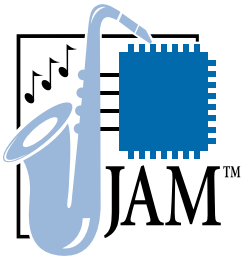
エンベデッド・プロセッサによるISPとICRにJam言語を使用する方法

1998年 10 月 ver.3.0 Application Note 88

イントロダクション



MAX® 9000、MAX 9000A、MAX 7000A、MAX 7000AE、MAX 7000S、およびFLEX® 10Kファミリの各デバイスでサポートされているエンベデッド・プロセッサによるイン・システム・プログラミングまたはイン・サーキット・リコンフィギュレーションは、デザインの試作を容易にし、製造工程の簡略化やフィールドでの効率的なアップグレードを実現します。イン・システム・プログラマビリティ（ISP）またはイン・サーキット・リコンフィギュラビリティ（ICR）をサポートしているデバイスに対して、ROM、FLASHカード、モデムまたは他のデータ・リンクから新しいデータをダウンロードすることによって、フィールドでのアップグレードを簡単に行うことができます。フィールドで設計変更データをエンベデッド・プロセッサを介してシステムにダウンロードさせることもできます。エンベデッド・プロセッサにプログラミング・データをメモリ・ソースからデバイスに転送させることによって、デザインのアップグレードが簡単に行えるようになります。

Jam™はISPのために開発された新しい標準ファイル・フォーマットのプログラミングおよびテスト用言語であり、IEEE Std. 1149.1のJTAG（Joint Test Action Group）インタフェースを使用してプログラムされるすべてのISP対応デバイスをサポートしています。Jam Playerは、Jamのウェッブ・サイト、http://www.jamisp.comからダウンロードすることができます。Jamのソース・コードは、インタプリタ・プログラムによってダイレクトに実行され、これらのソース・コードをバイナリの実行コードにコンパイルする必要はありません。（詳細については４ページの「Jam言語を使用したエンベデッド・プログラミング」を参照してください。）Jamのソース・コード、またはJamバイト・コード・ファイル（.jbc）には、１個または複数のデバイスをアップグレードするために必要なプログラミング・アルゴリズムとデータが含まれています。

このアプリケーション・ノートはJam言語を使用して、エンベデッド・プロセッサによるISPの利点を実現する方法について解説したものです。このアプリケーション・ノートでは以下の項目について解説します。

- エンベデッド・システムの構成と要求される機能
- Jam言語を使用したエンベデッド・プログラミング

☞ このアプリケーション・ノートの使用にあたっては、「*Jam Pro-gramming & Test Language Specification*」を参照してください。

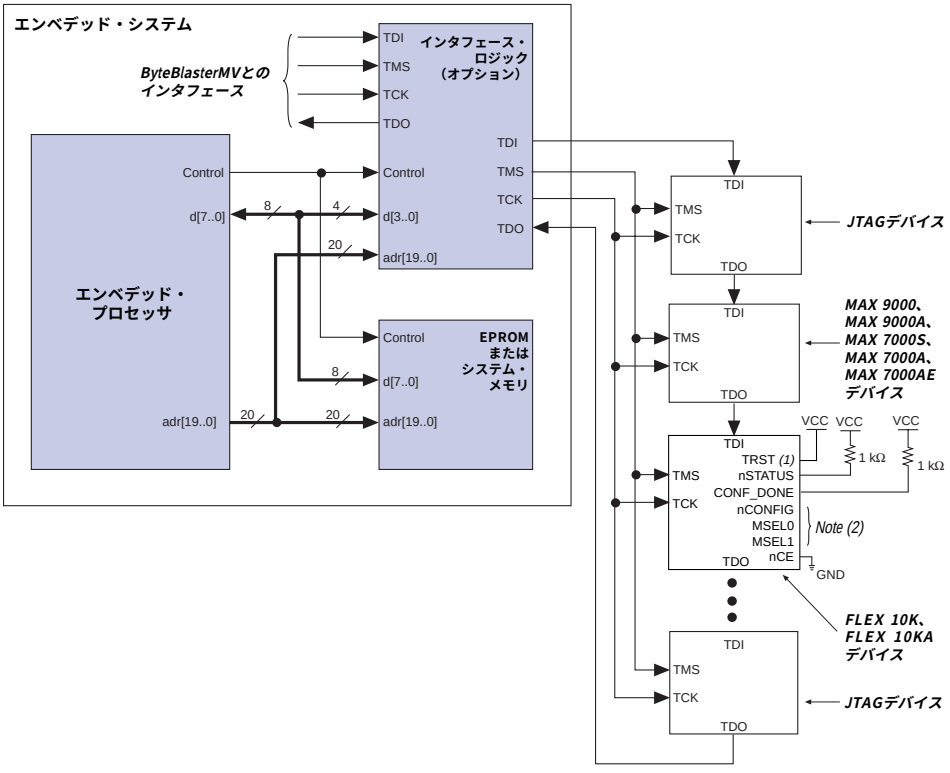
エンベデット・システムの構成と要求される機能

ISPまたはICRが提供する利点を活用するためには、エンベデッド・システムがターゲット・デバイスを小容量のシステム・メモリを使用してプログラムできるようになっている必要があり、複数のベンダの製品が含まれているデバイス・セットの変更にも対応できるような柔軟性を持っていなければなりません。通常、エンベデッド・システムは、エンベデッド・プロセッサ、

EPROMまたはシステム・メモリ、および複数のインタフェース・ロジックによって構成されます。プログラミング・データは、システム・メモリ（EPROMまたはFLASHメモリ）にストアされます。

ISPまたはICRを実行するときは、エンベデッド・プロセッサがプログラミング・データまたはコンフィギュレーション・データをシステム・メモリからISP対応デバイス（１個または複数）に転送します。図１はエンベデッド・システムの構成をブロック図で示したものです。

図１　エンベデッド・システムのブロック・ダイヤグラム



注：

(1)　144ピン薄型クワッド・フラット・パック（TQFP）パッケージのFLEX 10KAデバイスはTRSTピンを持っていないため、その場合はTRSTの接続を無視することができます。

(2)　nCONFIG、MSEL0、MSEL1の各ピンはFLEXデバイスの各コンフィギュレーション・モードに対応した接続となっている必要があります。JTAGのコンフィギュレーションだけが使用される場合は、nCONFIGピンをVCCに、MSEL0とMSEL1をグランドに接続します。

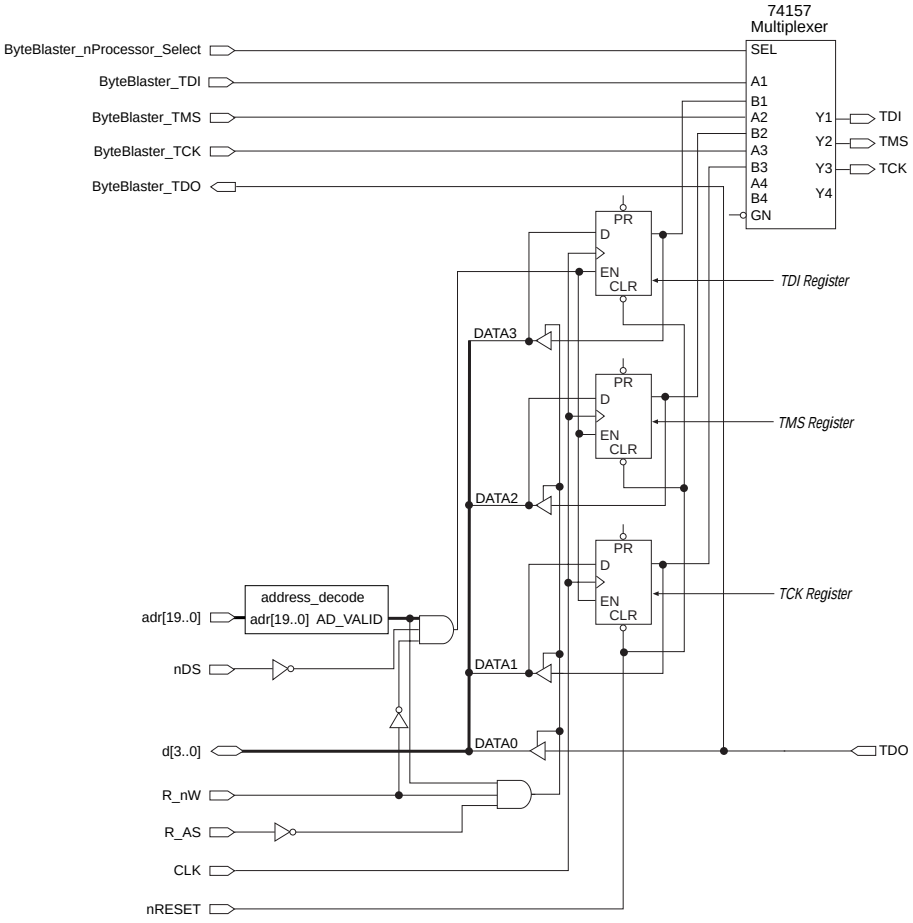
エンベデッド・プロセッサはEPROMまたはシステム・メモリ、およびオプションのインタフェース・ロジックを構成しているプログラマブル・ロジック・デバイス（PLD）と接続されます。JTAGチェインはエンベデッド・プロセッサの４本のデータ・ピンとダイレクトに接続することができますが、イン

タフェース・ロジックを追加することで、JTAGチェインを既存のバスのひとつのアドレス・ロケーションとして扱うことが可能になり、プロセッサの４本のポートを節約することができます。また、ボード上にByteBlasterMV™またはByteBlaster™用に10ピンのヘッダを実装しておくことによって、MAX+PLUS® II ソフトウェアとByteBlasterMVまたはByteBlasterパラレル・ポート・ダウンロード・ケーブルを使用したJTAGチェインのアクセスと検証が可能になります。

ByteBlasterMVパラレル・ポート・ダウンロード・ケーブルの詳細については、「*ByteBlasterMV Parallel Port Download Cable*」のデータシートを参照してください。

図２はエンベデッド・システムのインタフェース・ロジックを示したものです。

図２　インタフェース・ロジック



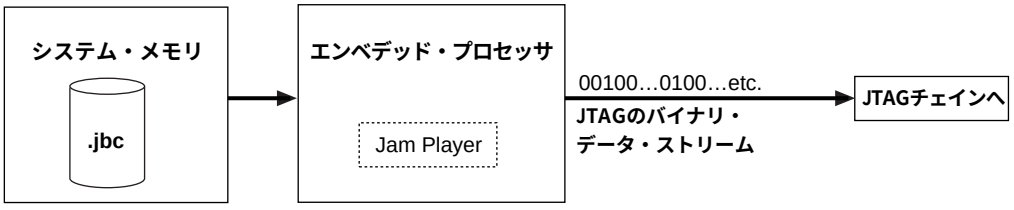
インタフェース・ロジックは、エンベデッド・プロセッサから適切なアドレスとコントロール信号を受信したときにアクティブとなります。このとき、レジスタはTDI、TCK、TMS信号のタイミング同期をとり、74157のマルチプレクサを経由して出力ピンをドライブします。このマルチプレクサは、ByteBlasterMVまたはByteBlasterによるJTAGチェインのアクセスと検証を可能にしています。

Jam言語を使用したエンベデッド・プログラミング

Jam言語はJamファイル（.jam）とJam Playerの２つの部分で実現されています。JamファイルはMAX+PLUS II 開発ツールによって生成され、システム・メモリにストアされます。Jamファイルには１個または複数のISP対応デバイスをプログラムするために必要なすべての情報が含まれています。これに対して、Jam Playerはエンベデッド・プロセッサ上で動作し、Jamファイル内の情報を解読して、デバイスのプログラミングに必要なバイナリのデータ・ストリームを生成します。アップグレード情報は必要に応じてJamファイルのみで供給されるため、Jam Playerはその都度変更されることなく、あらゆるベンダのデバイスをプログラムできるようになっている必要があります。

図３はJam言語を使用してイン・システム・プログラミングがどのように実現されるかをブロック図で示したものです。

図３　Jam PlayerとJBCファイルを使用したISPのブロック図



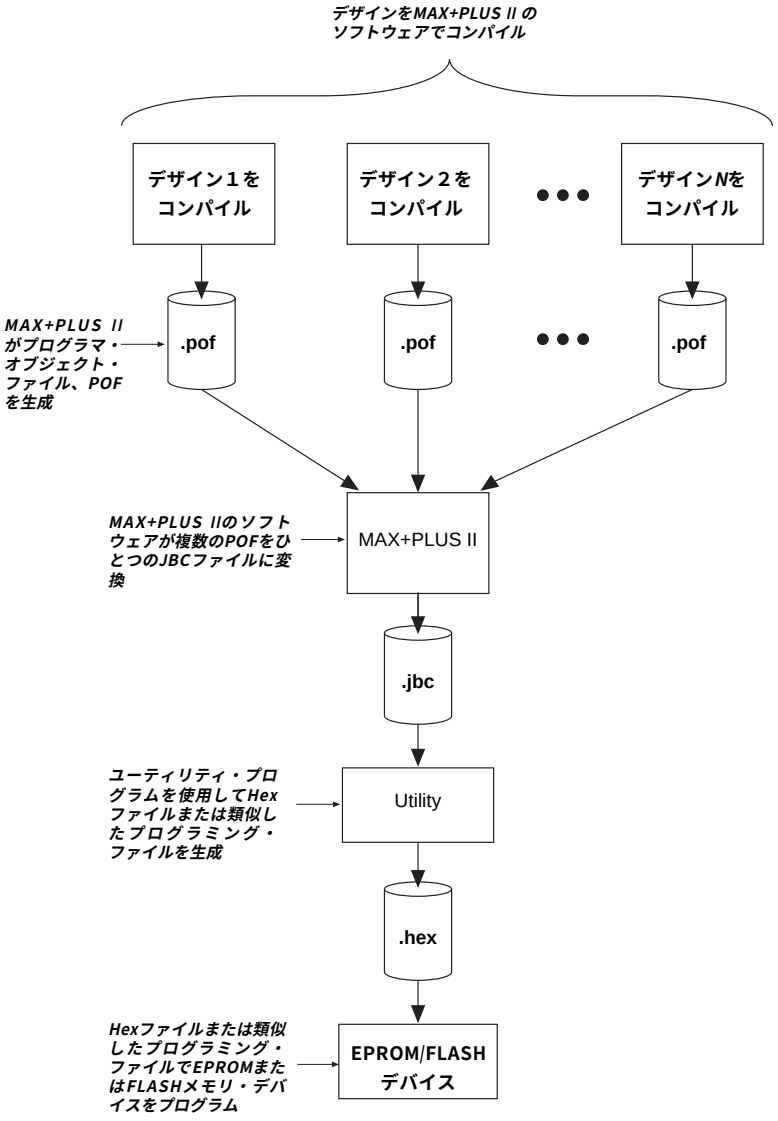
Jamファイル

Jamファイルはコンパクトなファイル・サイズとなっており、あらゆるISP対応デバイスをIEEE Std. 1149.1のJTAGポートを通じてプログラムするときに必要なデータとアルゴリズムを含んだものとなっています。アルテラは、Jamバイト・コード・ファイル（.jbc）とASCII Jamファイル（.jam）の２種類のフォーマットのJamファイルをサポートしています。JBCファイルはバイナリのファイルであるのに対して、ASCII Jamファイルはテキストのみのファイルとなっています。JBCファイルはASCII Jamファイルよりもサイズが小さく、より高速のプログラム時間を実現するため、アルテラは新しいデザインにはJBCファイルを採用することを推奨します。アルテラは、JBCファイルとの互換性を持っているASCII Jamファイルも引き続きサポートする予定です。

図４は、MAX+PLUS IIのソフトウェアを使用して、イン・システム・プログラミング用のJBCファイルを生成する方法を示したものです。

他社の開発ソフトウェアを使用した場合でも、同じようなフローで実行することができます。

図４　MAX+PLUS IIのソフトウェアでJBCファイルを生成する方法



イニシャライズの規定

MAX+PLUS IIのソフトウェアは、Jamの規定にしたがったイニシャライズが実行されるJBCファイルを生成します。このセクションでは、Jam言語でサポートされている特殊な規定について解説します。

DO_PROGRAM

DO_PROGRAMは、デバイスをプログラムするかどうかを決定する変数です。DO_PROGRAMが1にセットされると、Jam Playerが１個または複数のISP対応デバイスに対して、シリコンIDのチェック、デバイス全体の消去（バルク・イレース）、そしてプログラムを実行します。同一ファミリの複数のデバイスをプログラムする場合は、MAX+PLUS II のソフトウェアがコンカレント・プログラミング・アルゴリズムを使用します（同じファミリのすべてのデバイスにプログラミング・データが同時にシフト・インされる）。

プログラミングが開始されると、ターゲット・デバイスのI/Oピンはトライ・ステートとなり、最後のデバイスのプログラミングが完了すると、トライ・ステートのモードが解除されます。これらの変化はJTAGチェイン内のすべてのターゲット・デバイスで同時に発生します。

DO_VERIFY

DO_VERIFYはJBCファイルのデータとデバイスに書き込まれた内容が一致しているかを検証（ベリファイ）するための変数です。DO_VERIFYが1にセットされると、ターゲット・デバイスに対するベリファイ動作が実行されます。ベリファイの結果はJamプログラムの終了コードで表示されます。

DO_ERASE

DO_ERASEは、JBCファイルでバルク・イレースを実行させるための変数です（デバイスが完全に消去される）。このプロセスを実行することで、各ビットに対する適切なプログロミングと、複数のプログラミング・サイクルを実行した場合でもデバイスに対する適切な信頼性が確保されます。

DO_BLANKCHECK

DO_BLANKCHECKはプログラミング前にデバイス全体が適切に消去されていることを確認するための変数です。DO_BLANKCHECKの変数の設定でデバイス内のすべてのデータが消去されていることが検証されます。

DO_SECURE

DO_SECUREは、セキュリティ・ビットをオンに設定できるデバイスに対して、POFでセキュリティ・ビットをプログラムするときのイニシャライズ変数です。POFにセキュリティ・ビットをオンにする設定が含まれていない場合は、この変数を1にイニシャライズしても、デバイスが影響を受けることはありません。

DO_SECURE_ALL

DO_SECURE_ALLは、対応するPOFでセキュリリティ・ビットをオンにするかどうかの設定とは関係なく、セキュリティ・ビットがプログラムされるようにするためのイニシャライズ変数です。

DO_READ_UES

UES（User Electronic Signature）は、デザインのリビジョンを記録しておくための便利なコードとなっています。 DO_READ_UESの変数で、Jam Playerに対してターゲット・デバイスからUESコードを読み出し、それをレポートするように指示することができます。DO_READ_UESが1にイニシャライズされていると（-dDO_READ_UES=1）、Jam PlayerはUESのコードを読み出します。Jam Playerは、printfコマンドを使用して、jbi_export( )のルーチンを通じてUESコードの値を返してきます。UESをサポートしている２個のアルテラ・デバイスが接続されているJTAGチェインの出力は下記のようになります。

```
jbi -p378 -dDO_READ_UES=1 Altera.jbc
```

FFFAとFFFBがヘキサデシマルで表された16ビットのUESコードの値です。

DO_CONFIGURE

DO_CONFIGUREは、SRAMベースのデバイスをコンフィギュレーションするかどうかを決定する変数です。この変数を 1 に設定し、FLEX 10Kデバイス用に生成されたJBCファイルが適用されると、コンフィギュレーションが実行されます。

イニシャライズの規定に関する詳細については、「*Jam Programming & Test Language Specification*」を参照してください。

JBCファイルの構造

エンベデッド・システムにおいて、JBCファイルはアップデート可能なシステム・メモリにストアされます。JBCファイルはコンパクトなメモリ・サイズにストアできる構造となっており、Variable Declaration/ Initialization Section（変数宣言／イニシャライズ・セクション）およびAlgorithm Section（アルゴリズム・セクション）を持っています。図５はJBCファイルの構造を示したものです。

図５　JBCファイルの構造

Variable Declaration/Initialization Section

- 圧縮されたプログラム／ベリファイ・データ
- イニシャライズ変数

Algorithm Section

- シリコンIDのチェック
- ブランク・チェック（オプション）
- バルク・イレースとプログラム（オプション）
- UESのリード（オプション）
- ベリファイ（オプション）
- 終了コード

Variable Declaration/Initialization SectionにはJBCファイルで使用される変数に関する宣言が含まれます。また、このセクションでは、変数を特定の値にイニシャライズすることが可能です。BOOLEANのアレイ部では、変数がAdvanced Compression Algorithm（ACA）のフォーマットを使用して、圧縮されたデータ・アレイとしてイニシャライズされます。他のタイプの変数もこのセクションで宣言し、イニシャライズすることができます。なお、変数のイニシャライズはオプションです。

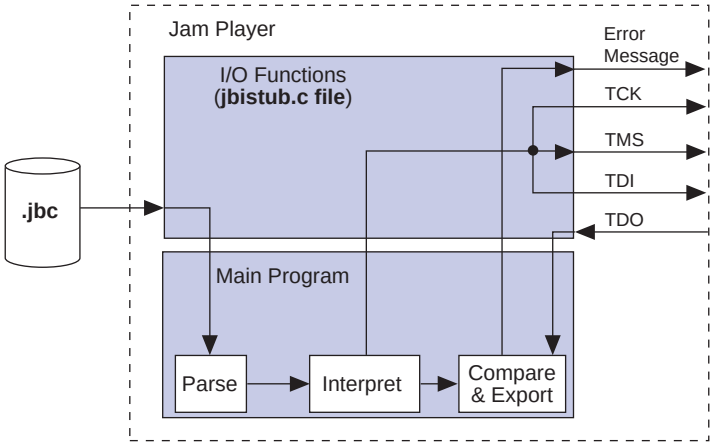
ACAに関する詳細については、「*Jam Programming & Test Language Specification*」を参照してください。

Algorithm Sectionはすべてテキストで記述され、ここに実際のプログラミング・コマンドや他の必要な機能（ベリファイ結果に応じたブランチ、複数のJTAGデータ・レジスタ・スキャンへのループ、ターゲットとなっているJTAGチェインをトラックするための管理機能など）を実行するプログラミング・コードが含まれます。Algorithm Sectionには、１個または複数のターゲット・デバイス上で実行される上位の変数（Blank-Check、Verifyなど）が含まれます。

Jam Player

Jam PlayerはJBCファイルの解析（parse）、Jam命令の解読、JTAGチェインに対するデータのリード／ライト動作を行うためのCプログラムです。このソース・コードは、16ビットまたは32ビットのプロセッサでコンパイルされるコードで記述されています。 Jam Playerは各変数をイニシャライゼーション・リストにしたがって処理、実行します（詳細については10ページの「Jam Playerの実行」を参照）。各アプリケーションには、それぞれ個別の要求が存在するため、Jam Playerのソース・コードは簡単に変更できるようになっている必要があります。図６はJam Playerのソース・コード構造を示したものです。

図６　Jam Playerのソース・コード構造



メイン・プログラム部分は、変更なしでJam Playerのすべての基本的な機能を実行できるようになっています。ここで、各アプリケーションに対応した変更が必要になるのは、jbistub.cのファイルに含まれるI/O機能の部分のみです。この部分には、I/Oピンに対するアドレス、遅延ルーチン、オペレーティング・システムに特化した機能、およびファイルの入出力に関するルーチンが規定されます。

Jam Playerはシステム・メモリ内に常駐し、JBCファイル内のコマンドの解読と、デバイス・プログラミングに使用されるバイナリ・データ・ストリームの生成を行います。この構造の実現により、すべてのアップグレードがJBCファイル内で行えるようになり、Jam Playerをあらゆるシステム・アーキテクチャに対応させることができます。

Jam Playerには、ASCII JamファイルとJBCファイルに対応した２種類が用意されています。表１は、これらのファイル・オプションとJam Playerの互換性をまとめたものです。

表1　Jam Player の互換性		
Jam ファイルのタイプ	Jam Player のタイプ	
	ASCII（jam.exe）	バイト・コード（jbi.exe）
ASCII	√	
Byte Code		√

JBCファイルはバイト・コードのJam Playerに対して適用され、ASCII Jamファイルは ASCII Jam Playerに対して適用されます。このようにJam Playerの機能を区別することによって、Jam Playerのバイナリ・ベースのファイル・サイズが縮小されています。アルテラは、ASCIIのJamファイルの使用が必要となる既存のプロジェクトを除き、どのような場合でもJBCファイルを採用することを推奨します。

Jam Playerのカスタマイズ

Jam Playerは、アプリケーションやプラットフォームの要求に応じて簡単にカスタマイズできる構造となっています。すべてのファイルのI/Oやポートの構成は、jbistub.cのファイルを書き換えることによって、簡単に変更することができます。jbistub.cファイルは、入力機能としてJBCファイルからのデータ検索、TDOから出力されたシフト・アウト・データの読み込みを行うことができます。また、出力機能として、jbistub.cファイルは３本のJTAGピン（TDI、TMS、TCK）に対する処理されたJTAGデータの送出、コールされたプログラムに対するフォーマット化されたエラーやインフォメーション・メッセージの返送、コール・プログラムに対するステータス・インフォメーションの返送などを行うことができます。

Jam Playerのソース・コードに含まれているreadmeファイルには、Jam Playerのポーティングに関する詳細な情報が提供されています。詳しいことは、日本アルテラの応用技術部へお問い合わせください。

Jam Playerの実行

Jam Playerは、実行されるISPの機能を指定できる高い柔軟性を実現しています。コマンド・ラインからのオプションがJam Playerに受け渡され、実行できるようになっています。Jam Playerの実行は、下記のフォーマットで行います。

```
Jam[-h][-v][-p<パラレル・ポートを示す16進のアドレス>]
    -d<イニシャライズ変数>[-d<イニシャライズ変数>]<Jamファイル名>
```

[] 内のフラグはオプションです。Jam Playerは１回で１個のJBCファイルのみを処理することができます。表２は各コマンド・ライン・オプションを解説したものです。

表 2　Jam Player のコマンド・ライン・オプション

コマンド・ライン・オプション	定義	機　　能
–h *(1)*	ヘルプ	Jam Player のバージョンをレポートさせる
–v *(1)*	レポート機能	詳細なリアルタイム情報とステータスおよびエラー・メッセージをレポートさせる
-d	イニシャライズ	Jam Player に対して実行するイニシャライズ機能を指定
-p *(1)*	ポート	Jam Player がデータを送出するパラレル・ポート・アドレスを指定
–s *(1)*	ポート	Jam Playerが送出すべきシリアル・ポート・アドレスを指定
–1 *(1)*	ケーブル	ダウンロード・ケーブルの使用を規定。デフォルトではByteBlasterパラレル・ポート・ダウンロード・ケーブルを使用

注：
(1)　このコマンドはオプションです。

-dのコマンド・ライン・オプションを使用する場合は、イニシャライゼーション・リストからJam Playerの初期化に必要な正しい変数が提供されなければなりません。-dのコマンド・ライン・オプション後に使用される変数は表3の通りです。

表 3　-d のコマンド・ライン・オプションに使用される変数名と機能

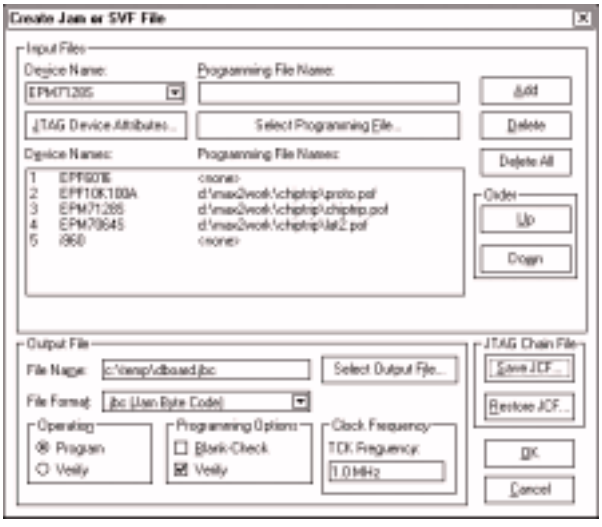
変数名	値	機　　能
DO_ERASE	0	デバイスのイレースを実行しない
	1	デバイスのイレースを実行する
DO_BLANKCHECK	0	デバイスがイレースされている状態をチェックしない
	1	デバイスがイレースされている状態をチェックする
DO_PROGRAM	0	デバイスをプログラムしない
	1	デバイスをプログラムする
DO_VERIFY	0	デバイスをベリファイしない
	1	デバイスをベリファイする
DO_SECURE	0	セキュリティ・ビットをセットしない
	1	対応するPOFがセキュリティ・ビットをプログラムするようになっていれば、セキュリティ・ビットをセットする
DO_SECURE_ALL	0	セキュリティ・ビットをセットしない
	1	POFとは関係なく、セキュリティ・ビットをセットする
DO_READ_UES	0	UES コードをリードしない
	1	UES コードをリードし、レポートする
DO_CONFIGURE	0	デバイスをコンフィギュレーションしない
	1	デバイスをコンフィギュレーションする

イニシャライズの変数は、ターゲットとなるJTAGチェイン内のアルテラのすべてのデバイスに対して適用されます。JBCファイルの生成時に、ターゲット・デバイスに対するプログラミング・ファイルまたはコンフィギュレーション・ファイルを指定します。JBCファイルの生成に使用されるダイアログ・ボックスは、以下の手順で開くことができます。

1. MAX+PLUS IIのソフトウェアを起動する。
2. Programmer（MAX+PLUS IIメニュー）を選択する。
3. Create Jam or SVF File（Fileメユー）を選択する。

図7は、MAX+PLUS IIのソフトウェアで、生成するJBCファイルを指定するときのダイアログ・ボックスを示したものです。

図7　マルチ・デバイスJTAGチェイン用JBCファイルの生成



上記の図で、デバイス名の右側が*<none>*となっているデバイスは、コンフィギュレーション時にバイパスされます。Jam Playerに受け渡されるイニシャライズ変数は、デバイス名の右側に示されているプログラミング・ファイルに対して適用されます。例えば、下記のコマンドで、EPF10K100Aのコンフィギュレーションは EPM7064S と EPM7128S に対するプログラムおよびベリファイと同時に実行され、EPF6016とi960プロセッサの両デバイスがバイパスされます。

```
jbi –v –p378 –dDO_CONFIGURE=1 –dDO_PROGRAM=1 –dDO_VERIFY=1
    dboard.jbc
```

Jam Playerに使用されるメモリ

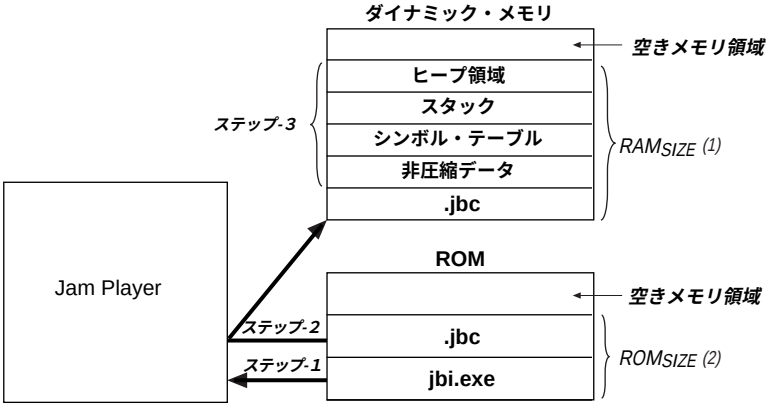
Jam Playerのバイナリ・コードとJBCファイルは不揮発性のシステム・メモリにストアされます。Jam Playerがコールされると、デバイスをプログラミングするタスクがダイナミックRAM（DRAM）を使用してすべて実行されます。Jamファイルをアップデートすることによって、フィールドでのアップグレードが簡単に行えます。

Jam Playerは、下記のような手順でメモリを使用します。

1. コントロール・ソフトウェアがJam Playerをコールする。
2. Jam PlayerがJBCファイルをリードし、DRAMにロードする。
3. Jam Playerが圧縮されたデータを解凍し、シンボル・テーブルとスタック用のメモリ領域をイニシャライズする。

図８は上記の手順と使用されるメモリとの関係を示したものです。

図８　Jam Playerとメモリの関係



注：
(1) RAM_{SIZE}はJam Playerに必要なDRAMの最大容量です。
(2) ROM_{SIZE}はJam PlayerとJBCファイルのストアに必要なROMの最大容量です。

Jam Playerがコールされると、JBCファイル全体がバッファにストアされ、JBCファイル内に含まれているプログラミング・データが伸張されます。なお、場合によっては、Jam Playerによるデータの伸張が不必要なJBCファイルの生成も可能です。

次に、Jam Playerがシンボル・テーブル、スタック、およびヒープ領域をイニシャライズします。シンボル・テーブルには、JBCファイル内に宣言された変数とラベル名がストアされます。スタックは、FORループ、CALLステートメント、およびPUSHステートメントに使用されます。ヒープ領域は、演算表現式の評価やパッディング・データのストアに使用されます。

シンボル・テーブル、スタックおよびヒープの領域がイニシャライズされると、Jam PlayerによるJBCファイルの解析（parse）と実行が可能になります。Jam PlayerがJBCファイルを処理している間は、コマンドが実行されるごとに、スタックとヒープの領域が圧縮されたり、拡張されたりします。このプロセスで、JBCファイル、非圧縮データ、シンボル・テーブルに使用されるメモリの容量は一定のままです。

使用されるROM容量の推定

下記の式を使用して、Jam PlayerとJBCファイルのストアに必要となるROMの最大容量を推定することができます。

ROM_{SIZE} = JBC File Size + Jam Player Size

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・アルゴリズムのために必要なメモリ空間とに分割することができます。JBCファイルのサイズの推定には、下記の式を使用します。

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

ここで

Alg = アルゴリズム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・ファミリのタイプを表わすインデックス
N = チェイン内のターゲット・デバイスの数

この計算式でJBCファイルのサイズが推定できますが、この値はデバイスの使用効率によって、±10%の範囲で変動します。デバイス・リソースの使用率が低い場合は、JBCファイルのサイズが小さくなる傾向があり、圧縮のアルゴリズムによって繰り返しのデータが処理されます。

また、上記の計算式では、デバイス・ファミリに対してアルゴリズムのサイズは一定になるが、ターゲット・デバイスの数が増加するにしたがって、プログラミング・データのサイズが増加することが示されています。一定のデバイス・ファミリに対して、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（データ数の増加による）、リニアに増加します。

表４はJam言語をサポートしているアルテラの各デバイス・ファミリの組み合わせた場合にアルゴリズム部分で使用されるファイル・サイズの値を示したものです。

表４　アルテラの各デバイス・ファミリの組み合わせに対応したアルゴリズム部分のファイル・サイズ

デバイス・ファミリ	標準的なアルゴリズムのサイズ（Kバイト）	
	ASCII Jam File	JBC File
MAX 7000S, MAX 7000A	22	18
MAX 9000	26	21
MAX 9000, MAX 7000S, MAX 7000A	42	35
FLEX 10K, MAX 7000S, MAX 7000A	42	35
FLEX 10K, MAX 9000, MAX 7000A, MAX 7000S *注(1)*	42	35
FLEX 10K	6	4
MAX 7000AE	*注(2)*	*注(2)*

注：

(1) FLEX 10KデバイスのコンフィギュレーションとMAX 9000、MAX 7000AまたはMAX 7000Sデバイスのプログラミングを行う場合、FLEX 10Kのアルゴリズムに追加されるメモリ領域は無視できるほど小さくなります。

(2) MAX 7000AEデバイスに関する情報については、日本アルテラの応用技術部へお問い合わせください。

表５は、ISP用のJam言語をサポートしているアルテラの各デバイスを組み合わせた場合のデータ部分のサイズを示したものです。

表５　データ部分のファイル・サイズ　*注(1)*

デバイス	標準的なアルゴリズムのサイズ（Kバイト）		
	圧縮		非圧縮
	ASCII Jam	JBC	JBC *注(2)*
EPM7032S	1	4	4
EPM7064S	3	9	9
EPM7128S, EPM7128A	6	5	20
EPM7160S	9	6	27
EPM7192S	10	7	34
EPM7256S, EPM7256A	14	10	49
EPM9320, EPM9320A	20	15	59
EPM9400	26	19	70
EPM9480	25	18	73
EPM9560, EPM9560A	27	20	96
EPF10K10, EPF10K10A	11	8	14
EPF10K20	23	17	28
EPF10K30, EPF10K30A, EPF10K30E	39	28	46
EPF10K40	51	37	61
EPF10K50, EPF10K50E, EPF10K50V	60	44	76
EPF10K70	95	69	109
EPF10K100, EPF10K100A, EPF10K100B, EPF10K100E	130	95	146
EPF10K130E, EPF10K130V	177	128	194
EPF10K200E	196	143	322
EPF10K250A, EPF10K250E	245	179	403

注：
(1) MAX 7000AEデバイスに関する情報は、日本アルテラの応用技術部へお問い合わせください。
(2) 非圧縮のプログラミング・データとなっているJBCファイルの生成方法については、日本アルテラの応用技術部へ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、表６を使用してJam Playerのサイズを推定します。

表６　Jam Player のバイナリ・サイズ

プロセッサ		標準的なサイズ（K バイト）	
サイズ	説　明	ASCII Jam Player	JBC Player
16-bit	BitBlaster™、またはByteBlasterMV™（またはByteBlaster）ダウンロード・ケーブルを使用したペンティアム／486プロセッサ	105	62
32-bit	BitBlaster™、またはByteBlasterMV™（またはByteBlaster）ダウンロード・ケーブルを使用したペンティアム／486プロセッサ	115	68

使用されるダイナミック・メモリの容量の推定

Jam Playerに要求されるDRAMの最大容量は、下記の式で推定することができます。

$$RAM_{SIZE} = \text{JBC File Size} + \sum_{k=1}^{N} \text{ACA variable } k$$

JBCファイルのサイズは、１個または複数のデバイスの計算式で決定されます（14ページの「使用されるROM容量の推定」を参照）。

上記の式で、ACA変数は*k*番目の圧縮アレイが解凍されたときのサイズを示し、*N*はJBCファイル内でACAで圧縮されたアレイのトータル数です。
ACA変数のサイズは、ASCII JamファイルのVariable Declaration/Initializationのセクションを見れば判明します。各アレイのサイズ（ビット数）は、 Variable Declarationのステートメントの[ ] 内に記述されています。例えば、

```
BOOLEAN A21[104320] = ACA mB300u...
```

この例では、ACA変数が、解凍されたときに104,320ビットになることを示しています。

スタックとヒープに要求されるメモリの容量は、バイト・コードのJam Playerに使用されるメモリ容量に比較して無視できるほど小さくなります。スタックの最大の深さは、Jbimain.cファイル内のJBI_STACK_SIZEのパラメータで指定されます。

メモリ・サイズの推定例

下記にモトローラ社のプロセッサ、68000を使用して、IEEE Std. 1149.1 のJTAGチェインに接続されたEPM7128SとEPM7064SをJBCファイルでプログラムする場合に使用されるメモリ・サイズの推定例を示します。使用されるメモリ・サイズを算出する場合は、まず最初に要求されるROMの容量を計算し、次にRAMの容量を計算します。バイト・コードのJam Playerに必要となるDRAMの容量は、下記の手順で計算できます。

1. JBCファイルのサイズを決定します。JBCファイルのサイズの推定にはマルチ・デバイスの計算式を使用します。

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

   ここで、

   Alg = 18Kバイト
   Data = EPM7064Sのデータ＋EPM7128Sのデータ＝9＋5= 14Kバイト

   したがって、JBCファイルのサイズは32Kバイトとなります。

2. バイト・コードのJam Playerのサイズを推定します。この例では、推定されるバイナリ・サイズを62Kバイトとします。下記の式を使用して、必要なROMの容量を計算します。

   ROM_{SIZE}= JBCファイルのサイズ＋Jam Playerのサイズ

   ROM_{SIZE}= 94Kバイト

3. 下記の式で、使用されるRAMのサイズを計算します。

   $$RAM_{SIZE} = 32\ \text{Kbytes} + \sum_{k=1}^{N} \text{ACA variable } k$$

   ACA Variableの値は、下記の通りです。（圧縮されたアレイを見つけるときには、ASCII Jam Fileをオープンしてください）

```
BOOLEAN A21[150120] = ACA Db400u...
BOOLEAN A22[97640] = ACA j_200u...
```

圧縮されたデータの解凍には、下記の容量のRAMが使用されます。

$$\frac{150{,}120 \text{ bits} + 97{,}640 \text{ bits}}{8 \frac{\text{bits}}{\text{byte}}} = 30 \text{ Kbytes}$$

使用されるトータルのDRAM容量は、次のように計算されます。

$$RAM_{SIZE} = 32 \text{ Kbytes} + 30 \text{ Kbytes} = 62 \text{ Kbytes}$$

一般的に、JamファイルにはROMよりもRAMに大きな容量が必要になりますが、RAMのほうが低価格であるため、これは好ましい傾向です。より多数のデバイスがプログラムされる場合ほど、簡単なアップグレードを実現するために必要となるコストの割合は低下してきます。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になります。

Jam Playerの動作

Jam Playerは、IEEE Std. 1149.1のJTAG Test Access Port（TAP）ステート・マシンを操作するためのインタフェースを提供しています。TAPコントローラはTCKの立ち上がりエッジで動作する16ビットのステート・マシンとなっており、デバイス内のJTAG動作のコントロールには、TMSピンが使用されます。図9は、IEEE Std. 1149.1 TAPコントローラのステート・マシンの動作を示す状態遷移図です。

図 9　JTAG TAPコントローラのステート・マシン動作

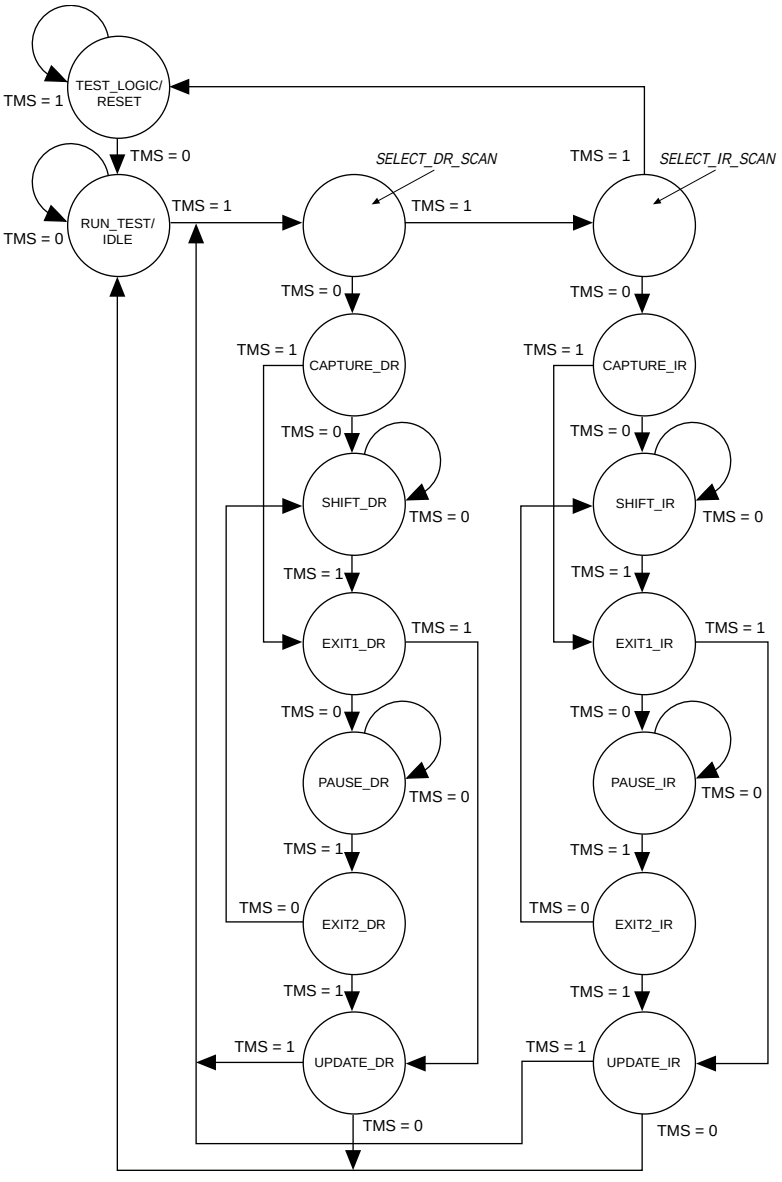


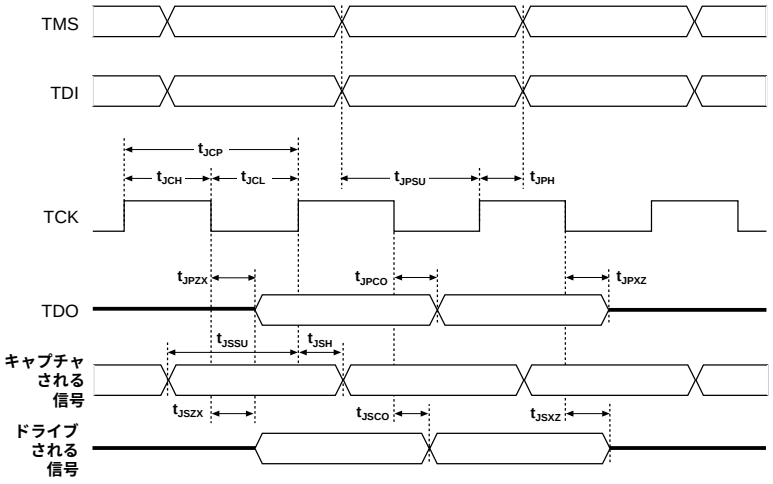
表7はTAPコントローラのタイミング規格を示したものです。これらのタイミング・パラメータはIEEE Std. 1149.1の仕様で規定されているものと同一です。

表7　JTAGのタイミング・パラメータ

シンボル	パラメータ	MAX 9000		MAX 7000A		MAX 7000AE		MAX 7000S		FLEX 10K		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のクロック・サイクル	100		100		100		100		100		ns
t_{JCH}	TCKのHigh期間	50		50		50		50		50		ns
t_{JCL}	TCK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出力確定までの遅延		25		25		25		25		25	ns
t_{JPXZ}	JTAGポートの確定出力からハイ・インピーダンスまでの遅延		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出力確定までの遅延		25		25		25		25		25	ns
t_{JSXZ}	アップデート・レジスタの確定出力からハイ・インピーダンスまでの遅延		25		25		25		25		25	ns

図10は各タイミング・パラメータを波形で示したものです。システムの設計にあたっては、これらのタイミング・パラメータを基準にして、あらゆるシステム上でJam Playerが適切に動作するようにしなければなりません。

図10　JTAGの波形とタイミング



Jam PlayerがTAPコントローラを操作する下位レベルのドライバを提供するのに対して、JBC Fileは与えらえたデバイスのプログラムに必要な上位レベルの機能を提供します。デバイスに対するJTAGデータの操作を行うためのすべてのJam命令は、ステート・マシンのデータ・レジスタの１部あるいはインストラクション・レジスタの１部を通じてTAPコントローラに与えられ、実行されます。例えば、JTAG命令のロードは、TAPコントローラをSHIFT_IRのステートに遷移させ、TDIピンから命令をインストラクション・レジスタにシフト・インさせます。次に、TAPコントローラはRUN_TEST/IDLEのステートに遷移し、ここでラッチされる命令の実行に必要な遅延が実現されます。このプロセスは、ステート・マシンのデータ・レジスタの一部が変化することを除いてデータ・レジスタのスキャン動作と同じです。

ハイ・レベルなJam命令としては、JTAGデータ・レジスタをスキャンするためのDRSCAN命令、インストラクション・レジスタをスキャンするためのIRSCAN命令、ステート・マシンを規定された時間まで特定のステートでウェイトさせるWAITコマンドがあります。TAPコントローラの各レジスタはJBCファイル内の命令に従って、ターゲット・デバイスがプログラムされるまで繰り返しスキャンされます。

Jam命令の詳細については、「*Jam Programming & Test Language Specification*」を参照してください。

図11はJam PlayerがJBCファイルの解析（parse）を行うときの動作フローを示したものです。DRSCAN、IRSCANまたはWAITの命令が与えられると、Jam PlayerはTCK、TMS、TDIの各ピンに対する適切なデータを生成して、命令を実行させます。この動作フロー図には、DRSCAN、IRSCAN、WAITの各命令に対応したブランチが示されています。Jam Playerは他の命令もサポートしていますが、簡略化するためにこのフローには示されていません。

図11　Jam Playerのフロー・ダイヤグラム　(1/2)

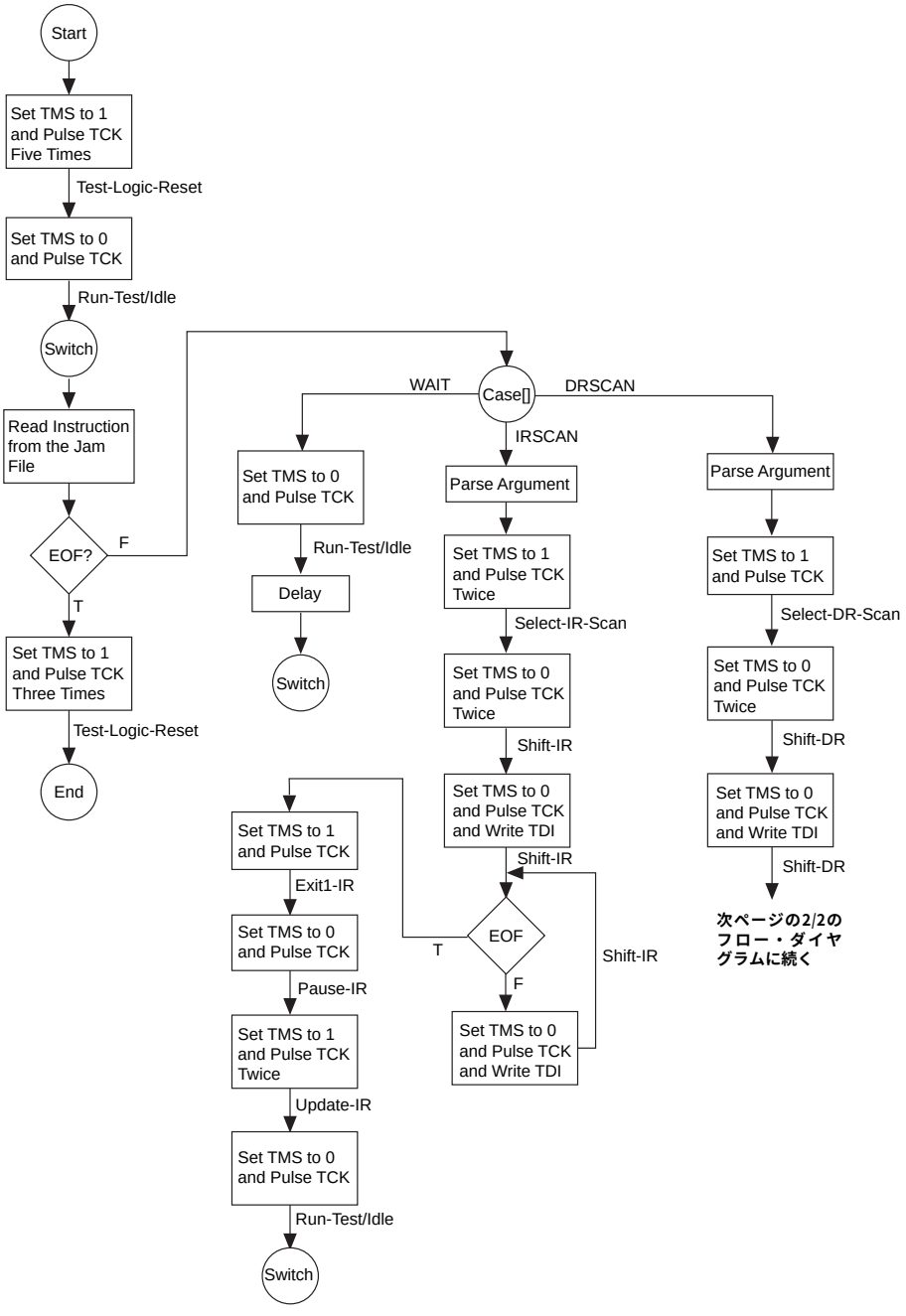
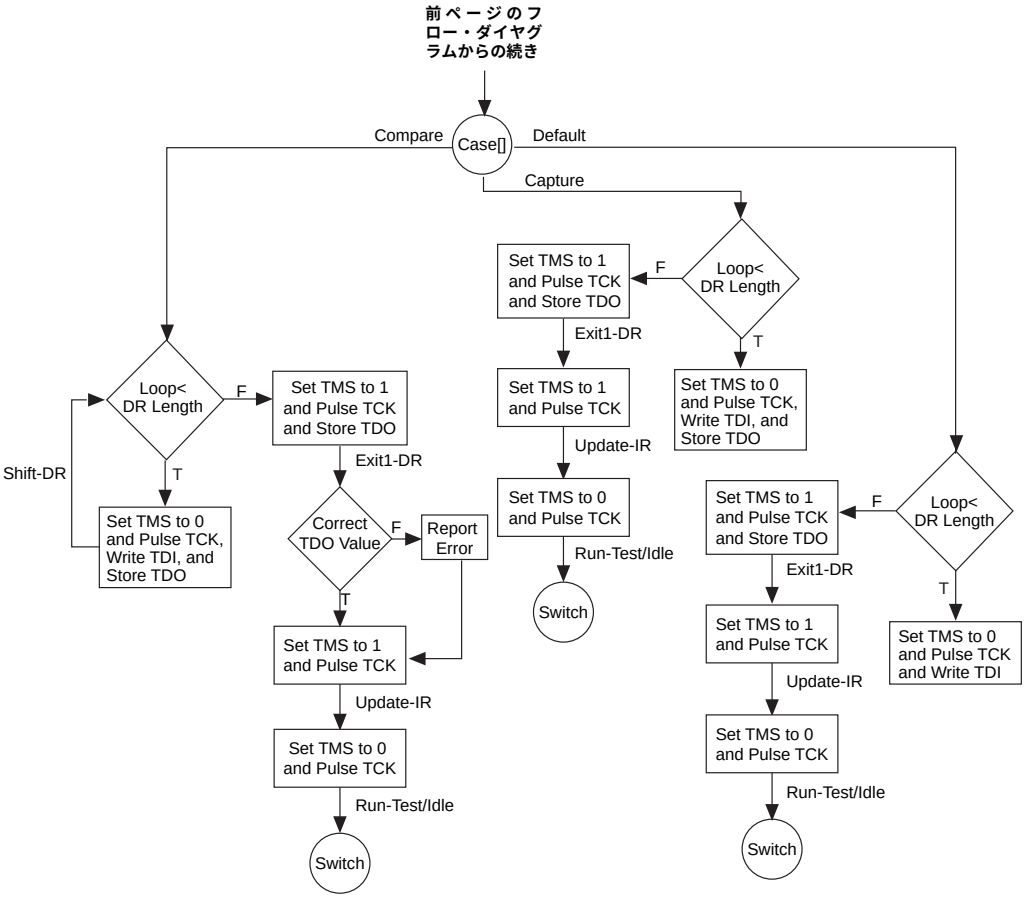


図11　Jam Playerのフロー・ダイヤグラム　(2/2)



まとめ

エンベデッド・プロセッサによるイン・システム・プログラミングやICRから得られる利点の実現には、小さなファイル・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、システム側に要求される各項目を満たしているJamプログラミング／テスト用言語の使用が最適です。エンベデッド・プロセッサによるISPとICRにJam言語を使用することによって、フィールドでのアップグレードがサポートされ、デザインの試作が容易になると共に、製造工程が短縮されます。これらの利点によって、最終製品の品質と柔軟性が強化されます。また、プログラムされたデバイスの在庫や管理が不要になるため、デバイスの在庫を減少させることができます。

日本アルテラ株式会社

〒163-0436
東京都新宿区西新宿2-1-1
新宿三井ビル私書箱261号
TEL. 03-3340-9480　FAX. 03-3340-9487
http://www.altera.com/japan/

本社 Altera Corporation

101 Innovation Drive,
San Jose, CA 95134
TEL : (408) 544-7000
http://www.altera.com