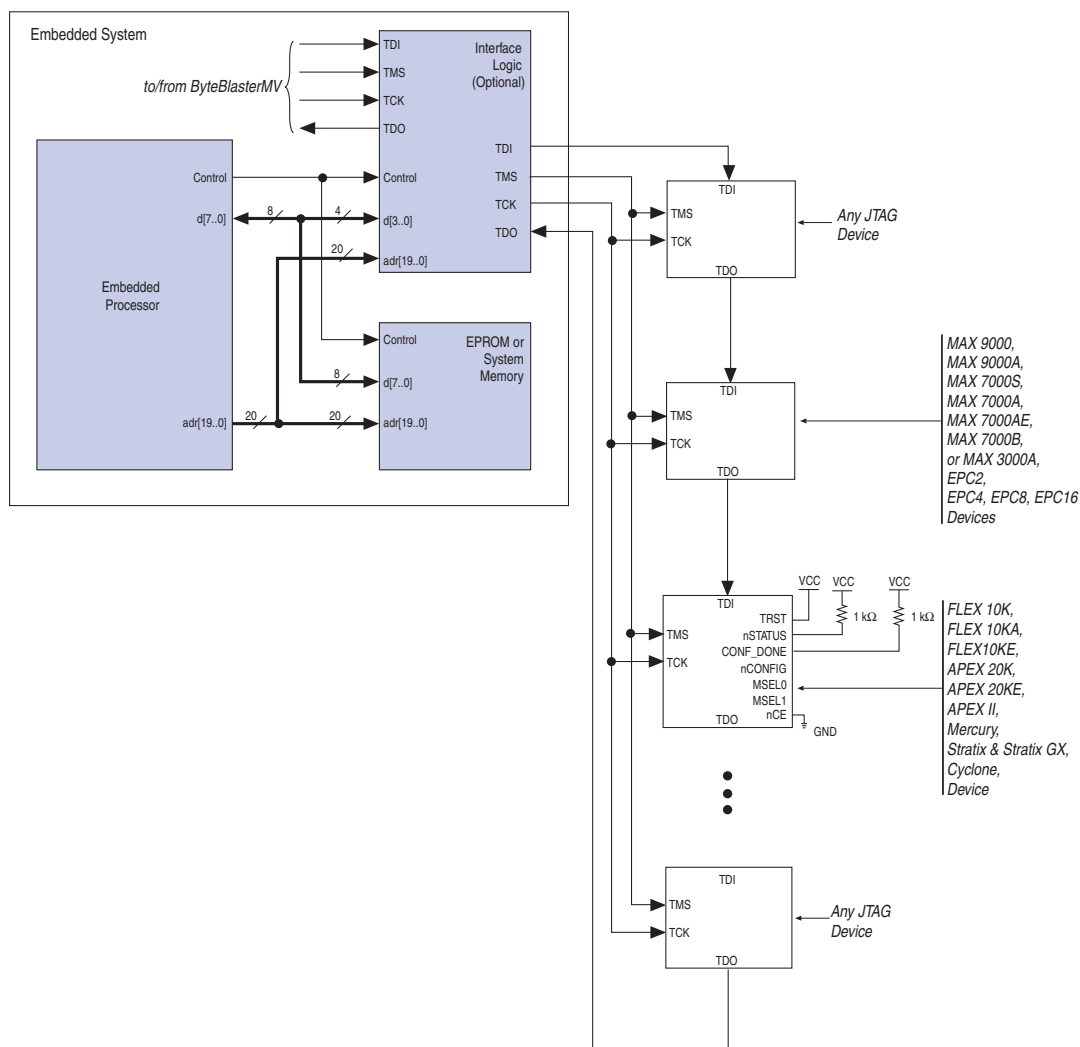
JTAG チェインをエンベデッド・プロセッサに接続するには、2 つの方法があります。

- エンベデッド・プロセッサを JTAG チェインに直接接続する
- インタフェース PLD を介して既存のバスに JTAG チェインを接続する

最初の方法は最も簡単です。この方法では、プロセッサ・ピンの 4 つは、JTAG インタフェースに専念しています。この方法は、ボードのスペースを節約しますが、使用可能なエンベデッド・プロセッサのピン数を低減することができます。

第二の方法では、[図 8](#) に示すように、JTAG チェインが既存のバス上のアドレスで表され、プロセッサはこのアドレスで読み出しおよび書き込み操作が実行されます。

図 8. エンベデッド・システムへの JTAG チェインの接続



両方の JTAG 接続方法で、MasterBlaster または ByteBlasterMV ヘッダ接続用のスペースを含める必要があります。ヘッダを使用すると設計者は PLD のコンテンツを素早く検証したり修正できるため、プロトタイプ作成に役立ちます。生産時にはヘッダを取り除いてコストを削減できます。

### インタフェース PLD デザイン例

図 9 には、インタフェース PLD の回路図の例を示しています。このデザイン例は参考用です。この例を使用する場合は、以下のことを確認する必要があります。

- TMS, TCK, および TDI が同期出力
- マルチプレクサ・ロジックを搭載し、MasterBlaster または ByteBlaster II ダウンロード・ケーブルによるボード・アクセスが可能

データ・バス data[3..0] を除く入力はすべてオプションであり、インタフェース PLD がエンベデッド・データ・バス上でアドレスとして動作する方法を示すためにだけに記載されています。

図 9. インタフェース・ロジック・デザイン例

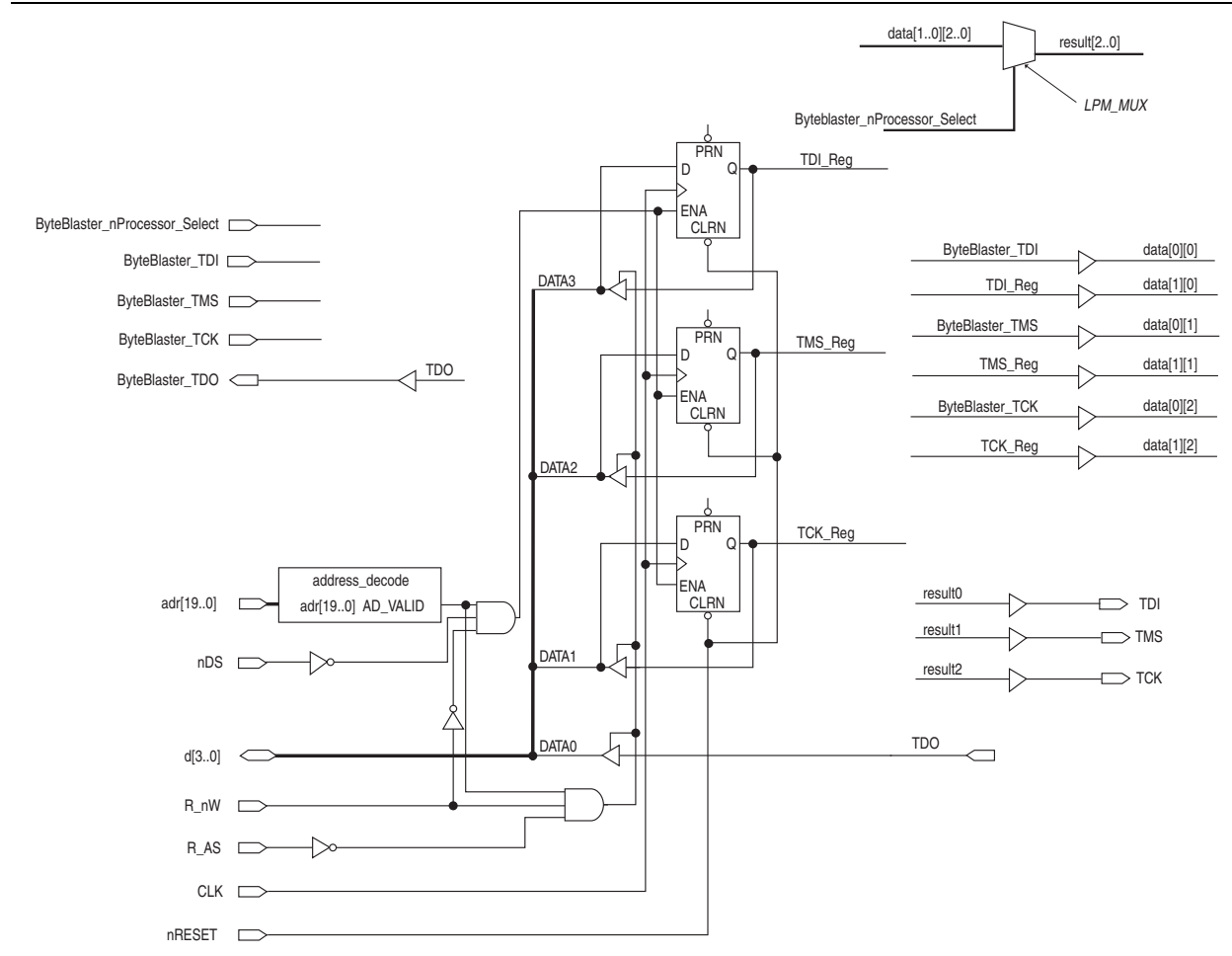


図 9 には、エンベデッド・プロセッサは、JTAG チェインのアドレスをアサートします。プロセッサがチェーンにアクセスするときには、インタフェース PLD に通知する R\_nW と R\_AS 信号を設定することができます。

書き込みを行うには、システム・クロック (CLK) でクロックされる 3 つの D レジスタを介して、データ・パス data[3..0] を PLD の JTAG 出力に接続します。このクロックは、プロセッサが使用するクロックと同じにすることができます。

同様に、読み出しを行うには、トライ・ステート・バッファをイネーブルし、TDO 信号をプロセッサに送り返す必要があります。

また、このデザインでは、TDI、TMS、および TCK 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提供します。このオプション機能を利用すると、インタフェース PLD 内のレジスタの有効なステートをソフトウェアでチェックでき、開発段階で役立ちます。さらに、マルチプレクサ・ロジックが搭載されているため、MasterBlaster または ByteBlasterMV ダウンロード・ケーブルでデバイス・チェーンをプログラムできます。さらに、デザイン例は、MasterBlaster または ByteBlasterMV ダウンロード・ケーブルはデバイスのチェーンをプログラムすることを許可するマルチプレクサ・ロジックが含まれています。



## ボード・レイアウト

IEEE Std. 1149.1 JTAG チェインでデバイスをプログラムまたはコンフィギュレーションするボードをレイアウトする場合、これらの重要な要素に従ってください。

- TCK 信号配線パターンをクロック・ツリーとして扱うこと
- TCK にプルダウン抵抗を使用すること
- JTAG 信号配線パターンを可能な限り短くすること
- 出力が規定のロジック・レベルになるように外部抵抗を追加すること

### TCK 信号配線の保護およびインテグリティ

TCK は、デバイスの JTAG チェイン全体に対するクロックです。これらのデバイスは TCK 信号でエッジ・トリガされるため、TCK を高周波ノイズから保護し、良好なシグナル・インテグリティを持つことが不可欠です。信号が該当するデバイス・ファミリー・データシートに記載された立ち上がり時間 ( $t_r$ ) および立ち下がり時間 ( $t_f$ ) パラメータに適合することを確認してください。



また、オーバーシュート、アンダーシュート、またはリングングを防止するために、信号に終端が必要な場合もあります。このステップは、この信号がソフトウェアで生成され、プロセッサの汎用 I/O ピンで発生するため、見落とされることがよくあります。

### TCK 信号のプルダウン抵抗

パワーアップ時に JTAG TAP (Test Access Port) を既知のステートに維持するために TCK 信号はプルダウン抵抗を介して Low に保持することが必要です。プルダウン抵抗がないとデバイスが JTAG BST ステートでパワーアップし、それによってボード上で競合が発生する可能性があります。一般的な抵抗値は 1 k $\Omega$  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と、ノイズやドライブ強度に関連した問題の解消に役立ちます。TCK ピンと TMS ピンには特別に注意が必要です。TCK と TMS は JTAG チェインのすべてのデバイスに接続されるため、これらのトレースは、TDI や TDO よりも負荷が大きくなります。JTAG チェインの長さや負荷によっては、プロセッサとの間で信号がインテグリティを維持しながら伝播できるように、いくつかの追加バッファリングが必要になることがあります。

### 外部抵抗

プログラミングまたはコンフィギュレーション中に出力を定義済みロジック・レベルにするには、出力ピンに外部抵抗を追加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中にはトライ・ステートになります。また、MAX 7000、FLEX 10K<sup>®</sup>、APEX<sup>™</sup> 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、ピンはウィーク内部抵抗、たとえば、50 k $\Omega$  でプルアップされています。しかし、すべてのアルテラ・デバイスは ISP かイン・サーキット・リコンフィギュレーションにウィーク・プルアップ抵抗を持つとは限りません。ウィーク・プルアップ抵抗のあるデバイスについて詳しくは、適切なデバイス・ファミリーのデータ・シートを参照してください。





センシティブな入力ピンをドライブする出力は、1 k $\Omega$  の順に外部抵抗を使用して適切なレベルに接続することを推奨します。

さらに、前述のボード・レイアウトの要素、特にシグナル・インテグリティのそれぞれを分析する必要がある場合があります。場合によっては、ディスクリート・バッファを使用する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。



詳しくは、[AN 100: In-System Programmability Guidelines](#) を参照してください。

## エンベデッド Jam STAPL Players

エンベデッド Jam STAPL Player は、標準 JEDEC ファイル・フォーマットに従い、レガシー Jam version 1.1 構文で後方互換性のある **.jam** を読み出すことができます。同様に、Jam STAPL Byte-Code Player は Jam STAPL と Jam version 1.1 **.jam** からコンパイルされた **.jbc** を再生することができます。

以下の項では、Jam STAPL Byte-Code Player の移植について説明します。

### Jam STAPL Byte-Code Player

Jam STAPL Byte-Code Player は 16 および 32 ビットのプロセッサ用の C プログラミング言語でコード化されています。プレーヤー・ソース・コードの特定のサブセットはまた、いくつかの 8 ビット・プロセッサをサポートします。



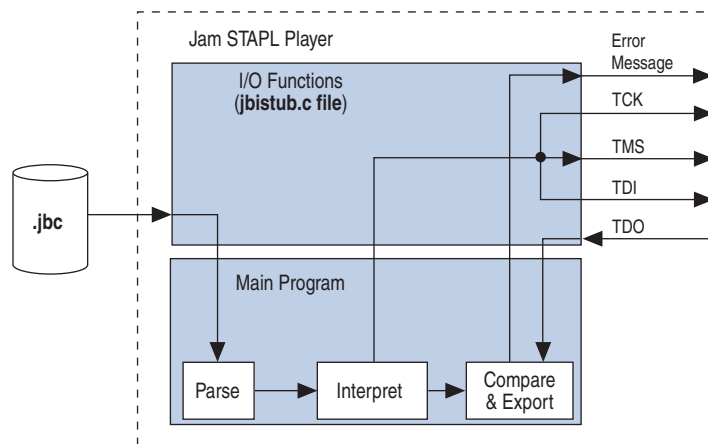
8 ビット・プロセッサのためのアルテラのサポートについて詳しくは、[AN 111: Embedded Programming Using the 8051 and Jam Byte-Code](#) を参照してください。

16 および 32 ビットの Jam STAPL Byte-Code Player のソース・コードは、2 つのカテゴリに分類されます。

- **jbistub.c**—I/O ファンクションを処理し、特定のハードウェアに適用されるプラットフォーム固有のコード
- 他のすべての C ファイル —Player の内部ファンクションを実行する汎用コード

図 10 に、ファンクションによるソース・コード・ファイルの構成を示します。プラットフォーム固有のコードが **jbistub.c** ファイル内に管理されているため、Jam STAPL Byte-Code Player を特定のプロセッサに移植するプロセスが簡略化されます。

図 10. Jam STAPL Byte-Code Player ソース・コードの構造



## Jam STAPL Byte-Code Player の移植

**jbistub.c** ファイルのデフォルト・コンフィギュレーションには、DOS、32 ビット Windows、および UNIX 用のコードが含まれているため、ソース・コードを簡単にコンパイルして、これらの定義済みオペレーティング・システムの機能性評価とデバッグを行うことができます。

エンベデッド環境の場合、このコードは単一のプリプロセッサ `#define` プリプロセッサ文を使用して除去できます。さらに、コードを移植するには、**jbistub.c** ファイルのコードの特定部分にわずかな変更が必要です。

Jam STAPL Byte-Code Player を移植するには、表 6 にリストされている **jbistub.c** の機能をカスタマイズしてください。

表 6. カスタマイズを必要とするファンクション

ファンクション	説明
<code>jbi_jtag_io ()</code>	このファンクションは、IEEE 1149.1 JTAG 信号 (TDI、TMS、TCK、および TDO) へのインタフェースを提供します。
<code>jbi_export ()</code>	UES (User Electronic Signature) などの情報を呼び出し側のプログラムに渡します。
<code>jbi_delay ()</code>	実行中に必要なプログラミング・パルスまたは遅延を実装します。
<code>jbi_vector_map ()</code>	非 IEEE 1149.1 JTAG 信号に対して信号からピンへのマップを処理します。
<code>jbi_vector_io ()</code>	VECTOR MAP で定義されるとおり、非 IEEE 1149.1 JTAG 信号をアサートします。

必要なコードをすべてカスタマイズしたことを確認するために、以下の 4 つのステップを実行します。

- 1-18 ページの「ステップ 1: プリプロセッサのステートメントを設定して、無関係なコードを除外する」
- 1-18 ページの「ステップ 2: JTAG 信号をハードウェア・ピンにマップする」
- 1-19 ページの「ステップ 3: `jbi_export()` からのテキスト・メッセージを処理する」
- 1-19 ページの「ステップ 4: 遅延校正をカスタマイズする」

## ステップ 1: プリプロセッサのステートメントを設定して、無関係なコードを除外する

デフォルトの PORT パラメータを EMBEDDED に変更して、すべての DOS、Windows、UNIX のソース・コード、およびインクルードされたライブラリを除外します。

**jbiport.h** のトップにこのコードを追加します。

DOS、Windows、および UNIX のソースコードおよび付属のライブラリをなくす EMBEDDED にデフォルトの PORT パラメータを変更する。**jbiport.h** の先頭にこのコードを追加します。

```
#define PORT EMBEDDED
```

## ステップ 2: JTAG 信号をハードウェア・ピンにマップする

**jbistub.c** での **jbi\_jtag\_io()** ファンクションには、バイナリ・プログラミング・データを送受信するコードが含まれています。デフォルトでは、ソース・コードは PC のパラレル・ポートに書き込みます。エンベデッド・プロセッサのピンに 4 つのすべての JTAG 信号を再マップする必要があります。

**jbi\_jtag\_io()** 信号は、図 11 に示す PC パラレル・ポート・レジスタに JTAG ピンをマップします。

図 11. デフォルトの PC パラレル・ポート信号マップ (注 1)

7	6	5	4	3	2	1	0	I/O Port
0	TDI	0	0	0	0	TMS	TCK	OUTPUT DATA - Base Address
TDO	X	X	X	X	---	---	---	INPUT DATA - Base Address + 1

図 11 の注:

(1) PC パラレル・ポート・ハードウェアは、最上位ビットの TDO を反転させます。

以下の **jbi\_jtag\_io()** ソース・コードのサンプルでは、マップ処理が示されています。

```
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);
}

```

PC パラレル・ポートは TDO の実際の値を反転させます。このため、上記のコードで `jbi_jtag_io()` ファンクションは、次のラインで元のデータを取得するために再び値を反転します。

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

ターゲット・プロセッサが TDO を反転させない場合、以下のコードを使用します。

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

信号を正しいアドレスにマップするには、左シフト (<<) または右シフト (>>) 演算子を使用します。例えば、TMS と TDI がそれぞれポート 2 とポート 3 の場合、コードは以下ようになります。

ターゲット・プロセッサが TDO を反転させない場合、以下のコードを使用します。

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

TCK および TDO にも同じ手法を適用します。

`read_byteblaster` および `write_byteblaster` 信号はそれぞれ、**conio.h** ライブラリの `inp()` および `outp()` ファンクションを使用して、ポートの読み出しと書き込みを行います。これらのファンクションが利用できない場合は、同等のファンクションで代替する必要があります。

### ステップ 3: `jbi_export()` からのテキスト・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()` 信号を使用して、オペレーティング・システムまたは Player を呼び出すソフトウェアに情報（デバイスの UES または USERCODE など）を渡します。ファンクションはテキスト（文字列形式）と数値（10 進整数形式）を渡します。



詳しくは、[IEEE 1149.1 JTAG Boundary-Scan Testing in Altera Devices](#) を参照してください。

`stdout` が利用できるデバイスが存在しない場合、情報はファイルまたはストレージ・デバイスにリダイレクトされるか、あるいは Player を呼び出すプログラムに変数として渡されます。

### ステップ 4: 遅延校正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 ミリ秒間に実行するループ数を決定します。プログラミングとコンフィギュレーションで正確な遅延が使用されるため、この校正は重要です。デフォルトでは、この数値は 1 ミリ秒あたり 1,000 ループとしてハードコード化され、以下のように表されます。

```
one_ms_delay = 1000
```

このパラメータが既知の場合、それに従って変更します。それ以外の場合は、それは 1 つのループの実行に必要なクロック・サイクル数をカウントする Windows および DOS プラットフォーム用に含まれるコードに同様のコードを使用してください。このコードは、正確な遅延の結果を得るために、複数回のテストを通じてサンプリングされ、平均化されます。この手法の利点は、ホスト・プロセッサの速度に基づいて較正を変更できることです。

Jam STAPL Byte-Code Player が移植され動作した後、ターゲット・デバイスでの JTAG ポートのタイミングとスピードを検証してください。MAX V、MAX II、および MAX デバイスのタイミング・パラメータは、関連するデバイス・ファミリのデータシートで提供する JTAG タイミング・パラメータと値を準拠する必要があります。

Jam STAPL Byte-Code Player がタイミング仕様で動作しない場合は、適切な遅延でコードを最適化する必要があります。タイミング違反は、プロセッサが非常に高性能で、18 MHz を超える高速レートで TCK を生成できる場合にのみ発生します。



予測不可能な Jam STAPL Byte-Code Player の動作を防止するには、`jbistub.c` 以外にソース・コード・ファイルをデフォルト・ステートに維持すると強く推奨されています。

## Jam STAPL Byte-Code Player のメモリ使用量

Jam STAPL Byte-Code Player は予測可能な方法でメモリを使用します。この項では、ROM および RAM 使用量を見積もる方法について説明します。

### ROM 使用量の見積もり

Jam Player および `.jbc` ファイルの格納に必要な ROM の最大容量を見積もるには、式 1 を使用してください。

式 1.

---


$$\text{ROM Size} = \text{jbc Size} + \text{Jam STAPL Byte-Code Player Size}$$


---

`.jbc` サイズは次の 2 つのカテゴリに分類されます。

- プログラミング・データの格納に必要なメモリ容量
- プログラミング・アルゴリズムに必要なスペース

`.jbc` のサイズを見積もるために、式 2 を使用します。

式 2. (注 1),(2),(3),(4)

---


$$\text{jbc Size} = \text{Alg} + \sum_{k=1}^N \text{Data}$$


---

式 2 の注:

- (1) *Alg* = アルゴリズムで使用するスペース
  - (2) *Data* = 圧縮されたプログラミング・データで使用するスペース
  - (3) *k* = ターゲットとなるデバイスを表すインデックス
  - (4) *N* = チェイン内のターゲット・デバイスの数
-

式 2 によって **.jbc** サイズが見積もられ、この値はデバイスの利用率によって  $\pm 10\%$  変動することがあります。デバイス利用率が低い場合、ファイル・サイズを最小化する圧縮アルゴリズムは、繰り返しデータを検出する可能性が高いため、**.jbc** サイズが小さくなる傾向があります。

式 2 は、アルゴリズム・サイズが 1 つのデバイス・ファミリに対しては一定となるが、プログラミング・データ・サイズは、ターゲットとするデバイスが増えるほど増大することも示しています。デバイス・ファミリでは、**.jbc** サイズ（データ・コンポーネントによる）の増加は線形となります。

表 7 に、1 つのアルテラ・デバイス・ファミリをターゲットとするアルゴリズム・ファイル・サイズ定数をリストします。

**表 7. 1 つのアルテラ・デバイス・ファミリをターゲットとするアルゴリズム・ファイル・サイズ定数**

デバイス	標準的な .jbc アルゴリズム・サイズ (KB)
Stratix デバイス・ファミリ	15
Cyclone デバイス・ファミリ	15
Arria デバイス・ファミリ	15
Mercury	15
EPC16	24
EPC8	24
EPC4	24
EPC2	19
MAX 7000AE	21
MAX 7000	21
MAX 3000A	21
MAX 9000	21
MAX 7000S	25
MAX 7000A	25
MAX 7000B	17
MAX II	24.3
MAX V	24.3

表 8 に、Jam Language をサポートするアルテラ・デバイス・ファミリの可能な組み合わせのためのアルゴリズム・ファイル・サイズ定数を示します。

表 8. 複数のアルテラ・デバイス・ファミリをターゲットするアルゴリズム・ファイル・サイズ定数

デバイス	標準的な .jbc アルゴリズム・サイズ (KB)
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

表 8 の注:

(1) FLEX または APEX デバイスのコンフィギュレーション時に、および MAX 9000 と MAX7000 デバイスをのプログラミング時に、FLEX または APEX のアルゴリズムは無視できるメモリを追加します。

表 9 に、ISP 用の Jam Language をサポートするアルテラ・デバイスのデータ・サイズ定数を示します。

表 9. Jam Language をサポートするアルテラ・デバイスのデータ定数 (ISP 用) (注 2),(3),(4),(5),(6) (その 1)

デバイス	標準的な Jam STAPL Byte-Code データ・サイズ (KB)	
	圧縮	非圧縮 (1)
EP1S10	105	448
EP1S20	188	745
EP1S25	241	992
EP1S30	320	1310
EP1S40	369	1561
EP1S60	520	2207
EP1S80	716	2996
EP1C3	32	82
EP1C6	57	150
EP1C12	100	294
EP1C20	162	449
EPC4(2),(5)	242	370
EPC8(2),(5)	242	370
EPC8(3),(5)	547	822
EPC16(2),(5)	242	370
EPC16(4),(5)	827	1344
EP1SGX25	243	992
EP1SGX40	397	1561
EP1M120	30	167
EP1M350	76	553
EP20K30E	14	48
EP20K60E	22	85

表 9. Jam Language をサポートするアルテラ・デバイスのデータ定数 (ISP 用) (注 2),(3),(4),(5),(6) (その 2)

デバイス	標準的な Jam STAPL Byte-Code データ・サイズ (KB)	
	圧縮	非圧縮 (1)
EP20K100E	32	130
EP20K160E	56	194
EP20K200E	53	250
EP20K300E	78	347
EP20K400E	111	493
EP20K600E	170	713
EP20K1000E	254	1124
EP20K1500E	321	1509
EP2A15	107	549
EP2A25	163	788
EP2A40	257	1209
EP2A70	444	2181
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



表 9. Jam Language をサポートするアルテラ・デバイスのデータ定数（ISP 用）（  
注 2),(3),(4),(5),(6)（その 3）

デバイス	標準的な Jam STAPL Byte-Code データ・サイズ (KB)	
	圧縮	非圧縮 (1)
EPF10K100, EPF10K100A, EPF10K100B	95	149
EPF10K100E	102	167
EPF10K130E	140	230
EPF10K130V	136	199
EPF10K200E	205	345
EPF10K250A	235	413
EP20K100	128	244
EP20K200	249	475
EP20K400	619	1,180
EPC2	136	212
EPM240	12.4(6)	12.4(6)
EPM570	11.4	19.6
EPM1270	16.9	31.9
EPM2210	24.7	49.3

表 9 の注:

- (1) 非圧縮プログラミング・データを使用した .jbc ルの生成方法について詳しくは、[www.altera.com/mysupport](http://www.altera.com/mysupport) を参照してください。
- (2) プログラミング・ファイルは一つの EP1S10 デバイスをターゲットとしています。
- (3) プログラミング・ファイルは一つの EP1S25 デバイスをターゲットとしています。
- (4) プログラミング・ファイルは一つの EP1S40 デバイスをターゲットとしています。
- (5) エンハンスド・コンフィギュレーション・デバイス (EPC) データ・サイズは、圧縮されたプログラマ・オブジェクト・ファイル (.pof) を使用してください。
- (6) .jbc コンパイラで圧縮アレイには最小値の 64 キロビットの (KB) があります。64 Kb—8 キロバイト (KB) より小さいプログラミング・データ・アレイは、圧縮されません。EPM240 のプログラミング・データ・アレイが限界以下であります。すなわち、.jbc は常に圧縮されていないことを意味します。メモリ・バッファは復元に必要です。小さなエンベデッド・システムの場合はアレイを圧縮解除するよりも、圧縮されていない小さなアレイを直接使用するほうが効率的であるからです。

Jam STAPL Byte-Code Player のバイナリ・サイズを見積もるには、表 10 の情報を使用してください。

表 10. Jam STAPL Byte-Code Player のバイナリ・サイズ

構築	説明	サイズ (KB)
16 ビット	MasterBlaster または ByteBlasterMV ダウンロード・ケーブルを使用する Pentium/486	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 Pentium/486	85

## ダイナミック・メモリ使用量の見積もり

Jam STAPL Byte-Code Player が必要とする DRAM の最大容量を見積もるには、式 3 を使用してください。

式 3.

$$\text{RAM Size} = \text{.jbc Size} + \sum_{k=1}^N \text{Data(Uncompressed Data Size)}_k$$

.jbc サイズは、シングル・デバイスまたはマルチ・デバイスの式で求められます (1-20 ページの「ROM 使用量の見積もり」を参照)。

Jam STAPL Byte-Code Player が使用する RAM の量とは、.jbc の合計サイズと各ターゲット・デバイスに必要なデータの合計です。.jbc ファイルが圧縮データを使用して生成される場合、データを解凍して一時的に格納するために、Player によって一部の RAM が使用されます。22 ページの表 9 に、非圧縮データ・サイズを示します。非圧縮 .jbc を使用する場合は、使用してください。

式 4.

$$\text{RAM Size} = \text{.jbc Size}$$



スタックおよび蓄積のためのメモリ要件は、Jam STAPL Byte-Code Player が使用する全メモリ容量に関しては無視できます。スタックの最大の深さは、jbmain.c 内の JBI\_STACK\_SIZE パラメータによって設定されます。

## メモリ見積もり例

以下の例では、16 ビット Motorola 68000 プロセッサを使用して、IEEE Std. 1149.1 JTAG チェイン内の EPM7128AE および EPM7064AE デバイスを圧縮するデータを使用する .jbc でプログラムします。メモリ使用量を算出するには、必要な ROM の容量を求めてから RAM の使用量を見積もります。

Jam Byte-Code Player が必要とする DRAM の容量を見積もるには、以下のステップを実行します。

1. .jbc サイズを決定する — 式 5 に示されるように、マルチ・デバイスの式を使用して、.jbc サイズを見積もります。.jbc ファイルは圧縮データを使用するため、22 ページの表 9 に示す圧縮データのファイル・サイズ情報を使用して Data サイズを算出します。

式 5. (注 1),(2)

$$\text{.jbc Size} = \text{Alg} + \sum_{k=1}^N \text{Data}$$

式 5 の注:

- (1) Alg = 21 KB であり、Data は EPM7064AE および EPM7128AE データ・サイズの合計 (8 KB + 4 KB = 12 KB)
- (2) .jbc ファイル・サイズは 33 KB です。

2. Jam STAPL Byte-Code Player サイズを見積もる —Motorola 68000 プロセッサは 16 ビット・プロセッサであるため、この例では 62 KB の Jam STAPL Byte-Code Player サイズを使用します。必要な ROM 容量を決定するために式 6 を使用してください。

式 6.

$$\begin{aligned}\text{ROM Size} &= \text{.jbc Size} + \text{Jam STAPL Byte-Code Player Size} \\ \text{ROM Size} &= 95 \text{ KB}\end{aligned}$$

3. 式 7 を使用して、RAM の使用率を見積もります。

式 7. (注 1),(2),(3)

$$\text{RAM Size} = 33 \text{ KB} + \sum_{k=1}^N \text{Data(Uncompressed Data Size)}_k$$

式 7 の注:

- (1) .jbc は圧縮データを使用するため、使用される全 RAM 容量を算出するには、各デバイスの非圧縮データのサイズを合計する必要があります。
- (2) EPM7064AE と EPM7128AE のための非圧縮データ・サイズの定数は、それぞれ 8 KB と 12 KB です。
- (3) DRAM の全使用量を以下のとおり計算します。RAM サイズ = 33 KB + (8 KB + 12 KB) = 53 KB。

一般に、.jam ファイルは ROM より RAM を多く使用します。RAM の方が安価であり、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されるため、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、メモリ・コストよりもアップグレードの容易さの方が重要です。

## Jam を使用したデバイスのアップデート

フィールドでデバイスを更新すると、多くの場合は program の Action 文によって新しい .jbc ファイルをダウンロードし、Jam STAPL Byte-Code Player を実行します。

Jam STAPL Byte-Code Player のためのメイン・エントリ・ポイントは、jbi\_execute() です。このルーチンは特定の情報を Player に渡します。Player が終了すると、終了コードが返され、併せてランタイム・エラーがあればそれにかんする詳細なエラー情報が返されます。インタフェースは、jbimain.c でのルーチンのプロトタイプ定義で定義されます。

```

JBI_RETURN_TYPE jbi_execute
(
    PROGRAM_PTR program
    long program_size,
    char *workspace,
    long workspace_size,
    char *action,
    char **init_list,
    int reset_jtag
    long *error_address,

```

```

    int *exit_code,
    int *format_version
)

```

**jbistub.c** にある `main()` 内のコードで決定されます。ほとんどの場合、このコードはエンベデッド環境には適用できません。したがって、このコードを削除し、エンベデッド環境用に `jbi_execute()` ルーチンを設定することができます。

`jbi_execute` ファンクションを呼び出す前に JEDEC Standard JESD71 仕様に指定されているように、**.jbc** で有効なアクションに対応する適切な引数で `init_list` を構築します。`init_list` は文字列へのポインタのヌル終端されたアレイです。

初期化リストには、Jam STAPL Byte-Code Player に実行する機能のタイプを指示します。例えば、プログラムと検証、そしてこのリストには `jbi_execute()` に渡されます。初期化リストを適切な方法で渡さなければなりません。無効な初期化リストが渡された場合、あるいは初期化リストが渡さない場合は、Jam STAPL Byte-Code Player は **.jbc** ファイルの構文チェックを実行します。エラーがない場合、プログラム機能を実行せずに成功した終了コードを返します。

プログラムの実行と動作の検証をするために、**例 1** のコードを使用して Jam STAPL Byte-Code Player を指示する `init_list` を設定します。

#### 例 1.

```
char CONSTANT_AREA init_list[] [] = "DO_PROGRAM=1", "DO_VERIFY=1";
```

Jam STAPL Byte-Code Player はデフォルト・コードは、異なる `init_list` を設定し、コマンド・プロンプトから Jam STAPL Byte-Code Player はに指示を与えるために使用されます。

適切なパラメータと等しくなるように設定しているときに**例 1**でのコードでは、`init_list` 変数を宣言します。`CONSTANT_AREA` 識別子は、プログラム・メモリに `init_list` アレイを格納するようにコンパイラに指示します。

Jam STAPL Byte-Code Player は、タスクを完了した後、Player は、`JB_I_RETURN_TYPE` または整数のステータス・コードを返します。「0」の戻り値は成功したアクションを示します。Jam STAPL は仕様で定義されているように `jbi_execute()` ルーチンは **8 ページの表 4** での終了コードのいずれかを返すことができます。**表 11** に、`jbi_execute()` ルーチンにパラメータをリストします。

**表 11. jbi\_execute () ルーチンでのパラメータ (注 1) (その 1)**

パラメータ	ステータス	説明
program	ルーチン	<b>.jbc</b> へのポインタ。ほとんどのエンベデッド・システムの場合、このパラメータを設定することは <code>jbi_execute()</code> を呼び出す前にポインタにアドレスを割り当てるのと同じくらい簡単です。
program_size	ルーチン	<b>.jbc</b> が占有するメモリの量 (バイト単位)。
workspace	オプション	その必要な機能を実行するために Jam STAPL Byte-Code Player が使用できる動的なメモリへのポインタ。- このメモリは、 <code>jbi_execute()</code> を呼び出す前に割り当てる必要があります。このメモリは、 <code>jbi_execute()</code> を呼び出す前に割り当てる必要があります。 ダイナミック・メモリの最大使用量が問題でない場合、このパラメータは <b>null</b> に設定します。それによって、Player は必要なメモリを動的に割り当てて、特定のアクションを実行することができます。
workspace_size	オプション	Workspace が指すメモリ容量を (バイト単位) を表すスケラ。

表 11. jbi\_execute () ルーチンでのパラメータ ( 注 1 ) ( その 2 )

パラメータ	ステータス	説明
action	必須	文字列 (Player に指示するテキスト) へのポインタ。Action の例に PROGRAM や VERIFY があります。ほとんどの場合、このパラメータは文字列 PROGRAM に設定されます。Player では大文字と小文字が区別されないため、テキストは大文字と小文字のどちらでも構いません。 Jam STAPL Byte-Code Player は、Jam STAPL Specification で定義されるすべてのアクションをサポートしています (24 ページの表 10 を参照)。 文字列は null で終了する必要があることに注意してください。
init_list	オプション	文字列へのポインタの配列。Jam バージョン 1.1 ファイルを適用する、またはサブアクション・オプションをオーバーライドするときにするときに、このパラメータを使用します。 アルテラは init_list で STAPL ベース .jbc を使用することをお勧めします。あなたが STAPL ベース .jbc を使用する場合、init_list は文字列へのポインタの NULL で終了するアレイを指定する必要があります。
error_address	—	長い整数へのポインタ。実行中にエラーが発生した場合、Player はエラーが発生した .jbc の行を記録します。
exit_code	—	長い整数へのポインタ。 .jbc の構文または構造に関するエラーが発生した場合、コードを返します。このようなエラーが発生した場合は、サポートしているベンダに問い合わせ、終了コードが発生した状況を詳しく説明する必要があります。

表 11 の注:

(1) Jam STAPL Byte-Code Player を実行するために、必須パラメータを渡す必要があります。

## Jam STAPL Byte-Code Player の実行

Jam STAPL Byte-Code Player の呼び出しは、その他のサブルーチンの呼び出しと類似しています。この場合、サブルーチンはアクションとファイル名が指定され、その関数を実行します。

イン・フィールド・アップグレードは、現在のデバイス・デザインが最新かどうかによって実行できる場合があります。多くの場合、JTAG USER CODE は、PLD デザインのリビジョンを示す電子スタンプとして使用されます。USERCODE が古い値に設定されると、エンベデッド・ファームウェアはデバイスをアップデートします。

以下の擬似コードは、Jam Byte-Code Player を複数回呼び出して、ターゲット PLD をアップデート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?;
```

ここで Jam STAPL Byte-Code Player は JTAG USERCODE を読み出し、jbi\_export() ルーチンを使用してこれをエクスポートします。次に、コードはその結果に基づいて分岐できます。

例 2 に示されるように、Switch 文を使用すると、どのデバイスにアップデートが必要で、どのデザイン・リビジョンを使用するかを決定できます。

**例 2.**


---

```

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_excecute?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?;
}

```

---

JamSTAPL Byte-Code ソフトウェア・サポートによって、PLD アップデートはコードに数行追加するのと同じくらい簡単なものになります。

## 改訂履歴

表 12 に、本資料の改訂履歴を示します。

表 12. 改訂履歴

日付	バージョン	変更内容
2010 年 12 月	5.0	<ul style="list-style-type: none"> <li>■ (1-2 ページの「Jam STAPL Players および quartus_jli の相違点」、1-3 ページの「ASCII テキスト・ファイル」、1-3 ページの「Byte-Code ファイル」、1-3 ページの「Jam STAPL ファイルの生成」、1-9 ページの「quartus_jli コマンドライン実行コマンドの使用」、および 1-16 ページの「エンベデッド Jam STAPL Players」) の章のタイトルを変更。</li> <li>■ すべてのスクリーンショットを更新。</li> <li>■ 表および図のタイトルを更新</li> <li>■ MAX V デバイスについての情報を追加。</li> <li>■ 図 9 でのテキストのエラーを修正</li> <li>■ 「ステップ 2: JTAG 信号をハードウェア・ピンにマップする」および「Jam を使用したデバイスのアップデート」のコードを更新。</li> <li>■ 式を更新。本資料全体で番号付け式の変化を更新。</li> <li>■ 1-24 ページの「表 9 の注 :」でのマイナー・エラーを修正。</li> <li>■ 「結論」の章を削除。</li> <li>■ ドキュメント全体を通したテキストのマイナー・チェンジ。</li> </ul>
2010 年 7 月	4.0	<ul style="list-style-type: none"> <li>■ すべてのスクリーンショットを更新。すべてのスクリーンショットを更新。</li> </ul>
2009 年 7 月	3.0	<ul style="list-style-type: none"> <li>■ すべてのスクリーンショットを更新。内容的には変更なし。</li> </ul>
2008 年 8 月	2.1	<ul style="list-style-type: none"> <li>■ 「Jam によるデバイスをアップデート」の章を追加。</li> <li>■ 表 3 を更新。</li> <li>■ 表 1 を更新。</li> </ul>
2007 年 11 月	2.0	<ul style="list-style-type: none"> <li>■ 「Introduction」の章を更新。</li> <li>■ 「Jam STAPL Players」、「Jam STAPL Files」、「Using the Jam STAPL for ISP via an Embedded Processor」、「Embedded Jam Players」、および「Updating Devices Using Jam」の章を追加。</li> </ul>
2006 年 12 月	1.1	<ul style="list-style-type: none"> <li>■ 章のタイトルを更新。</li> <li>■ 「Introduction」の章を更新。</li> <li>■ 「Differences Between Jam STAPL Player and quartus_jli Command-Line Executable」の章を更新。</li> <li>■ 図 6、図 7、および図 8 を更新。</li> </ul>