Nios® II

Nios II フラッシュ・プログラマ

ユーザガイド

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

この資料は英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。こちらの日本語版は参考用としてご利用ください。設計の際には、最新の英語版で内容をご確認ください。

UG-NIOSIIFLSHPROG-1.4

目次

付録 D. トラブルシューティング

このユーザガイドについて

改訂履歴

以下の表にこのユーザガイドの章の改訂履歴を示します。

Nios II フラッシュ・プログラマ・ユーザガイドの改訂履歴

章	日付	バージョン	変更内容
第 1 章	2007 年 5 月	7.1	• ボードの説明ファイルの記述の削除（提供中止） • `--instance` および`--device` コマンドライン・パラメータの説明の修正と更新 • SOPC Builder スクリーン・ショットの更新
第 3 章	2007 年 5 月	7.1	**nios2-flash-programmer** オプション `--sidp`、`--id`、`--timestamp`、`--accept-bad-sysid` の記述の追加
付録 C	2007 年 5 月	7.1	欠落したインストール手順の訂正
付録 D	2007 年 5 月	7.1	ボードの説明ファイルの記述の削除（提供中止）
すべて	2006 年 11 月	6.1	Nios II バージョン 6.1 のリリースにより更新。フラッシュ・プログラマユーザ・インタフェースの改良点を追記
すべて	2005 年 10 月	5.1	Nios II バージョン 5.1 のリリースにより更新。フラッシュ・プログラマのターゲット・デザインの主な変更点を追記
すべて	2004 年 12 月	1.1	Nios II バージョン 1.1 のリリースにより更新。
すべて	2004 年 5 月	1.0	Nios II 開発ボードのフラッシュ・プログラマユーザ・ガイドの初版

アルテラへのお問い合わせ

アルテラ製品に関する最新情報は、アルテラのウェブサイト、www.altera.co.jp をご覧ください。テクニカル・サポートについては、www.altera.co.jp/mysupport にアクセスしてください。また、アルテラの販売代理店にもお問い合わせいただけます。

表記規則

本書では、以下の表記規則を使用しています。

書体	意味
太字かつ文頭が大文字	コマンド名、ダイアログ・ボックス・タイトル、チェックボックス・オプション、およびダイアログ・ボックス・オプションは、太字かつ文頭が大文字で表記されています。例 : **Save As** ダイアログ・ボックス
太字	外部タイミング・パラメータ、ディレクトリ名、プロジェクト名、ディスク・ドライブ名、ファイル名、ファイルの拡張子、およびソフトウェア・ユーティリティ名は、太字で表記されています。 例 : **f_{MAX}**, **\qdesigns** ディレクトリ、**d:** ドライブ、**chiptrip.gdf** ファイル
斜体かつ文頭が大文字	資料のタイトルは、斜体かつ文頭が大文字で表記されています。 例 : *AN 75: High-Speed Board Design*
斜体	内部タイミング・パラメータおよび変数は、斜体で表記されています。 例 : t_{PIA}, *n* + 1 変数は、山括弧 (< >) で囲み、斜体で表記されています。 例 : < *ファイル名* >、< *プロジェクト名* >**.pof** ファイル
文頭が大文字	キーボード・キーおよびメニュー名は、文頭が大文字で表記されています。 例 : Delete キー、Options メニュー
「小見出しタイトル」	資料内の小見出しおよびオンライン・ヘルプ・トピックのタイトルは、鉤括弧で囲んでいます。例 : 「表記規則」
`Courier フォント`	信号およびポート名は、Courier フォントで表記されています。 例 : `data1`、`tdi`、`input`。アクティブ Low 信号は、サフィックス `n` で表示されています (例 : `resetn`)。 表示されているとおりに入力する必要があるものは、Courier フォントで表記されています (例 : `c:\qdesigns\tutorial\chiptrip.gdf`)。また、Report ファイルのような実際のファイル、ファイルの構成要素 (例 : AHDL キーワードの `SUBDESIGN`)、ロジック・ファンクション名 (例 : `TRI`) も Courier フォントで表記されています。
1.、2.、3. および a.、b.、c. など	手順など項目の順序が重要なものは、番号が付けられリスト形式で表記されています。
■ ● •	箇条書きの黒点などは、項目の順序が重要ではないものに付いています。
✓	チェックマークは、1 ステップしかない手順を表します。
☞	指差しマークは、要注意箇所を表しています。
CAUTION	CAUTION マークは、特別な配慮および理解が必要であり、手順またはプロセスを始める前、または続ける際に確認すべき情報を示しています。
⚠	注意マークは、手順またはプロセスを始める前、または続ける際に確認すべき情報を示しています。
↵	矢印は、Enter キーを押すことを示しています。
👣	足跡マークは、詳細情報の参照先を示しています。

1. Nios II フラッシュ・プログラマの概要

はじめに

Nios® II フラッシュ・プログラマの目的は、アルテラの FPGA に接続されたフラッシュ・メモリ・デバイスにデータをプログラムすることです。フラッシュ・プログラマは、ファイルの内容を USB-Blaster® などのアルテラ・ダウンロード・ケーブルを使用して、FPGA 上で動作している Nios II システムに送信します。Nios II プロセッサを含む多くのハードウェア・デザインでは、FPGA コンフィギュレーション・データや Nios II プログラム・データを保存する手段としてフラッシュ・メモリもボード上に搭載されています。Nios II フラッシュ・プログラマは、Nios II 開発ツールの一部であり、このメモリをプログラミングする手軽な手段です。

Nios II フラッシュ・プログラマは、3 種類の内容をフラッシュ・メモリにプログラムできます。

- Nios II ソフトウェア実行可能ファイル－多くのシステムは不揮発性プログラム・コードまたはファームウェアを保存するのにフラッシュ・メモリを使用します。Nios II システムはフラッシュ・メモリからブートできます。
- FPGA コンフィギュレーション・データ－システムのパワーアップ時に、ボード上の FPGA コンフィギュレーション・コントローラはフラッシュ・メモリから FPGA コンフィギュレーション・データを読み込むことができます。コンフィギュレーション・コントローラのデザインに応じて、フラッシュ・メモリに保存された複数の FPGA コンフィギュレーション・ファイルから選択できる場合があります。
- その他の任意のデータ・ファイル－ Nios II フラッシュ・プログラマは、目的を問わず、バイナリ・ファイルをフラッシュ・メモリの任意のオフセットにプログラムできます。例えば、Nios II プログラムはこのデータを係数テーブルまたはサイン・ルックアップ・テーブルとして使用する場合があります。

フラッシュ・プログラマを使用して、以下のタイプのメモリをプログラムすることができます。

- コモン・フラッシュ・インタフェース（CFI）準拠のフラッシュ・メモリ － CFI はフラッシュ・メモリ・デバイスに、ベンダに依存しない共通インタフェースを提供する業界規格です。
- アルテラの EPCS（Erasable Programmable Configurable Serial）デバイス －アルテラ EPCS シリアル・コンフィギュレーション・デバイスは、FPGA コンフィギュレーション・データと Nios II 実行可能ソフトウェアを格納できます。

CFI 仕様について詳しくは、**www.intel.com/design/flash/swb/cfi.htm** を参照してください。EPCS デバイスについて詳しくは、「シリアル・コンフィギュレーション・デバイス (EPCS1、EPCS4、EPCS16、EPCS64 および EPCS128) データシート」および「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」の「Avalon インタフェース対応 EPCS デバイス・コントローラ・コア」の章を参照してください。

この資料では、「フラッシュ・メモリ」という用語は、特に記述がない限り CFI と EPCS の両方のメモリ・デバイスを指します。

前提条件

このユーザガイドは、Nios II ハードウェアとソフトウェアの開発フローを十分に理解しているユーザを対象としています。以下のチュートリアルの内容を把握していなければなりません。

- Nios II ハードウェア開発チュートリアル
- Nios II ソフトウェア開発チュートリアル。Nios II 統合開発環境（IDE）ヘルプ・システムで利用可能

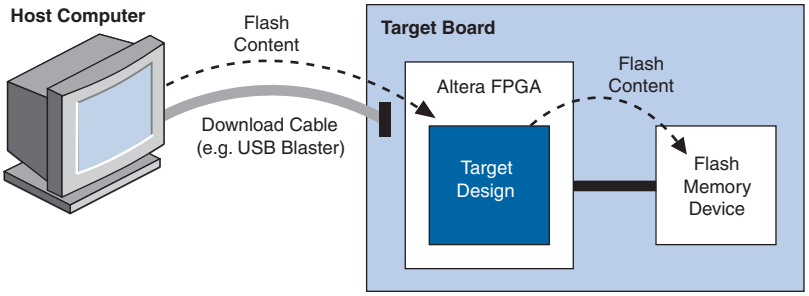
Nios II フラッシュ・プログラマを使用して、FPGA コンフィギュレーション・データをフラッシュ・メモリにプログラムする場合、ボードで使用されているコンフィギュレーション方法も理解しておく必要があります。

「AN346: Using the Nios Development Board Configuration Controller Reference Design」または特定の Nios II 開発ボードのリファレンス・マニュアル参照してください。

フラッシュ・プログラマの動作

図 1-1 に示すように、フラッシュ・プログラマにはホストとターゲットの 2 つの部分があります。ホスト部分はコンピュータ上で実行されます。ホストはダウンロード・ケーブルを使用してフラッシュ・プログラミング・ファイルをターゲットに送信します。ターゲット部分は FPGA で実行されるハードウェア・デザインです。ターゲットはホストが送信したプログラミング・データを受け入れて、フラッシュ・メモリ・デバイスにデータを書き込みます。Nios II フラッシュ・プログラマで作業するには、FPGA デザインが特定の要件を満たしている必要があります。1–3 ページの「フラッシュ・プログラマ・ターゲット・デザイン」を参照してください。

図 1-1. Nios II フラッシュ・プログラマの動作



IDE およびコマンドライン・モード

Nios II フラッシュ・プログラマは、次の 2 つのモードのいずれかで実行できます。

- IDE モード − Nios II IDE は、使いやすいフラッシュ・プログラマ機能へのインタフェースを提供しています。IDE モードは、ほとんどのフラッシュ・プログラミングのニーズに適しています。
- コマンドライン・モード− 上級ユーザ向けのコマンドライン・モードは、フラッシュ・プログラマ機能を完全に制御します。コマンドラインフラッシュ・プログラマユーティリティは、Nios II Command Shell などのコマンド・シェルから実行できます。一部のパラメータは手動で計算する必要があります。

この資料では、「Nios II フラッシュ・プログラマ」および「フラッシュ・プログラマ」という用語は、特に記述がない限り、IDE モードとコマンドライン・モードの両方を指します。

フラッシュ・プログラマ・ターゲット・デザイン

Nios II フラッシュ・プログラマを使用するには、有効なフラッシュ・プログラマ・ターゲット・デザインを用意しておく必要があります。ターゲット・デザインには、最低でも表 1–1 に示す SOPC Builder コンポーネントを含む、SOPC Builder システムが含まれます。

ホストで Nios II フラッシュ・プログラマを実行するには、FPGA でターゲット・デザインが実行されていなければなりません。

最小コンポーネント・セット

最小コンポーネント・セットには、ターゲット・デザインがホストと通信し、フラッシュ・メモリに書き込むための機能があります。最小コンポーネント・セットは、プログラムするフラッシュ・メモリのタイプによって異なります。表 1–1 に最小コンポーネント・セットをリストします。

表 1–1. フラッシュ・プログラマ・ターゲット・デザインの最小コンポーネント・セット

コンポーネント	プログラムするフラッシュ・メモリ		
	CFI のみ	**EPCS のみ**	**CFI および EPCS**
Nios II プロセッサ、JTAG（Joint Text Action Group）デバッグ・モジュール・レベル 1 以上を含む	必須	必須	必須
システム **ID** ペリフェラル	推奨される (1)	推奨される (1)	推奨される (1)
Avalon-MM トライステート・ブリッジ	必須		必須
フラッシュ・メモリ（コモン・フラッシュ・インターフェイス）	必須 (2)		必須 (2)
EPCS シリアル・フラッシュ・コントローラ		必須	必須

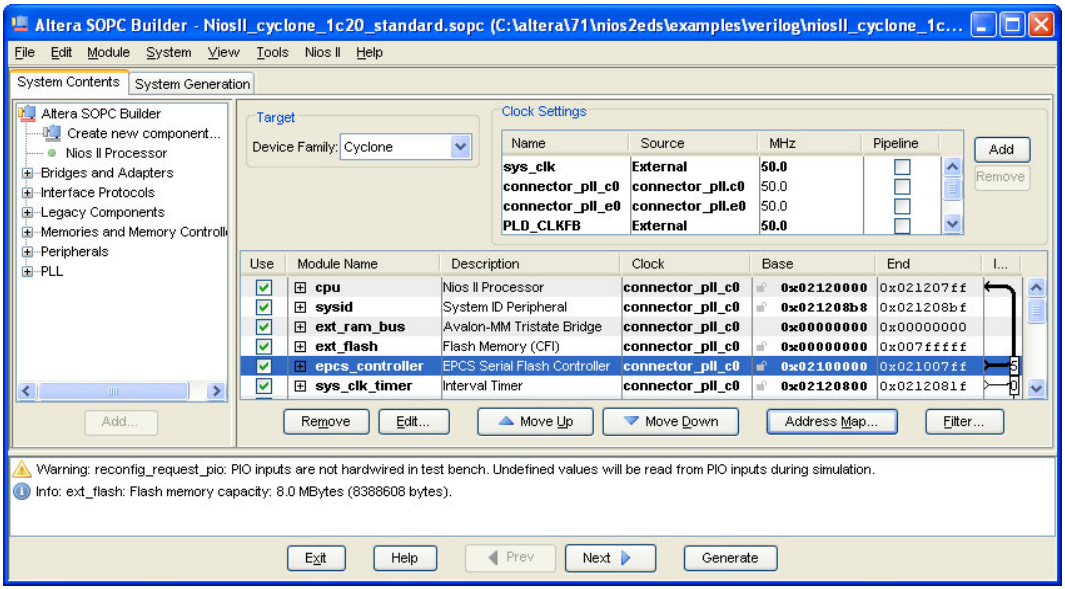
表 1–1 の注：

(1) システム **ID** ペリフェラル・コンポーネントが存在する場合、フラッシュ・プログラマでターゲット・デザインを検証してから、フラッシュ・メモリをプログラムすることができます。

(2) システムには複数の CFI フラッシュ・メモリを搭載できます。システムは、各フラッシュ・メモリに対してボード上に 1 個のフラッシュ・メモリ（コモン・フラッシュ・インターフェイス）コンポーネントを搭載していなければなりません。

図 1-2 に、1 個の CFI フラッシュ・メモリと EPCS シリアル・コンフィギュレーション・デバイスを搭載したシステムに対する最小コンポーネント・セットを含む、SOPC Builder システムの例を示します。システムには、フラッシュ・プログラマ以外のシステムの目的に関連する他のコンポーネントも含まれています。

図 1-2. 最小コンポーネント・セットを含むターゲット・デザインの例



Nios 開発ツールに付属する **full_featured** または **standard** ハードウェア・デザイン例は、アルテラ開発ボードで動作する既製のターゲット・デザインです。カスタム・ボードを開発する場合は、これらのデザイン例のいずれかを最初のターゲット・デザインを作成するための出発点として使用することを考慮してください。

2. IDE モードでのフラッシュ・プログラマの使用

Nios II 統合開発環境（IDE）は、フラッシュ・メモリのプログラミングのプロセスを自動化して、使いやすいグラフィカル・インターフェイスによるプログラミング・パラメータの制御を可能にします。IDE により、ソフトウェア、ハードウェア、バイナリ・データの任意の組み合わせを一度の操作でフラッシュ・メモリにプログラムできます。IDE モードは、Nios II フラッシュ・プログラマを使用する場合の推奨方法です。

コマンドライン・モードでのフラッシュ・プログラマの使用について詳しくは、第 3 章　コマンドライン・モードでのフラッシュ・プログラマの使用を参照してください。

フラッシュ・プログラマ ダイアログ・ボックス

Nios II IDE で Nios II フラッシュ・プログラマを開くには、まずフラッシュにプログラムするソフトウェア・プロジェクトをハイライトし、**Tools** メニューから **Flash Programmer** をクリックします。**Flash Programmer** ダイアログ・ボックスが表示されます。フラッシュを初めてプログラムする場合、ダイアログ・ボックスは図 2-1 のように表示されます。既存のフラッシュ・コンフィギュレーションが存在する場合は、2–2 ページの図 2-2 のように表示されます。

図 2-1. 最初に開いたときの Flash Programmer ダイアログ・ボックス

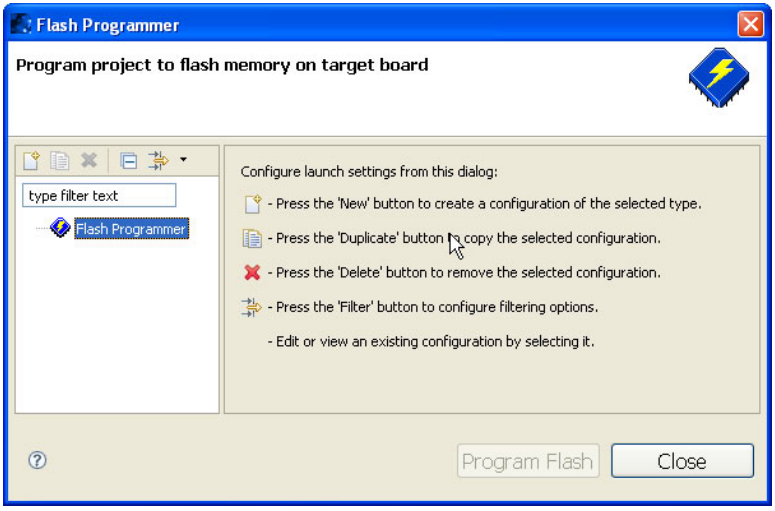
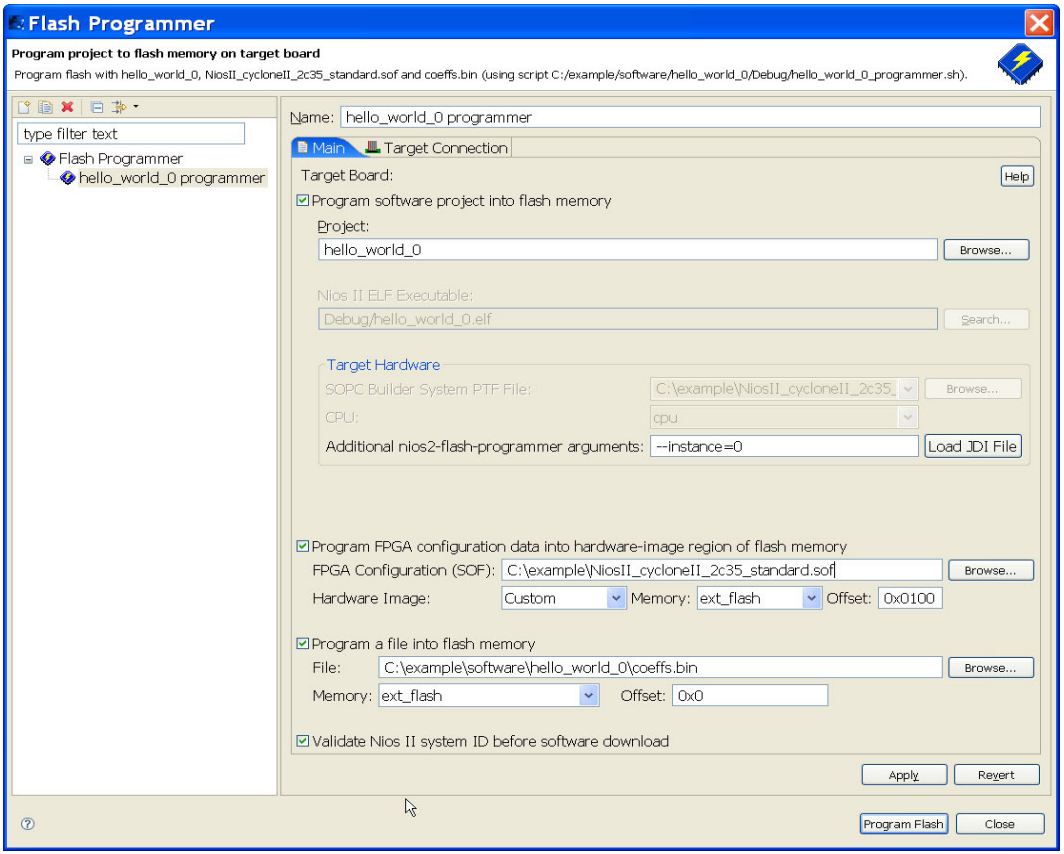


図 2-2. 既存のコンフィギュレーションの Flash Programmer ダイアログ・ボックス



フラッシュに書き込む前に、フラッシュ・コンフィギュレーションを開く必要があります。既存のフラッシュ・プログラマのコンフィギュレーションを使用できるかどうかを判断し、使用できない場合は新しいフラッシュ・プログラマコンフィギュレーションを作成する必要があります。以前にプロジェクトをフラッシュにプログラムし、特別なオプションを選択していた場合は、そのフラッシュ・コンフィギュレーションを再利用したいことがあるかも知れません。

新しいフラッシュ・プログラマ・コンフィギュレーションを作成する場合は、以下のステップを実行します。

1. ダイアログ・ボックスの左側で **Flash Programmer** を選択します。

2. 2–3 ページの図 2-3 に示す、**Flash Programmer** ウィンドウの左上隅にある **New launch configuration** ボタンをクリックします。Nios II IDE で、新しいフラッシュ・プログラミング・コンフィギュレーションが作成されます。

既存のフラッシュ・コンフィギュレーションを再利用する場合、フラッシュ・コンフィギュレーションの作成後に Quartus® II プロジェクトが再コンパイルされていれば、以下のステップを実行します。

√ **Load JDI File** をクリックします。JDI ファイルをロードすると、追加の **nios2-flash-programmer Arguments** ボックスに正しいインスタンス ID が格納されます。インスタンス ID について詳しくは、3–3 ページの表 3–2 を参照してください。

フラッシュ・コンフィギュレーションを使用してフラッシュに書き込むには、以下のステップを実行します。

1. Nios II IDE プロジェクトのソフトウェア、または Nios II IDE プロジェクトに関連する読み出し専用 Zip ファイル・システムを使用してフラッシュをプログラムする場合は、**Program software project into flash memory** のタイトルが付いたボックスをチェックします。

図 2-3. 新しいフラッシュ・プログラマ・コンフィギュレーションの作成

2. フラッシュ・プログラマに追加の引数を渡す場合は、**Additional nios2-flash-programmer arguments**のタイトルが付いたフィールドに入力します。

3. フラッシュに FPGA コンフィギュレーション・データをプログラムする場合、**Program FPGA configuration data into hardware-image region of flash memory** のタイトルが付いたボックスをチェックします。

   a. **FPGA Configuration (SOF)** フィールドに、プログラムする SRAM オブジェクト・ファイル（**.sof**）を入力するか参照して指定します。

   b. **Hardware Image** フィールドで、SRAM オブジェクト・ファイルをプログラムする既定の場所を選択するか、**Custom** を選択します。**Custom** を選択した場合、メモリ名およびそのメモリ内のオフセット（バイト）も指定しなければなりません。

4. フラッシュに任意のバイナリ・ファイルをプログラムする場合は、**Program a file into flash memory** のタイトルが付いたボックスをチェックします。

   
   プログラムするファイル、フラッシュ・メモリ名、およびオフセットを指定する必要があります。

   Nios II EDS には、フラッシュ・メモリ内のデータの格納およびアクセスのための使いやすいツールである、アルテラ Zip 読み出し専用ファイル・システム・ソフトウェア・コンポーネントも付属しています。使用するアプリケーションに応じて、フラッシュ・メモリにロウ・バイナリ・データを格納するよりも Zip 読み出し専用ファイル・システムを使用する方が便利な場合もあります。


詳細については、Nios II IDE ヘルプ・システムの Zip 読み出し専用ファイル・システムのトピックを参照してください。

5. **Program Flash** をクリックします。IDE が指定されたすべてのファイルをフラッシュ・メモリにプログラムするのに必要な一連の動作を実行します。


Flash Programmer ダイアログ・ボックスのコントロールの説明については、Nios II IDE ヘルプ・システムを参照してください。**Flash Programmer** ダイアログ・ボックスの右上隅にある **Help** をクリックして、ヘルプ・システムを開きます。

ターゲット・デザインがシステム ID コンポーネントを持っている場合、IDE はフラッシュ・メモリのプログラムを試みる前に、期待したシステム ID 値を備えたシステムが FPGA で動作中かどうかを検証します。期待したシステムが動作中でない場合、フラッシュ・プログラマはフラッシュ・メモリのプログラムを中断します。ターゲット・システムにシステム ID コンポーネントが存在しない場合、IDE はこのチェックをスキップします。

システム ID に関係なく、FPGA でコンフィギュレーションされたハードウェア・デザインが有効なフラッシュ・プログラマのターゲット・デザインではない場合、フラッシュ・メモリをプログラムすることはできません。したがって、C/C++ アプリケーション・プロジェクトのための Nios II ハードウェア・システムは、有効なフラッシュ・プログラマのターゲット・システムでなければならず、最低でも 1–4 ページの表 1–1 に指定される最小コンポーネント・セットを備えている必要があります。

3. コマンドライン・モードでのフラッシュ・プログラマの使用

Nios II 開発ツールは、Nios II フラッシュ・プログラマの機能を完全に制御するための 4 つのコマンドライン･ユーティリティを備えています。フラッシュ・プログラミング作業を自動化するカスタム・スクリプト・ファイルを作成することができます。コマンドライン・モードでフラッシュ・プログラマを使用すると、IDE モードよりも細かく制御できますが、より複雑になります。可能な場合は、IDE モードを使用してフラッシュをプログラムすることを推奨します。

表 3–1 にコマンドライン・ユーティリティのリストを示します。

表 3–1. フラッシュ・プログラマのコマンドライン・ユーティリティ

`nios2-flash-programmer`	S レコード・ファイルをフラッシュ・メモリにプログラムします。またデータのリード・バック、データの検証、フラッシュ・チップに関するデバッグ情報の提供なども実行できます。
`sof2flash`	SRAM オブジェクト・ファイルを S レコード・ファイルに変換します。
`elf2flash`	Nios II 実行可能およびリンク・フォーマット・ファイル（**.elf**）を S レコード・ファイルに変換します。
`bin2flash`	任意のデータ・ファイルを S レコード・ファイルに変換します。

Nios II IDE はコマンドライン・ユーティリティに基づいてスクリプトを作成することによって、フラッシュをプログラムします。このスクリプトは、体裁よくまとめられ、プロジェクトに合わせてカスタマイズされていて、人間が読み取ることができます。このスクリプトは、フラッシュ・プログラマのコマンドライン構文のリファレンスとして使用できます。

IDE で作成されたスクリプトが特に役立つのは、`--instance` パラメータ（3–3 ページの表 3–2 に示す）を使用する必要がある場合です。

IDE を使用してフラッシュ・メモリを正常にプログラムすると、プロジェクトの **Debug** フォルダまたは **Release** フォルダの **C/C++ Projects** ビューに、Flash プログラマ・スクリプトが表示されます。フラッシュ・プログラマ・スクリプトは、**<** プロジェクト名 **>_programmer.sh** という名前が付けられた、拡張子 **.sh** を持つファイルです。**Flash Programmer** ダイアログ・ボックスには、2–2 ページの図 2-2 に示すように、スクリプトのフル・パスとファイル名が表示されます。

コマンドラインからフラッシュ・メモリをプログラムするための主なユーティリティは、**nios2-flash-programmer** です。業界標準 S レコード入力ファイルが必要になります。ファイル変換ユーティリティ **sof2flash**、**elf2flash**、および **bin2flash** は、**nios2-flash-programmer** 用の S レコード・ファイルを作成します。これらのユーティリティによって、入力とフラッシュ・プログラマとの互換性が保証されます。すべてのユーティリティの入力ファイル名には、**.elf** または **.flash** などの明示的な拡張子が付いていなければなりません。

Windows コンピュータでは、Nios II Command Shell を起動すると、デフォルトの検索パスでフラッシュ・プログラマ・ユーティリティを使用できます。

Nios II Command Shell について詳しくは、「Nios II ソフトウェア開発ハンドブック」の「アルテラの開発ツール」の章を参照してください。

以下の項では、ユーティリティとそれらの機能をリストしています。

nios2-flash-programmer

nios2-flash-programmer ユーティリティは、フォーマット済みのファイルを指定されたフラッシュ・メモリにプログラムします。この入力は通常、変換ユーティリティ **sof2flash**、**elf2flash**、または **bin2flash** のいずれかで作成された業界標準 S レコード・ファイルです。**nios2-flash-programmer** は入力に、任意の S レコード・ファイルを使用できますが、この S レコード・ファイルに指定されるアドレスがフラッシュ・メモリの先頭からのオフセットを表している必要があります。Nios II IDE は、拡張子 **.flash** を持つフラッシュ・プログラマ・ファイルを作成します。

nios2-flash-programmer ユーティリティは、ハードウェア・ターゲット・デザインの任意の CFI 互換フラッシュ・メモリまたは EPCS シリアル・コンフィギュレーション・デバイスのプログラミング、消去、読み出し機能を備えています。

nios2-flash-programmer コマンドライン構文は、次のとおりです。

```
nios2-flash-programmer [--help] [--cable=<cable name>] [--device=<device index>]\
    [--instance=<instance>] [--sidp=<address>] [--id=<id>] [--timestamp=<time>]\
    [--accept-bad-sysid] --base=<address>\ [--epcs] { <file> } [--go]
```

表 3–2 に、**nios2-flash-programmer** で一般的に使用されるパラメータをリストしています。

表 3–2. nios2-flash-programmer パラメータ （1 / 3）

名称	必要性	説明
汎用パラメータ		
`--cable=<cable name>`	ホスト・コンピュータに接続されるダウンロード・ケーブルが複数ある場合に必須。	使用するダウンロード・ケーブルを指定します。
`--device=<device index>`	JTAG チェイン内に複数のデバイスが存在する場合に必須。	JTAG チェイン内の FPGA のデバイス番号を指定します。デバイス・インデックスは、フラッシュ・プログラマが Nios II JTAG デバッグ・モジュールを検索するデバイスを指定します。JTAG デバイスには、JTAG チェインに対して 1 から始まる番号が付けられます。(1)
`--instance=<instance>`	FPGA のターゲット・デザイン内に、JTAG デバッグ・モジュールを持つ Nios II プロセッサが複数存在する場合に必須。	FPGA で検索する Nios II JTAG デバッグ・モジュールを指定します。インスタンス ID は、フラッシュ・メモリのプログラミングに使用される JTAG デバッグ・モジュールを指定します。(2)
`--sidp=<address>`	オプション。システム ID 検証の場合は必須。	システムのシステム ID コンポーネントのベース・アドレスを含みます。この値は 16 進数形式です（例：`0x01000000`）。(3)
`--id=<id>`	オプション。システム ID 検証の場合は必須。	システムのシステム ID コンポーネントにプログラムされた ID 値を含みます。この値は、SOPC Builder システムを再生成するたびにランダムに選択されます。この値は符号なし 10 進数形式です（例：`2056847728u`）。(4)
`--timestamp=<time>`	オプション。システム ID 検証の場合は必須。	システムのシステム ID コンポーネントにプログラムされたタイムスタンプ値を含みます。SOPC Builder はシステムの生成時間に基づいて、この値を設定します。この値は符号なし 10 進数形式です（例：`1177105077u`）。(5)

表 3–2. nios2-flash-programmer パラメータ （2 / 3）

名称	必要性	説明
--accept-bad-sysid	オプション。デフォルトはオフ。	システム ID 検証をバイパスするのに使用されます。フラッシュ・プログラマにフラッシュ・イメージのダウンロードを強制します。このパラメータをオンにすることは、Nios II IDE の **Validate Nios II System ID before software download** チェックボックスをオフにすることと同じです。
--erase=<*start*>,<*size*>	オプション。デフォルトはオフ。	フラッシュ・メモリの特定のバイト範囲を消去します。
--erase-all	オプション。デフォルトはオフ。	フラッシュ・メモリ全体を消去します。入力ファイルがプログラミングに提供される場合は、プログラミングの前に消去動作が実行されます。
--program	オプション。入力ファイルが指定される場合はデフォルトでオン。	入力ファイルからフラッシュ・メモリをプログラムします。
--no-keep-nearby	オプション。デフォルトはオフ。	セクタ・データの一部を破棄します。先頭または最終セクタがプログラムするデータで完全に埋まらない場合、フラッシュ・プログラマは通常はそれらのセクタの元のデータを保持して、再プログラムします。この機能は、--no-keep-nearby パラメータでディセーブルされます。このオプションはプログラミング・プロセスを高速化しますが、既存のフラッシュ・メモリ内容が重要でない場合にのみ使用してください。
--verify	オプション。デフォルトはオフ。	フラッシュ・メモリの内容が入力ファイルに一致することを確認します。
{ <*file*> }	オプション	プログラムまたは検証する入力ファイル名を指定します。複数のファイル名をスペースで区切ります。
--read=<*file*>	オプション。デフォルトはオフ。	フラッシュ・メモリの内容を指定されたファイルに読み出します。
--read-bytes=<*start*>,<*size*>	--read が指定されている場合はオプション。デフォルトはオフ。	読み出すアドレス範囲を指定します（バイト・アドレス）。
--go	オプション。デフォルトはオフ。	フラッシュ・メモリのプログラミングの完了後、リセット・ベクタからプロセッサを実行します。

表 3–2. nios2-flash-programmer パラメータ （3 / 3）

名称	必要性	説明
CFI パラメータ		
--debug	オプション。デフォルトはオフ。	フラッシュ・メモリのクエリ・テーブルを含むデバッグ情報をプリントします。
--base=<*address*>	必須	CFI フラッシュ・メモリのベース・アドレスを指定します。このパラメータはターゲット・デザインのアドレス空間における絶対アドレスです。**nios2-flash-programmer** は、S レコード・ファイル内のアドレスをベース・アドレスのオフセットとして扱います。
EPCS パラメータ		
--epcs	EPCS 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場合は必須。デフォルトはオフ。	ターゲット・フラッシュ・メモリを EPCS シリアル・コンフィギュレーション・デバイスとすることを指定します。
--debug	オプション。デフォルトはオフ。	EPCS デバイス内の物理メモリに関するデバッグ情報をプリントします。
--base=<*address*>	必須	EPCS デバイスのベース・アドレスを指定します。

表 3–2 の注：

(1) --device パラメータが必要なのは、異なるデバイスに同じインスタンス ID を持つプロセッサが複数存在する場合のみです。JTAG デバイス・インデックスを決定するには、**jtagconfig** を実行します。

(2) プロセッサのインスタンス ID の正しい値を見つける方法は 2 通りあります。最も簡単な方法は、Nios II IDE を使用してサンプル・フラッシュ・プログラミング・スクリプトを作成することです。詳細については、第 2 章 IDE モードでのフラッシュ・プログラマの使用を参照してください。あるいは、Quartus II プロジェクト・ディレクトリで、<Quartus II プロジェクト名 >**.jdi** を開きます。< プロセッサ・モジュール名 >= を含む hpath の値を探して、Nios II プロセッサ・ノードを特定します。インスタンス ID は instance_id として指定されます。

(3) システム・ライブラリまたはボード・サポート・パッケージ（BSP）の **system.h** では、システム ID ベース・アドレスは SYSID_BASE で指定されます。

(4) システム・ライブラリまたは BSP の **system.h** では、システム ID 値は SYSID_ID で指定されます。

(5) システム・ライブラリまたは BSP の **system.h** では、システム ID タイムスタンプは SYSID_TIMESTAMP で指定されます。

その他のパラメータについては、コマンドラインで nios2-flash-programmer --help と入力します。

nios2-flash-programmer コマンドラインの例

```
nios2-flash-programmer --cable="Usb-blaster [USB-0]" --base=0x200000\
    --program ext_flash.flash
```

ケーブル「Usb-blaster [USB-0]」を使用し、入力ファイル **ext_flash.flash** でアドレス 0x200000 をベースにする CFI フラッシュ・メモリをプログラムします。

```
nios2-flash-programmer --epcs --base=0x02100000 epcs_controller.flash
```

入力ファイル **epcs_controller.flash** を使用し、アドレス 0x02100000 をベースにする EPCS デバイスをプログラムします。

```
nios2-flash-programmer --base=0x200000 --read=current.srec --read-bytes=0,0x10000
```

アドレス 0x200000 をベースにする CFI フラッシュ・メモリから 0x10000 バイトを読み出し、ファイル **current.srec** に内容を書き込みます。

```
nios2-flash-programmer --base=0x200000 --erase=0x8000,0x10000
```

アドレス 0x200000 をベースにする CFI フラッシュ・メモリの 0x8000 ～ 0x10000 のアドレス範囲を消去します。

```
nios2-flash-programmer --base=0x200000 --debug
```

アドレス 0x200000 をベースにする CFI フラッシュ・メモリを問い合わせて、結果を報告します。このコマンドはフラッシュ・メモリのクエリ・テーブルをダンプします。

sof2flash

sof2flash ユーティリティは、SRAM オブジェクト・ファイルを取り込み、フラッシュ・メモリへのプログラミングに適したSレコード・ファイルに変換します。

表 3–3 に、**sof2flash** で使用される一般的なパラメータをリストします。

表 3–3. sof2flash パラメータ

名称	必要性	説明
汎用パラメータ		
`--compress`	オプション。デフォルトはオフ。	Stratix® II 圧縮ではオン。Stratix II FPGA 以外では使用不可。
`--input=<file>`	必須	入力 SRAM オブジェクト・ファイルの名称。
`--output=<file>`	必須	出力ファイルの名称。
CFI パラメータ		
`--offset=<addr>`	必須	FPGAコンフィギュレーションがプログラムされるフラッシュ・メモリ・デバイス内のオフセット。
EPCS パラメータ		
`--epcs`	EPCS デバイスでは必須。デフォルトはオフ。	EPCS デバイスに出力が送られるように指定します。

その他のパラメータについては、コマンドラインで`sof2flash --help`と入力します。

sof2flash コマンドラインの例

```
sof2flash --offset=0x0 --input=standard.sof --output=standard_cfi.flash
```

standard.sof を、CFI フラッシュ・メモリ用のSレコード・ファイル **standard_cfi.flash** に変換します。Sレコード・オフセットは 0x0 から開始します。

```
sof2flash --epcs --input=standard.sof --output=standard_epcs.flash
```

standard.sof を EPCS デバイス用のSレコード・ファイル **standard_epcs.flash** に変換します。

elf2flash

elf2flash ユーティリティは、実行可能およびリンク・フォーマット・ファイルを取り込み、フラッシュ・メモリへのプログラミングに適したSレコード・ファイルに変換します。

elf2flash はまた、必要に応じて、ブート・コピアをフラッシュ・ファイルに挿入します。**elf2flash** は、以下の条件の下でアプリケーション・コードの前にブート・コピア・コードを挿入します。

- プロセッサのリセット・アドレスがプログラム中のフラッシュ・メモリのアドレス範囲内にある。
- 実行可能コードが、プログラム中のフラッシュ・メモリの外部メモリ位置にリンクされる。

elf2flash がブート・コピアを挿入する場合、アプリケーション実行可能およびリンク・フォーマット・ファイルを、ブート・コピアで使用するブート・レコードに変換します。このブート・レコードには、すべてのアプリケーション・コードが格納されますが、レコードは実行可能ではありません。リセット後に、ブート・コピアはフラッシュ・メモリからブート・レコードを読み出し、アプリケーション・コードを正しくリンクされたアドレスにコピーしてから、新たにコピーされたアプリケーション・コードに分岐します。

表 3–4 に、**elf2flash** で使用される一般的なパラメータをリストします。

表 3–4. elf2flash パラメータ （1 / 2）

名称	必要性	説明
汎用パラメータ		
`--input=<file>`	必須	入力の実行可能およびリンク・フォーマット・ファイルの名称。
`--output=<file>`	必須	出力ファイルの名称。
CFI パラメータ		
`--base=<addr>`	必須	フラッシュ・メモリ・コンポーネントのベース・アドレス。**elf2flash** はこのパラメータを `--end` および `--reset` と一緒に使用して、システムがブート・コピアを必要としているかどうかを判断します。
`--end=<addr>`	必須	フラッシュ・メモリ・コンポーネントのエンド・アドレス。**elf2flash** はこのパラメータを `--base` および `--reset` と一緒に使用して、システムがブート・コピアを必要としているかどうかを判断します。

表 3–4. elf2flash パラメータ （2 / 2）

名称	必要性	説明
--reset=<*addr*>	必須	プロセッサ・リセット・アドレス。SOPC Builder で指定されます。**elf2flash** はこのパラメータを--base および--end と一緒に使用して、システムがブート・コピアを必要としているかどうかを判断します。
--boot=<*file*>	以下の条件で必須。 ● プロセッサのリセット・アドレスがプログラム中のフラッシュ・メモリのアドレス範囲内にある。 ● 実行可能コードが、プログラム中のフラッシュ・メモリの外部メモリ位置にリンクされる。	ブート・コピア・オブジェクト・コード・ファイルを指定します。ブート・コピアが要求されない場合は無視されます。**elf2flash** がブート・コピアが必要であると判断したが、--boot パラメータが指定されていない場合、**elf2flash** はエラー・メッセージを表示します。アルテラ提供のブート・コピアは、<Nios II EDS インストール・パス >/**components/altera_nios2/boot_loader_cfi.srec**に存在します。
EPCS パラメータ		
--epcs	EPCS デバイス用のファイルを作成する場合は必須。デフォルトはオフ。	EPCS デバイスに出力が送られるように指定します。
--after=<*file*>		**elf2flash** はこのパラメータを使用して、Nios II 実行コマンドを FPGA コンフィギュレーションに併せて EPCS デバイスに配置します。詳しくは、「EPCS デバイスへのハードウェアとソフトウェアの同時プログラミング」を参照してください。

その他のパラメータについては、コマンドラインで elf2flash --help と入力します。

EPCS デバイスへのハードウェアとソフトウェアの同時プログラミング

--baseパラメータはEPCSデバイスでは利用できません。EPCSデバイスでは、FPGA コンフィギュレーション・データはアドレス 0x0 から開始する必要があるためです。ただし、EPCS デバイスで FPGA コンフィギュレーションと Nios II ソフトウェア実行可能コードの両方をプログラムする場合、--after パラメータにより、ソフトウェア実行可能コードを FPGA コンフィギュレーション・データの直後に配置できます。

最初に **sof2flash** を使用して、FPGA コンフィギュレーション・ファイルを変換します。Nios II ソフトウェア実行可能コードを変換するときは、`--after` パラメータを使用して、FPGA コンフィギュレーション S レコード・ファイルを指定します。ソフトウェア実行可能コードの S レコード出力は、FPGA コンフィギュレーションで使用されていない最初のアドレスから開始します。3–10 ページの「elf2flash コマンドラインの例」の 2 番目の例を参照してください。

elf2flash は FPGA コンフィギュレーションを出力ファイルに挿入しません。オフセット 0x0 からコンフィギュレーション・データを収めるのに十分なスペースを残すだけです。

elf2flash コマンドラインの例

```
elf2flash --base=0x0 --reset=0x0 --boot=boot_loader_cfi.srec --input=myapp.elf\
    --output=myapp.flash
```

myapp.elf を、0x0 をベースにする CFI フラッシュ・メモリ用の S レコード・ファイル **myapp.flash** に変換します。この例では、`--base` と `--reset` が等しいために必要となる（**boot_loader_cfi.srec** の）ブート・コピアをインクルードします。

```
elf2flash --epcs --after=standard.flash --input=myapp.elf --output=myapp.flash
```

myapp.elf を、EPCS デバイス用の S レコード・ファイル **myapp.flash** に変換します。S レコードの出力は、**standard.flash** で使用されていない最初のアドレスから開始します。

bin2flash

bin2flash ユーティリティは、任意のファイルをフラッシュ・プログラマで使用するのに適した S レコード・ファイルに変換します。**bin2flash** を使用すると、ソフトウェア・コンフィギュレーション・テーブルなど、Nios II プログラムで必要とされる読み出し専用バイナリ・データを変換することができます。

ユーザ・アプリケーションによっては、同じ目的に使用される読み出し専用 Zip ファイル・システムを使用する方が便利な場合もあります。

詳細については、Nios II IDE ヘルプ・システムの Zip 読み出し専用ファイル・システムのトピックを参照してください。

実行可能ソフトウェア・ファイルまたは FPGA コンフィギュレーション・ファイルを変換する場合、**bin2flash** を使用しないでください。Nios II ソフトウェア実行可能ファイルを変換する場合、**elf2flash** を使用します。FPGA コンフィギュレーション・ファイルを変換するには、**sof2flash** を使用します。

表 3–5 に、**bin2flash** で使用される一般的なパラメータをリストします。

表 3–5. bin2flash パラメータ

名称	必要性	デフォルト	説明
`--location=<addr>`	必須	N/A	データがプログラムされるフラッシュ・メモリ内のオフセット。
`--input=<file>`	必須	N/A	変換される入力バイナリ・ファイルの名称
`--output=<file>`	必須	N/A	出力ファイルの名称

その他のパラメータについては、コマンドラインで`elf2flash --help`と入力します。

bin2flash コマンドラインの例

```
bin2flash --location=0x40000 --input=data.bin --output=data.flash
```

data.bin を S レコード・ファイル **data.flash** に変換します。S レコード・ファイル内のアドレスにより、フラッシュ・メモリの先頭からオフセット 0x40000 で開始するデータが配置されます。

付録 A. 非標準フラッシュ・メモリ

この項では、非標準 CFI フラッシュ・メモリをサポートする最新のトピックを扱います。この項の手順を使用するには、使用しているフラッシュ・メモリ・デバイスのデータシートが必要です。デバイスの CFI 機能を完全に理解していることを確認してください。

CFI フラッシュ・メモリ・デバイスによっては、CFI テーブル情報が欠落していたり、正しくないことがあります。この場合、Nios II フラッシュ・プログラマは CFI テーブルの間違った情報に基づいて動作するため、失敗する可能性があります。これらのデバイスの場合、Nios II フラッシュ・プログラマは以下の方法で CFI テーブルを上書きして、フラッシュ・メモリを正しくプログラムします。

- ビルトイン認識およびオーバライド
- フラッシュ・オーバライド・ファイル
- 幅モード・オーバライド

ビルトイン認識およびオーバライド

Nios II フラッシュ・プログラマには、既知の CFI テーブル問題が発生したいくつかのデバイスを認識するためのコードが含まれています。これらのデバイスでは、フラッシュ・プログラマが間違ったテーブル・エントリを上書きします。オーバライド・ファイルの作成を試みる前に、必ずビルトイン認識およびオーバライドを使用するようにしてください。フラッシュ・プログラマがデバイスを認識するかどうかを確認するには、コマンドラインから `--debug` オプションを指定して、フラッシュ・プログラマを実行します。フラッシュ・プログラマが CFI テーブルを上書きする場合、フラッシュ・プログラマはメッセージ「Override data for this device is built in」を表示します。

コマンドライン・モードでのフラッシュ・プログラマの使用について詳しくは、第 3 章　コマンドライン・モードでのフラッシュ・プログラマの使用を参照してください。

フラッシュ・オーバライド・ファイル

CFI テーブルに問題がある可能性のある新しくリリースされたフラッシュ・メモリ・デバイスをサポートするために、Nios II フラッシュ・プログラマは、CFI テーブルのエントリをフラッシュ・オーバライド・ファイルで上書きする機能を備えています。フラッシュ・オーバライド・ファイルを使用して、CFI テーブルの間違った情報を手動で上書きできます。これによって、Nios II フラッシュ・プログラマが正しく機能するようになります。

オーバライド・ファイルを作成する前に、コマンドライン・モードで--debug パラメータを指定して、**nios2-flash-programmer** を実行します。このコマンドは、デバイス内の CFI テーブルをリストします。デバッグ出力をデバイスのデータシートと比較します。

フラッシュ・オーバライド・ファイルのフォーマット

フラッシュ・オーバライド・ファイルには、上書きするフラッシュ・メモリごとに 2 つのセクションがあります。最初のセクションでは、フラッシュ・メモリのタイプを宣言します。2 番目のセクションは、CFI テーブルのオーバライド・データです。フラッシュ・オーバライド・ファイルには、'#' 文字を先頭に付けてコメントを記述することができます。

例えば、SST 39VF800 フラッシュ・メモリでは、CFI テーブルの 0x13、0x14、0x2C の位置にある 3 つのエントリが間違っています。以下の例では、それらのアドレス値を上書きする方法を示します。

```
[FLASH-00BF-2781] # キーワード FLASH に、製造 ID とデバイス ID が続く
                  # これらの ID 値は以下の 3 通りの方法で検出できる：
                  # - フラッシュ・メモリ・デバイスのデータシートを参照する
                  # - "autoselect" コマンドを使用する
                  # - nios2-flash-programmer --debug を実行する
CFI[0x13] = 0x02  # CFI テーブル内で見つかったプライマリ・コマンド・セット -
CFI[0x14] = 0x00  # アドレス 0x13 と 0x14 が上書きされて 0x02、0x00 になる
CFI[0x2C] = 0x01  # CFI 消去ブロック領域の番号が
                  # CFI テーブル -アドレス 0x2C で見つかった場合は上書きされて 0x1 になる
```

これは例証にすぎません。**nios2-flash-programmer** は、SST 39VF800 を非標準 CFI デバイスとして認識し、その CFI テーブルを上書きします。このパーツに対しては、オーバライド・ファイルを作成する必要はありません。

フラッシュ・オーバライド・ファイルの使用方法

フラッシュ・オーバライド・ファイルは以下の 2 通りの方法で展開できます。

1. オーバライド・ファイルを <Nios II EDS インストール・パス >**/bin** に置きます。フラッシュ・プログラマは、このディレクトリでパターン nios2-flash-override* に一致するすべてのファイル名を検索します。フラッシュ・プログラマは、これらのすべてのファイルをオーバライド・ファイルとしてロードします。

2. オーバライド・ファイルを--override パラメータでフラッシュ・プログラマに渡します。以下の例で、このパラメータを示します。

```
nios2-flash-programmer --base 0x0 --override=my_override.txt sw.flash
```

幅モード・オーバライド・パラメータ

A–1 ページの「フラッシュ・オーバライド・ファイル」で説明するオーバライド・プロシージャは、フラッシュ・プログラマが CFI クエリ・テーブルで正しいデータ幅モードを検出するものと仮定しています。場合によっては、8 ビット・モードで配線された 16 ビット CFI フラッシュ・メモリ・デバイスが 16 ビット・モードを示すクエリ・テーブルを返すことがあります。このような状況では、フラッシュ・プログラマがクエリ・テーブルの残りを正しく解釈できなくなります。フラッシュ・プログラマはデバイス・タイプが読み出せないため、このような状況を検出できません。フラッシュ・メモリ・デバイスでこの問題が発生した場合は、コマンドラインからフラッシュ・メモリ・デバイスをプログラムする必要があります。

この場合、コマンドラインのデータ幅を隠しパラメータ--width=8 で上書きします。

このパラメータは、ST Micro ST29W800 と ST29W640 の 2 つのフラッシュ・メモリ・デバイスでのみ必要なことが分かっています。これらのデバイスを使用しない場合、このパラメータが必要になることはないはずです。

付録 B. サポートされるフラッシュ・メモリ・デバイス

Nios II フラッシュ・プログラマは、プログラミング・アルゴリズム 1、2、または 3 をサポートするすべての CFI 互換パラレル・フラッシュ・メモリ、およびすべてのアルテラ EPCS シリアル・コンフィギュレーション・デバイスで動作します。すべてのフラッシュ・メモリ・デバイスが、Nios II フラッシュ・プログラマでテストされているわけではありません。この章では、アルテラが Nios II 開発ツールを使用してハードウェアで検証したすべてのフラッシュ・メモリ・デバイスを記載しています。

Nios II フラッシュ・プログラマで動作しない CFI 互換デバイスが見つかった場合は、アルテラのテクニカル・サポートに報告してください。

以下のアルテラ EPCS デバイスは、Nios II フラッシュ・プログラマで動作することが確認されています。

- アルテラ EPCS4
- アルテラ EPCS16
- アルテラ EPCS64

以下のアルテラ CFI 互換デバイスは、Nios II フラッシュ・プログラマで動作することが確認されています。

- AMD AM29LV128
- AMD AM29LV641
- AMD AM29LV065
- Sharp LH28F160S3T
- Intel TE28F320C3
- Intel TE28F320J3
- SST SST39VF800
- SST SST39VF400
- SST SST39VF200
- Atmel AT49BV332AT
- Atmel AT49BV162AT
- ST Micro ST29W800
- ST Micro ST29W640

付録 C. スタンドアロン・モード

ハードウェアまたはソフトウェアを開発する間、Quartus II ソフトウェアおよび Nios II 開発ツールがインストールされたコンピュータで、Nios II フラッシュ・プログラマを使用します。ただし、生産またはサービス環境では、フル・セットのアルテラ開発ツールをインストールしないで、フラッシュ・メモリをプログラムするようにコンピュータをセットアップしたい場合があります。

スタンドアロン・モードでは、フラッシュ・プログラマの機能が制限されてきました。どのタイプの CFI または EPCS フラッシュ・メモリでもプログラムすることができます。ただし、**elf2flash**、**sof2flash**、および **bin2flash** ユーティリティは使用できません。Nios II 開発ツールがフル・インストールされたコンピュータで、フラッシュ・プログラマの入力ファイルを作成する必要があります。

Nios II スタンドアロン・フラッシュ・プログラマのインストール

フラッシュ・プログラマをスタンドアロン・モードでインストールするには、以下のステップを実行します。

1. Quartus II スタンドアロン・プログラマを、Quartus II CD からインストールします。**nios2-flash-programmer** ユーティリティは、Quartus II スタンドアロン・プログラマがボード上の JTAG チェインにアクセスするために必要です。
2. Nios II 開発ツールがフル・インストールされたコンピュータ上で、実行可能ファイル <Nios II EDS インストール・パス >**/bin/nios2-flash-programmer.exe** を探します。
3. スタンドアロン・コンピュータで、**nios2-flash-programmer.exe** をディレクトリ <Quartus II スタンドアロン・プログラマ・パス >**/bin** にコピーします。
4. Nios II がフル・インストールされたコンピュータで、Windows ライブラリ・ファイル **c:/cygwin/bin/cygwin1.dll** を探します。Nios II EDS とは別に **cygwin** をインストールする場合、**cygwin** DLL が異なるディレクトリに常駐している場合があります。
5. スタンドアロン・コンピュータで、**cygwin1.dll** を **c:/cygwin/bin**（または **cygwin** のインストールに基づく同等のパス）にインストールします。

Nios II スタンドアロン・フラッシュ・プログラマの実行

フラッシュ・プログラマをスタンドアロン・モードで実行するには、以下のステップを実行します。

1. スタンドアロン・コンピュータで、コマンド・プロンプトを開きます。ディレクトリをフラッシュ・メモリにプログラムするファイルがある場所に変更します。
2. **nios2-flash-programmer** ユーティリティを、第 3 章　コマンドライン・モードでのフラッシュ・プログラマの使用の説明に従って実行します。

付録 D. トラブルシューティング

概要

この章では、Nios II フラッシュ・プログラマのトラブルシューティング情報を記載しています。この章の各項で、Nios II フラッシュ・プログラマを使用するときに発生する可能性がある一般的な問題について説明します。

IDE でグレー表示されるプログラム・フラッシュ・ボタン

Nios II IDE の **Flash Programmer** ダイアログ・ボックスで、**Program Flash** ボタンがグレー表示されます。

考えられる原因

Flash Programmer ダイアログ・ボックスで、必要なパラメータを完全に指定していません。

推奨される処置

- JTAG ケーブルが正しく設定されていることを確認します。**Target Connection** タブで JTAG 設定を指定します。
- フラッシュ・メモリにプログラムするファイルを選択していることを確認します。

“No Nios II processors available” エラー

フラッシュ・プログラマを実行すると、以下のエラーが表示される。“There are no Nios II processors available which match the values specified. Please check that your PLD is correctly configured, downloading a new **.sof** file if necessary.”（指定された値に一致する使用可能な Nios II プロセッサがありません。PLD が正しく設定されているか確認し、必要に応じて新しいファイルをダウンロードしてください。）

考えられる原因

フラッシュ・プログラマは、FPGA 内の Nios II JTAG デバッグ・モジュールに接続できません。

推奨される処置

- FPGA が有効なフラッシュ・プログラマのターゲット・デザインを実行していることを確認します。実行していない場合は、Quartus II Programmer を使用して FPGA をコンフィギュレーションする必要があります。1–3 ページの「フラッシュ・プログラマ・ターゲット・デザイン」を参照してください。

- フラッシュ・プログラマをコマンドライン・モードで使用する場合は、正しい--device、--cable、および--instance パラメータ値を指定していることを確認してください。詳細については、第 3 章　コマンドライン・モードでのフラッシュ・プログラマの使用を参照してください。

“No CFI table found” エラー

フラッシュ・プログラマを実行して CFI フラッシュ・メモリをプログラムすると、次のエラーが表示される。“No CFI table found at address <base address>（<base address> アドレスで CFT テーブルが見つかりません）”

考えられる原因

フラッシュ・プログラマは FPGA の Nios II JTAG デバッグ・モジュールに接続できますが、指定されたベース・アドレスでフラッシュ・メモリへのクエリを正しく実行できません。

推奨される処置

- **nios2-flash-programmer** をコマンドライン・モードで使用する場合、CFI デバイスの正しいベース・アドレスを指定しているか確認します。SOPC Builder でフラッシュ・メモリのベース・アドレスを見つけることができます。
- **nios2-flash-programmer** を、コマンドライン・モードで--debug パラメータを指定して実行します。このコマンドはフラッシュ・メモリのクエリ・テーブルをダンプします。出力をフラッシュ・メモリ・デバイスのデータ・シートと比べます。詳しくは、第 3 章　コマンドライン・モードでのフラッシュ・プログラマの使用を参照してください。
- フラッシュ・メモリ・ハードウェアが正しく接続され、SOPC Builder で指定されるベース・アドレスに配置されていることを確認します。Nios II IDE に付属する “Memory Test” ソフトウェア・テンプレートの “Test Flash” ルーチンを実行して、ベース・アドレスを確認します。テストが失敗する場合は、メモリ接続に問題があります。問題は次の 2 つの場所で探します。
  - ターゲット・ボード上の物理的接続
  - トップレベル FPGA デザインのピン・アサインメント
- ここで見つからない場合は、使用しているフラッシュ・メモリ・デバイスがオーバライド・ファイルを必要としていないか確認します。詳細については、付録 A　非標準フラッシュ・メモリを参照してください。

“No EPCS registers found” エラー

フラッシュ・プログラマを実行して EPCS デバイスをプログラムすると、次のエラーが表示される。“No EPCS registers found: tried looking at addresses....”（EPCS レジスタが見つかりません。アドレスの確認を試みました。）

考えられる原因

フラッシュ・プログラマは FPGA の Nios II JTAG デバッグ・モジュールに接続できていますが、指定されたベース・アドレスに配置されている EPCS デバイスを見つけることはできません。

推奨される処置

- Quartus II Programmer を使用し、JTAG による有効なターゲット・デザインで FPGA を再コンフィギュレーションします。コンフィギュレーション・コントローラなど、FPGA が別の方法でコンフィギュレーションされている場合、EPCS デバイスに接続されるピンがディセーブルされている可能性があります。
- **nios2-flash-programmer** をコマンドライン・モードで使用する場合、EPCS デバイスの正しいベース・アドレスを指定しているか確認します。SOPC Builder でフラッシュ・メモリのベース・アドレスを見つけることができます。
- EPCS デバイスがボード上の EPGA に正しく接続されているか確認します。Nios II IDE の “Memory Test” ソフトウェア・テンプレートの “Test EPCS” ルーチンを実行して、EPCS 接続を確認します。テストが失敗する場合は、メモリ接続に問題があります。問題は次の 2 つの場所で探します。
  - ターゲット・ボード上の物理的接続
  - トップレベル FPGA デザインのピン・アサインメント
- Quartus II Programmer を使用して、JTAG ダウンロード・ケーブルから直接 EPCS デバイスをプログラムし、EPCS デバイスが正しく FPGA をコンフィギュレーションしているか確認します。
- **nios2-flash-programmer** を、コマンドライン・モードで`--epcs` パラメータを指定して実行します。このコマンドは EPCS デバイスのフラッシュ・メモリに関する情報を表示します。詳しくは、第 3 章 コマンドライン・モードでのフラッシュ・プログラマの使用を参照してください。

“System does not have any flash memory” エラー

フラッシュ・プログラマを実行すると、以下のエラーが表示されます。“The SOPC Builder system does not have any flash memory.（SOPC Builder システムにフラッシュ・メモリが装備されていません）”

考えられる原因

FPGA で現在有効なフラッシュ・プログラマのターゲット・デザインがコンフィギュレーションされていません。

推奨される処置

実行可能な場合、フラッシュ・プログラマのターゲット・デザインの基準に適合するように FPGA デザインをアップグレードします。詳細については、1–3 ページの「フラッシュ・プログラマ・ターゲット・デザイン」を参照してください。

“Reading System ID at address 0x<address>: FAIL” エラー

フラッシュ・プログラマを IDE モードで実行すると、以下のエラーが表示されます。“Reading System ID at address 0x<address>: FAIL”（アドレス 0x<address> でシステム ID の読み込み中 : 失敗）

考えられる原因

FPGA が現在、IDE の C/C++ アプリケーション向けシステム・ライブラリ･プロジェクトに対応するターゲット･デザインでコンフィギュレーションされていません。

推奨される処置

Quartus II を使用して正しい FPGA コンフィギュレーション・ファイルを FPGA にダウンロードし、再度 Nios II フラッシュ・プログラマの使用を試みます。

デバイスのサイズにアライメントされていないベース・アドレス

フラッシュ･プログラマを実行すると、エラー･メッセージ **Base address not aligned on size of device** が表示されます。

考えられる原因

フラッシュ・プログラマに渡されるフラッシュ・デバイスのベース・アドレスがフラッシュ・デバイスのサイズの倍数になっていません。

推奨される処置

- フラッシュ・デバイスが、CFI テーブルにリストされるフラッシュ・サイズの倍数である SOPC Builder のベース・アドレスにマップされているか確認します。
- コマンドライン・モードで、**nios2-flash-programmer** に渡す `--base` パラメータが、SOPC Builder システムでのフラッシュ・デバイスの正しいベース・アドレスになっているか確認します。