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 It may contain legacy content and trademarks which may be outdated.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

SignalTap II エンベデッド・ロジック・アナライザを使用した Nios II システムのデバッグ

2008 年 6 月 ver. 1.2 Application Note 446

はじめに

FPGA システム・デザインは、プロセッサ、ペリフェラル、バス、およびブリッジの数を増やして、より複雑でシステム集約化の傾向にあるため、設計者はより良質で洗練されたシステム・レベルのデバッグ・ツールを必要としています。アルテラの SignalTap® II エンベデッド・ロジック・アナライザは、ロジック・アナライザをシステム内に組み込むことにより、リアルタイムのハードウェア・デバッグ機能を提供します。SignalTap II エンベデッド・ロジック・アナライザ用の Nios® II プラグ・インは、Nios II プロセッサのプログラム実行のキャプチャを可能にすることにより SignalTap II のシステム・デバッグ能力を拡張します。

このアプリケーション・ノートでは、Nios II プロセッサによるソフトウェア実行中に提供される機能的な情報を使用してシステム・デザインをデバッグする方法について説明します。このアプリケーション・ノートでは、SignalTap II ロジック・アナライザ用の Nios II プラグ・インの使い方について検討し、このプラグ・インの性能、コンフィギュレーション・オプション、および使用モデルを示します。簡単なチュートリアルでは、Nios II プラグ・イン、SignalTap II ロジック・アナライザ、および Nios II 統合開発環境（IDE）を使用して、RS-232 UART インタフェースからの文字の受信をトリガ、キャプチャ、およびトレースする方法を示します。

Nios II プラグ・インによって SignalTap II エンベデッド・ロジック・アナライザの能力が拡張され、ユーザーは、Nios II プロセッサ・コアで実行中の命令のトレース・データを簡単にトリガおよびキャプチャすることができます。ユーザーは、命令トレースのトリガを指定することができ、これによって、ユーザー・プログラムからシンボル名を入力するか、またはユーザー自身の SignalTap II トリガ条件を指定することにより、プロセッサが特定のアドレスに達したときに SignalTap II ロジック・アナライザがトリガをかけます。Nios II プラグ・インはプロセッサ・トレースを指定されたソフトウェア・イメージと自動的に関連付け、デコードされた Nios II 機械語オペレーション・コード（命令コード）とともにシンボル名とトレースのオフセット・ビューをユーザーに提供します。

前提条件

このアプリケーション・ノートの目的は、SignalTap II フレームワーク中での Nios II プラグ・インの使い方を示すことです。このアプリケーション・ノートは、SignalTap II エンベデッド・ロジック・アナライザの基本的な使い方を含め、特定のアルテラ・ソフトウェア・ツールを理解して

いる方を対象にしています。さらに、チュートリアルは、特定のアルテラ・ツールおよび IP（Intellectual Property）にアクセス可能な方を対象にしています。

このアプリケーション・ノートは、Quartus® II ソフトウェア、SOPC Builder、Nios II エンベデッド・プロセッサの使い方、および SignalTap II エンベデッド・ロジック・アナライザの基本的な機能と操作を理解している方を対象にしています。Nios II プラグ・インは SignalTap II ロジック・アナライザを拡張したものであるため、このプラグ・インの基本的な使い方は SignalTap II ロジック・アナライザとほぼ同じです。

SignalTap II のコンフィギュレーション・オプションおよび使用モードについて詳しくは、「Quartus II ハンドブック Volume 3」の「SignalTap II エンベデッド・ロジック・アナライザを使用したデザインのデバッグ」の章を参照してください。

必要なツール

Nios II プラグ・インを使用するには、以下のツールが必要です。

- Quartus II ソフトウェア v8.0 以降
- Nios II Embedded Development Suite 8.0 以降

このアプリケーション・ノートの最後に記載されているチュートリアルを完了するには、以下の追加リソースが必要です。

- Nios II 開発キット
- RS-232 シリアル・ケーブル（オプション）
- JTAG ByteBlaster™ または USB-Blaster™ ケーブル
- ソース・ファイル **signal_tap_test.c**（このアプリケーション・ノートとともにダウンロード可能なファイルとして提供されます）

チュートリアルに必要なハードウェアおよびソフトウェアの詳細については、12 ページの「チュートリアル：Nios II プラグ・インの使用」を参照してください。

Nios II プラグ・イン

Nios II プラグ・インは、SignalTap II エンベデッド・ロジック・アナライザに対するデバッグ用拡張機能です。このプラグ・インを使用すると、Nios II エンベデッド・プロセッサにより実行されるオペレーション・コードをキャプチャすることができます。Nios II プラグ・インは、Nios II プロセッサの ALU 内部のデバッグ・ノードをインスタンス化することにより動作します。Nios II プラグ・インは、全種類の Nios II プロセッサ・コアをサポートします。

各 Nios II プラグ・インのインスタンス化は、特定の Nios II プロセッサと関連付けられ、他の Nios II プラグ・インのインスタンスおよび他の SignalTap II インスタンスとともに動作できます。

以下の種類のトリガ条件を指定することができます。

- 命令アドレス
- Nios II の実行可能かつロード可能フォーマット（**.elf**）ファイル内に存在するシンボル名（ファンクション）
- シンボル名とオフセット

プラグ・インは、キャプチャしたすべての命令を文字に変換します。命令アドレスは、（可能な場合）シンボル名とアドレス・オフセットに変換され、命令オペレーション・コードはそれぞれ等価なアセンブリ言語ニーモニックに変換されます。

Quartus II プロジェクトおよび SignalTap II ロジック・アナライザのセットアップ

Nios II プラグ・インで使用するために、SignalTap II ファイルを Quartus II プロジェクト内に作成し、コンフィギュレーションしなければなりません。システムに SignalTap II ファイルを追加するには、以下の手順を実行します。

1. Quartus II の File メニューの **New** をクリックします。
2. **New** ダイアログ・ボックスで、**Verification/Debugging Files** カテゴリ内の **SignalTap II Logic Analyzer File** をクリックします。
3. **OK** をクリックします。

あるいは、以下の手順を実行することにより、新規または既存の SignalTap II ファイルを開くことができます。

✓ Quartus II の Tools メニューから **SignalTap II Logic Analyzer** をクリックします。

この手順を実行すると、SignalTap II ウィンドウが表示されます。

Nios II プラグ・インは、インクリメンタル・コンパイル・オプションが設定された Quartus II プロジェクトには現在のところ対応していません。したがって、Quartus II プロジェクトでインクリメンタル・コンパイルがディセーブルされていることを確認する必要があります。インクリメンタル・コンパイルをディセーブルするには、以下の手順を実行します。

1. Assignments メニューの **Settings** をクリックします。
2. **Settings** ダイアログ・ボックスで、**Compilation Process Settings** を展開し、**Incremental Compilation** をクリックします。
3. **Incremental Compilation** の下で、**Off** を選択します。
4. **OK** をクリックします。

Nios II プラグ・インの追加

Nios II プラグ・インをシステムに追加する場合、モニタする Nios II プロセッサ、およびオプションで、そのプロセッサのソフトウェア・イメージである **.elf** ファイルを指定しなければなりません。Nios II プラグ・インは、**.elf** ファイルを処理してシンボル情報を抽出します。この情報は、コンフィギュレーション中にトリガ条件指定のために使用されます。

システムに Nios II プラグ・インを追加するには、以下の手順を実行します。

1. Quartus II ウィンドウで、Processing メニューの **Start** をポイントして **Start Analysis & Elaboration** をクリックします。
2. SignalTap II ウィンドウの SignalTap II ノード・リスト内で右クリックします。**Add Nodes with Plug-In** をポイントして、**Nios II** をクリックします。**Select Hierarchy Level** ダイアログ・ボックスが表示されます。
3. **Select Hierarchy Level** ダイアログ・ボックスで、プラグ・インでモニタする Nios II プロセッサのインスタンスを選択します。
4. **OK** をクリックします。**Plug-In Options** ダイアログ・ボックスが表示されます。
5. **Plug-In Options** ダイアログ・ボックスで、オプションで **.elf** ファイルの場所を指定します。

2 つの Nios II プラグ・イン・インスタンスが同じ Nios II プロセッサをモニタすることはできません。1 つのプロセッサあたり 2 つ以上の Nios II プラグ・イン・インスタンスを追加しようとすると、エラー・メッセージが表示されます。

6. **OK** をクリックします。

Nios II プラグ・インが使用する **.elf** ファイルは、以下の手順を実行することにより、いつでも変更できます。

1. SignalTap II ウィンドウで、**Setup** タブをクリックします。

2. SignalTap II ノード・リスト内で、変更する Nios II プラグ・イン・インスタンスを右クリックし、**Plug-In Options** をクリックします。Nios II プラグ・インのコンフィギュレーション・オプションが表示されます。

☞ **.elf** ファイルは、ソフトウェアのビルド・プロセス中に Nios II IDE によって生成されます。ほとんどの場合、ユーザーのソフトウェア・プロジェクト用 **.elf** ファイルは、Nios II IDE によって作成された **Debug** ディレクトリまたは **Release** ディレクトリに置かれています。ユーザーは、Nios II ソフトウェア・ビルド・ツールのフロー内で **.elf** ファイルを作成することもできます。

トリガ条件の指定

標準 SignalTap II ロジック・アナライザのトリガ条件がハードウェアまたはロジックのイベントとして記述されるのとは異なり、Nios II プラグ・インのトリガ条件は命令アドレスとして指定します。Nios II プラグ・インは、プログラム実行中に Nios II プロセッサが指定された命令アドレスに達したときにトリガをかけます。

基本トリガ

基本トリガ・モードでは、Nios II プラグ・インは、トレース・キャプチャ開始のトリガとしてプロセッサが認識できるシステム・アドレスを使用します。トリガを設定するには、Nios II プラグ・インの **Trigger Conditions** 欄をクリックし、命令アドレスを入力します。

以下のサポートされるトリガ条件のいずれかを入力することができます。

- 16 進整数：0x<*32 ビットの数値*>（例えば、0x20000000）
- **.elf** ファイル・シンボル：< *文字列*> ― **.elf** ファイル内でシンボルとして表示される英数字による C/C++ ファンクション名（例えば、`foo`）
- **.elf** ファイル・シンボル + オフセット：< *文字列*>+<*16 進数*>（例えば、`foo+0x80`）

シンボル名オプションを使ってトリガ条件を指定するには、Nios II プロセッサの **.elf** ファイルを使用して Nios II プラグ・インのコンフィギュレーションを行わなければなりません。**.elf** ファイルに指定されたシン

ボル名が含まれていない場合は、エラー・メッセージが表示され、トリガ条件のフィールドは不定値に設定されます。トリガ条件フィールドに値を入力すると、Nios II プラグ・インは参照する **.elf** ファイルの内容が変更されているかを検査します。このチェックによって、ソフトウェアのデバッグ中などのように **.elf** ファイルが頻繁に変更されている場合でも、トリガ条件が **.elf** ファイルの内容と一致していることが保証されます。

.elf ファイル内のシンボルを確認するには、以下の方法のうちいずれかを使って **objdump** ファイルを生成します。

- makefile 内で CREATE_OBJDUMP オプションをオンにして、プロジェクトをビルドします。
- **.elf** ファイルを作成した後、Nios II コマンド・シェルで以下のコマンドを入力します。

```
nios2-elf-objdump -s <filename>.elf ↵
```

詳細については、15 ページの「Nios II ソフトウェアのビルド」を参照してください。

標準 SignalTap II ロジック・アナライザのトリガ条件は、SignalTap II ウィンドウ内で **Trigger Conditions** 欄を右クリックし、トリガ・パターンを選択することにより指定できます。しかし、明確にするためには、Nios II プラグ・インのトリガ条件を手動で入力しなければなりません。

Nios II プラグ・インでアドレス・トリガ条件を指定しない場合、通常の SignalTap IIインスタンスでトリガ条件を指定しないのと同じ影響をもたらします。トリガ条件を指定しない場合、ロジック・アナライザは直ちにトリガをかけ、表示されるデータは通常、役に立ちません。

複数のトリガ

Nios II プラグ・インは SignalTap II の複数トリガ条件機能をサポートしており、より複雑なキャプチャ・シーケンスの一部として組み込むことができます。Nios II プラグ・インのトリガ・パターンを、任意の SignalTap II トリガ条件の一部として組み込むことができます。

拡張トリガ

SignalTap II の拡張トリガ・オプションを使用して複雑なトリガを作成することができます。しかし、このオプションを使用した場合、Nios II プラグ・インの多くの利点が失われます。Nios II プラグ・インに含まれる信号グループ化は Advanced Trigger Configuration Editor 内に表示され、こ

れらは通常の信号として扱われます。Advanced Trigger Configuration Editor は、アドレスまたはシンボルによってトリガする Nios II プラグ・インの機能を備えていません。

パワーアップ・トリガ

Nios II プラグ・インは、SignalTap II のパワーアップ・トリガ機能とともに使用することができます。パワーアップ・トリガは、SignalTap II ロジック・アナライザを手動で開始する前にイネーブルされるため、Nios II プロセッサが自己ブート・モードで、つまり FPGA がコンフィギュレーションされた直後に、動作するシステムをモニタするのに役立ちます。この場合、Nios II プロセッサは、デバッガを使用せずにシステム・メモリから直接ソフトウェアの実行を開始し、プロセッサのランタイム・メモリを開始、停止およびロードします。

アクイジション・クロックの割り当て

サンプルのアクイジションを制御するには、クロック信号を指定する必要があります。クロック信号は、SignalTap II ウィンドウの Signal Configuration 枠内で指定します。Nios II プロセッサが使用するクロック信号を、SignalTap II のアクイジション・クロックとして選択することを推奨します。Nios II プロセッサのクロックを使用することで、キャプチャされた命令トレース・データが、Nios II プロセッサの命令実行に正確に対応するようになります。

サンプル容量、メモリ・タイプ、およびバッファ収集モードの選択

Nios II プラグ・インでは、SignalTap II ロジック・アナライザの場合と同様に、キャプチャ・セッションについてサンプル容量、メモリ・タイプ、およびバッファ収集モードを設定する必要があります。これらのコンフィギュレーション・オプションは、通常の SignalTap II ロジック・アナライザのキャプチャ・セッションと同じように動作し、Signal Configuration 枠を通じてアクセスできます。

サンプル容量の大きさを選択する際には注意してください。Nios II プラグ・インは収集するサンプルごとに多くの信号をキャプチャする必要があるため、利用可能なメモリ・リソースは急速に減少します。サンプル容量パラメータの調整がデザインに与える影響をよりよく理解するために、SignalTap II に内蔵されているリソース使用量推定機能を使用してください。

デザインのコンパイルおよびターゲット・デバイスのプログラミング

Nios II プラグ・インは、SignalTap II ロジック・アナライザを使用して Quartus II デザインでコンパイルされます。ただし、Quartus II のインクリメンタル・コンパイル・オプションは Nios II プラグ・インではサポートされないため、プラグ・インを追加した後で Quartus II プロジェクトの完全なコンパイルを実行する必要があります。

コンパイル後、Nios II プラグ・インを使用していない場合と同様に、SignalTap II ウィンドウからの SRAM オブジェクト・ファイル（**.sof**）を使用して FPGA ターゲット・デバイスをプログラムすることができます。

キャプチャ・セッションの実行

Nios II プラグ・インによるデータ収集は、SignalTap II ロジック・アナライザを使用してデータを収集する場合と同じ方法で実行します。まず、Quartus II ソフトウェアで生成された **.sof** ファイルを使用して FPGA をプログラムします。次に、SignalTap II Instance Manager を使って手動で、または FPGA がプログラム済みでパワーアップ・トリガが選択されている場合は自動的に、SignalTap II 解析を実行します。システムがトリガ条件を満たしている場合、SignalTap II ロジック・アナライザは、収集したデータを SignalTap II 結果ウィンドウに表示します。

Nios II プラグ・インは、Nios II IDE によるものとスタンドアロン・モードの2種類のデータ・キャプチャ・セッションで使用することができます。

Nios II IDE によるデータ・キャプチャ・セッションの実行

Nios II IDE で Nios II プラグ・インを使用するには、Nios II ソフトウェア・イメージを手動でダウンロードし、デバッガを使用してプロセッサの動作を制御する必要があります。この種のキャプチャ・セッションは通常、Nios II ソフトウェア・アプリケーションを開発しデバッグする際に実行されます。

Nios II IDE によって制御された Nios II プロセッサで SignalTap II キャプチャ・セッションを実行するには、以下の手順を実行します。

1. SignalTap II ウィンドウで、以下の手順を実行することにより、Quartus II ソフトウェアにより生成された **.sof** ファイルで FPGA ターゲット・デバイスをプログラムします。

    a. Hardware メニューで、使用している Nios II 開発ボードに接続されたプログラミング・ケーブルを選択します。

b. **SOF Manager** フィールド内で、ブラウズ・ボタンをクリックします。

c. **Select Programming File** ダイアログ・ボックスで、現在のプロジェクト用に生成された **.sof** ファイルを選択します。

d. **Open** をクリックします。これで **Program Device** ボタンが使用できる状態になります。

e. **Program Device** ボタンをクリックして、**.sof** ファイルを FPGA にダウンロードします。 

2. SignalTap II ウィンドウの Instance Manager 枠内で、**Run Analysis** ボタンをクリックして、ロジック・アナライザのキャプチャ・セッションを開始します。 

3. Nios II IDE で、Nios II プロセッサで実行するソフトウェア・プロジェクトの名前を右クリックして、**Debug As** をクリックします。この操作により、デバッガが起動し、**.elf** ファイルがシステム・メモリにダウンロードされ、プロセッサは `main()` へのエントリ・ポイントで停止します。

4. **Debug** タブ上で、**Resume** ボタンをクリックし、Nios II プロセッサの実行を開始します。 

SignalTap II ロジック・アナライザは、Nios II プラグ・インで指定されたトリガ条件に達するまで継続して動作します。SignalTap II ロジック・アナライザが動作中、Nios II IDE のすべてのデバッグ操作を安全に使用することができます（例えば、ブレークポイントを設定してプロセッサを停止することができます）。

デバッガを起動すると、プロセッサがトリガ・アドレスまで進んだ場合に Nios II プラグ・インがトリガをかけます。スタートアップ・ブレークポイントの位置が、Nios II プラグ・インについて指定されたトリガ・アドレス位置の後に生じた場合は、デバッガによる検出誤りの原因になることがあります。デバッガのブレークポイント開始位置を変更するには、Nios II IDE で以下の手順を実行します。

1. **Run** メニューの **Debug** をクリックします。Debug ウィンドウが表示されます。

2. Debug ウィンドウで、**Debugger** タブをクリックします。

3. Nios II IDE デバッガを実行させるブレークポイント位置を選択し、**Apply** をクリックします。

あるいは、**Debug As** オプションを使用する代わりに **Run As** オプションを使用することができます。**Run As** オプションを使用すると、Nios II IDE は、Nios II IDE のデバッガ機能を呼び出さずに、システム・メモリからソフトウェア・イメージをダウンロードして実行します。

外部ソフトウェアからダウンロードせずに実行するデータ・キャプチャ方法

使用している Nios II プロセッサ・ベース・システムが、外部ソフトウェアのダウンロードを必要としない自己ブートに設定され、SignalTap II パワーアップ・トリガ機能が選択されている場合、SignalTap II ロジック・アナライザは、FPGA がプログラムされると自動的に実行を開始します。

この場合、SignalTap II ロジック・アナライザが既に有効なデータをキャプチャしている場合があります。キャプチャされたデータが有効かどうか、またはロジック・アナライザがまだ動作中かどうかを確認するには、SignalTap II Instance Manager で **Run Analysis** をクリックします。

自己ブート Nios II プロセッサ・システムの作成について詳しくは、「Nios II ソフトウェア開発ハンドブック」を参照してください。

解析結果

Nios II プラグ・インを使用すると、キャプチャされた Nios II プロセッサのトレース・データを見ることができます。この項では、Nios II プラグ・インのキャプチャ後の機能のいくつかについて説明します。

データの表示

キャプチャされた SignalTap II データは、SignalTap II ウィンドウの **Data** タブに表示されます。Nios II プラグ・インでキャプチャされたすべてのサンプルについて、以下の情報が表示されます。

- **アドレス**－16 進数で表された命令アドレス位置。さらに、プラグ・インのコンフィギュレーション中に **.elf** ファイルが指定された場合、命令アドレスはシンボル名とオフセットにさらに解決されることがあります。
- **アセンブリ言語ニーモニック**－命令のバイナリ・オペレーション・コードと等価な Nios II アセンブリ言語

プラグ・インのコンフィギュレーション中に **.elf** ファイルを指定した場合、ソフトウェアはデータ収集が完了した直後にファイルを再検査し、命令アドレスをシンボル名とオフセットによる表現に変換します。この再検査は、古いソフトウェア・イメージを誤って使用しないように保護するのに役立ちます。

SignalTap II のタブ・コントロールを使用して、Nios II プロセッサのプログラム実行全体をスクロールすることができます。指定したアクイジション・クロックが Nios II プロセッサのクロックに対応している場合、クロックの各立ち上がりエッジが新しい命令サイクルに対応します。

Nios II プラグ・インで収集されたトレース・データの中に、1 つまたは複数の「**空の**」命令エントリが見られる場合があります。このようなエントリは、その特定のクロック・サイクル中に Nios II プロセッサによって命令が実行されなかったことを示します。この挙動は正常であり、以下の理由で生じる可能性があります。

- **キャッシュ・ミス**－要求された命令アドレス位置により命令キャッシュ内にミスが発生し、キャッシュ・ラインを満たして命令を戻すために追加のクロック・サイクルが必要です。
- **メモリ競合または速度**－命令アドレス位置が、アクセスに複数のクロック・サイクルを必要とするメモリ内、または現在他のペリフェラルまたはプロセッサにより制御されているメモリ内にあります。

Nios II プラグ・インのトレース・データは、SignalTap II のリスト・ファイル・フォーマットでも見ることができます。この表形式フォーマットでは、トレース・サンプルは、サンプル番号および関連するアセンブリ言語ニーモニックによってデコードされ、時系列的に行に表示されます。このリスト・ファイル・フォーマットは、`nios2-elf-objdump` コマンドの出力フォーマットと似ていて、解析プロセスが簡略化されるため、役に立ちます。SignalTap II リスト・ファイルを作成するには、File メニューの **Create/Update** をポイントして、**Create SignalTap II List File** をクリックします。

Nios II プロセッサ命令には、実行時に複数のクロック・サイクルを必要とするものがあります。特定の命令に必要なクロック・サイクル数について詳しくは、「Nios II プロセッサ・リファレンス・ハンドブック」の「Nios II コア実装の詳細」の章を参照してください。

プロセッサの .elf ファイルへのトレース・データの関連付け

objdump ファイルの内容を調べることによって、キャプチャされた Nios II プラグ・インの命令トレースを、Nios II プロセッサによって実行されるソフトウェア・イメージと比較することができます。

nios2-elf-objdump コマンドを使用して **objdump** ファイルを作成することができます。このコマンドは、プロセッサの **.elf** ファイルを、C/C++ コードの一部分、記号ファンクション名、アセンブリ命令、およびアドレス位置を含む **objdump** ファイルの文字列にコピーします。

nios2-elf-objdump 実行コマンドは、Nios II エンベデッド・デザイン・スイートの一部として含まれています。一連のコマンドライン・オプションを使用してツールをコンフィギュレーションすることができます。nios2-elf-objdump コマンドの変換オプションのリストを表示するには、Nios II コマンド・シェルで以下のコマンドを入力します。

```
nios2-elf-objdump --help ↵
```

objdump ファイルには **.elf** ファイルからデコードされた膨大な情報が含まれていますが、Nios II プロセッサの命令は、このファイルに以下のフォーマットで 1 行につき 1 つずつ表示されます。

< アドレス >:*< オペレーション・コード > < アセンブリ・ニーモニック >*

例えば、有効な命令は次のとおりです。

```
200a8c0:e0800417 ldw r2,16(fp)
```

キャプチャされたデータの保存と変換

Nios II プラグ・インでキャプチャされたデータは、SignalTap II のデータ・ログ機能を使用して保存することができます。Nios II プラグ・インのデータ・セットは、プロセッサの **.elf** ファイル情報とともに保存されます。データ・ログをイネーブルするには、SignalTap II ウィンドウで **Data Log** オプションをオンにします。

Nios II プラグ・インは、SignalTap II のデータ変換機能もサポートしています。キャプチャされたデータをエクスポートするには、File メニューの **Export** をクリックして、**File Name**、**Export Format**、および **Clock Period** を指定します。

チュートリアル：Nios II プラグ・インの使用

このチュートリアルでは、SignalTap II ロジック・アナライザ用の Nios II プラグ・インを使用して Nios II 開発ボード上で RS-232 文字の伝送をトレースする方法を示します。このチュートリアルでは、Nios II プラグ・インのセットアップとコンフィギュレーションの方法、Nios II ソフトウェア・デバッグ・サイクル中のプラグ・インの使い方（Nios II IDE）、および **.elf** ファイルをリファレンスとして使用してキャプチャしたデータを解析する方法を説明します。

ハードウェアおよびソフトウェア要件

このチュートリアルを実行するには、以下のものが必要です。

- Quartus II ソフトウェア v8.0 以上 — デザイン例では、Quartus II Web Edition とフル・ライセンス・バージョンの両方が動作します。
- Nios II Embedded Development Suite（EDS）v8.0 以降
- Nios II 開発キット —**full_featured** デザイン例をサポートする任意のアルテラ Nios II 開発キットを使用することができます。以下の開発キットがこのデザインをサポートしています。
  - Stratix® II Edition
  - Cyclone® II Edition
- RS-232 UART ケーブル — チュートリアルは、このケーブル無しで実行できます。ただし、ケーブルが無い場合、Nios II 開発ボードからの出力は Nios II IDE コンソールに表示されません。
- ソフトウェア・ファイル —**signal_tap_test.c** ファイルを、使用中のハード・ドライブのフォルダにダウンロードします。このソフトウェア・ファイルは、アルテラの資料ページ（www.altera.com/literature/lit-nio2.jsp）に、このアプリケーション・ノートと一緒に掲載されています。
- JTAG ByteBlaster または USB-Blaster ケーブル

ハードウェア・プロジェクトの設定

最初に、Nios II プロセッサを含むシステム・ハードウェアを作成し、Nios II プラグ・イン用に Quartus II プロジェクトをコンフィギュレーションする必要があります。

開発ボードのセットアップ

使用している Nios II 開発ボードの電源が入っており、JTAG ダウンロード・ケーブルを通してワークステーションに接続されていることを確認します。さらに、使用中の Nios II 開発ボードの RS-232 UART ポートがワークステーションの使用中のシリアル・ポートに接続されていることを確認します。

Nios II 開発ボードがシリアル・ケーブルによって使用中のワークステーションに接続されていることが理想的ですが、このチュートリアルは開発ボード無しでも実行できます。しかし、シリアル・ケーブルが無いと、ホスト・ワークステーションで受信中の文字を見ることができません。

Quartus II ソフトウェアのコンフィギュレーション

Quartus II ソフトウェアをコンフィギュレーションするには、以下の手順を実行します。

1. 使用しているボード用の Nios II full_featured のデザイン例を、編集することができる場所にコピーします。デザイン例は以下のディレクトリに置かれています。

   $SOPC_KIT_NIOS2/examples/verilog/
   *<Nios2 ボード・タイプ>***/full_featured**

2. Quartus II ソフトウェアを起動します。

3. **File** メニューの **Open Project** をクリックします。

4. **full_featured** デザイン例をコピーした場所に移動します。これがユーザーのプロジェクト・ディレクトリです。

5. *<Nios2 ボード・タイプ>*_**full_featured.qpf** プロジェクト・ファイルを選択します。

6. **Open** をクリックします。

Nios II ハードウェアの生成

Quartus II ソフトウェアでデザイン例を正しくコンパイルするには、SOPC Builder を使用して **full_featured** デザイン例を再生成する必要があります。

1. Tools メニューの **SOPC Builder** をクリックします。SOPC Builder が表示されます。

2. SOPC Builder で、**Generate** をクリックします。この操作によって、**full_featured** デザイン例のためにハードウェアが再構築されます。

   
   この操作は完了するのに数分かかることがあります。

3. 生成が完了したら、**Exit** をクリックします。

Nios II ソフトウェアのセットアップ

これで、Nios II コマンド・シェルを使用してソフトウェア・アプリケーションをコンパイルし、**.elf** および **objdump** ファイルを生成することができます。ファイルが生成されると、Nios II プラグ・インで使用するトリガ位置を特定できます。

Nios II ソフトウェアのビルド

ソフトウェア・アプリケーションをビルドし **objdump** ファイルを生成するには、以下の手順を実行します。

1. Nios II Command Shell を開きます。
2. プロジェクト・ディレクトリ、つまり新しくコピーした **full_featured** デザイン例のディレクトリに移動します。
3. 以下のコマンドを入力して、後でソフトウェア・ファイルを置く新しいディレクトリを作成します。

   ```
   mkdir software software/app software/bsp ↵
   ```

4. C ソース・ファイル **signal_tap_test.c** を **software/app** ディレクトリにコピーします。
5. 新しいディレクトリ **software/bsp** に移動します。
6. 以下のコマンドを入力して、ボード・サポート・パッケージ（BSP）を作成します。

   ```
   nios2-bsp hal . ../.. ↵
   ```

7. 新しいディレクトリ **software/app** に移動します。
8. 以下のコマンドを入力して、アプリケーション・プロジェクトの makefile を作成します。

   ```
   nios2-app-generate-makefile \
       --bsp-dir ../bsp \
       --src-files ./signal_tap_test.c \
       --elf-name signal_tap_test.elf \
       --set CREATE_OBJDUMP 1 \
       --set OBJDUMP_INCLUDE_SOURCE 1 ↵
   ```

   CREATE_OBJDUMP が 1 に設定されている場合は、プロジェクトのビルドによって **objdump** ファイルが生成されます。

9. プロジェクトをビルドするには、以下のコマンドを入力します。

```
make ↵
```

この手順によって **.elf** および **objdump** ファイルが作成されます。

Nios II IDE へのソフトウェア・プロジェクトのインポート

Nios II IDE でプロジェクトをデバッグするには、まずプロジェクトをインポートする必要があります。Nios II IDE にソフトウェア・プロジェクトをインポートするには、以下の手順を実行します。

1. Nios II IDE を開きます。
2. File メニューの **Import** をクリックします。**Import** ダイアログ・ボックスが表示されます。
3. **Altera Nios II** フォルダを展開します。
4. **Existing Nios II software build tools project or folder into workspace** をクリックします。
5. **Next** をクリックします。Import ウィザードが表示されます。
6. ユーザーの **software/app** ディレクトリに移動します。
7. **OK** をクリックします。Import ウィザードにより、プロジェクト名とパスのフィールドに値が入力されます。
8. **Project name** フィールドに `signal_tap_test` を入力して、プロジェクト名を signal_tap_test に変更します。
9. **Finish** をクリックします。

命令トリガ条件の検索

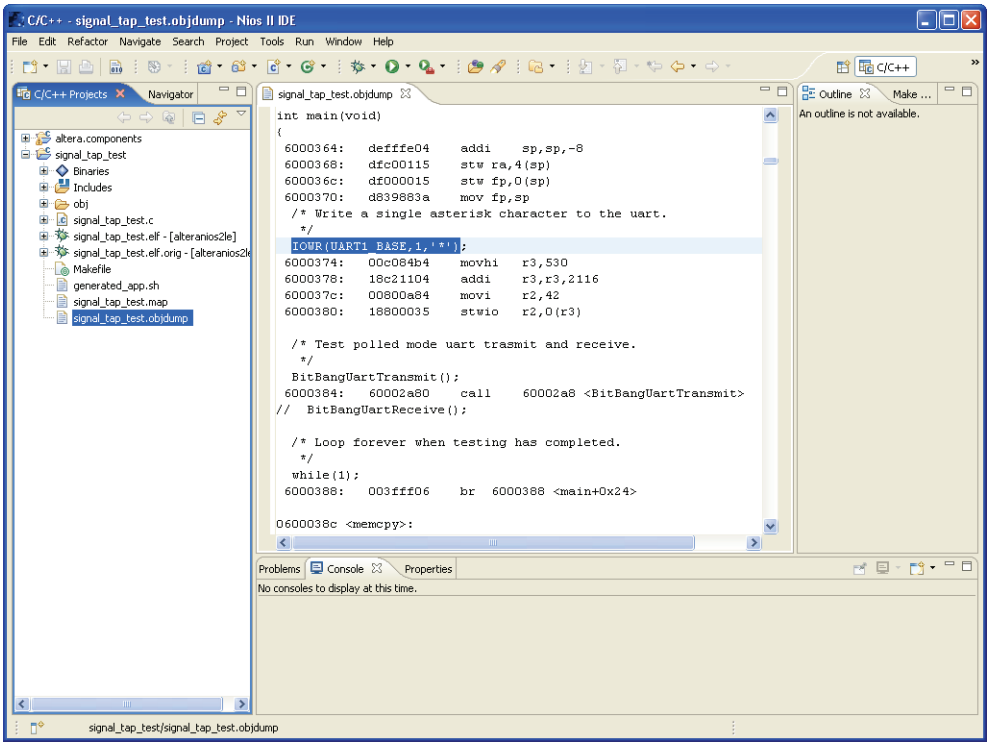
Nios II プラグ・インは、`main()` 関数へのエントリ・ポイントの直後、UART ペリフェラルへの最初の書き込み時にトリガします。UART の送信レジスタに書き込むアセンブリ命令のアドレスを特定するには、ソフトウェアのコンパイル中に生成された **objdump** ファイルを使用します。

objdump ファイルのフォーマットは、コンパイルした文字列化したプログラムを含んでいます。このファイルには、C 命令とそれに対応するアセンブリ命令、および Nios II プロセッサのアドレス空間内でのそれらの位置が示されています。UART の送信レジスタへの書き込み動作を行う

C 命令を見つけ、次にその C 命令に対応するアセンブリ命令を調べることによって、UART の送信レジスタへの書き込み動作を行うアセンブリ命令のアドレスを取得することができます。以下の手順を実行します。

1. **Nios II C/C++ Projects** タブ内の **signal_tap_test** フォルダを展開します。
2. **signal_tap_test.objdump** ファイルを右クリックし、**Open** をクリックします。
3. Edit メニューの **Find/Replace** をクリックします。
4. **Find/Replace** ダイアログ・ボックスの **Find** フィールドに `IOWR(UART1_BASE,1,'*')` を入力します。この文字列は、UART の送信レジスタへの書き込みを行う C 関数呼び出しです。
5. **Find** をクリックします。図 1 に示すように、`IOWR(UART_BASE,1,'*')` を含む行がハイライトされます。

図 1. Nios II の Objdump 表示



6. アセンブリ命令 `stwio r2,0(r3)` が含まれる行が見つかるまで数行下にスクロールして、その命令の左にあるアドレスを調べます。この命令は、UART の送信レジスタへの Nios II プロセッサによる書き込みに対応します。この例では、この命令のアドレスは 0x6000380 です。ただし、ユーザーの特定の場合には、このアドレスは異なることがあります。この命令のアドレスを Nios II プラグ・インへのトリガとして使用して、命令トレース・データのキャプチャを開始することができます。

この例では、Nios II プロセッサは、アセンブリ命令 `stwio r2,0(r3)` を実行するときにUART送信制御レジスタに書き込みを行い、次にUART送信制御レジスタは文字を送信します。書き込み命令は、以下のインジケータから特定できます。

- この命令は、C 命令 `IOWR(UART1_BASE,1,'*')` に対応するアセンブリ命令のブロック内にあります

- このブロックに含まれるその他のアセンブリ命令は、Nios II プロセッサのレジスタに対する操作のみを実行します。
- アセンブリ言語の `stwio` 命令は、ペリフェラルへの store word 操作を実行します。

アセンブリ言語の命令の詳細については、「Nios II プロセッサ・リファレンス・ハンドブック」の「命令セット・リファレンス」の章を参照してください。

ハードウェア・プロジェクトのコンフィギュレーション

次に、SignalTap II ロジック・アナライザで動作するように Nios II プラグ・インをコンフィギュレーションします。

full_featured ディレクトリ内に作成されたプロジェクトで、インクリメンタル・コンパイルは既にオフになっています。

Quartus II ソフトウェアで、以下の操作を実行します。

1. Processing メニューで、**Start** をポイントして **Start Analysis & Elaboration** をクリックします。

   この操作は完了するのに数分かかることがあります。

2. Tools メニューの **SignalTap II Logic Analyzer** をクリックします。

3. SignalTap II ノード・リスト内で右クリックします。**Add Nodes with Plug-In** をポイントして **Nios II** をクリックします。**Select Hierarchy Level** ダイアログ・ボックスが表示されます。

4. **Select Hierarchy Level** ダイアログ・ボックスで、**<***Nios2 ボード・タイプ***>_sopc_instance|cpu:the_cpu** を選択します。

5. **OK** をクリックします。**Plug-In Options** ダイアログ・ボックスが表示されます。

6. **Plug-In Options** ダイアログ・ボックスで、**Setting** テキスト・フィールドの横にあるブラウズ・ボタンをクリックします。

7. **Select File** ダイアログ・ボックスで、Nios II コマンド・シェルでコンパイルした **.elf** ファイルの位置に移動します。この **.elf** ファイルはユーザーの **software/app** ディレクトリ内にあり、**signal_tap_test.elf** という名前が付いているはずです。このファイルを選択します。

8. **Open** をクリックします。**Select File** ダイアログ・ボックスが閉じます。

9. **OK** をクリックします。**Plug-In Options** ダイアログ・ボックスが閉じます。

10. Nios II プラグ・インの **Trigger Conditions** 欄に、UART の送信レジスタへの書き込みに対応するアドレスを入力します。この例では、18 ページの図 1 に示すように、トリガ条件のアドレスは 0x6000380 です。

11. 以下の手順を実行して、RS-232 UART の送信データ信号をノード・リストに追加します。

    a. ノード・リストの枠内を右クリックし、**Add Nodes** をクリックします。

    b. **Look in** フィールドが |*<Nios2 ボード・タイプ>*_full_featured| に設定され、**Named** フィールドにはアスタリスク（*）が 1 つだけ入っていることを確認します。

    c. **Filter** フィールドが **SignalTap II: pre-synthesis** に設定されていることを確認します。

    d. **List** をクリックします。

    e. **Nodes Found** 枠内で、ノード `txd_from_the_uart1` をダブルクリックします。ノードが **Selected Nodes** 枠内に追加されます。

    f. **OK** をクリックします。

12. Signal Configuration 枠内で、**Clock** テキスト・フィールドの横にあるブラウズ・ボタンをクリックし、**Node Finder** ダイアログ・ボックスを表示させます。

13. **Node Finder** ダイアログ・ボックスで、以下の手順を実行します。

    a. **Look in** テキスト・フィールドの横にあるブラウズ・ボタンをクリックします。**Select Hierarchy Level** ダイアログ・ボックスが表示されます。

    b. **Select Hierarchy Level** ダイアログ・ボックスで、*<Nios2 ボード・タイプ>*_full_featured_sopc_instance の下にある階層リストを展開します。

c. 展開した階層リストで、エンティティ **pll:the_pll** をクリックします。

d. **OK** をクリックして、リストを閉じます。

e. **Filter** フィールドが **SignalTap II: pre-synthesis** に設定されていることを確認します。

f. Node Finder で、**List** をクリックして、すべてのノードのリストを表示させます。

g. ノード **c0** をダブルクリックして、**Selected Nodes** フィールドに追加します。

h. **OK** をクリックします。**Node Finder** ダイアログ・ボックスが閉じます。

14. Signal Configuration 枠内で、**Sample Depth** フィールドを 256 に設定します。

15. Quartus II ウィンドウで、Processing メニューの **Start Compilation** をクリックして、このプロジェクト用にハードウェアをビルドします。SignalTap II インスタンスを保存し、このプロジェクトのために SignalTap II ロジック・アナライザをイネーブルするように促すメッセージが表示されたら、**Yes** をクリックします。

.stp ファイルは、ユーザーのプロジェクト・ディレクトリに保存しなければなりません。

Quartus II のハードウェア・デザインは、コンパイルするのに数分以上かかる場合があります。

トレース・キャプチャ・セッションの実行

次のステップは、以下の手順を実行することにより、生成されたシステムを使用して Nios II プラグ・インでトレース・キャプチャ・セッションを実行することです。

1. SignalTap II ウィンドウの JTAG Chain Configuration 枠内で、以下の手順を実行します。

a. Hardware メニューで、使用している Nios II 開発ボードに接続されたプログラミング・ケーブルを選択します。

b. **SOF Manager** フィールド内で、ブラウズ・ボタンをクリックします。

c. **Select Programming File** ダイアログ・ボックスで、ユーザーのプロジェクト・ディレクトリに移動し、**.sof** ファイル *<Nios2 ボード・タイプ>***_full_featured.sof** を選択します。

d. **Open** をクリックします。これで **Program Device** ボタンが使用できる状態になります。

e. **Program Device** ボタンをクリックして、**.sof** ファイルを FPGA にダウンロードします。 

2. SignalTap II ウィンドウの Instance Manager 枠内で、**Run Analysis** ボタンをクリックして、ロジック・アナライザのキャプチャ・セッションを開始します。 

   解析は、Instance Manager 枠内の **Status** が **Waiting for trigger** に設定された状態で停止するはずです。

3. Nios II IDE で、Projects リスト内のプロジェクト **signal_tap_test** を選択します。

4. Nios II IDE の Run メニューの **Debug** をクリックします。**Debug** ダイアログ・ボックスが表示されます。

5. **Debug** ダイアログ・ボックスで、以下の手順を実行します。

   a. 左側枠内の **Nios II Hardware** をクリックします。

   b. **New launch configuration** ボタンをクリックして、新しいデバッグ起動コンフィギュレーションを作成します。 

   c. **Project** テキスト・フィールドに `signal_tap_test` を入力します。

   d. Target hardware 枠内で、**Browse** ボタンをクリックします。

   e. **SOPC Builder System PTF File** ダイアログ・ボックスで、ユーザーのプロジェクト・ディレクトリに移動し、*<Nios2 ボード・タイプ>***_full_featured_sopc.ptf** を選択します。

   f. **Open** をクリックします。

g. **Target Connection** タブで、**Nios II Terminal communication device** を **uart1** に設定し、使用中の開発ボードに接続されているシリアル・ポートについての正しい設定が **Host COM port** フィールドに入っていることを確認します。

このステップは、シリアル・ポートを通じて入ってくる文字を見る場合に必要です。このステップを省略した場合、SignalTap II ロジック・アナライザはデータをキャプチャすることはできますが、文字は Nios II IDE のコンソール・ウィンドウに表示されません。

h. **Debugger** タブの **Breakpoints at Start-up** フィールドで、**Break at program entry point** オプションをオンにします。その他のオプションはオフにします。

i. **Apply** をクリックし、続いて **Debug** をクリックします。**.elf** ファイルがボードにダウンロードされ、その後デバッガが起動します。

j. **Confirm Perspective Switch** ダイアログ・ボックスで、**Yes** をクリックします。

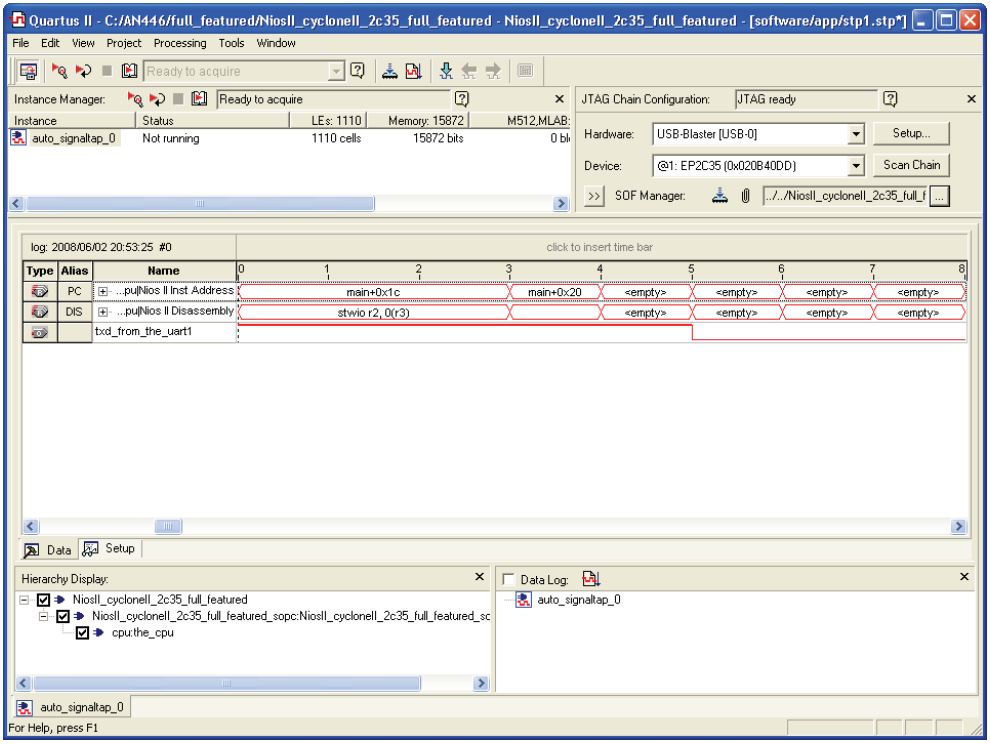
6. Nios II IDE の Debug 枠内で、**Resume** ボタンをクリックし、Nios II プロセッサの実行を開始します。 

UART ポートが接続されている場合、以下のメッセージがコンソール・ウィンドウに表示されます。

```
*BIT BANG
```

7. SignalTap II ウィンドウで、Nios II プラグ・インのトリガ条件が満たされたのでキャプチャ・セッションが終了したことを確認します。SignalTap II ウィンドウは、図 2 のようになっているはずです。

図 2. SignalTap II キャプチャ後のウィンドウ



キャプチャしたデータの解析

ここで、Nios II プラグ・インによりキャプチャしたデータを解析し、Nios II **.elf** ファイルの **objdump** と比較することができます。Nios II プラグ・インからのデータ・サンプルは、**signal_tap_test.objdump** ファイルにリストされているアセンブリ言語プログラムに正確に対応しているはずです。

図 2 は、以下の結果を示したものです。

- Nios II プラグ・インによってキャプチャされた 0 番目のサンプルは、ユーザーが設定したトリガ条件と一致し、命令 `stwio r2, 0(r3)` をキャプチャしています。

- RS-232 UART の送信データ信号は、stwio 命令の 2 クロック・サイクル後に始まります。この命令は Nios II IDE による UART 送信レジスタへの書き込みに対応しています。SignalTap II ロジック・アナライザは、UART がアスタリスク文字の送信を完了できる前に、そのサンプル・バッファが満杯になっているため、UART の完全な 8 ビット・シリアル送信シーケンスはキャプチャされません。8 ビット文字シーケンスの最初のビットだけがキャプチャされます。

結論

FPGA デザインは、エンベデッド・プロセッサ、ペリフェラル、バス、およびブリッジの数の増加とともに、サイズおよび複雑さが増加し続けており、設計者は、システム開発時間の短縮のために、より包括的なデバッグ・ツールを必要としています。SignalTap II エンベデッド・ロジック・アナライザ用の Nios II プラグ・インによって、Nios II プロセッサのプログラム実行をキャプチャすることが可能になります。Nios II プラグ・インは SignalTap II ロジック・アナライザと一緒に動作するため、Nios II プロセッサからの命令トレースは、他のハードウェア・イベントとともにキャプチャされます。プロセッサ命令の表示によって、システム・デザイン上の問題を効率的に見つけることができます。

参考資料

このアプリケーション・ノートでは、以下のドキュメントを参照しています。

- 「Quartus II ハンドブック volume 3」の「SignalTapII エンベデッド・ロジック・アナライザを使用したデザインのデバッグ」の章
- 「Nios II ソフトウェア開発ハンドブック」
- 「Nios II プロセッサ・リファレンス・ハンドブック」の「Nios II コア実装の詳細」の章
- 「Nios II プロセッサ・リファレンス・ハンドブック」の「Instruction Set Reference」の章

改訂履歴

表 1 に、このアプリケーション・ノートの改訂履歴を示します。

表 1. 改訂履歴

日付およびドキュメント・バージョン	変更内容	概要
2008 年 6 月 v1.2	この改訂版には以下の変更が含まれます。 • 多数の Nios II IDE スクリーンショットの削除を含め、Nios II ソフトウェア・ビルド・フローの「チュートリアル：Nios II プラグ・インの使用」の項を更新。 • カウント・バイナリのデザイン例を signal_tap_test 例に置き換え。	Quartus II ソフトウェアおよび Nios II EDS v8.0 のドキュメントを更新。
2007 年 10 月 v1.1	この改訂版には以下の変更が含まれます。 • **bit_bang_uart.c** へのすべての参照を **signal_tap_test.c** に置き換え。 • 関連するデザイン・ファイル **bit_bang_uart.c** をファイル **signal_tap_test.c** に置き換え。	Quartus II ソフトウェアおよび Nios II EDS v7.2 のドキュメントを更新。
2007 年 5 月 v1.0	初版	—

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support:
www.altera.com/support/