

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。

QI151009-8.0.0

はじめに

プログラマブル・ロジック・デザイン (PLD) がより複雑になり、性能の向上が要求される中で、高度な合成がデザイン・フローで重要な部分を占めるようになりました。この章では、Quartus® II ソフトウェアにおける Synplicity Synplify および Synplify Pro ソフトウェアのサポートについて説明し、アルテラ・デバイスで良好な結果を達成するためのデザイン・フロー、方法論、および手法を紹介します。この章は、以下のトピックスで構成されています。

- Synplify および Quartus II ソフトウェアでの一般的なデザイン・フロー
- タイミング・ドリブン・コンパイル設定、最適化オプション、およびアルテラ固有の属性など、Synplify ソフトウェアの最適化方法
- NativeLink® 統合を使用した Quartus II ソフトウェアへのデザインおよび制約のエクスポート
- アルテラ・メガファンクションおよび LPM (Library of Parameterized Modules) ファンクションのガイドライン、MegaWizard® Plug-In Manager によるこれらのメガファンクション / ファンクションのインスタンス化、および HDL (ハードウェア記述言語) コードからのこれらのメガファンクション / ファンクションの推測方法のヒント
- Synplify Pro ソフトウェアの MultiPoint フローなど、インクリメンタル・コンパイルおよびブロック・ベースのデザイン

この章の内容は、特に記載がない限り、Synplify および Synplify Pro ソフトウェアの両方に適用されます。この章は、以下の項で構成されています。

- 9-2 ページの「サポートされるアルテラ・デバイス・ファミリ」
- 9-3 ページの「デザイン・フロー」
- 9-9 ページの「Synplify 最適化方法」
- 9-19 ページの「NativeLink 統合を使用する Quartus II ソフトウェアへのデザインのエクスポート」
- 9-36 ページの「アルテラ・メガファンクションおよびアーキテクチャ固有の機能に関するガイドライン」
- 9-51 ページの「インクリメンタル・コンパイルおよびブロック・ベース・デザイン」

この章では、Synplify または Synplify Pro ソフトウェアをすでにセットアップし、ライセンスを取得済みで、ソフトウェアの操作に慣れていることを前提としています。

サポートされる アルテラ・ デバイス・ ファミリ

Synplify ソフトウェアは、合成結果をアルテラ・デバイス・ファミリにマップします。以下に、Quartus II ソフトウェア v8.0 で、Synplify ソフトウェア v9.4 によってサポートされるアルテラ・デバイス・ファミリを示します。

- Arria™ GX
- Stratix® IV
- Stratix III
- Stratix II、Stratix II GX、Hardcopy® II
- Stratix、Stratix GX、HardCopy Stratix
- Cyclone® III
- Cyclone II
- Cyclone
- MAX® II
- MAX® 7000、MAX 3000
- APEX™ II
- APEX 20K、APEX 20KC、APEX 20KE
- FLEX® 10K、FLEX 6000
- ACEX® 1K

また、Synplify ソフトウェアは、Quartus II ソフトウェアでのみサポートされる Excalibur™ ARM® レガシー・デバイスもサポートします。ただし、www.altera.co.jp/mysupport でライセンスをリクエストする必要があります。

また、Synplify ソフトウェアは、アルテラ MAX+PLUS II ソフトウェアでのみサポートされる以下のレガシー・デバイスもサポートします。

- FLEX 8000
- MAX 9000



特定の Synplify バージョンでの新しいデバイスのサポートについては、Synplicity の Web サイト (www.synplicity.com) でリリース・ノートを参照してください。

デザイン・フロー

Synplify ソフトウェアを使用した、Quartus II ソフトウェアのデザイン・フローは、以下のステップで構成されます。

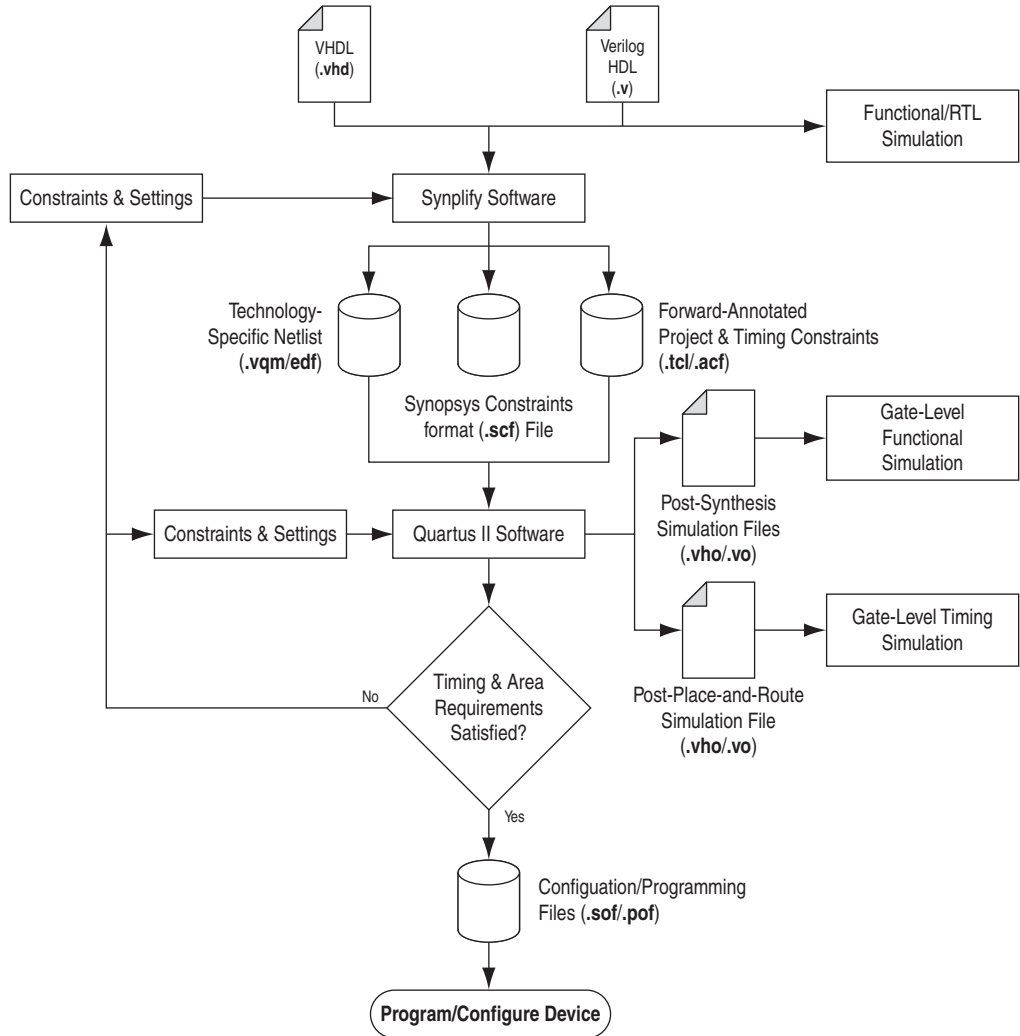
1. Quartus II ソフトウェア、Synplify ソフトウェア、またはテキスト・エディタで、Verilog HDL または VHDL デザイン・ファイルを作成します。
2. Synplify ソフトウェアでプロジェクトをセットアップし、合成のために HDL デザイン・ファイルを追加します。
3. ターゲット・デバイスを選択し、Synplify ソフトウェアでタイミング制約およびコンパイラ・ディレクティブを追加して、合成時にデザインを最適化します。
4. Synplify ソフトウェアで合成を実行します。
5. Quartus II プロジェクトを作成し、Synplify ソフトウェアによって生成されるファイルを Quartus II ソフトウェアにインポートします。これらのファイルは、配置配線および性能評価に使用されます。
 - テクノロジ固有の Verilog Quartus Mapping File (。**.vqm**) ネットリストまたはレガシー・デバイス用の EDIF (。**.edf**) ネットリストも、MAX+PLUS II ソフトウェアでサポートされます。
 - TimeQuest タイミング制約用の Synopsys Constraints Format (。**.scf**) ファイル。
 - ツール・コマンド言語 (。**.tcl**) 制約ファイル。

Quartus II ソフトウェアは、Synplify ソフトウェア内から実行することもできます。詳細は、[9-20 ページの「Synplify ソフトウェア内から Quartus II ソフトウェアを実行する」](#)を参照してください。

6. ニーズを満たす配置配線結果が得られた後、アルテラ・デバイスをコンフィギュレーションまたはプログラムします。

図 9-1 に、Synplify および Quartus II ソフトウェアを使用するときの推奨デザイン・フローを示します。

図 9-1. 推奨デザイン・フロー



Synplify および Synplify Pro ソフトウェアは、VHDL および Verilog HDL ソース・ファイルの両方をサポートします。ただし、VHDL および Verilog HDL ソース・ファイルを組み合わせて使用するミックスド・シンセシスは、Synplify Pro ソフトウェアでのみサポートされています。

Synplify ソフトウェアの SCOPE ウィンドウを使用して、Synplify Constraints File (.sdc) にデザインのタイミング制約および属性を指定します。これらは、HDL ソース・ファイルに直接指定することもできます。また、HDL ソース・ファイルにはコンパイラ・ディレクティブも定義できます。これらの制約の多くは、Quartus II ソフトウェアで使用するためにフォワード・アノテートされます。

Synplify ソフトウェアに付属する HDL Analyst は、テクノロジーに依存しない RTL (Register Transfer Level) ビュー・ネットリスト (.srs) ファイルと、テクノロジー・ビュー・ネットリスト (.srm) ファイルの回路図ビューを生成するグラフィカル・ツールです。Synplify HDL Analyst では、デザインの解析およびデバッグを視覚的に行えます。HDL Analyst は、RTL および Technology ビュー、HDL ソース・コード、および FSM (Finite State Machine) ビューの間でクロス・プロービングをサポートしています。また、Synplify HDL Analyst は、Technology ビューと Quartus II ソフトウェアのタイミング・レポート・ファイルの間でもクロス・プロービングをサポートしています。



Synplify ソフトウェアで HDL Analyst をイネーブルするには、別途ライセンス・ファイルが必要です。Synplify Pro ソフトウェアには HDL Analyst が付属しています。

合成が完了したら、配置配線のために、.vqm または .edf ネットリストを Quartus II ソフトウェアにインポートします。Synplify ソフトウェアによって生成される Tcl ファイルを使用して、制約 (デバイスの選択を含む) をフォワード・アノテートできます。また、オプションで、Quartus II ソフトウェアでプロジェクトをセットアップすることもできます。

Stratix III、Cyclone III、Arria GX、または新しいデバイスを選択した場合、Quartus II ソフトウェアはデフォルトで、TimeQuest タイミング・アナライザにより .scf ファイルからの SDC フォーマットのタイミング制約を使用します。Stratix II または Stratix II GX デバイスが選択されている場合、Synplify ソフトウェアの **Implementation Options** ダイアログ・ボックスで、**Device** タブの **Use TimeQuest Timing Analyzer** オプションをオンにすることによって、クラシック・タイミング・アナライザから TimeQuest タイミング・アナライザに切り替えることができます。その他のデバイスの場合、Quartus II ソフトウェアは、クラシック・タイミング・アナライザで Quartus Setting File (.qsf) からの Tcl フォーマットのタイミング制約を使用します。Quartus II ソフトウェアで、TimeQuest タイミング・アナライザからクラシック・タイミング・アナライザに手動で変更する方法については、[9-23 ページの「TimeQuest SDC タイミング制約を .scf ファイルで Quartus II ソフトウェアに渡す」](#)を参照してください。

領域およびタイミング要求を満たしている場合は、Quartus II ソフトウェアによって生成されるファイルを使用して、アルテラ・デバイスをプログラムまたはコンフィギュレーションします。図 9-1 に示すように、領域またはタイミング要求を満たしていない場合は、Synplify ソフトウェアまたは Quartus II ソフトウェアで制約を変更して合成を繰り返すことができます。アルテラでは、可能な限り、Synplify ソフトウェア・レベルでタイミング制約を提供し、Quartus II ソフトウェア・レベルで配置制約を提供することを推奨しています。領域およびタイミング要求を満たすまで、この処理を繰り返します。

シミュレーションはプロセスのさまざまな時点で実行できますが、最終的なタイミング解析は、配置配線の完了後に実行する必要があります。また、フォーマル検証もデザイン・プロセスのさまざまな段階で実行できます。



Synplify ソフトウェアでのフォーマル検証のサポート方法について詳しくは、「Quartus II ハンドブック Volume 3」の「[セクション III フォーマル検証](#)」を参照してください。

また、Quartus II ソフトウェアの他のオプションや手法を使用して、領域およびタイミング要求を満たすこともできます。例えば、WYSIWYG Primitive Resynthesis というオプションがあります。このオプションを使用すると、Quartus II ソフトウェア内で .vqm ネットリストに最適化を実行できます。



ネットリスト最適化については、「Quartus II ハンドブック Volume 2」の「[Netlist Optimizations and Physical Synthesis](#)」の章を参照してください。

Synplify および Quartus II ソフトウェアのオプションを使用しても、領域およびタイミング要求を満足できない場合があります、そのときはソース・コードを修正する必要があります。

合成後、Synplify ソフトウェアは、いくつかの中間ファイルと出力ファイルを生成します。それらのファイル・タイプを表 9-1 に示します。

| ファイル拡張子 | ファイルの説明 |
|-----------|--|
| .srs | テクノロジーに依存しない RTL ネットリスト。Synplify ソフトウェアでのみ読み出し可能。 |
| .srm | Technology ビュー・ネットリスト。 |
| .srr (1) | Synthesis レポート・ファイル。 |
| .edf/.vqm | .edf または .vqm ファイル形式の、テクノロジー固有のネットリスト。 .edf ファイルは、ACEX 1K、FLEX 10K、FLEX 10KA、FLEX 10KE、FLEX 6000、FLEX 8000、MAX 7000、MAX 9000、および MAX 3000 デバイスで作成されます。.vqm ファイルは、これら以外のすべてのアルテラ・デバイス・ファミリで作成されます。 |
| .acf/.tcl | 制約およびアサインメントを含む、フォワード・アノテートされた制約ファイル。 Quartus II ソフトウェアの .tcl ファイルは、すべてのデバイスで作成されます。.tcl ファイルには、Quartus II プロジェクトを作成およびセットアップし、配置制約を引き渡すのに適切な Tcl コマンドが含まれています。 適用可能な場合、.acf ファイルから MAX+PLUS II アサインメントがインポートされます。 |
| .scf | TimeQuest タイミング・アナライザ制約用のタイミング制約を含む Synopsys Constraint Format ファイル。 |

表 9-1 の注:

- (1) このレポート・ファイルには、多くの場合は配置配線前の情報に基づく性能見積もりが含まれています。配置配線後は、Quartus II ソフトウェアがレポートする f_{MAX} を使用します。これが唯一信頼できるタイミング情報源です。このレポート・ファイルには、合成後デバイス・リソース使用率統計情報が含まれていますが、この情報は配置配線後のリソース使用率を正確に予測しているとは限りません。Synplify ソフトウェアは、ブラック・ボックス・ファンクションを考慮せず、また Quartus II ソフトウェアが実行するレジスタ・パッキングで達成されるロジック使用率の低下も考慮しません。レジスタ・パッキングにより、1 個のレジスタとルック・アップ・テーブル (LUT) が 1 つのロジック・セルに統合され、ロジック・セルの使用率が Synplify ソフトウェアの見積もりよりも低くなります。配置配線後は、Quartus II ソフトウェアがレポートするデバイス使用率を使用してください。

出力ネットリスト・ファイル名と結果の形式

以下のステップを実行して、Synplify ソフトウェアの出力ネットリスト・ディレクトリの場所と名前を指定します。

1. Project メニューの **Implementation Options** をクリックします。
2. **Implementation Results** タブをクリックします。
3. **Results Directory** ボックスに、出力ネットリスト・ファイル・ディレクトリの場所を入力します。

4. **Result File Name** ボックスに、出力ネットリスト・ファイル名を入力します。

デフォルトでは、ディレクトリはプロジェクト実装ディレクトリに設定され、ファイル名はトップレベル・デザイン・モジュール名またはエンティティ名に設定されます。

また、**Implementation Results** タブでは、**Result Format** および **Quartus Version** オプションも使用できます。**Result Format** リストには、デバイス・ファミリに応じて、**.edf** または **.vqm** ネットリストが指定されます。**.edf** 出力ネットリスト・ファイルは、ACEX 1K、FLEX 10K、FLEX 10KA、FLEX 10KE、FLEX 6000、FLEX 8000、MAX 7000、MAX 9000、および MAX 3000 デバイスでのみ作成されます。その他のアルテラ・デバイスでは、**.vqm** フォーマットのネットリストが生成されます。

Quartus Version リストから、使用中の **Quartus II** ソフトウェアのバージョンを選択します。このオプションにより、ネットリストとソフトウェア・バージョンの互換性が保証され、最新機能がサポートされます。アルテラでは、可能な限り、**Quartus II** ソフトウェアの最新バージョンを使用することを推奨しています。**Quartus II** ソフトウェアが、**Quartus Version** リストで使用可能なバージョンよりも新しい場合は、現在の **Quartus II** ソフトウェア・バージョンをサポートする新しいバージョンの **Synplify** ソフトウェアを入手可能かどうかを確認します。そうでない場合は、最良の互換性を達成するために、リストで最新バージョンを選択します。



Quartus Version リストは、アルテラ・デバイスを選択した後のみ利用できます。

Synplify ソフトウェアで使用する **Quartus II** ソフトウェアのバージョンを設定するには、以下のステップを実行します。

1. **Synplify** ソフトウェアで、**Project** メニューの **Implementation Options** をクリックします。
2. **Implementation Results** タブをクリックし、**Quartus Version** をクリックします。
3. リストから適切なバージョン番号を選択します。

また、コマンドラインで以下のコマンドを使用することもできます。

```
set_option -quartus_version <version number>
```

Synplify 最適化方法

デザインがより複雑になり、性能の向上が要求される中で、さまざまな最適化方法を使用することが重要になってきました。Synplify ソフトウェアの制約を VHDL および Verilog HDL コーディング手法、および Quartus II ソフトウェア・オプションと組み合わせると、必要な結果を得ることができます。



追加のデザインおよび最適化手法については、「Quartus II ハンドブック Volume 1」の「[Design Recommendations for Altera Devices and the Quartus II Design Assistant](#)」および「Quartus II ハンドブック Volume 2」の「[Area and Timing Optimization](#)」の章を参照してください。

Synplify ソフトウェアは、デザインの性能を向上させるための制約および最適化手法を多数備えています。Synplify Pro ソフトウェアは、基礎的な Synplify ソフトウェアでサポートされていない手法もいくつか備えています。このドキュメントで Synplify サポートと記述されている箇所では、基礎的な Synplify ソフトウェアと Synplify Pro ソフトウェアの両方を意味し、Synplify Pro に限定される機能は、Synplify Pro と記述されます。この項では、結果の品質を向上させるのに使用できるいくつかの手法の概要を示します。



この項で説明する属性の適用について詳しくは、「Synplify Software Reference Manual」の「[Altera Constraints, Attributes, and Options](#)」の章を参照してください。

Synplify Pro の実装

他の合成結果を上書きしないで、別の合成結果を作成するには、Synplify Pro ソフトウェアで、Project メニューの **New Implementation** をクリックします。各実装について、ターゲット・デバイス、合成オプション、および制約ファイルを指定します。各実装では、特定の実装のコンパイルからのすべての結果ファイル（.vqm/.edf、.scf、および .tcl ファイルなど）を含む独自のサブディレクトリが生成されます。したがって、異なる実装の結果と比較して、デザインにとって最適な合成オプションおよび制約のセットを見つけることができます。

タイミング・ドリブンのシンセシス設定

Synplify ソフトウェアは、ユーザが割り当てたタイミング制約によるタイミング・ドリブンのシンセシスをサポートして、デザインの性能を最適化します。Synplify ソフトウェアは、デザインを最適化して、これらの制約を満足するよう試みます。

Quartus II NativeLink 機能により、タイミング・ドリブン配置配線のために **.tcl** スクリプト・ファイルまたは **.scf** ファイルを使用して、Synplify ソフトウェアで適用するタイミング制約を Quartus II ソフトウェアにフォワード・アノテートすることができます。クロック周波数、フォルス・パス、およびマルチサイクル・パスなどの制約をフォワード・アノテートする方法について詳しくは、9-23 ページの「TimeQuest SDC タイミング制約を .scf ファイルで Quartus II ソフトウェアに渡す」または 9-25 ページの「Tcl コマンドを使用して Quartus II ソフトウェアに制約を渡す」を参照してください。この項では、Synplify ソフトウェアのいくつかの重要なタイミング制約について説明します。



Synplify Synthesis Report File ファイル (**.srr**) には、配置配線遅延の見積もりについてのタイミング・レポートが含まれています。Quartus II ソフトウェアは、サードパーティ合成ツールから合成後ネットリストでさらに最適化を実行できます。また、デザインにはブラック・ボックスや、サードパーティ合成ソフトウェアで最適化されていない IP (Intellectual Property) ファンクションが含まれている場合もあります。実際のタイミング結果は、Quartus II ソフトウェアでデザインに対して完全な配置配線を実行した後でのみ得られます。これらの理由から、Quartus II 配置配線後タイミング・レポートは、より正確なデザイン表現を提供します。デザインの性能評価には、これらのレポートの統計情報を使用する必要があります。

クロック周波数

シングル・クロック・デザインの場合、プッシュボタン・フローを使用するときはグローバル周波数を指定します。このフローはシンプルであり、良好な結果が得られますが、より高度なデザインの性能要件を満たさないことがよくあります。タイミング制約、コンパイラ・ディレクティブ、および他の属性を使用すると、デザインの性能を最適化するのに役立ちます。これらの属性およびディレクティブは、HDL コードに直接入力できます。あるいは、属性は Synplify ソフトウェアの SCOPE ウィンドウから **.sdc** ファイルに入力できます (ディレクティブは入力できません)。

SCOPE ウィンドウを使用して、デザイン全体に対するグローバル周波数要件の設定と、個別クロック設定を行います。SCOPE ウィンドウの **Clocks** タブを使用して、周波数 (または周期)、立ち上がり時間、立ち下がり時間、デューティ・サイクル、その他の設定を指定します。グローバル周波数を過度に制約しないで、個別にクロック設定を割り当てると、Quartus II ソフトウェアおよび Synplify ソフトウェアは、デザイン全体に対して最速のクロック周波数を達成します。define_clock 属性はクロック制約を割り当てます。

複数クロック・ドメイン

Synplify ソフトウェアは、関連しないクロック・ドメインに対してタイミング解析を実行できます。各クロック・グループは異なるクロック・ドメインであり、他のすべてのクロック・グループのクロックとは関連しないものとして扱われます。1 つのクロック・グループ内のすべてのクロックが関連すると想定され、Synplify ソフトウェアはクロック間の関係を自動的に計算します。SCOPE ウィンドウの **Clocks** タブを使用するか、`define_clock` 属性を使用して、新しいクロック・グループにクロックを割り当てたり、同じクロック・グループに関連するクロックを配置できます。

入力 / 出力遅延

SCOPE ウィンドウの **Input/Output** タブを使用するか、`define_input_delay` および `define_output_delay` 属性を使用して、デザインのポートの入力および出力遅延を指定します。Synplify ソフトウェアでは、 t_{CO} および t_{SU} 値を入力および出力に直接割り当てることができません。ただし、 t_{CO} 値は外部出力遅延を設定して推測でき、 t_{SU} 値は外部入力遅延を設定して推測できます。下の **計算式 1** および **2** は、 t_{CO} / t_{SU} の関係と入力 / 出力遅延の関係を示します。

- (1) $t_{CO} = \text{clock period} - \text{external output delay}$
- (2) $t_{SU} = \text{clock period} - \text{external input delay}$

`syn_forward_io_constraints` 属性が 1 に設定されていると、Synplify ソフトウェアは、NativeLink 統合を使用して外部入力遅延および出力遅延を Quartus II ソフトウェアに渡します。次に、Quartus II ソフトウェアは外部遅延を使用して、最大システム周波数を計算します。

マルチサイクル・パス

SCOPE ウィンドウの **Multi-Cycle Paths** タブを使用するか、`define_multicycle_path` 属性を使用して、デザインのマルチサイクル・パスを指定します。マルチサイクル・パスとは、伝播するために複数クロック・サイクルを必要とするパスです。どのパスがマルチサイクルかを指定して、Quartus II および Synplify コンパイラが非クリティカル・パスで過剰に動作しないようにすることが重要です。これらのパスを指定していない場合も、タイミング解析時に不正確なクリティカル・パスがレポートされる可能性があります。

フォルス・パス

フォルス・パスとは、タイミング解析で考慮する必要がないパスで、最適化時には低い優先順位が割り当てられる（または、優先順位が割り当てられない）パスです。フォルス・パスの例としては、低速非同期リセット、デザインに追加されるテスト・ロジックなどがあります。これらのパスは、SCOPE ウィンドウの **False Paths** タブで設定します。define_false_path 属性を使用します。

FSM コンパイラ

FSM コンパイラがオンの場合、コンパイラはデザイン内の状態・マシンを自動的に検出します。次に、状態・マシンを抽出して最適化します。FSM コンパイラは状態・マシンを解析し、状態数に基づいて、“sequential”、“gray”、または“one-hot”のどのエンコーディングを実装するかを決定します。また、FSM コンパイラは、未使用状態解析、到達不能状態の最適化、および遷移ロジックの最小化も実行します。

FSM コンパイラがオフの場合、コンパイラはロジックを状態・マシンとして最適化しません。状態・マシンは、HDL コードでのコーディングとして実装されます。したがって、状態・マシンのコーディング・スタイルが“sequential”の場合、実装も“sequential”です。FSM コンパイラがオンの場合、状態・マシンを推測して最適化します。実装は HDL コードのコーディング・スタイルには関係なく、状態数に基づいて行われます。

syn_state_machine コンパイラ・ディレクティブを使用して、状態・マシンを抽出および最適化するか否かを指定します。FSM コンパイラのデフォルトのエンコーディングを無効にするには、syn_encoding ディレクティブを使用します。

syn_encoding ディレクティブの値を表 9-2 に示します。

| 値 | 説明 |
|------------|---|
| Sequential | 可能な限り、フリップフロップ数が少ない状態・マシンを生成します。Sequential (Binary と呼ばれる) 状態・マシンは、タイミングが主要な問題ではない場合に、面積が重視されるデザインに有効です。 |
| Gray | 遷移ごとに 1 個のフリップフロップだけが変化する状態・マシンを生成します。Gray でエンコードされる状態・マシンはグリッチが発生しない傾向があります。 |

表 9-2. syn_encoding ディレクティブの値 (2 / 2)

| 値 | 説明 |
|---------|---|
| One-hot | 各ステートに 1 個のフリップフロップを持つステート・マシンを生成します。One-hot ステート・マシンは一般に、最高の性能を提供し、clock-to-output 遅延が最短です。ただし、One-hot 実装は、通常 Binary 実装よりも大きくなります。 |
| Safe | 無効なステートに達した場合に、ステート・マシンをリセット・ステートに強制する、追加のコントロール・ロジックを生成します。値 safe は、他の 3 つの値とともに使用でき、これらの値を併用すると、ステート・マシンは要求されたエンコーディング方式で実装され、リセット・ロジックも生成されます。 |

例 9-1 に、syn_encoding ディレクティブを適用するための VHDL コード例を示します。

例 9-1. syn_encoding の VHDL コード

```
SIGNAL current_state : STD_LOGIC_VECTOR(7 DOWNTO 0);
ATTRIBUTE syn_encoding : STRING;
ATTRIBUTE syn_encoding OF current_state : SIGNAL IS "sequential";
```

デフォルトでは、ステート・マシン・ロジックは速度と面積に対して最適化されますが、これはクリティカル・システムにとっては望ましくない場合もあります。値 safe は、無効なステートに達した場合に、ステート・マシンをリセット・ステートに強制する、追加のコントロール・ロジックを生成します。

Synplify Pro の FSM エクスプローラ

Synplify Pro ソフトウェアは、FSM エクスプローラを使用して、ステート・マシンに適したエンコーディング・スタイルを探し、デザイン全体の制約に基づいて最適なエンコーディングを実装できます。FSM エクスプローラは、FSM コンパイラを使用して、ステート・マシンを識別してデザインから抽出します。ただし、FSM コンパイラはエンコーディング・スタイルをステート数に基づいて選択しますが、FSM エクスプローラは特定のエンコーディング・スタイルを選択する前に、いくつかのスタイルを試します。トレード・オフは、コンパイルではステート・マシンの解析を実行するのにより多くの時間が必要ですが、ステート・マシンにとって最適なエンコーディング方式を検出することです。

最適化属性およびオプション

以下の項では、デザイン性能を高めるために Synplify ソフトウェアで変更できる他の属性およびオプションについて説明します。

Synplify Pro でのリタイミング

Synplify Pro ソフトウェアは、デザインをリタイミングできます。これにより、組み合わせエレメント間でレジスタを移動（レジスタ・バランシング）することにより、シーケンシャル回路のタイミング性能を向上させることができます。レジスタをリタイミングすると名前が変更されるので注意してください。デザインをリタイミングするには、**Device** タブの **Implementation Options** セクションにある **Retiming** オプションをオンにするか、`syn_allow_retiming` 属性を使用します。

最大ファンアウト

デザインにファンアウトが大きいクリティカル・パス・ネットがある場合は、`syn_maxfan` 属性を使用して、ネットのファンアウトを制御できます。この属性を特定のネットに設定すると、ネットのドライバが複製されて、全体的なファンアウトが低下します。`syn_maxfan` 属性は整数値を取り、それを入力またはレジスタに適用します（`syn_maxfan` 属性は、コントロール信号の複製には使用できず、この属性の許容最小値は 4）。この属性を使用すると、ロジック・リソース使用率が増加することがあるため、配線リソースに歪みが生じて、コンパイル時間が長くなり、フィッティングが困難になります。

出力レジスタまたは出力イネーブル・レジスタを複製する必要がある場合は、`syn_useioff` 属性を使用して、出力ピンごとにレジスタを作成できます（「[レジスタ・パッキング](#)」を参照）。

ネットの維持

合成時、コンパイラは、ポート、レジスタ、およびインスタンス化されたコンポーネントを保持します。ただし、ネットによっては保持できずに、最適化された回路が作成される場合があります。`syn_keep` デイレクティブを適用すると、コンパイラの最適化が無効になり、合成時にネットが維持されます。`syn_keep` デイレクティブは、**Boolean** 値を取り、これをワイヤ（Verilog HDL）および信号（VHDL）に適用できます。この値を **true** に設定すると、合成を通じてネットが維持されます。

レジスタ・パッキング

アルテラ・デバイスは、レジスタを I/O セルにパッキングすることができます。アルテラでは、Quartus II ソフトウェアで I/O レジスタ・アサインメントの作成を許可することを推奨しています。ただし、`syn_useioff` 属性を使用してレジスタ・パッキングを制御できます。`syn_useioff` 属性は、Boolean 値を取り、ポートまたはモジュール全体に適用できます。この値を **1** に設定すると、コンパイラはレジスタを I/O セルにパッキングするよう指示します。この値を **0** に設定すると、Synplify および Quartus II ソフトウェアの両方でレジスタ・パッキングが行われません。

リソース共有

Synplify ソフトウェアは、合成時に、デフォルトでリソース共有手法を使用して面積を削減します。**Implementation Options** ダイアログ・ボックスの **Options** タブで **Resource Sharing** オプションをオフにすると、デザインによって性能結果が改善される場合があります。また、`syn_sharing` 属性により特定のモジュールに対してこのオプションをオフにすることもできます。このオプションをオフにする場合は、結果をチェックしてタイミング性能に有効かどうか判断してください。オフにしても効果がない場合は、**Resource Sharing** をオンのままにしておく必要があります。

階層の維持

Synplify ソフトウェアは、デフォルトで境界間最適化を実行します。これにより、デザインがフラット化されて最適化が可能になります。`syn_hier` 属性を使用して、デフォルトのコンパイラ設定を無効にします。`syn_hier` 属性は文字列値を取り、それをモジュールやアーキテクチャに適用します。この値を **hard** に設定すると、モジュールやアーキテクチャの境界が保持されますが、一定の伝播が許容されます。この値を **locked** に設定すると、すべての境界間最適化は実行されません。9-54 ページの「**Synplify Pro MultiPoint シンセシスとインクリメンタル・コンパイルの併用**」で説明するとおり、**locked** 設定は、パーティション設定とともに使用して、個々のデザイン・ブロックおよび複数の出力ネットリストをインクリメンタル・コンパイル用に作成します。

デフォルトでは、Synplify ソフトウェアは、階層的な **.vqm** ファイルを生成します。ファイルをフラット化するには、`syn_netlist_hierarchy` 属性を 0 に設定します。

レジスタ入力遅延および出力遅延

`define_reg_input_delay`および`define_reg_output_delay`という高度なオプションを使用すると、レジスタに供給されるパスまたはレジスタからのパスを、特定のナノ秒数だけ高速化できます。Synplify ソフトウェアは、デザインのグローバル・クロック周波数目標や、個別のクロック周波数目標（`define_clock` で設定）を満たすよう試みます。これらの属性を使用すると、レジスタに供給するパスまたはレジスタからのパスに遅延を追加して、クリティカル・パスをさらに制約できます。また、この設定では負の数値を使用して、過剰に最適化されているパスを低速化することもできます。

これらのオプションは、配置配線後の配線遅延が Synplify ソフトウェアが予測する遅延を超過するため、デザインがタイミング目標を満たさない場合にタイミングを閉じるのに便利です。このオプションを使用して合成を再実行し、実際の配線遅延（配置配線結果から）を指定して、ツールが必要なクロック周波数を満たすようにします。Synplify では、最良の結果を得るには、これらのアサインメントを過大にしないことを推奨しています。例えば、配線遅延値を増やしても、最後のコンパイルからの完全配線遅延を使用しないでください。

SCOPE 制約ウィンドウで、以下の項目を持つレジスタ・パネルを使用します。

- **Register**— レジスタの名前を指定します。コンパイル済みデザインを初期化した場合は、リストから名前を選択できます。
- **Type**— 遅延が入力遅延または出力遅延のいずれであるかを指定します。
- **Route**— Quartus II ソフトウェアにフォワード・アノテートされるクロック周期に影響を与えることなく、制約されているレジスタの有効周期を指定された値だけ短縮します。

以下の Tcl コマンド構文を使用して、入力または出力レジスタ遅延をナノ秒で指定します。

例 9–2. Tcl コマンド構文を使用して、入力または出力レジスタ遅延を指定する

```
define_reg_input_delay {<register>} -route <delay in ns>
define_reg_output_delay {<register>} -route <delay in ns>
```

syn_direct_enable

この属性は、レジスタの専用イネーブル・ピンへのクロック・イネーブル・ネットのアサインメントを制御します。この属性を使用すると、Synplify マップに対して、デザインに複数のクロック・イネーブル候補がある場合に、特定のネットを唯一のクロック・イネーブルとして使用するように指示できます。

また、この属性をコンパイラ・ディレクティブとして使用して、クロック・イネーブルを持つレジスタを推測することもできます。これを行うには、SCOPE スプレッドシートではなく、ソース・コードに `syn_direct_enable` ディレクティブを入力します。

`syn_direct_enable` のデータ型は Boolean です。値 **1** または **true** を設定すると、クロック・イネーブル・ピンへのネット・アサインメントがイネーブルされます。Verilog HDL の構文は以下のとおりです。

```
object /* synthesis syn_direct_enable = 1 */ ;
```

標準 I/O パッド

特定のアルテラ・デバイスおよび同等のデバイス I/O 規格については、Synplify SCOPE ウィンドウの **I/O Standard** パネルを使用して、デザインの I/O パッドで使用する I/O 規格のタイプを指定できます。

例 9-3 に、`define_io_standard` 制約の Synplify SDC 構文を示します。この制約の `delay_type` は、`input_delay` または `output_delay` のいずれかでなければなりません。

例 9-3. `define_io_standard` 制約の Synplify SDC 構文

```
define_io_standard [-disable|-enable] {<objectName>} -delay_type \
[input_delay|output_delay] <columnTclName>{<value>} \
[<columnTclName>{<value>} ...]
```




サポートされている I/O 規格について詳しくは、「Synplify Reference Manual」の「Altera I/O Standards」を参照してください。

アルテラ固有の属性


以下の属性は、特定のアルテラ・デバイス機能で使用するものです。これらの属性は、Quartus II プロジェクトにフォワード・アノテートされ、配置配線プロセスで使用されます。

altera_chip_pin_lc

この属性は、ピン・アサインメントを実施するために使用します。この属性は文字列値を取り、それを入力および出力に適用します。この属性は、デザイン内のトップレベル・エンティティのポートでのみ使用でき、デザイン階層の低レベルにおけるエンティティからのピン位置を割り当てることはできません。

 この属性はどの MAX シリーズ・デバイスでもサポートされているわけではありません。SCOPE ウィンドウで、属性 **altera_chip_pin_lc** を選択し、その値をピン番号またはピン番号のリストに設定します。

例 9-4 に、ACEX 1K および FLEX 10KE デバイスに対してロケーション・アサインメントを実施する VHDL コードを示します。

 “@” 記号は、ACEX 1K および FLEX 10KE デバイスのピン・ロケーションを指定する場合にのみ、プリフィックスとして使用します。これらのデバイスの場合、ピン・ロケーション・アサインメントは、出力 **.edf** ファイルに書き込まれます。


例 9-4. ACEX 1K および FLEX 10KE デバイスに対するロケーション・アサインメントの実施 (VHDL)

```
ENTITY sample (data_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
               data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
ATTRIBUTE altera_chip_pin_lc : STRING;
ATTRIBUTE altera_chip_pin_lc OF data_out : SIGNAL IS "@14, @5,@16, @15";
```

例 9-5 に、他のアルテラ・デバイスに対してロケーション・アサインメントを実施する VHDL コードを示します。これらのデバイスのピン・ロケーション・アサインメントは、出力 Tcl スクリプトに書き込まれます。

例 9-5. その他のデバイスに対するロケーション・アサインメントの実施 (VHDL)

```
ENTITY sample (data_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
               data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
ATTRIBUTE altera_chip_pin_lc : STRING;
ATTRIBUTE altera_chip_pin_lc OF data_out : SIGNAL IS "14, 5, 16,
15";
```

 data_out 信号は 4 ビット信号で、data_out[3] はピン 14、data_out[0] はピン 15 に割り当てられます。

altera_implement_in_esb または altera_implement_in_eab

これらの属性を使用すると、ロジックをロジック・リソースではなく、エンベデッド・システム・ブロック (ESB) またはエンベデッド・アレイ・ブロック (EAB) に実装して、面積使用率を高めることができます。このような実装に選択されるモジュールは、フィードバック・パスを持つことができず、すべての I/O をラッチする必要があるか、まったくラッチしないかのいずれかです。この属性は、Boolean 値を取ってインスタンスに適用できます (このオプションは、ESB/EAB を持つデバイスにのみ適用可能。例えば、Stratix デバイス・ファミリは、このオプションでサポートされない。この属性は、ESB または EAB を持たないデバイスを対象にしたデザインでは無視される)。

altera_io_powerup

この属性を使用すると、セットまたはリセットを持たない I/O レジスタのパワーアップ値を定義できます。この属性は、文字列値 (high|low) を取りそれを I/O レジスタを持つポートに適用します。デフォルトでは、I/O のパワーアップ値は low に設定されます。

altera_io_opendrain

この属性は、オープン・ドレイン・モード I/O ポートを指定するのに使用します。この属性は、Boolean 値を取りそれをオープン・ドレイン・モードをサポートするデバイスの出力または双方向ポートに適用します。

NativeLink 統合を使用する Quartus II ソフトウェアへの デザインの エクスポート

Quartus II ソフトウェアの NativeLink 機能により、Quartus II ソフトウェアと EDA ツールの間でシームレスに情報転送が可能で、他の EDA デザイン・エントリまたは合成、シミュレーション、およびタイミング解析ツールを Quartus II ソフトウェア内から自動的に実行できます。Synplify ソフトウェアでデザインを合成した後、デザインは .vqm または .edf ネットリスト・ファイル、TimeQuest タイミング・アナライザのタイミング制約に使用する .scf ファイル、および .tcl ファイルを使用して、配置配線のために Quartus II ソフトウェアにインポートされます。Quartus II ソフトウェアは、Synplify ソフトウェア内から、あるいはスタンドアロン・アプリケーションとして実行できます。デザインを Quartus II ソフトウェアにインポートしたら、さまざまなオプションを指定して、さらにデザインを最適化することができます。



NativeLink 統合を使用する場合、プロジェクトへのパスに空白文字を含めないでください。Synplify ソフトウェアは、Tcl スクリプトを使用して、Quartus II ソフトウェアと通信します。また、Tcl 言語では、パスに空白文字を含む引数は使用できません。

NativeLink 統合を使用して、Synplify ソフトウェアと Quartus II ソフトウェアを 1 つの GUI に統合して、合成と配置配線の両方の操作を行うことができます。NativeLink 統合により、Quartus II ソフトウェアを Synplify ソフトウェア GUI 内から実行したり、Synplify ソフトウェアを Quartus II ソフトウェア GUI 内から実行することができます。

この項では、さまざまな Nativelink フローについて説明し、Quartus II ソフトウェアに制約を渡す方法について詳しく解説します。この項では、以下の内容について説明します。

- 9-20 ページの「Synplify ソフトウェア内から Quartus II ソフトウェアを実行する」
- 9-21 ページの「Quartus II ソフトウェアを使用して Synplify ソフトウェアを実行する」
- 9-22 ページの「Synplify が生成した Tcl スクリプトを使用して、Quartus II ソフトウェアを手動で実行する」
- 9-23 ページの「TimeQuest SDC タイミング制約を .scf ファイルで Quartus II ソフトウェアに渡す」
- 9-25 ページの「Tcl コマンドを使用して Quartus II ソフトウェアに制約を渡す」

Synplify ソフトウェア内から Quartus II ソフトウェアを実行する

Synplify ソフトウェア内から Quartus II ソフトウェアを使用するには、まず `QUARTUS_ROOTDIR` 環境変数に、`<Altera Design Suite Installation Directory>\quartus` にある Quartus II ソフトウェア・インストール・ディレクトリが含まれていることを確認しなければなりません。この環境変数は、Synplify ソフトウェアと Quartus II ソフトウェアを一緒に使用するのに必要です。

Windows オペレーティング・システムでは、`QUARTUS_ROOTDIR` 変数は、Quartus II ユーザー・インタフェースを開いたときに設定されるため、この変数はユーザー・インタフェースで開いた最新バージョンに自動的に設定されます。ソフトウェアを別のマシンにインストールしている場合は、この変数が正しく設定されていることを確認します。この変数は、Control Panel の System アイコンを使用して手動で変更できます。

UNIX および Linux オペレーティング・システムでは、この変数は自動的に設定されません。したがって、環境変数 `QUARTUS_ROOTDIR` を作成して、`<Altera Design Suite Installation Directory>/quartus` を参照する必要があります。

Synplify Pro ソフトウェアの各実装で、`pr_<number>` Altera Place and Route という配置配線実装を作成できます。新しい配置配線実装は、GUI の **New P&R** ボタンを使用して作成できます。各合成の実行後に、Quartus II ソフトウェアをコマンドライン・モードで実行するには、テキスト・ボックスを使用して配置配線実装をオンにします。配置配線の結果は、現在の実装ディレクトリ内の `pr_<number>` ディレクトリにあるログ・ファイルに書き込まれます。

また、合成が正常に完了した後は、Quartus II メニューのコマンドを使用して、いつでも Quartus II ソフトウェアを実行できます。Synplify ソフトウェアで、Options メニューの **Quartus II** をクリックし、以下のいずれかのコマンドを選択します。

- **Launch Quartus**—Quartus II ソフトウェア GUI を開いて、Quartus II プロジェクトを作成します。このプロジェクトには、合成された出力ファイル、フォワード・アノートされたタイミング制約、およびピン・アサインメントが含まれています。このオプションを使用して、プロジェクト用のオプションをコンフィギュレーションしたり、任意の Quartus II コマンドを実行できます。
- **Run Background Compile**—合成実行からのプロジェクト設定を使用して、Quartus II ソフトウェアをコマンドライン・モードで実行します。配置配線の結果は、ログ・ファイルに書き込まれます。

`<project_name>_cons.tcl` ファイルは、Quartus II プロジェクトをセットアップするために使用され、`<project_name>.tcl` ファイルを呼び出して、Synplify ソフトウェアから Quartus II ソフトウェアに制約を渡します。デフォルトでは、`<project_name>.tcl` ファイルには、デバイス、タイミング、およびロケーション・アサインメントが含まれています。プロジェクトが TimeQuest タイミング・アナライザを使用するようにセットアップされている場合、`<project_name>.tcl` ファイルには、Tcl 制約をクラシック・タイミング・アナライザで使用するのではなく、Synplify が生成する `.scf` 制約ファイルを TimeQuest で使用するためのコマンドが含まれています。

Quartus II ソフトウェアを使用して Synplify ソフトウェアを実行する

Quartus II ソフトウェアは、NativeLink 統合を使用して合成のために Synplify ソフトウェアを実行するように設定できます。この機能を使用すると、Quartus II ソフトウェアで通常のコンパイル処理の一部として、デザインをすばやく合成できます。この機能を使用すると、Synplify ソフトウェアは、Quartus II ソフトウェアで設定したインクリメンタル・コンパイル・パーティションなどのタイミング制約やアサインメントを使用しません。



Synplify では、最良の結果を得るために、制約は Synplify ソフトウェアで設定し、Quartus II ソフトウェアには Tcl スクリプトを使用して渡すことを推奨しており、Quartus II ソフトウェア内から Synplify を呼び出す方法は推奨していません。

Synplify を Quartus II で設定するには、Tools メニューの **Options** をクリックします。**Options** ダイアログ・ボックスで、**EDA Tool Options** をクリックし、Synplify または Synplify Pro ソフトウェアのパスを指定します。



Synplify ソフトウェアでの NativeLink 統合の使用方法については、Quartus II Help を参照してください。

Quartus II ソフトウェア v7.1 から、Synplify ソフトウェアを NativeLink 統合で実行できるようになり、フローティング・ネットワーク・ライセンスとノード・ロックされたシングル PC ライセンスの両方でサポートされています。両種類のライセンスともバッチ・モード・コンパイルをサポートしています。

Synplify が生成した Tcl スクリプトを使用して、Quartus II ソフトウェアを手動で実行する

Quartus II ソフトウェアは、Synplify ソフトウェアとは別々に使用することもできます。Synplify ソフトウェアが生成する Tcl スクリプトを実行して、プロジェクトのセットアップおよびデバイスの選択などのアサインメントの設定を行うには、以下のステップを実行します。

1. **.vqm/.edf**、**.scf** (TimeQuest タイミング・アナライザのタイミング制約を使用している場合)、および Tcl ファイルが同じディレクトリ内に配置されている (これらのファイルは、デフォルトでは実装ディレクトリに配置されている必要がある) ことを確認します。
2. Quartus II ソフトウェアで、View メニューの **Utility Windows** をポイントし、**Tcl Console** をクリックします。Quartus II Tcl Console が開きます。
3. Tcl Console のコマンド・プロンプトで、以下のように入力します。

```
source <path>/<project name>_cons.tcl
```

TimeQuest SDC タイミング制約を .scf ファイルで Quartus II ソフトウェアに渡す

TimeQuest タイミング・アナライザは、ASIC スタイルの強力なタイミング解析ツールで、業界標準の制約フォーマットである Synopsys Design Constraint (SDC) を使用して、デザイン内のすべてのロジックのタイミング性能を検証します。この項では、Synplify ソフトウェアで設定するタイミング制約が、どのような方法で Quartus II ソフトウェアに渡されて、TimeQuest タイミング・アナライザで使用されるかについて説明します。

Synplify ソフトウェアで設定するタイミング制約は、Synplify Design Constraint (.sdc) ファイルに格納されます。.tcl ファイルには常に、デバイス指定やロケーション制約など、Quartus II ソフトウェアに対する他のすべての制約が含まれます。タイミング制約は、Quartus II クラシック・タイミング・アナライザ用の .tcl ファイルを使用して、フォワード・アノテートされます (9-25 ページの「Tcl コマンドを使用して Quartus II ソフトウェアに制約を渡す」を参照)。TimeQuest タイミング・アナライザの場合、タイミング制約は Synopsys Constraints Format (.scf) ファイルでフォワード・アノテートされます。

アルテラでは、TimeQuest タイミング・アナライザの使用方法について、Quartus II プロジェクトを最新のデバイスに対応するように設定する Synplify .tcl ファイルに準拠することを推奨しています。ただし、必要に応じて、クラシック・タイミング・アナライザの Tcl コマンドを使用することもできます。Quartus II ソフトウェアで、TimeQuest タイミング・アナライザからクラシック・タイミング・アナライザに手動で変更するには、以下のステップを実行します。

1. Assignments メニューから **Settings** をクリックします。
2. **Category** リストで、**Timing Analysis Settings** を選択します。
3. **Timing analysis processing** で、**Use Classic Timing Analyzer during compilation** を選択します。OK をクリックします。



TimeQuest タイミング・アナライザについて詳しくは、「Quartus II ハンドブック Volume 3」の「**Quartus II TimeQuest タイミング・アナライザ**」の章を参照してください。

Synplicity では、制約を変更する場合は、生成された .sdc、.scf、または .tcl ファイルを使用するのではなく、SCOPE 制約エディタ・ウィンドウを使用することを推奨しています。

以下に示す Synplify 制約は、同等の Quartus II SDC コマンドに変換され、.scf ファイル内で Quartus II にフォワード・アノテートされます。

- `define_clock`
- `define_input_delay`
- `define_output_delay`
- `define_multicycle_path`
- `define_false_path`

以下の項で説明する Synplify 制約ではすべて、9-25 ページの「Tcl コマンドを使用して Quartus II ソフトウェアに制約を渡す」で説明するコマンドと同じ Synplify コマンドを使用しています。ただし、制約は TimeQuest タイミング・アナライザ用の SDC コマンドにマップされます。



これらのコマンドの構文および引数については、該当する項を参照するか、Synplify Help を参照してください。Quartus II ソフトウェアで対応するコマンドのリストについては、Quartus II Help を参照してください。

個別クロックおよび周波数

Synplify ソフトウェアでは、`define_clock` コマンドを使用して、個々のクロックのクロック周波数を指定できます。このコマンドは、`create_clock` によって Quartus II ソフトウェアに渡されます。

入力および出力遅延

Synplify ソフトウェアでは、`define_input_delay` および `define_output_delay` コマンドを使用して、入力遅延および出力遅延の制約を指定できます。これらのコマンドは、`set_input_delay` および `set_output_delay` 使用して、Quartus II ソフトウェアに渡されます。

マルチサイクル・パス

Synplify ソフトウェアでは、`define_multicycle_path` コマンドを使用して、マルチサイクル・パス制約を指定できます。このコマンドは、`set_multicycle_path` によって Quartus II ソフトウェアに渡されます。

フォルス・パス

Synplify ソフトウェアでは、`define_false_path` コマンドを使用して、フォルス・パス制約を指定できます。このコマンドは、`set_false_path` によって Quartus II ソフトウェアに渡されます。


Tcl コマンドを使用して Quartus II ソフトウェアに制約を渡す

この項では、Synplify 制約がどのような方法で同等の Quartus II アサインメントに変換され、Tcl コマンドによって Quartus II ソフトウェアにフォワード・アノテートされるかについて説明します。

この項では、Quartus II クラシック・タイミング・アナライザのためのタイミング制約について説明します。TimeQuest タイミング・アナライザを使用する場合は、この項で説明する Quartus II タイミング制約は適用されません。TimeQuest でサポートされるタイミング制約については、9-23 ページの「TimeQuest SDC タイミング制約を .scf ファイルで Quartus II ソフトウェアに渡す」を参照してください。

グローバル信号

Synplify ソフトウェアは、クロック信号を自動的にグローバル配線ラインに昇格させ、**Global Signal** アサインメントを Quartus II ソフトウェアに渡します。このアサインメントにより、配置配線時に、同じグローバル配線制約が適用されます。

 グローバル配線に昇格された信号と、Quartus II ソフトウェアがデフォルトでグローバル配線に昇格させた信号は、異なる場合があります。Synplify ソフトウェアが昇格させるのはクロック信号のみで、リセットやイネーブルなどの他のコントロール信号は昇格させません。Synplify ソフトウェアからの制約がない場合、Quartus II ソフトウェアはデフォルトにより、ファンアウトが大きいコントロール信号をグローバル配線に昇格させます。

デフォルトまたはグローバル・クロック周波数

以下の Synplify コマンドを使用して、プロジェクト全体に適用する Synplify のデフォルトまたはグローバル・クロック周波数を設定します。

```
set_option -frequency <frequency>
```

<frequency> は MHz 単位で指定されます。グローバル周波数が指定されていない場合は、デフォルトのグローバル・クロック周波数 1 MHz が使用されます。

set_option 制約は、以下のコマンドで Quartus II ソフトウェアに渡されます。

```
set_global_assignment -name FMAX_REQUIREMENT \
<frequency> MHz
```

Quartus II ソフトウェアに周波数が指定されていない場合、デフォルトのグローバル・クロック周波数 1 GHz が使用されます。

個別クロックおよび周波数

以下の Synplify コマンドを使用して、個々のクロックのクロック周波数を指定できます。

例 9-6. 個別クロックのクロック周波数の指定

```
define_clock -name {<clock_name>} -freq <frequency> -clockgroup <clock_group> \
-rise <rise_time> -fall <fall_time>
define_clock -name {<clock_name>} -period <period> -clockgroup <clock_group> \
-rise <rise_time> -fall <fall_time>
```

表 9-3 に、コマンド引数を示します。

| 引数 | 説明 |
|----------------|--|
| -name | <clock_name> は、デザイン・ポート名またはレジスタ出力信号名を指定し、合成後は <mapped_clock_name> に対応します。 |
| -freq (1) | <frequency> は MHz 単位で指定されます。 |
| -period (2) | <period> は ns 単位で指定されます。 |
| -clockgroup | <clock_group> が指定されていない場合は、デフォルトで default_clkgroup が指定されます。Synplify ソフトウェアは、同じクロック・グループに属するすべてのクロックが関連すると想定しています。クロック・グループを指定しない場合、クロックはデフォルトのクロック・グループに属します。したがって、クロック・グループを指定しない場合、すべてのクロックはデフォルトで関連するとみなされます。 |
| -rise -fall | <rise_time> および <fall_time> は、デフォルト以外のデューティ・サイクルを指定します。Synplify 合成ツールがデフォルトで想定するクロックは、デューティ・サイクル 50% クロック、立ち上がりエッジ 0、立ち下がりエッジは周期 /2 です。別のデューティ・クロック・サイクルがある場合は、適切な Rise At および Fall At 値を指定できます。 |

表 9-3 の注:

- (1) <frequency> が指定されているとき、Synplify ソフトウェアは、<fall_time> および <frequency> を使用して duty_cycle を計算します。計算式は、 $duty_cycle = (\langle fall_time \rangle - \langle rise_time \rangle) \times \langle frequency \rangle \div 10$ です。
- (2) <period> が指定されているとき、Synplify ソフトウェアは、<fall_time> および <period> を使用して duty_cycle を計算します。計算式は、 $duty_cycle = 100 \times (\langle fall_time \rangle - \langle rise_time \rangle) \div \langle period \rangle$ です。

Quartus II クラシック・タイミング・アナライザの同等のコマンドは、クロック・グループがどのように定義されるかによって異なります。Quartus II ソフトウェアでは、同じまたは関連するクロック設定に属するクロックは、関連クロックとみなされます。関連しないクロック設定に割り当てられるクロックは、非関連クロックです。各 Quartus II クロック設定と Synplify クロック・グループは、一対一で対応します。



以下の項では、周波数制約についてのみ説明します。周期には対応する制約を使用できます。

仮想クロック

Quartus II ソフトウェアは、仮想クロックをサポートします。Synplify ソフトウェアで仮想クロック設定を使用する場合、設定は Quartus II ソフトウェアの制約にマップされます。

配線遅延オプション

Synplify ソフトウェアのクロック制約の `-route` オプションは、配置配線後の配線遅延が Synplify ソフトウェアで予測される遅延を超過するために、タイミング目標を満足しない場合にのみ、合成で使用するように設計されています。この制約は、Quartus II ソフトウェアにフォワード・アノテートする必要はありません。

さまざまなクロック・グループでの複数クロック

例 9-7 に示す Synplify コマンドを使用して、複数クロックのクロック周波数を指定できます。

例 9-7. 複数クロックのクロック周波数の指定

```
define_clock -name {<clock_name1>} -freq <frequency1> \  
-clockgroup <clock_group1> -rise <rise_time1> -fall <fall_time1>
```

```
define_clock -name {<clock_name2>} -freq <frequency2> \  
-clockgroup <clock_group2> -rise <rise_time2> -fall <fall_time2>
```

<clock_group1> および <clock_group2> は、Synplify ソフトウェアで定義される固有の名前で、Quartus II クラシック・タイミング・アナライザのベース・クロック設定で使用されます。

クロック *<rise_time>* がゼロ (“0”) の場合、各クロックは、例 9-8 に示すコマンドによって、別々に Quartus II ソフトウェアに渡されます。

例 9-8. クロックの立ち上がり時間がゼロの場合の複数クロックに対する Quartus II アサインメント

```
create_base_clock -fmax <frequency1>MHz -duty_cycle <duty_cycle1> \  
-target mapped_clock_name1 <base_clock_setting1>  
  
create_base_clock -fmax <frequency2>MHz -duty_cycle <duty_cycle2> \  
-target mapped_clock_name2 <base_clock_setting2>
```

クロック *<rise_time>* がゼロ以外の場合、各クロックは、例 9-9 に示す以下のコマンドによって個別に Quartus II ソフトウェアに渡されます。

例 9-9. クロックの立ち上がり時間がゼロ以外の場合の複数クロックに対する Quartus II アサインメント

```
create_base_clock -fmax <frequency1>MHz -duty_cycle <duty cycle1> \  
-no_target <base clock setting1>  
  
create_base_clock -fmax <frequency2>MHz -duty_cycle <duty cycle2> \  
-no_target <base clock setting2>  
  
create_relative_clock -base_clock <base clock setting1> -offset <rise time1>ns \  
-duty_cycle <duty cycle1> -multiply <multiply by> -divide <divide by> \  
-target <mapped clock name1> <derived clock setting1>  
  
create_relative_clock -base_clock <base clock setting2> -offset <rise time2>ns \  
-duty_cycle <duty cycle2> -multiply <multiply by> -divide <divide by> \  
-target <mapped clock name2> <derived clock_setting2>
```

同じクロック・グループ内で周波数が異なる複数クロック

Synplify ソフトウェアでは、同じクロック・グループ内に、相対クロック設定を持ち周波数が異なる複数のクロックを指定できます。このコマンドを例 9-10 に示します。

例 9-10. 同じクロック・グループ内で周波数が異なる複数のクロックを指定

```
define_clock -name {<clock_name1>} -freq <frequency1> -clockgroup <clock_group1> \  
-rise <rise_time1> -fall <fall_time1>  
  
define_clock -name {<clock_name2>} -freq <frequency2> -clockgroup <clock_group1> \  
-rise <rise_time2> -fall <fall_time2>
```



同じクロック・グループに周波数が異なるクロックを指定すると、クロック・グループ設定で `<frequency1>` および `<frequency2>` から相対クロック設定のために、`<multiply_by>` および `<divide_by>` 係数が計算されます。

クロックの `<rise_time>` がゼロの場合、同じグループ内で周波数が異なる相対クロック設定を持つ複数のクロックが、例 9-11 に示すコマンドで Quartus II ソフトウェアに渡されます。

例 9-11. 同じクロック・グループ内で周波数が異なる複数クロックに対する Quartus II

アサインメント (クロックの立ち上がり時間がゼロの場合)

```
create_base_clock -fmax <frequency1>MHz -duty_cycle <duty_cycle1> \
-target <mapped_clock_name1> <base_clock_setting1>

create_relative_clock -base_clock <base_clock_setting1> \
-duty_cycle <duty_cycle2> -multiply <multiply_by> -divide <divide_by> \
-target <mapped_clock_name2> <derived_clock_setting2>
```

インター・クロック関係 — クロック間の遅延およびフォルス・パス

例 9-12 に示すコマンドを使用して、Synplify でクロック間遅延制約を設定できます。

例 9-12. クロック間遅延制約の指定

```
define_clock_delay -fall <clock_name1> -rise <clock_name2> <delay_value>
define_clock_delay -rise <clock_name1> -fall <clock_name2> <delay_value>
define_clock_delay -rise <clock_name1> -rise <clock_name2> <delay_value>
define_clock_delay -fall <clock_name1> -fall <clock_name2> <delay_value>
```

`<delay_value>` を `false` に設定すると、Synplify でのこれらの制約は、2つのクロック間のフォルス・パスを指定します。Synplify ソフトウェアで、立ち上がり / 立ち下がりクロック・エッジの4つのペアがすべて指定されている場合、Synplify の制約は、Quartus II ソフトウェアの以下の制約にマップされます。

```
set_timing_cut_assignment -from <clock_name1> \
-to <clock_name2>
```

Synplify で、4つのクロック・エッジのペアの一部が指定されていない場合、その制約を Quartus II クラシック・タイミング・アナライザの制約にマップすることはできません。

<delay_value> が false 以外の値に設定される場合、Synplify でのこれらの制約は、Quartus II ソフトウェアの制約にマップされません。Quartus II クラシック・タイミング・アナライザは、クロック・エッジ間の遅延制約をサポートしません。

フォルス・パス

以下のコマンドを使用して、Synplify ソフトウェアでフォルス・パス制約を指定できます。

```
define_false_path -from <sig_name1> -to <sig_name2>
```

<sig_name1> および <sig_name2> 信号には、デザイン・ポート名またはレジスタ・インスタンス名を指定できます。

Synplify ソフトウェアの **define_false_path** 制約は、以下に示すように、Quartus II ソフトウェアの制約にマップされます。

```
set_timing_cut_assignment -from <sig_name1> \  
-to <sig_name2>
```

Synplify ソフトウェアは、信号セットのペアを、これらの 2 つのセットの製品間の各メンバが有効なフォルス・パス制約になるように識別できます。信号グループは、以下のコマンドを使用して、Quartus II クラシック・タイミング・アナライザで定義できます。

```
timegroup -add_member sig_name1_i <sig_group1>  
(<sig_group1> のすべての信号が対象)
```

```
timegroup -add_member sig_name2_i <sig_group2>  
(<sig_group2> のすべての信号が対象)
```

```
set_timing_cut_assignment -from <sig_group1> \  
-to <sig_group2>
```

<sig_name1> または <sig_name2> 信号が、ワイルドカード、グループ、またはバスなどの複数の信号を表す場合、制約は Quartus II ソフトウェアで表現するために適宜拡張できます。Quartus II ソフトウェアはタイミング・アサインメントのために、ワイルドカードを使用した信号名と、信号グループをサポートしています。Quartus II ソフトウェアは、A[7:4] などのバス表記はサポートしていません。

信号からのフォルス・パス

以下のコマンドを使用して、Synplify ソフトウェアで信号からのフォルス・パス制約を指定できます。

```
define_false_path -from <sig_name>
```

Quartus II クラシック・タイミング・アナライザは、“-from” のみのパス指定はサポートしていません。“-to” パスの指定も含める必要があります。ただし、-to 信号にはワイルドカードを指定できます。Synplify でのこの制約は、Quartus II ソフトウェアの以下の制約にマップされます。

```
set_timing_cut_assignment -from <sig_name> -to {*}
```

信号へのフォルス・パス

以下のコマンドを使用して、Synplify ソフトウェアで信号へのフォルス・パス制約を指定できます。

```
define_false_path -to <sig_name>
```

Quartus II クラシック・タイミング・アナライザは、-to のみのパス指定はサポートしていません。-from パスの指定も含める必要があります。ただし、-from 信号にはワイルドカードを指定できます。Synplify ソフトウェアでのこの制約は、Quartus II ソフトウェアの以下の制約にマップされます。

```
set_timing_cut_assignment -from {*} -to <sig_name>
```

信号を介したフォルス・パス

以下のコマンドを使用して、Synplify ソフトウェアで信号を介したフォルス・パス制約を指定できます。

```
define_false_path -from <sig_name1> -to <sig_name2> \  
-through <sig_name3>
```

Quartus II クラシック・タイミング・アナライザは、“スルー・パス”を指定したフォルス・パスをサポートしていません。-through が指定された Synplify ソフトウェアの制約は、Quartus II タイミング・アナライザのための制約にマップされません。

マルチサイクル・パス


以下のコマンドを使用して、Synplify ソフトウェアでマルチサイクル・パス制約を指定できます。

```
define_multicycle_path -from <sig_name1> \  
-to <sig_name2> <clock_cycles>
```

Synplify ソフトウェアでのこの制約は、Quartus II ソフトウェアの以下の制約にマップされます。

```
set_multicycle_assignment -from <sig_name1> \  
-to <sig_name2> <clock_cycles>
```

<sig_name1> または <sig_name2> 信号が、ワイルドカード、グループ、またはバスなどの複数の信号を表す場合、制約は Quartus II ソフトウェアで表現するために適切に適宜拡張できます (9-12 ページの「フォルス・パス」で説明)。

 <clock_cycles> は、マルチサイクル・パスのクロック・サイクル数です。

信号からのマルチサイクル・パス

以下のコマンドを使用して、Synplify ソフトウェアで信号からのマルチサイクル・パス制約を指定できます。

```
define_multicycle_path -from <sig_name> <clock_cycles>
```

フォルス・パス制約の場合と同様、この制約は Quartus II クラシック・タイミング・アナライザで -to 値にワイルドカードを使用してマップされます。

```
set_multicycle_assignment -from <sig_name> \  
-to {*} <clock_cycles>
```

信号へのマルチサイクル・パス

以下のコマンドを使用して、Synplify ソフトウェアで信号へのマルチサイクル・パス制約を指定できます。

```
define_multicycle_path -to <sig_name> <clock_cycles>
```

フォルス・パス制約の場合と同様、この制約は Quartus II クラシック・タイミング・アナライザで -from 値にワイルドカードを使用してマップされます。

```
set_multicycle_assignment -from {*} <sig_name> \  
<clock_cycles>
```

信号を介したマルチサイクル・パス

以下のコマンドを使用して、Synplify ソフトウェアで信号を介したマルチサイクル・パス制約を指定できます。

```
define_multicycle_path -from <sig_name1> -to <sig_name2> \  
-through <sig_name3> <clock_cycles>
```

Quartus II クラシック・タイミング・アナライザは、“スルー・パス”を指定したマルチサイクル・パスをサポートしていません。`-through` が指定された Synplify ソフトウェアの制約は、Quartus II タイミング・アナライザのための制約にマップされません。

最大パス遅延

以下のコマンドを使用して、Synplify ソフトウェアで信号間の最大パス遅延関係を指定できます。

```
define_path_delay -from <sig_name1> -to <sig_name2> \  
-max <delay_value>
```

Synplify ソフトウェアでのこの制約は Quartus II ソフトウェアの以下の制約にマップされます。

```
set_instance_assignment -from <sig_name1> \  
-to <sig_name2> -name SETUP_RELATIONSHIP <delay_value>ns
```

Quartus II クラシック・タイミング・アナライザは、信号グループまたはパス表記をサポートせず、この制約のためにレジスタ名のみサポートします。

信号からの最大パス遅延

以下のコマンドを使用して、Synplify ソフトウェアで信号からの最大パス遅延制約を指定できます。

```
define_path_delay -from <sig_name> -max <delay_value>
```

フォルス・パス制約の場合と同様、この制約は Quartus II クラシック・タイミング・アナライザで `-to` 値にワイルドカードを使用してマップされます。

```
set_instance_assignment -from <sig_name> -to {*} \  
-name SETUP_RELATIONSHIP <delay_value>ns
```

信号への最大パス遅延

以下のコマンドを使用して、Synplify ソフトウェアで信号への最大パス遅延制約を指定できます。

```
define_path_delay -to <sig_name> -max <delay_value>
```

フォルス・パス制約の場合と同様、この制約は Quartus II クラシック・タイミング・アナライザで `-from` 値にワイルドカードを使用してマップされます。

```
set_instance_assignment -from {*}<sig_name> \  
-to <sig_name> -name SETUP_RELATIONSHIP <delay_value>ns
```

信号を介した最大パス遅延

以下のコマンドを使用して、Synplify ソフトウェアで信号経路の最大パス遅延制約を指定できます。

```
define_path_delay -from <sig_name1> -to <sig_name2> \  
-through <sig_name3> -max <delay_value>
```

Quartus II クラシック・タイミング・アナライザは、“スルー・パス”を指定した最大パス遅延制約をサポートしていません。`-through` が指定された Synplify の制約は、Quartus II タイミング・アナライザのための制約にマップされません。

レジスタ入力および出力遅延

Synplify ソフトウェアでのレジスタ入力遅延およびレジスタ出力遅延の制約は、合成でのみ使用するためのものなので、Quartus II ソフトウェアにフォワード・アノテートされません。

デフォルトの外部入力遅延

以下のコマンドを使用して、Synplify ソフトウェアでデフォルトの入力遅延制約を指定できます。

```
define_input_delay -default <delay_value>
```

この制約は、Quartus II ソフトウェアの以下の制約にマップされます。

```
set_input_delay -clock {*} <delay_value> {*}
```

ポート固有の外部入力遅延

以下のコマンドを使用して、Synplify ソフトウェアでポート固有の入力遅延制約を指定できます。

```
define_input_delay <input_port_name> <delay_value> \  
-ref <clock_name>:<clock_edge>
```

`<clock_edge>` は、r (立ち上がりエッジ) または f (立ち下がりエッジ) に設定できます。

クロック・エッジが *r* (立ち上がりエッジ) の場合、この制約は Quartus II ソフトウェアの以下の制約にマップされます。

```
set_input_delay -clock <clock_name> <delay_value> \  
<input_port_name>
```

<clock_edge> が *f* (立ち下がりエッジ) の場合、この制約は Quartus II ソフトウェアの制約にマップされません。Quartus II クラシック・タイミング・アナライザは、クロックの立ち下がりエッジに対する入力遅延の指定をサポートしません。

デフォルトの外部出力遅延

以下のコマンドを使用して、Synplify ソフトウェアでデフォルトの出力遅延制約を指定できます。

```
define_output_delay -default <delay_value>
```

この制約は、Quartus II ソフトウェアの以下の制約にマップされます。

```
set_output_delay -clock {*} <delay_value> {*}
```

ポート固有の外部出力遅延

以下のコマンドを使用して、Synplify ソフトウェアでポート固有の入力遅延制約を指定できます。

```
define_output_delay <output_port_name> <delay_value> \  
-ref <clock_name>:<clock_edge>
```

<clock_edge> は、*r* (立ち上がりエッジ) または *f* (立ち下がりエッジ) に設定できます。クロック・エッジが *r* (立ち上がりエッジ) の場合、この制約は Quartus II ソフトウェアの以下の制約にマップされます。

```
set_output_delay -clock <clock_name> <delay_value> \  
<output_port_name>
```

クロック・エッジが *f* (立ち下がりエッジ) の場合、この制約は Quartus II ソフトウェアの制約にマップされません。Quartus II クラシック・タイミング・アナライザは、クロックの立ち下がりエッジに対する出力遅延の指定をサポートしません。

アルテラ・ メガファンク ションおよび アーキテクチャ 固有の機能 に関する ガイドライン

アルテラでは、LPM、デバイス固有のアルテラ・メガファンクション、アルテラ MegaCore® ファンクションとして使用可能な IP、AMPPSM (Altera Megafunction Partners Program) を通じて使用可能な IP などのパラメータ化可能なメガファンクションを提供しています。これらのメガファンクションおよび IP ファンクションは、HDL コードでインスタンス化して使用できます。あるいは、汎用 HDL コードから特定のメガファンクションを推測できます。

HDL で、メガファンクションをインスタンス化する場合は、MegaWizard Plug-In Manager でインスタンス化してファンクションをパラメータ化するか、ポートおよびパラメータ定義を使用してファンクションをインスタンス化できます。MegaWizard Plug-In Manager は、Quartus II ソフトウェア内でデザインに使用可能なメガファンクションをカスタマイズおよびパラメータ化するためのグラフィカル・インタフェースを提供します。9-37 ページの「[MegaWizard Plug-In Manager を使用したアルテラ・メガファンクションのインスタンス化](#)」および 9-39 ページの「[MegaWizard Plug-In Manager および IP Toolbench を使用した Intellectual Property \(IP\) のインスタンス化](#)」では、Synplify ソフトウェアでの MegaWizard Plug-In Manager フローについて説明します。



特定のアルテラ・メガファンクションについて詳しくは、Quartus II Help を参照してください。IP ファンクションについて詳しくは、該当する IP マニュアルを参照してください。

また、Synplify ソフトウェアは、特定タイプの HDL コードを自動的に認識し、メガファンクションが最適な結果を提供するときは適切なメガファンクションを推測します。Synplify ソフトウェアには、特定タイプのメガファンクションの推測を制御するオプションがあります。これについては 9-44 ページの「[HDL コードからのアルテラ・メガファンクションの推測](#)」で説明します。




メガファンクションのインスタンス化と推測について詳しくは、「Quartus II ハンドブック Volume 1」の「[Recommended HDL Coding Styles](#)」の章を参照してください。この章では、Quartus II ソフトウェアでの MegaWizard Plug-In Manager の使用方法について詳しく説明し、ウィザードで生成されるファイルについて解説します。さらに、推奨コーディング・スタイル、およびアルテラ・デバイスでメガファンクションを推測するための HDL の例についても紹介します。

MegaWizard Plug-In Manager を使用したアルテラ・メガファンクションのインスタンス化

この項では、MegaWizard Plug-In Manager を使用してアルテラ・メガファンクションをインスタンス化する方法について説明します。

MegaWizard Plug-In Manager を使用してメガファンクションをセットアップおよびパラメータ化するときは、MegaWizard Plug-In Manager は、メガファンクションをインスタンス化する VHDL または Verilog HDL ラッパー・ファイル `<output file>.v|vhd` を作成します。

Synplify ソフトウェアは、Quartus II タイミングおよびリソース見積もりネットリスト機能を使用して、リソース使用率およびタイミング性能見積もりをより正確にレポートし、メガファンクションを“ブラック・ボックス”として扱う場合よりも良好にタイミング・ドリブン最適化を利用します。MegaWizard が生成するメガファンクション・バリエーション・ラッパー・ファイルを Synplify プロジェクトに取り込んで、Synplify ソフトウェアがメガファンクションに関するすべての情報を持つようにする必要があります。

 MegaWizard Plug-In Manager には、リソースおよびタイミング見積もり用のネットリストを生成するオプションがあります。Synplify ソフトウェアは、リソースおよびタイミング見積もりに関する情報を個別のネットリストがなくてもバックグラウンドで生成するため、このオプションは Synplify ソフトウェアには推奨されません。実際に、個別のネットリスト `<output file>_syn.v|vhd` を作成して、そのファイルを合成プロジェクトで使用する場合は、Quartus II プロジェクトに `<output file>.v|vhd` ファイルも含める必要があります。

Synplify ソフトウェアに適切な Quartus II バージョンを設定してから、MegaWizard が生成したファイルをコンパイルします。これにより、Synplify ソフトウェアはメガファンクションに対して適切なライブラリ定義を使用します。Quartus Version 設定は、MegaWizard Plug-In Manager でカスタマイズされたメガファンクションを生成するのに使用する Quartus II ソフトウェアのバージョンと一致する必要があります。

Synplify ソフトウェアで Quartus II バージョンを設定する方法について詳しくは、[9-7 ページ](#)の「出力ネットリスト・ファイル名と結果の形式」を参照してください。

また、QUARTUS_ROOTDIR 環境変数に、適切な Quartus II バージョンのインストール・ディレクトリが設定されていることを確認します。Synplify ソフトウェアは、この情報を使用して Quartus II ソフトウェア

をバックグラウンドで起動します。この環境変数の設定は、MegaWizard Plug-In Manager でカスタマイズされたメガファンクションを生成するために使用する Quartus II ソフトウェアのバージョンと一致する必要があります。詳細は、9-21 ページの「[Quartus II ソフトウェアを使用して Synplify ソフトウェアを実行する](#)」を参照してください。

MegaWizard Plug-In Manager で生成された Verilog HDL ファイルを使用した、メガファンクションのインスタンス化

ウィザードの最後のページで、<output file>_inst.v オプションをチェックすると、MegaWizard Plug-In Manager は、Synplify デザインで使用するための Verilog HDL インスタンス化テンプレート・ファイルを生成します。インスタンス化テンプレート・ファイルを使用すると、メガファンクション・バリエーション・ラッパー・ファイル <output file>.v をトップレベル・デザインで容易にインスタンス化できます。Synplify プロジェクトにメガファンクション・バリエーション・ラッパー・ファイル <output file>.v を取り込みます。これにより、Synplify ソフトウェアは、output.vqm ネットリスト・ファイルにメガファンクション情報を取り込みます。MegaWizard で生成されたメガファンクション・バリエーション・ファイルを Quartus II プロジェクトに取り込む必要はありません。

MegaWizard Plug-In Manager で生成される VHDL ファイルを使用した、メガファンクションのインスタンス化

ウィザードの最後のページで、<output file>.cmp および <output file>_inst.vhd オプションをチェックすると、MegaWizard Plug-In Manager は、Synplify デザインで使用するための VHDL コンポーネント宣言ファイルおよび VHDL インスタンス化テンプレート・ファイルを生成します。これらのファイルを使用すると、メガファンクション・バリエーション・ラッパー・ファイル <output file>.vhd をトップレベル・デザインで容易にインスタンス化できます。Synplify プロジェクトにメガファンクション・バリエーション・ラッパー・ファイル <output file>.vhd を取り込みます。これにより、Synplify ソフトウェアは、output.vqm ネットリスト・ファイルにメガファンクション情報を取り込みます。MegaWizard で生成されたメガファンクション・バリエーション・ファイルを Quartus II プロジェクトに取り込む必要はありません。

インスタンス化したアルテラ・メガファンクションに対する Synplify のデフォルトの動作の変更

Synplify ソフトウェアは、デフォルトにより、バックグラウンドで Quartus II ソフトウェアを呼び出して、メガファンクション用のリソースおよびタイミング見積もりネットリストを生成します。これについては前項で説明しています。

この動作を変更して、Synplify ソフトウェアの実行時間を短縮することができます（大規模デザインの場合、ネットリスト・ファイルを生成するのに数分かかることがあります）。また、この動作は、Synplify ソフトウェアが Quartus II ソフトウェアにアクセスしてファイルを生成できない場合も変更できます。

Synplify ソフトウェアは、Quartus II ソフトウェアを呼び出して、2 通りの方法で情報を生成します。メガファンクションによっては、“クリア・ボックス”・モデルを提供するものがあります。Synplify ソフトウェアは、このモデルを完全に合成し、output.vqm ネットリスト・ファイルにデバイス・アーキテクチャ固有のプリミティブを取り込みます。メガファンクションによっては、“グレイ・ボックス”・モデルを提供するものもあります。Synplify ソフトウェアは、リソース情報を読み出しますが、ネットリストにすべてのロジック・ファンクションが含まれるわけではありません。これらのファンクションのために、Synplify ソフトウェアはリソースとタイミングの見積もりおよび最適化にロジック情報を使用し、さらに output.vqm ネットリスト・ファイルのメガファンクションをインスタンス化するため、Quartus II ソフトウェアは適切なデバイス・プリミティブを実装できるようになります。デフォルトでは、Synplify ソフトウェアはクリア・ボックス・モデルを使用しますが、使用できない場合はグレイ・ボックス・モデルを使用します。この動作を変更するには、**Implementation Options** をクリックし、**Device** タブで **Altera Models** 設定を変更します。デフォルトは **on** です。クリア・ボックス・モデルをイネーブルし、グレイ・ボックスをイネーブルしないようにするには、**clearbox_only** を選択します。また、この機能全体をオフにするには、**off** を選択します。

MegaWizard Plug-In Manager および IP Toolbench を使用した Intellectual Property (IP) のインスタンス化

多くのアルテラ IP ファンクションには、リソース使用率とタイミング性能の見積もりをより正確にレポートするために Synplify ソフトウェアで利用できるリソースおよびタイミング見積もりネットリストが含まれており、ブラック・ボックス・ファンクションよりも良好にタイミング・ドリブン最適化を利用します。

このネットリスト・ファイルを作成するには、まず MegaWizard Plug-In Manager で IP ファンクションを選択し、**Next** をクリックして IP Toolbench を開きます。**Step 2: Set Up Simulation** をクリックします。ここで、EDA オプションをすべて設定します。**Generate netlist** オプションをイネーブルにして、リソースおよびタイミングの見積もりのためのネットリストを生成します。このネットリスト・ファイルは、**Step 3: Generate** をクリックすると生成されます。

Quartus II ソフトウェアは、ファイル `<output file>_gb.v|vhd` を生成します。このネットリストには、リソースとタイミングを見積もるための“グレイ・ボックス”情報が含まれていますが、実際の実装は含まれていません。このネットリスト・ファイルを Synplify プロジェクトに取り込みます。次に、Quartus II プロジェクトに、Synplify .vqm 出力ネットリストとともに、メガファンクション・バリエーション・ラッパー・ファイル `<output file>.v|vhd` を取り込みます。

IP ファンクションに、リソースおよびタイミング見積もりネットリストが含まれていない場合、Synplify ソフトウェアは、IP ファンクションをブラック・ボックスとして扱う必要があります。この場合、ブラック・ボックスの作成について詳しくは、以下の項を参照してください。

Synplify プロジェクトに Quartus II 固有のファイルを取り込んで、これらのファイルを `output.vqm` ファイルとともに Quartus II ソフトウェアに自動的に渡す方法については、[9-43 ページの「Quartus II 配置配線専用ファイルのインクルード」](#)を参照してください。

生成された Verilog HDL ファイルを使用した、 ブラック・ボックス IP ファンクションのインスタンス化

`syn_black_box` コンパイラ・ディレクティブを使用して、モジュールをブラック・ボックスとして宣言できます。トップレベル・デザイン・ファイルには IP ポート・マッピングと中空体モジュール宣言が含まれていなければなりません。`syn_black_box` ディレクティブを、トップレベル・ファイルまたはプロジェクトに含まれる別のファイル内のモジュール宣言に適用して、Synplify ソフトウェアにこれがブラック・ボックスであることを指示します。このソフトウェアは、このディレクティブがなくても正常にコンパイルしますが、追加の警告メッセージがレポートされます。このディレクティブを使用すると、他のディレクティブを追加できます ([9-42 ページの「ブラック・ボックスを作成するための他の Synplify ソフトウェアの属性」](#)を参照)。

例 9-13 に、**my_verilogIP.v** (MegaWizard Plug-In Manager および IP Toolbench) によって生成される簡略化カスタマイズ・バリエーションをインスタンス化するトップレベル・ファイルの例を示します。

例 9-13. IP のブラック・ボックス・インスタンス化を含むトップレベル Verilog HDL コード

```
module top (clk, count);
    input clk;
    output [7:0] count;
    my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule
// モジュール宣言
// このモジュール用のブラック・ボックスを作成するために、
// 以下の属性が追加されます。
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
    input clock;
    output [7:0] q;
endmodule
```

生成された VHDL ファイルを使用した、ブラック・ボックス IP ファンクションのインスタンス化

`syn_black_box` コンパイラ・ディレクティブを使用して、コンポーネントをブラック・ボックスとして宣言できます。トップレベルのデザイン・ファイルには、メガファンクション・バリエーションのコンポーネント宣言とポート・マッピングが含まれていなければなりません。トップレベル・ファイルでのコンポーネント宣言には、`syn_black_box` ディレクティブを適用します。このソフトウェアは、このディレクティブがなくても正常にコンパイルしますが、追加の警告メッセージがレポートされます。このディレクティブを使用すると、「**ブラック・ボックスを作成するための他の Synplify ソフトウェアの属性**」の項のディレクティブなど、他のディレクティブを追加できます。

例 9-14 に、**my_vhdlIP.vhd** (MegaWizard Plug-In Manager および IP Toolbench) によって生成される簡略化カスタマイズ・バリエーションをインスタンス化するトップレベル・ファイルの例を示します。

例 9-14. IP のブラック・ボックス・インスタンス化を含むトップレベル VHDL コード

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY top IS
  PORT (
    clk:IN STD_LOGIC ;
    count:OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END top;

ARCHITECTURE IP OF top IS
  COMPONENT my_vhdlIP
  PORT (
    clock:IN STD_LOGIC ;
    q:OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
end COMPONENT;
attribute syn_black_box :boolean;
attribute syn_black_box of my_vhdlIP:component is true;
BEGIN
  vhdlIP_inst :my_vhdlIP PORT MAP (
    clock => clk,
    q => count
  );
END rtl;
```

ブラック・ボックスを作成するための他の Synplify ソフトウェアの属性

ファンクションをブラック・ボックス手法としてインスタンス化した場合、合成ツールではファンクション・モジュールの可視性が実現されません。そのため、合成ツールのタイミング・ドリブンの最適化の利点が十分に活かされません。特にブラック・ボックスにレジスタ付き入力および出力がない場合は、より良好なタイミング最適化を実現するために、タイミング・モデルをブラック・ボックスに追加します。これは `syn_tpd` 属性、`syn_tsu` 属性、および `syn_tco` 属性を追加することによって可能になります。Verilog HDL の例は、例 9-15 を参照してください。

例 9–15. Verilog HDL の例

```

module ram32x4 (z,d,addr,we,clk);
  /* synthesis syn_black_box syn_tco1="clk->z[3:0]=4.0"
     syn_tpd1="addr[3:0]->z[3:0]=8.0"
     syn_tsu1="addr[3:0]->clk=2.0"
     syn_tsu2="we->clk=3.0" */
  output [3:0] z;
  input [3:0] d;
  input [3:0] addr;
  input we;
  input clk;
endmodule

```

HDL コード内のブラック・ボックス・モジュールの特性の詳細を通知するために、以下の追加属性が Synplify ソフトウェアによってサポートされています。

- `syn_resources`— 特定のブラック・ボックスで使用されるリソースを指定。
- `black_box_pad_pin`— I/O セルへのマッピングを防止。
- `black_box_tri_pin`— トライ・ステート信号を通知。



これらの属性の適用について詳しくは、「Synplify Reference Manual」の「Altera Constraints, Attributes, and Options」の章を参照してください。

Quartus II 配置配線専用ファイルのインクルード

Synplify ソフトウェアでは、Quartus II ソフトウェアでの配置配線時にものみ使用されるファイルをプロジェクトに追加できます。これは、全部のデザイン・ファイルを Quartus II ソフトウェアでコンパイルすることを要求する、Synplify シンセシス用のグレー・ボックスまたはブラック・ボックスがある場合に役立つことがあります。

これらのファイルは、他のソース・ファイルと同様に Synplify プロジェクトに追加できます。次に、ファイルを右クリックし、**File options** を選択します。**Use for Place and Route Only** オプションをイネーブルします。**job_owner par** オプションを使用して、スクリプトでこのオプションを設定することも可能です。

例えば、例 9–16 のコマンドは、トップレベル・デザイン・ファイル、グレー・ボックス・ネットリスト・ファイル、IP ラッパー・ファイル、および暗号化された IP ファイルを含む Synplify プロジェクト用のファイルを定義します。Synplify ソフトウェアは、これらのファイルを使用して `.vqm` ファイルに“コア”の空のインスタンス化を書き込みます。また、

グレー・ボックス・ネットリストをリソースの見積もりとタイミングの見積もりに使用します。**core.v** ファイルおよび **core_enc8b10b.v** ファイルは、Synplify ソフトウェアではコンパイルされず、配置配線ディレクトリにコピーされます。Quartus II ソフトウェアは、これらのファイルをコンパイルして、“コア”IP ブロックを実装します。

例 9-16. Synplify プロジェクト用のファイルを定義するコマンド

```
add_file -verilog -job_owner par "core_enc8b10b.v"  
add_file -verilog -job_owner par "core.v"  
add_file -verilog "core_gb.v"  
add_file -verilog "top.v"
```

HDL コードからのアルテラ・メガファンクションの推測

Synplify ソフトウェアは、Behavior Extraction Synthesis Technology (BEST) アルゴリズムを使用して、RAM、ROM、演算子、FSM、DSP 乗算演算などの高レベル構造を推測します。次に、Synplify ソフトウェアは、合成プロセス中に、構造をできるだけ長く抽象的な状態に保ちます。これにより、メガファンクションが最適な結果を達成したときに適切なアルテラ・メガファンクションを推測することにより、テクノロジー固有のリソースを使用してこれらの構造を実装できます。以下の項では、アルテラ・メガファンクションを推測する際の Synplify 固有の詳細のいくつかについて概説します。Synplify ソフトウェアには、特定のタイプのメガファンクションの推測を制御するオプションがあります。これについても以下の項で説明しています。

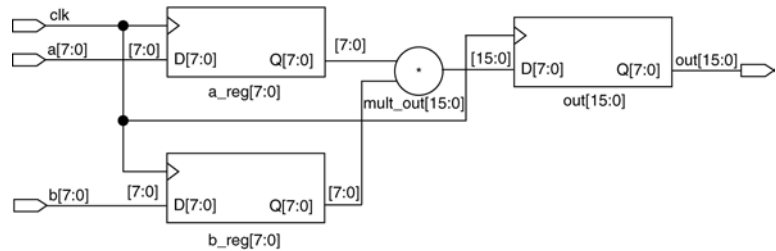


アルテラ・デバイスでのメガファンクションの推測に対するコーディング・スタイルの推奨事項と例については、「Quartus II ハンドブック Volume 1」の「[Recommended HDL Coding Styles](#)」の章を参照してください。

マルチプライヤの推測

図 9-2 に、Synplify ソフトウェアの HDL Analyst に表示される、合成後のパイプライン・ステージ数が 2 の符号なし 8x8 マルチプライヤの HDL Analyst ビューを示します。このマルチプライヤは、ALTMULT_ADD メガファンクションまたは ALTMULT_ACCUM メガファンクションに変換されます。DSP ブロックを備えたデバイスの場合、ソフトウェアはデバイス使用率に応じて、通常のロジックではなく DSP ブロック内にファンクションを実装できます。特定のデバイスの場合、ソフトウェアは .vqm ファイル内でメガファンクションをインスタンス化する代わりに、DSP ブロックのデバイス・プリミティブに直接マップします。

図 9-2. LPM_MULT メガファンクションの HDL Analyst ビュー
(パイプライン数が 2 の符号なし 8 × 8 マルチプライヤ)



リソース・バランシング

Synplify ソフトウェアは、マルチプライヤを DSP ブロックにマップしながら、最適な性能を実現するためにリソース・バランシングを実行します。

アルテラ・デバイスは固定数の DSP ブロックを備えており、固定数のエンベデッド・マルチプライヤを内蔵しています。デザインで、提供されているマルチプライヤよりも多くのマルチプライヤが使用される場合、Synplify ソフトウェアは追加のマルチプライヤをロジック・エレメント (LE) またはアダプティブ・ロジック・モジュール (ALM) に自動的にマップします。

デザインで、DSP ブロック内で提供されているマルチプライヤよりも多くのマルチプライヤが使用される場合、Synplify ソフトウェアはクリティカル・パス内のマルチプライヤを DSP ブロックにマップします。次に、クリティカル・パス内またはクリティカル・パス外にかかわらず、任意のワイド・マルチプライヤが DSP ブロックにマップされます。次に、より小さいマルチプライヤおよびクリティカル・パス内に存在しないマルチプライヤが、ロジック (LE または ALM) 内に実装されます。これにより、デザインがデバイスに正しくフィットすることが保証されます。

DSP ブロックの推測の制御

マルチプライヤは、DSP ブロック内、または DSP ブロックを内蔵するアルテラ・デバイスのロジックに実装できます。この実装を、Synplify ソフトウェアの属性設定を通じて制御することができます。

信号レベル属性

以下の Verilog HDL コードに示す `syn_multstyle` 属性を使用して、個々のマルチプライヤの実装を制御できます。

```
<signal_name> /* synthesis syn_multstyle = "logic" */;
```

ここで、*signal_name* は信号の名前です。


 この設定はワイヤにのみ適用され、レジスタには適用できません。

表 9-4 に、DSP ブロックまたは LE 内へのマルチプライヤの実装を制御する、Synplify ソフトウェアでの信号レベル属性の値を示します。

| 属性名 | 値 | 説明 |
|---------------|------------|---|
| syn_multstyle | lpm_mult | LPM ファンクションが推測され、マルチプライヤが DSP ブロック内に実装される。 |
| syn_multstyle | logic | 推測されない LPM ファンクションおよび Synplify ソフトウェアにより LE 内に実装されるマルチプライヤ |
| syn_multstyle | block_mult | DSP メガファンクションは推測され、マルチプライヤは DSP ブロックのデバイス・プリミティブ(サポートされているデバイスの) に直接マップされる。 |

例 9-17 および例 9-18 に、syn_multstyle 属性を使用した簡単な Verilog HDL コードと VHDL コードを示します。

例 9-17. Verilog HDL コードの DSP ブロックの推測を制御するための信号属性

```
module mult(a,b,c,r,en);
  input [7:0] a,b;
  output [15:0] r;
  input [15:0] c;
  input en;
  wire [15:0] temp /* synthesis syn_multstyle="logic" */;

  assign temp = a*b;
  assign r = en ? temp :c;
endmodule
```

例 9-18. VHDL コードで DSP ブロックの推測を制御するための信号属性

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity onereg is port (
  r :out std_logic_vector(15 downto 0);
  en :in std_logic;
  a :in std_logic_vector(7 downto 0);
  b :in std_logic_vector(7 downto 0);
  c :in std_logic_vector(15 downto 0)
);
end onereg;

architecture beh of onereg is
signal temp :std_logic_vector(15 downto 0);
attribute syn_multstyle :string;
attribute syn_multstyle of temp :signal is "logic";

begin
  temp <= a * b;
  r <= temp when en='1' else c;
end beh;
```

RAM の推測

HDL デザインから RAM ブロックが推測された場合、ソフトウェアはアルテラ・メガファンクションを使用して、デバイスのメモリ・アーキテクチャをターゲットにします。特定のデバイスの場合、ソフトウェアは **.vqm** ファイル内でメガファンクションをインスタンス化する代わりに、メモリ・ブロックのデバイス・プリミティブに直接マップします。

デザインで RAM を正しく推測するために、Synplify ソフトウェアに対する以下のガイドラインに従ってください。

- アドレス・ラインの幅は 2 ビット以上でなければなりません。
- メモリでのリセットはサポートされていません。リード・ポートとライト・ポートが同期している必要があるか否かについては、デバイス・ファミリの資料を参照してください。
- ブロッキング・アサインメントを持ついくつかの Verilog HDL ステートメントは、RAM ブロックにマップされないことがあります。したがって、Verilog HDL で RAM をモデル化するときはブロッキング・ステートメントを避けてください。

特定のデバイス・ファミリの場合、`syn_ramstyle` 属性で推測された RAM に使用する実装が指定されます。`syn_ramstyle` をモジュールまたは RAM インスタンスにグローバルに適用して、`registers` または `block_ram` 値を指定できます。RAM の推測をオフにするには、属性値を `registers` に設定します。

特定のアルテラ・デバイス・ファミリの RAM を推測する場合、Synplify ソフトウェアが追加のバイパス・ロジックを生成します。このロジックは、RTL シミュレーションと合成後シミュレーションの間のハーフサイクル・リード/ライト動作の違いを解消するために生成されます。RTL シミュレーションは更新中のメモリをクロックのポジティブ・エッジで示し、合成後シミュレーションは更新中のメモリをネガティブ・エッジで示します。バイパス・ロジックをなくすには、RAM の出力をラッチする必要があります。このレジスタを追加することによって、RAM の出力は 1 フル・クロック・サイクル後に現れますが、この時点までに更新が行われるのでバイパス・ロジックは不要になります。

TriMatrix メモリ・ブロックを備えたデバイスの場合、`syn_ramstyle` 値を `no_rw_check` に設定することで、グルー・ロジックの作成をディセーブルできます。`syn_ramstyle` を `no_rw_check` の値で使用して、デュアル・ポート・モードでのグルー・ロジックの作成をディセーブルします。

例 9-19 に、デュアル・ポート RAM を推測するための VHDL コードの例を示します。

例 9-19. 推測されたデュアル・ポート RAM の VHDL コード

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY dualport_ram IS
PORT ( data_out:OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      data_in:IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      wr_addr, rd_addr:IN STD_LOGIC_VECTOR (6 DOWNTO 0);
      we:IN STD_LOGIC ;
      clk:IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL mem:Mem_Type;
SIGNAL addr_reg:STD_LOGIC_VECTOR (6 DOWNTO 0);
```

```
BEGIN
  data_out <= mem (CONV_INTEGER(rd_addr));
  PROCESS (clk, we, data_in) BEGIN
    IF (clk='1' AND clk'EVENT) THEN
      IF (we='1') THEN
        mem(CONV_INTEGER(wr_addr)) <= data_in;
      END IF;
    END IF;
  END PROCESS;
END ram_infer;
```

例 9-20 に、デュアル・ポート RAM を推測するためのバイパス・ロジックを排除する VHDL コードの例を示します。推測手法のために余分なレイテンシ動作が生じますが、この動作はメガファンクションをインスタンス化するときには不要です。

例 9-20. バイパス・ロジックを排除する推測されたデュアル・ポート RAM の VHDL コード

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY dualport_ram IS
  PORT ( data_out:OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        data_in:IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        wr_addr, rd_addr :IN STD_LOGIC_VECTOR (6 DOWNTO 0);
        we:IN STD_LOGIC ;
        clk:IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
  TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
  SIGNAL mem:Mem_Type;
  SIGNAL addr_reg:STD_LOGIC_VECTOR (6 DOWNTO 0);
  SIGNAL tmp_out :STD_LOGIC_VECTOR(7 DOWNTO 0); --output register

BEGIN
  tmp_out <= mem (CONV_INTEGER(rd_addr));
  PROCESS (clk, we, data_in) BEGIN
    IF (clk='1' AND clk'EVENT) THEN
      IF (we='1') THEN
        mem(CONV_INTEGER(wr_addr)) <= data_in;
      END IF;
      data_out <= tmp_out; --registers output preventing
                          -- bypass logic generation.
    END IF;
  END PROCESS;
END ram_infer;
```

RAM の初期化

HDL コードで Verilog HDL システム・タスク、`$readmemb` または `$readmemh` を使用して、RAM メモリを初期化できます。Synplify コンパイラは `.srs` ファイル (テクノロジーに依存しない RTL ネットリスト) の初期化値をフォワード・アノテートし、マッパーは対応する 16 進メモリ初期化 (`.hex`) ファイルを生成します。デザインで推測される各 `altsyncram` メガファンクションに対して 1 つの `.hex` ファイルが作成されます。`.hex` ファイルは、`init_file` 属性を使用して、`.vqm` ファイル内の `altsyncram` インスタンスに関連付けられます。

例 9-21 および例 9-22 に、HDL コードでの RAM メモリの初期化方法、および Verilog HDL を使用した対応する `.hex` ファイルの生成方法を示します。

例 9-21. `$readmemb` システム・タスクの使用による Verilog HDL コードでの推測された RAM の初期化

```
initial
begin
    $readmemb("mem.ini", mem);
end

always @(posedge clk)
begin
    raddr_reg <= raddr;
    if (we)
        mem[waddr] <= data;
end
```

例 9-22. 例 9-21 からのメモリ初期化ファイルを含む `.vqm` インスタンスの例

```
altsyncram mem_hex( .wren_a(we), .wren_b(GND), ...);

defparam mem_hex.lpm_type = "altsyncram";
defparam mem_hex.operation_mode = "Dual_Port";
...
defparam mem_hex.init_file = "mem_hex.hex";
```

ROM の推測

HDL デザインから ROM ブロックが推測された場合、ソフトウェアはアルテラ・メガファンクションを使用して、デバイスのメモリ・アーキテクチャをターゲットにします。特定のデバイスに関しては、ソフトウェアは、**.vqm** ファイル内でメガファンクションをインスタンス化する代わりに、メモリ・ブロックのデバイス素子に直接マップします。デザインで ROM を正しく推測するために、Synplify ソフトウェアに対する以下のガイドラインに従ってください。

- アドレス・ラインの幅は 2 ビット以上でなければなりません。
- ROM が 50% 以上満たされていないければなりません。
- CASE ステートメントまたは IF ステートメントは、同じ幅の定数値を使用して 16 以上のアサインメントを作成しなければなりません。

シフト・レジスタの推測

ソフトウェアは、ALTSHIFT_TAPS メガファンクションを使用したサポートされているデバイス・アーキテクチャ内の専用メモリ・ブロック内に配置できるように、シーケンシャル・シフト・コンポーネント用のシフト・レジスタを推測します。

必要に応じて、`syn_srlstyle` 属性を使用して実装スタイルを設定します。コンポーネントを自動的にシフト・レジスタにマップしたくない場合は、値を `registers` に設定します。値は、グローバルに、あるいは個々のモジュールまたはレジスタに設定可能です。

デザインによっては、シフト・レジスタの推測をオフしてデザインの性能を改善することができます。

インクリメンタル・コンパイルおよびブロック・ベース・デザイン

デザインが複雑になり設計者がチームで作業するようになると、ブロック・ベースのインクリメンタル・デザイン・フローが有効なデザイン・アプローチになる場合がよくあります。インクリメンタル・コンパイル・フローでは、デザインの一部を変更しながら、変更されていない部分の配置と性能を維持することができます。新しいコンパイルを特定のデザイン・パーティションに集中させ、結果を他のパーティションの以前のコンパイル結果とマージすることにより、デザインの繰り返しが大幅に高速化されます。ボトムアップまたはチーム・ベースのアプローチでは、個々のサブブロックに対して最適化を実行し、結果を維持した上で、ブロックを最終的なデザインに統合し、デザインをトップレベルで最適化します。

Synplify Pro ソフトウェアで特定のデバイス・テクノロジーに使用できる MultiPoint シンセシスは、自動化されたブロック・ベースのインクリメンタル・シンセシス・フローを提供します。MultiPoint 機能はデザイン

階層を管理して、プロジェクト全体をトップダウンで合成するのに時間がかかりすぎるデザインをインクリメンタルに設計および合成できるようにします。MultiPoint シンセシスは、デザイン階層の異なるセクションに対して異なるネットリスト・ファイルの作成を可能にし、また Quartus II インクリメンタル・コンパイル手法をサポートします。また、デザインのコンパイル時にデザインの更新されたセクションのみ再合成されるようにして、合成実行時間を短縮し、変更されていないブロックの結果を維持します。デザインのあるセクションを他のセクションに影響を与えずに、変更および再合成することができます。

さらに、デザインの各パーティションのロジックに対して個別プロジェクトを作成することにより、Synplify ソフトウェア（基本 Synplify および Synplify Pro）で手動によってデザインを分割し、異なるネットリスト・ファイルを作成することができます。デザインの各パーティションに対して異なるネットリスト・ファイルを作成することは、各パーティションを他のパーティションから独立させることが可能なことも意味します。

階層的デザイン手法により、デザイン・プロセスの効率を高めることができ、チーム環境で作業する際に、より良好なデザイン再利用の機会が提供され、統合問題が軽減されます。これらのインクリメンタル・シンセシス手法を使用すれば、Quartus II ソフトウェアでインクリメンタル・コンパイルを活用できます。デザインの変更されたパーティションに対してのみ配置配線を実行し、配置配線時間を短縮し、フィッティング結果を維持することができます。これらの手法により良好な結果を達成するには、この項のガイドラインに従ってください。

Quartus II ソフトウェアのこれらの機能を使用する際の一般的なトップダウン・コンパイル・フローを以下に示します。

1. 通常のデザイン・フローの場合と同様に、Verilog HDL または VHDL デザイン・ファイルを作成します。
2. デザインでどの階層ブロックを個別パーティションとして扱うかを決めます。
3. デザインの各パーティションに対して個別のネットリスト・ファイルが作成されるように、MultiPoint 機能または個別プロジェクトを使用してデザインをセットアップします。
4. 個別プロジェクトを使用する場合、実装において下位レベル・パーティションに対する I/O パッドの挿入をディセーブルにします。
5. Synplify Pro または Synplify ソフトウェアで各パーティションをコンパイルおよびテクノロジ・マップし、通常のデザイン・フローの場合と同様に制約を作成します。

6. **.vqm** ネットリストと各パーティションに対する **.tcl** ファイルを Quartus II ソフトウェアにインポートし、インクリメンタル・コンパイルを使用するように Quartus II プロジェクトをセットアップします。
7. Quartus II ソフトウェアでデザインをコンパイルし、インクリメンタル・コンパイルでフィッティング後のネットリストを使用して、コンパイル結果を維持します。
8. デザインの一部に対してデザインまたは合成の最適化の変更を行う場合は、変更されたパーティションのみを再合成して、新しいネットリストと **.tcl** ファイルを生成します。変更されていないパーティションのネットリストを再生成してはなりません。
9. 新しいネットリストと **.tcl** ファイルを Quartus II ソフトウェアにインポートし、インクリメンタル・コンパイルを使用して、Quartus II ソフトウェアでデザインをリコンパイルします。



Quartus II ソフトウェアでのパーティションの作成およびインクリメンタル・コンパイルの使用について詳しくは、「[Quartus II ハンドブック Volume 1](#)」の「[Quartus II Incremental Compilation for Hierarchical and Team-Based Design](#)」の章を参照してください。

インクリメンタル・コンパイル用の個別ネットリスト・ファイルによるデザインの作成

階層的またはインクリメンタル・デザイン・フローの最初のステージは、デザインの異なる部分が互いに影響しないことを確認することです。デザインの各パーティションに個別のネットリストを持たせて、Quartus II ソフトウェアでインクリメンタル・コンパイルを利用できるようにします。デザイン全体が1つのネットリスト・ファイルにある場合、1つのパーティションが変更されると、デザインの再合成時にノード名が変更される可能性があるため、他のパーティションが影響を受けることがあります。合成フローを正しく機能させるために、モジュールおよびエンティティ専用の個別ネットリスト・ファイルを作成することができます。また、各モジュールまたはエンティティは独自のデザイン・ファイルを持っている必要があります。2つの異なるモジュールが同じデザイン・ファイルにあるが異なるパーティションの一部と定義されている場合は、モジュールのいずれか一方を変更すると両方のパーティションをリコンパイルする必要があるため、インクリメンタル・コンパイルを維持することはできません。

アルテラでは、各パーティションのすべての入力および出力をラッチすることを推奨しています。これにより、ロジックを同期させて、パーティション境界を通過する信号の遅延ペナルティを回避することができます。

低レベル・ブロックで境界トライ・ステートを使用する場合、アルテラ・デバイスの出力ピンのトライ・ステート・ドライバを利用するために、Synplify ソフトウェアは階層を通じてトライ・ステートをトップレベルに押し上げ（または“バブル”させ）ます。トライ・ステートをバブルさせるには、階層を通じた最適化が必要なため、ブロック・ベースのコンパイル手法では低レベルのトライ・ステートはサポートされていません。デバイスの外部出力ピンおよび階層内のトップレベル・ブロックでのみ、トライ・ステート・ドライバを使用する必要があります。



階層の設計およびパーティションの作成に関する推奨事項について詳しくは、「Quartus II ハンドブック Volume 1」の「[Best Practices for Incremental Compilation Partitions and Floorplan Assignments](#)」の章を参照してください。

Synplify Pro ソフトウェアの MultiPoint シンセシスを使用して、あるいは個別の Synplify プロジェクトを手動で作成し、個別デザイン・パーティションとして扱いたい各ブロックに対してブラック・ボックスを作成することによって、複数の .vqm ネットリスト・ファイルを生成できます。

MultiPoint シンセシス・フロー（Synplify Pro のみ）では、管理しやすい1つのトップレベル合成プロジェクトから複数の .vqm ネットリスト・ファイルを作成できます。手動ブラック・ボックス手法（Synplify または Synplify Pro）を使用して、複数の合成プロジェクトを持つことができます。このプロジェクトは1つのトップレベル・プロジェクトが望ましくない特定のチーム・ベース・デザインまたはボトムアップ・デザインに必要となる場合があります。

これら2つの方法の1つを使用して複数の .vqm ファイルを作成したら、適切な Quartus II プロジェクトを作成して、デザインの配置配線を行う必要があります。

Synplify Pro MultiPoint シンセシスとインクリメンタル・コンパイルの併用

この項では、Synplify Pro MultiPoint シンセシス・フローを使用して複数の .vqm ファイルを生成する方法について説明します。最初に制約ファイルと Synplify Pro オプションを設定し、次に複数の .vqm ファイルに

書き込むための適切な“コンパイル・ポイント”設定を適用して、インクリメンタル・コンパイル用のデザイン・パーティション・アサインメントを作成します。

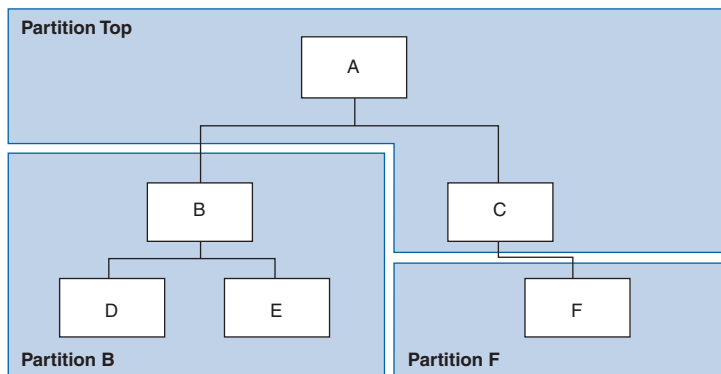
コンパイル・ポイントの設定と制約ファイルの作成

MultiPoint フローでは、デザインを“コンパイル・ポイント”と呼ばれるより小さな合成単位に分割できます。合成ソフトウェアは、各コンパイル・ポイントをインクリメンタル・マッピング用のパーティションとして扱います。これにより、個別のコンパイル・ポイント・モジュールを、他のデザイン・モジュールに影響を及ぼすことなく、より大規模なデザインの独立したセグメントとして切り離して作業することができます。デザインは任意の数のコンパイル・ポイントを持つことができ、コンパイル・ポイントはネストさせることができます。トップレベル・モジュールは常にコンパイル・ポイントとして扱われます。

コンパイル・ポイントは、別のコンパイル・ポイントまたはトップレベル・デザインであるそれぞれの親から切り離された状態で最適化されます。コンパイル・ポイントを使用して作成された各ブロックは、クリティカル・パスやその親または他のブロック上の制約に影響されません。コンパイル・ポイントは独立しており、独自の制約が与えられています。合成中、まだ合成されていないコンパイル・ポイントは、トップレベルの前に合成されます。ネストされたコンパイル・ポイントは、それらが含まれる親のコンパイル・ポイントの前に合成されます。コンパイル・ポイントに適切な設定を適用すると、そのコンパイル・ポイントに対して個別ネットリストが作成され、そのロジックはデザインの他のロジックから切り離されます。

図 9-3 に、複数のパーティションに分割可能なデザイン階層の例を示します。各パーティションのトップレベル・ブロックは、個別のコンパイル・ポイントとして合成できます。

図 9-3. 階層構造デザインのパーティション



この場合、コンパイル・ポイントはモジュール A、B、および F です。トップレベルのコンパイル・ポイントは、別のコンパイル・ポイントの下で定義されていないロジックを含む、デザインのトップレベル・ブロック（この例では、ブロック A）で構成されています。この例では、トップレベルのコンパイル・ポイント A のデザインには、そのサブブロックの 1 つである C のロジックも含まれています。ブロック F は、独自のコンパイル・ポイントとして定義されているため、トップレベルのコンパイル・ポイント A の一部として扱われません。もう 1 つの独立したコンパイル・ポイント B には、ブロック B、D、E のロジックが含まれます。トップレベル・モジュール A とサブモジュール C に対して 1 つのネットリストが作成され、B およびそのサブモジュールである D と E に対してもう 1 つのネットリストが作成され、F に対して 3 番目のネットリストが作成されます。

コンパイル・ポイントは、Synplify Pro SCOPE スプレッドシートまたは .sdc ファイル内のモジュールまたはアーキテクチャに適用します。Verilog HDL または VHDL ソース・コードでコンパイル・ポイントを設定することはできません。以下の項で説明するように、Tcl を使用して、または .sdc ファイルを編集することにより、制約を手動で設定できます。あるいは、GUI の 2 つの方法のうち 1 つを使用することができます。

.tcl または **.sdc** ファイルを使用したコンパイル・ポイントの定義
.tcl ファイルまたは **.sdc** ファイルを使用してコンパイル・ポイントを設定するには、define_compile_point コマンドを例 9-23 に示すとおり使用します。

例 9-23. define_compile_point コマンド

```
define_compile_point [-disable] {<objname>} -type {locked, partition}
```

上記の構文で、*objname* はデザイン内の任意のモジュールを表します。コンパイル・ポイントのタイプ {locked, partition} は、コンパイル・ポイントが Quartus II インクリメンタル・コンパイル・フロー用のパーティションを表していることを示します。

各コンパイル・ポイントには、以下に示すように SCOPE 環境を設定するための `define_current_design` コマンドで始まる制約ファイルのセットがあります。

```
define_current_design {<my_module>}
```

トップレベルの **SCOPE** ウィンドウでのコンパイル・ポイントの定義
以下の方法では、トップレベルおよび下位レベルのコンパイル・ポイント用の制約ファイルを個別に作成する必要があります。

1. トップレベルの SCOPE ウィンドウで、**Compile Points** タブを選択します。
2. コンパイル・ポイントとして定義したいモジュールを選択し、Type を **locked, partition** に設定します。
3. 各モジュール用の制約ファイルを手動で作成して、各コンパイル・ポイントの制約を設定します。

新しい **SCOPE** ファイルの作成によるコンパイル・ポイントの定義
以下のプロセスを使用する場合、下位レベルの制約ファイルが自動的に作成されます。

1. **File** メニューで、**New** をクリックし、新しい **Constraint File** を作成します。あるいは、ツール・バーの **SCOPE** アイコンをクリックします。
2. **Create a New SCOPE File** ダイアログ・ボックスの **Select File Type** タブから、**Compile Point** を選択します。
3. コンパイル・ポイントとして指定したいモジュールを選択して、**OK** をクリックします。ソフトウェアは、トップレベルの制約ファイルに自動的にコンパイル・ポイントを設定し、各コンパイル・ポイントに対して下位レベルの制約ファイルを作成します。

コンパイル・ポイントに対する追加検討事項

コンパイル・ポイントに対する変更がトップレベルの親モジュールに影響しないようにするために、**Implementation Options** ダイアログ・ボックスの **Update Compile Point Timing Data** オプションをディセーブルにします。このオプションがイネーブルされている場合、子モジュールを更新するとトップレベル・モジュールが影響を受ける可能性があります。

`syn_allowed_resources` 属性を任意のコンパイル・ポイント・ビューに適用して、特定のモジュールのリソース数を制限することができます。

コンパイル・ポイントをインクリメンタル・コンパイルと併用する場合は、以下の制限事項に留意してください。

- コンパイル・ポイントを効果的に使用するには、各コンパイル・ポイントにタイミング制約（タイミング・バジェット）を与える必要があります。制約が正確であるほど、結果が良好になります。制約は自動的に設定されません。したがって、手動のタイミング・バジェットが不可欠です。アルテラでは、各パーティションのすべての入力および出力をラッチすることを推奨しています。これによって、パーティション境界を通過する信号の遅延ペナルティを回避できます。
- Synplify Pro の属性 `syn_useioff` を使用してアルテラ・デバイスの I/O エlement (IOE) でレジスタをバックする場合、これらのレジスタは、下位レベルではなくトップレベル・モジュールになければなりません。そうでない場合は、`syn_useioff` 属性の代わりに、Quartus II ソフトウェアが I/O レジスタのパッキングを実行できるようにする必要があります。**Fast Input Register** オプションまたは **Fast Output Register** オプションを使用するか、Quartus II ソフトウェアの **Settings** ダイアログ・ボックスの Fitter Settings ページで I/O タイミング制約を設定し、**Optimize I/O cell register placement for timing** をオンにします。
- トップレベル・ロジック用のインクリメンタル・シンセシスはサポートされていません。トップレベルのどのロジックも Synplify Pro ソフトウェアでの毎回のコンパイル中に再合成されます。



コンパイル・ポイントの使用、およびトップレベルと下位レベルの両方のコンパイル・ポイントに対する Synplify 属性および制約の設定について詳しくは、Synplcity 社のウェブサイト (www.synplcity.com) にある Synplify Pro User Guide and Reference Manual を参照してください。

コンパイル・ポイント用の Quartus II プロジェクトおよび複数の .vqm ファイルの作成

コンパイル中、Synplify Pro ソフトウェアは Quartus II ソフトウェアに適切な制約とデザイン・パーティション・アサインメントを提供する <トップレベル・プロジェクト >.tcl ファイルを作成し、Quartus II プロジェクトをセットアップするための情報と併せて、各 .vqm ファイル用のパーティションを作成します。Quartus II プロジェクトをセットアップし、制約を渡すための Synplify ソフトウェアで生成された Tcl スクリプトの使用については、9-22 ページの「Synplify が生成した Tcl スクリプトを使用して、Quartus II ソフトウェアを手動で実行する」を参照してください。

デザイン手法に応じて、すべてのネットリストに対して1つの Quartus II プロジェクト (トップダウンの配置配線フロー)、または各ネットリストに対して個別の Quartus II プロジェクト (ボトムアップ配置配線フロー) を作成できます。トップダウンのインクリメンタル・コンパイル・デザイン・フローでは、デザイン・パーティション・アサインメントを作成し、オプションで、1つの Quartus II プロジェクト内のデザインの各パーティションに対して LogicLock フロアプラン・ロケーション・アサインメントを作成します。この手法により、最高の品質結果を達成でき、またデザインへのインクリメンタル変更時に性能を維持することができます。

特定のチーム・ベースのデザイン・フローの場合など、各パーティションを個別に最適化する必要がある場合は、ボトムアップのデザイン・フローが必要になることがあります。このフローを使用する場合は、各パーティション間の配置競合を回避するためにデザイン・フロアプランを作成することを推奨します。Quartus II ソフトウェアでボトムアップ・コンパイルを実行するには、個別の Quartus II プロジェクトを作成し、インクリメンタル・コンパイルのエクスポート機能とインポート機能を使用して各デザイン・パーティションをトップレベル・デザインにインポートし、配置結果を維持します。

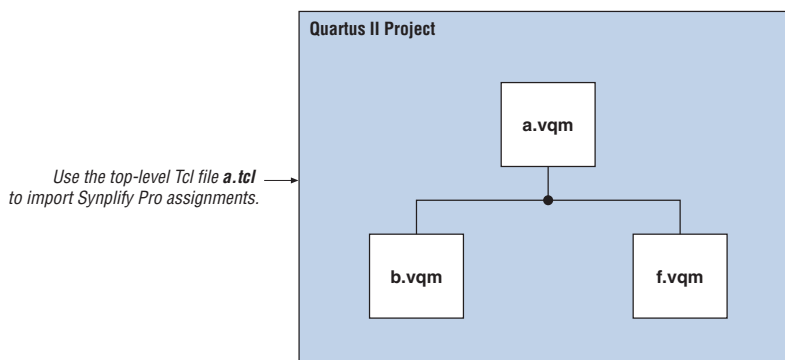
以下の項では、これらの2つのデザイン・フローのために Quartus II プロジェクトを作成する方法について説明します。

トップダウン・インクリメンタル・コンパイル・フローのための1つの Quartus II プロジェクトの作成

プロジェクト内のすべてのパーティションに対する Synplify Pro アサインメントを含む <トップレベル・プロジェクト >.tcl ファイルを使用します。この方法により、インクリメンタル・コンパイルが提供する性能の維持とコンパイル時間の短縮の利点を活かして、すべてのパーティ

ションを1つの Quartus II プロジェクトにインポートし、プロジェクト内のすべてのモジュールを一度に最適化することができます。図 9-4 に、図 9-3 のデザイン例のデザイン・フローを視覚的に示します。

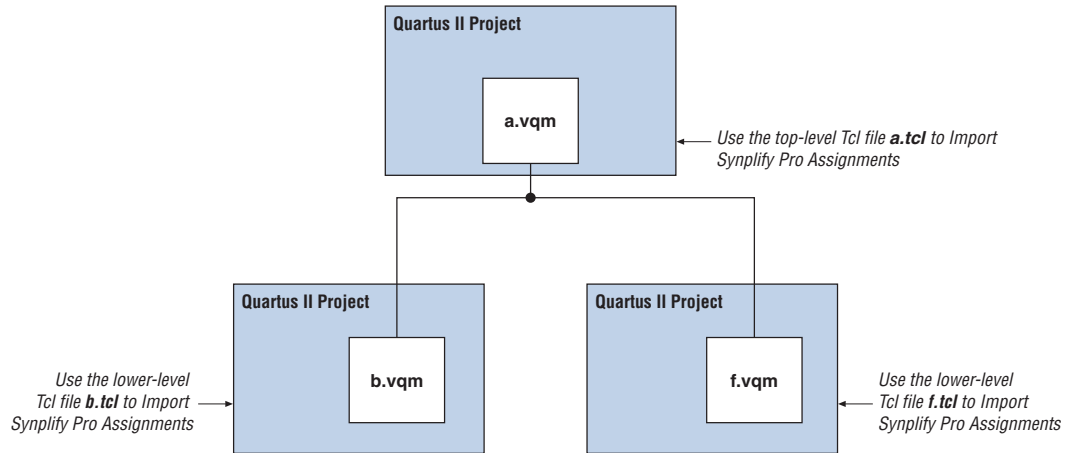
図 9-4. 複数の .vqm ファイルと 1つの Quartus II プロジェクトを併用したデザイン・フロー



ボトムアップ・インクリメンタル・コンパイル・フローのための複数の Quartus II プロジェクトの作成

各コンパイル・ポイントに対する Synplify Pro アサインメントを含む <下位レベル・コンパイル・ポイント>.tcl ファイルを使用します。デザインの各パーティションおよびネットリストに対して 1 つずつ、複数の Quartus II プロジェクトを生成します。プロジェクトの各設計者は、各自のパーティションを Quartus II ソフトウェア内で別々に最適化し、それらのパーティションの結果をエクスポートすることができます。図 9-5 に、図 9-3 のデザイン例のデザイン・フローを視覚的に示します。最適化されたサブデザインをエクスポートし、次にインクリメンタル・コンパイルを使用してそのサブデザインを 1 つのトップレベル Quartus II プロジェクトにインポートして、デザインを完成させることができます。

図 9-5. 複数の .vqm ファイルと複数の Quartus II プロジェクトを使用したデザイン・フロー



個別の Synplify プロジェクトを使用した、複数のインクリメンタル・コンパイル用 .vqm ファイルの作成

この項では、ブラック・ボックスと各デザイン・パーティション用の Synplify プロジェクトを使用して複数の .vqm ファイルをインクリメンタル・コンパイル用に手動で生成する方法について説明します。この手動フローは、MultiPoint シンセシス機能を持たない Synplify ソフトウェアのバージョンでサポートされます。

ブラック・ボックスを使用した複数の .vqm ファイルの手動による作成

Synplify ソフトウェアで複数の .vqm ファイルを手動で作成するには、各下位レベル・モジュール用の個別プロジェクト、およびインクリメンタル・コンパイル・パーティションのための個別 .vqm ファイルとして維持したいトップレベル・デザインを作成します。トップレベル・プロジェクトで下位レベルのパーティションのブラック・ボックス・インスタンス化を実装します。

下位レベル・モジュール用のプロジェクトを合成するときは、以下のステップを実行します。

1. **Implementation Options** ダイアログ・ボックスで、ターゲット・テクノロジの **Disable I/O Insertion** をオンにします。
2. モジュール用の HDL ファイルを読み出します。



モジュールには、個別の **.vqm** ファイルとしても維持される下位レベル・モジュールのブラック・ボックス・インスタンス化が含まれている可能性があります。

3. SCOPE 制約ウィンドウで制約を追加します。
4. クロック周波数を入力して、サブデザインが正しく最適化されるようにします。
5. **Attributes** タブで、**syn_netlist_hierarchy** を **0** に設定します。

トップレベル・デザイン・プロジェクトを合成するときは、以下のステップを実行します。

1. ターゲット・テクノロジーの **Disable I/O Insertion** をオフにします。
2. トップレベル・デザイン用の HDL ファイルを読み出します。
3. トップレベル・デザイン内の下位レベル・レベル・モジュールを使用してブラック・ボックスを作成します。
4. SCOPE 制約ウィンドウで制約を追加します。
5. クロック周波数を入力して、デザインが正しく最適化されるようにします。
6. **Attributes** タブで、**syn_netlist_hierarchy** を **0** に設定します。

以下の項では、個別の **.vqm** ファイルを作成するためのブラック・ボックス実装の例について説明します。複数のパーティションに分割されるデザイン階層の例は、[図 9-3](#) を参照してください。

図 9-6. 階層構造デザインのパーティション

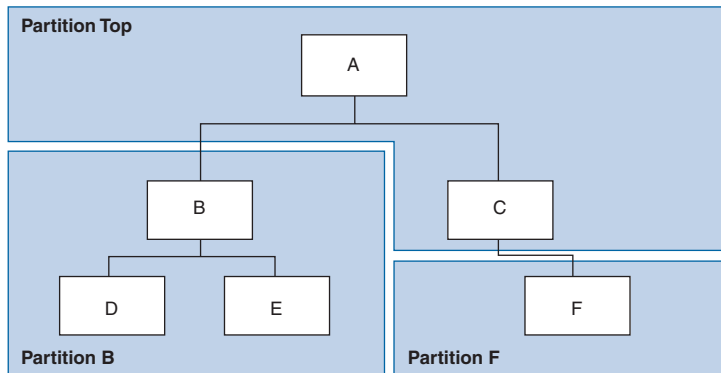


図 9-3 のパーティション Top には、デザイン内のトップレベル・ブロック（ブロック A）と、別のパーティションの一部として定義されないロジックが含まれます。この例では、トップレベル・ブロック A のパーティションには、そのサブブロック C の 1 つであるロジックも含まれています。ブロック F は、独自のパーティションに含まれるため、トップレベル・パーティション A の一部として扱われません。もう 1 つの独立したパーティション B には、ブロック B、D、E のロジックが含まれます。チーム・ベースのデザインでは、異なるエンジニアが異なるパーティションのロジックを扱うことができます。トップレベル・モジュール A とサブモジュール C に対して 1 つのネットリストが作成され、B およびそのサブモジュールである D と E に対してもう 1 つのネットリストが作成され、F に対して 3 番目のネットリストが作成されます。このデザイン用に複数の **.vqm** ファイルを作成するには、以下のステップに従います。

1. モジュール B 用の **.vqm** ファイルを生成します。ソース・ファイルとして、**B.v.vhd**、**D.v.vhd**、および **E.v.vhd** を使用します。
2. モジュール F 用の **.vqm** ファイルを生成します。ソース・ファイルとして **F.v.vhd** を使用します。
3. モジュール A 用のトップ・レベル **.vqm** ファイルを生成します。ソース・ファイルとして、**A.v.vhd** および **C.v.vhd** を使用します。必ず前のステップで別々に最適化されたブラック・ボックス・モジュール B および F を使用してください。

Verilog HDL でのブラック・ボックスの作成

ソフトウェアは、プロジェクトで定義されていない、あるいはプロジェクトのために読み出されるファイルのリストに含まれるすべてのデザイン・ブロックをブラック・ボックスとして扱います。syn_black_box 属性を使用して、特定のモジュールのためにブラック・ボックスを作成するよう指示します。Verilog HDL では、ブラック・ボックスとして扱われるモジュールに空のモジュール宣言を提供する必要があります。

例 9-24 に、トップレベル・ファイル **A.v** の例を示します。現在の階層レベルの下にあるモジュールのブラック・ボックスも含む下位レベル・ファイルに対して、以下の同じ手順に従います。

例 9-24. トップレベル・ファイル **A.v の Verilog HDL ブラック・ボックス**

```
module A (data_in, clk, e, ld, data_out);
    input data_in, clk, e, ld;
    output [15:0] data_out;

    wire [15:0] cnt_out;

    B U1 (.data_in (data_in), .clk(clk), .ld (ld), .data_out(cnt_out));
    F U2 (.d(cnt_out), .clk(clk), .e(e), .q(data_out));

    // A.v 内の他のコードをここに入れる。
endmodule

// サブブロック B および F の空のモジュール宣言をこの後に置く。
// これらのモジュール宣言（ポートを含む）はブラック・ボックスに必要。//

module B (data_in, clk, ld, data_out) /* synthesis syn_black_box */ ;
    input data_in, clk, ld;
    output [15:0] data_out;
endmodule

module F (d, clk, e, q) / *synthesis syn_black_box */ ;
    input [15:0] d;
    input clk, e;
    output [15:0] q;
endmodule
```

VHDLでのブラック・ボックスの作成

ソフトウェアは、プロジェクトで定義されていない、あるいはプロジェクトのために読み出されるファイルのリストに含まれるすべてのデザイン・ブロックをブラック・ボックスとして扱います。syn_black_box属性を使用して、特定のコンポーネントをブラック・ボックスとして扱うことを指示します。VHDLでは、デザイン内の他の任意のブロックと同様に、ブラック・ボックスに対するコンポーネント宣言が必要です。



VHDLは大文字と小文字を区別しませんが、.vqm (Verilog HDLのサブセット) ファイルは大文字と小文字を区別します。エンティティ名とエンティティのポート宣言が.vqm ファイルに転送されます。ブラック・ボックス名とポート宣言も.vqm ファイルに渡されます。大文字と小文字の不一致が生じないように、VHDLデザインでのブラック・ボックスとエンティティ宣言には大文字を使用してください。

例 9-25 に、トップレベル・ファイル **A.vhd** の例を示します。現在の階層レベルの下にあるブロックのブラック・ボックスを含む下位レベル・ファイルに対して、同じ手順に従います。

例 9-25. トップレベル・ファイル A.vhd の VHDL ブラック・ボックス

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY synplify;
use synplify.attributes.all;

ENTITY A IS
PORT ( data_in :IN INTEGER RANGE 0 TO 15;
      clk, e, ld :IN STD_LOGIC ;
      data_out:OUT INTEGER RANGE 0 TO 15 );
END A;

ARCHITECTURE a_arch OF A IS

COMPONENT B PORT(
  data_in:IN INTEGER RANGE 0 TO 15;
  clk, ld :IN STD_LOGIC ;
  d_out :OUT INTEGER RANGE 0 TO 15 );
END COMPONENT;

COMPONENT F PORT(
  d :IN INTEGER RANGE 0 TO 15;
  clk, e:IN STD_LOGIC ;
  q:OUT INTEGER RANGE 0 TO 15 );
END COMPONENT;

attribute syn_black_box of B:component is true;
attribute syn_black_box of F:component is true;

```

-- A.vhd のその他のコンポーネント宣言をここに入れる。

```
signal cnt_out :INTEGER RANGE 0 TO 15;
```

```
BEGIN
```

```
U1 :B
```

```
PORT MAP (  
    data_in => data_in,  
    clk => clk,  
    ld => ld,  
    d_out => cnt_out );
```

```
U2 :F
```

```
PORT MAP (  
    d => cnt_out,  
    clk => clk,  
    e => e,  
    q => data_out );
```

-- A.vhd 内のその他のコードをここに置く。

```
END a_arch;
```

この項に記載されるステップの完了後に、デザインの各パーティションに対するネットリスト・ファイルが得られます。これらのファイルは、Quartus II ソフトウェアでのインクリメンタル・コンパイルに使用できます。

複数の .vqm ファイル用の Quartus II プロジェクトの作成

Synplify ソフトウェアは、Quartus II ソフトウェアに適切な制約を提供する各 .vqm ファイル用の .tcl ファイルと、プロジェクトを設定するための情報を作成します。Quartus II プロジェクトをセットアップし、制約を渡すための Synplify ソフトウェアで生成された Tcl スクリプトの使用について詳しくは、9-22 ページの「Synplify が生成した Tcl スクリプトを使用して、Quartus II ソフトウェアを手動で実行する」を参照してください。

デザイン手法に応じて、すべてのネットリストに対して1つの Quartus II プロジェクト（トップダウンの配置配線フロー）、または各ネットリストに対して個別の Quartus II プロジェクト（ボトムアップ配置配線フロー）を作成できます。トップダウンのインクリメンタル・コンパイル・デザイン・フローでは、デザイン・パーティション・アサインメントを作成し、オプションで、1つの Quartus II プロジェクト内のデザインの各パーティションに対して LogicLock フロアプラン・ロケーション・アサイン

メントを作成します。この手法により、最高の品質結果を達成でき、またデザインへのインクリメンタル変更時に性能を維持することができます。特定のチーム・ベースのデザイン・フローの場合など、各パーティションを個別に最適化する必要がある場合は、ボトムアップのデザイン・フローが必要になることがあります。

Quartus II ソフトウェアでボトムアップ・コンパイルを実行するには、個別の Quartus II プロジェクトを作成し、インクリメンタル・コンパイルのエクスポート機能とインポート機能を使用して各デザイン・パーティションをトップレベル・デザインにインポートし、結果を維持します。

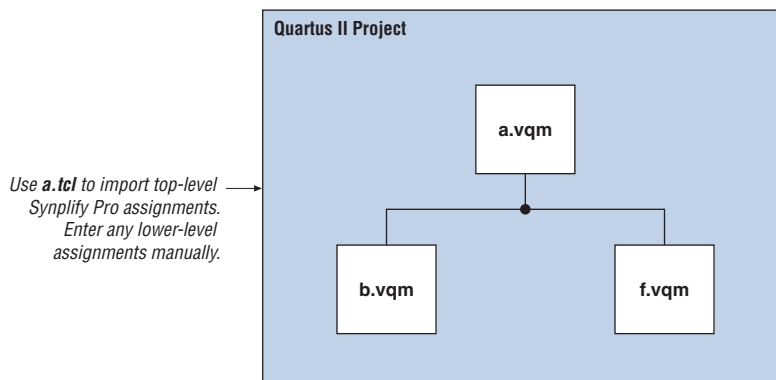
以下の項では、これらの2つのデザイン・フローのために Quartus II プロジェクトを作成する方法について説明します。

トップダウン・インクリメンタル・コンパイル・フローのための1つの Quartus II プロジェクトの作成

トップレベル・デザインに対する Synplify アサインメントを含む <トップレベル・プロジェクト>.tcl ファイルを使用します。この方法により、インクリメンタル・コンパイルが提供する性能の維持とコンパイル時間の短縮の利点を活かして、すべてのパーティションを1つの Quartus II プロジェクトにインポートし、プロジェクト内のすべてのモジュールを一度に最適化することができます。図 9-7 に、図 9-3 のデザイン例のデザイン・フローを視覚的に示します。

トップレベル・プロジェクトからのすべての制約がトップレベル.tcl ファイル内の Quartus II ソフトウェアに渡されますが、Synplify ソフトウェア内の下位レベル・プロジェクトで作成された制約はフォワード・アノテートされません。これらの制約は Quartus II プロジェクトに手動で入力します。

図 9-7. 複数の .vqm ファイルと 1 つの Quartus II プロジェクトを併用したデザイン・フロー

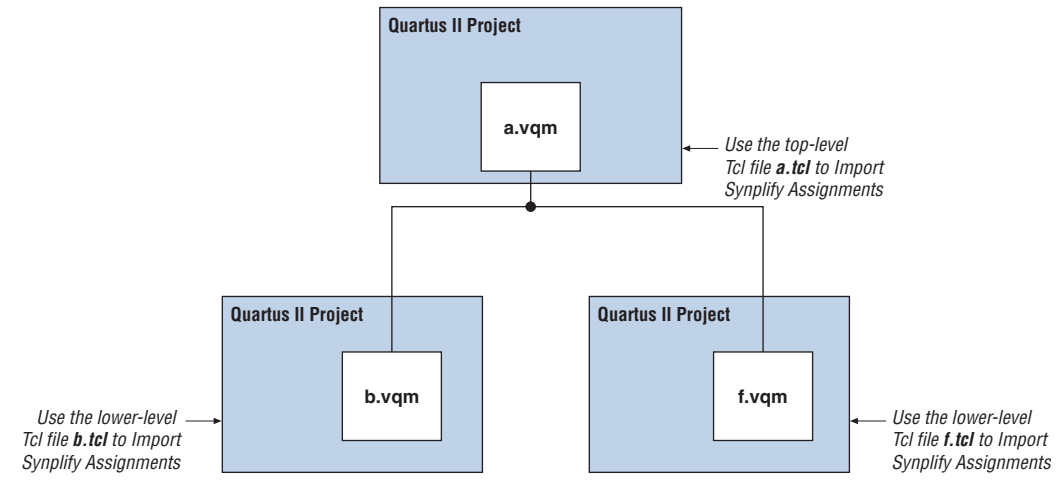


ボトムアップ・インクリメンタル・コンパイル・フローのための複数の Quartus II プロジェクトの作成

Synplify ソフトウェアで各 .vqm ファイル用に作成された .tcl ファイルを各 Synplify プロジェクトに使用します。この方法により、デザインの各ブロックに対して 1 つずつ、複数の Quartus II プロジェクトが生成されます。プロジェクトの各設計者は、各自のブロックを Quartus II ソフトウェア内で個別に最適化し、それらのブロックの配置をエクスポートすることができます。図 9-8 に、9-56 ページの図 9-3 の例のデザイン・フローを視覚的に示します。

設計者は、パーティション間の競合を回避するために LogicLock 領域を作成して、各ブロックのデザイン・フロアプランを作成する必要があります。トップレベルの設計者は、次にすべてのブロックとアサインメントをトップレベル・プロジェクトにインポートする必要があります。この方法により、デザイン内の各ブロックを個別に最適化してから、1 つのトップレベル・プロジェクトにインポートすることができます。

図 9-8. 複数のSynplify プロジェクトと複数のQuartus IIプロジェクトを使用するデザイン・フロー



Quartus II ソフトウェアでのインクリメンタル・コンパイルの実行

Multipoint シンセシスを使用するトップダウンのデザイン・フローでは、Synplify ソフトウェアは Quartus II トップレベル .tcl ファイルを使用して、2つのツールのデータベースが同期状態で維持されるようにします。Tcl は、Synplify で作成、変更、または削除されたコンパイル・ポイントに対する、Quartus II ソフトウェア内のパーティション・アサインメントを作成、変更、または削除します。ただし、Quartus II ソフトウェアでパーティションを作成、変更、または削除しても、Synplify ソフトウェアはコンパイル・ポイントの設定を変更しません。Synplify プロジェクトで対応する変更を行って、正しい .vqm ファイルを作成する必要があります。



9-21 ページの「Quartus II ソフトウェアを使用して Synplify ソフトウェアを実行する」で説明した Nativelink 統合機能を使用する場合、Synplify ソフトウェアは、Quartus II ソフトウェアで設定したデザイン・パーティション・アサインメントに関する情報は使用しません。

複数の Synplify プロジェクトを使用してネットリストを作成する場合、または Quartus II プロジェクトの制約を更新する際に Synplify Pro で生成された .tcl ファイルを使用しない場合、Synplify の .vqm ネットリストが Quartus II のパーティション設定に適合していることを確認する必要があります。

.vqm ネットリスト・ファイルで Quartus II プロジェクトを個別デザイン・パーティションとしてセットアップした後、コンパイル結果が維持されるように適切な Quartus II オプションを設定します。Assignments メニューで、**Design Partitions Window** をクリックします。以前のコンパイルのフィッティング後の配置結果を維持するために、Netlist Type を **Post-Fit** に変更します。配線結果も維持するには、Fitter Preservation Level を **Placement and Routing** に設定します。これらの設定を行わない場合、Quartus II ソフトウェアは以前のコンパイルからの配置または配線結果を再利用しません。

インクリメンタル・コンパイルと Synplify デザインの利点を活かして、Quartus II ソフトウェアでコンパイル時間を短縮し、変更されていないデザイン・ブロックに対する結果を維持することができます。



Quartus II インクリメンタル・コンパイルの使用について詳しくは、「[Quartus II ハンドブック Volume 1](#)」の「[Quartus II Incremental Compilation for Hierarchical and Team-Based Design](#)」の章を参照してください。

まとめ

高度な合成はデザイン・フローの重要な部分です。Synplicity 社の Synplify とアルテラの Quartus II デザイン・フローの利点を活かすことにより、Quartus II 配置配線プロセスのためのデザイン・ファイルの準備方法を制御できるだけでなく、性能を向上させアルテラ・デバイスで使用するためにデザインを最適化することもできます。この章で概要を述べた手法のいくつかは、デザインを最適化して性能目標を達成し、デザイン時間を短縮するのに役立つことがあります。

参考資料

この章では以下のドキュメントを参照しています。

- 「Synplify Software Reference Manual」の「Altera Constraints, Attributes, and Options」の章
- 「Synplify Reference Manual」の「Altera I/O Standards」
- 「Quartus IIハンドブック Volume 2」の「Area and Timing Optimization」の章
- 「Quartus IIハンドブック Volume 1」の「アルテラ・デバイスのデザイン推奨事項」の章
- 「Quartus IIハンドブック Volume 2」の「Netlist Optimizations and Physical Synthesis」の章
- 「Quartus II ハンドブック Volume 1」の「Quartus II Incremental Compilation for Hierarchical and Team-Based Design」の章
- Synplify Pro User Guide and Reference Manual (Synplicity ウェブサイト www.synplicity.com/literature/index.html)
- 「Quartus IIハンドブック Volume 2」の「Tcl Scripting」の章

改訂履歴

表 9-5 に、本資料の改訂履歴を示します。

| 表 9-5. 改訂履歴 (1 / 2) | | |
|---------------------|---|---|
| 日付およびバージョン | 変更内容 | 概要 |
| 2008年5月 v8.0.0 | <ul style="list-style-type: none"> ● サポートされているデバイス・リストを更新。 ● TimeQuest タイミング・アナライザの制約の注釈情報を更新。 ● RAM および MAC の制約に関する制限事項を更新。 ● 表 9-1 を改訂。 ● 新しい項、「インスタンス化したアルテラ・メガファンクションに対する Synplify のデフォルトの動作の変更」を追加。 ● 新しい項、「MegaWizard Plug-In Manager および IP Toolbench を使用した Intellectual Property (IP) のインスタンス化」を追加。 ● 新しい項、「Quartus II 配置配線専用ファイルのインクルード」を追加。 ● 新しい項、「コンパイル・ポイントに対する追加検討事項」を追加。 ● 「Apply the LogicLock Attributes」の項を削除。 ● Figures 9-4、9-5、9-7、および 9-8 を変更。 ● 新しい項、「Quartus II ソフトウェアでのインクリメンタル・コンパイルの実行」を追加。 ● 章全体を通して、多数のテキストを変更および追加。 ● 複数の項のタイトルを変更。 ● 「参考資料」の項を更新。 | Quartus II ソフトウェア v8.0 および Synplify ソフトウェア v9.4 のリリースによる更新。 |
| 2007年10月 v7.2.0 | <p>このドキュメントに対する変更:</p> <ul style="list-style-type: none"> ● Synplicity 社のバージョン・サポートを更新。 ● MegaWizard で生成されたファイルをコンパイルする前に、適切な Quartus II バージョンの設定方法に関する情報を追加。 | Quartus II ソフトウェア v7.2 のリリースでサポートされる Synplicity 機能に基づいた章の更新。 |
| 2007年5月 v7.1.0 | <ul style="list-style-type: none"> ● 図 7-2 を削除 (該当しなくなったため)。 ● Synplify ソフトウェア v8.8 を使用するときのみ MegaWizard Plug-In Manager で生成されたラッパー・ファイルを使用することを規定するために、「Instantiating Altera Megafunctions Using the MegaWizard Plug-In Manager」の項を更新。 ● 「クリアボックス」手法への参照を合成エリアおよびタイミング見積もりネットリストをサポートする情報と差し替え。 ● Quartus II ソフトウェア v7.1 のためのマイナー・アップデート。 ● 参考資料の項を追加。 | Synplify ソフトウェア v8.0 および Quartus II ソフトウェア v7.1 のリリースによる章の更新。 |
| 2007年3月 v7.0.0 | <ul style="list-style-type: none"> ● サポートされているデバイスのリストに Cyclone III を追加 ● Synplify ソフトウェアが選択されたデバイスに関係なく、.scf ファイルを生成することを明記。 | このリリースによる Cyclone III のサポートを記載するために、この章を更新。 |

表 9-5. 改訂履歴 (2 / 2)

| 日付およびバージョン | 変更内容 | 概要 |
|--------------------|---|--|
| 2006年11月 v6.1.0 | <ul style="list-style-type: none"> ● v6.0.0 の 8 章を 9 章に変更。 ● TimeQuest 用の SDC 制約を渡すために SCF が生成されることを追記。 ● TimeQuest を使用するときのタイミング制約情報を追加。 ● alt_pll メガファンクションに関する注をクリアボックスの項からブラック・ボックスの項に移動。 ● Synplify が Stratix および Cyclone シリーズ・デバイス用の alt_pll メガファンクション・ブラック・ボックス・ファイルを読み出すことを明記。 | Stratix III のサポートを記載するための更新、および TimeQuest に対するタイミング制約情報を渡す方法についての情報の追加 |
| 2006年5月 v6.0.0 | Quartus II ソフトウェア v6.0.0 のための更新： <ul style="list-style-type: none"> ● クロス・プロービング情報を更新。 ● NativeLink® 統合情報を追加。 ● Synplify デザイン・フローのサポートを追記。 ● アルテラのメガファンクションのガイドラインとアーキテクチャ固有の機能を追記。 | — |
| 2005年12月 v5.1.1 | 誤字・脱字のマイナー修正。 | — |
| 2005年10月 v5.1.0 | <ul style="list-style-type: none"> ● Quartus II ソフトウェア v5.1 のための更新。 ● v5.0 の 9 章を 8 章に変更。 | — |
| 2005年5月 v5.0.0 | v4.2 の 7 章を 9 章に変更。 | — |
| 2004年12月 v2.1.0 | <ul style="list-style-type: none"> ● v4.1 の 9 章を 8 章に変更。 ● 情報を更新。 ● Quartus II ソフトウェア v4.2 の新機能。 ● 図 8-1 を更新。 | — |
| 2004年6月 v2.0.0 | <ul style="list-style-type: none"> ● 表および図を更新。 ● Quartus II ソフトウェア v4.1 の新機能。 | — |
| 2004年2月 v1.0.0 | 初版 | — |

