

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。

QI151016-8.0.0

## はじめに

この数年間で、FPGA デバイスの集積度が飛躍的に向上したため、デザインはますます複雑になり、複数の設計者が開発に携わる場合もあります。最新の FPGA は柔軟性を備えていますが、これは各デザイン・ブロックのピン・レイアウト、消費電力、面積使用率、タイミング性能がすべて最終的なデザイン実装に依存することを意味します。システム設計者は、全体的な「Time-to-Market」に影響し、コスト増を招く問題を引き起こすことも多いデザイン・ブロックの統合を行う場合は、これらのデザイン問題を解決しなければなりません。多くの潜在的な問題は、適切なデザイン・プランニングを実行することにより、デザイン・サイクルの早期段階で解決できます。

この章では、このような重要な FPGA のデザイン・プランニング問題について説明し、推奨事項を提示し、アルテラの FPGA で使用できるデザインの生産性向上に役立つ各種ツールを紹介します。この章は、以下の項で構成されています。

- 1-2 ページの「デザイン仕様の作成」
- 1-3 ページの「デバイスの選択」
- 1-5 ページの「デバイス・プログラミング / コンフィギュレーションのプランニング」
- 1-6 ページの「早期消費電力見積もり」
- 1-7 ページの「早期ピン・プランニングおよび I/O 解析」
- 1-10 ページの「サードパーティ EDA ツールのフローの選択」
- 1-12 ページの「オンチップ・デバッグ・プランニングのオプション」
- 1-15 ページの「デザイン手法と HDL コーディング・スタイル」
- 1-17 ページの「階層およびチーム・ベース・デザインのプランニング」
- 1-23 ページの「高速合成と早期タイミング見積もり」

この章で説明するデザイン・プランニングのガイドラインを読む前に、デザインの重要な要素は何か、デザインの優先順位を検討してください。デバイスの機能、集積度、または性能を高くすると、システム・コストが上昇します。シグナル・インテグリティとボードの問題が、I/O ピンの位置に影響を与える場合があります。消費電力、タイミング性能、ロジック使用率は相互に影響し合い、コンパイル時間はこれらの要素の最適化に影響を受けます。

Quartus® II ソフトウェアは、最良の平均的結果が得られるようにデザインを最適化しますが、設定を変更してデザイン結果の特定項目にのみ目を絞る、他の項目のトレードオフを図ることも可能です。ツールまたは

デバッグ・オプションによっては、デザイン・フローが制約を受ける可能性があります。デザインで何が重要であるかが分かれば、この情報がデザインで使用すべきツール、機能、方法を選択するのに役立ちます。この章で、複雑な FPGA デザインをプランするための考慮事項をすべて列挙することはできませんが、デザインの優先順位が分かったら、ここで述べるデザイン・プランニング問題を指針として使用して、生産的かつ適切な FPGA デザイン・フローを確立するのに役立てることができます。



この章では、Quartus II ソフトウェアでのさまざまなデザインおよびプランニング機能の概要を述べます。Quartus II デザイン・フローおよび機能の概要については、「[Introduction to Quartus II Software](#)」を参照してください。Quartus II の特定の機能および方法の詳細については、この章では「[Quartus II ハンドブック](#)」の該当する章を参照先として示しています。デザイン・ファミリを選択したら、アルテラ・ウェブサイトのデバイス資料ページの「[デザイン・ガイドライン](#)」セクションを参照し、他にもそのデバイス・ファミリに関するガイドラインがないか調べることができます。

## デザイン仕様の作成

ロジック・デザインを作成したり、システム・デザインを完成させる前に、デザインの詳細な仕様を確定する必要があります。仕様では、システムの動作を定義し、FPGA の I/O インタフェースを指定します。また、基本デザイン機能のブロック図も含めます。これらの仕様を作成することにより、デザイン効率が向上します。

また、この段階でテスト・プランを作成すると、DFT (Design-For-Testability) および DFM (Design-For-Manufacturability) も容易になります。例えば、ビルトイン・セルフ・テスト機能を実行して、インタフェースをドライブしますか？その場合は、FPGA デバイス内で Nios® II プロセッサとともに UART インタフェースを使用することができます。すべてのデザイン・インタフェースを検証する機能が必要な場合があります。システムに実装されたデバイスの解析およびデバッグに関連するガイドラインについては、[1-12 ページ](#)の「[オンチップ・デバッグ・プランニングのオプション](#)」を参照してください。

デザインに複数の設計者が携わっている場合、この時点で共通のデザイン・ディレクトリ構造について検討することも得策です。これによって、デザインの統合ステージが容易になります。[1-17 ページ](#)の「[階層およびチーム・ベース・デザインのプランニング](#)」に、チーム・ベースのデザインに関するより詳細な提案が記載されています。

## デバイスの 選択

デザイン・プランニングの最初のステージでは、アプリケーションに最適なデバイスを選択します。デバイスの選択は、ボードの仕様とレイアウトを含む、以降のデザイン・サイクルに影響します。このプランのほとんどは、Quartus II ソフトウェアの外部で実行されますが、この項ではプラン・プロセスで役立ついくつかの提案を示します。

デザインのニーズに最も適したデバイス・ファミリーを選択することが重要です。ファミリーごとに、コスト、性能、ロジックおよびメモリ集積度、I/O 集積度、消費電力、パッケージングなど提供するトレードオフもさまざまです。I/O 規格のサポート、高速トランシーバ、グローバル / リージョナル・クロック・ネットワーク、PLL (Phase-Locked Loop) などの機能要件も考慮に入れる必要があります。各デバイス・ファミリーの重要な機能は、アルテラ・ウェブサイト ([www.altera.co.jp/literature/lit-sg.jsp](http://www.altera.co.jp/literature/lit-sg.jsp)) のセレクト・ガイドで確認できます。各デバイス・ファミリーには、デバイス機能の詳細をまとめたデバイス・ハンドブックやデータシートも用意されています。

必要なデバイス集積度の決定は、デザイン・プランニング・プロセスの中でも困難な作業になる可能性があります。ロジック・リソースが多く、I/O 数の多いデバイスほど、大きく複雑なデザインを実装できますが、コストも高くなります。デザイン・サイクルの後半でロジックを追加したり、オンチップ・デバッグ用にロジックとメモリを確保する場合は、ある程度の安全マージンを持ってデザインのニーズに最適なデバイスを選択します (1-12 ページの「オンチップ・デバッグ・プランニングのオプション」を参照)。異なるサイズのメモリ・ブロックや特定の演算機能を実装するためのデジタル信号処理 (DSP) ブロックなど、特定のタイプの専用ロジック・ブロックの必要性を検討してください。

アルテラ・デバイスをターゲットとする以前のデザインがある場合、それらのリソース使用率を新しいデザインの見積もりとして使用することができます。既存のデザインを Quartus II ソフトウェアでコンパイルしますが、この場合デバイス選択を **Auto** に設定してリソース使用率を確認し、デザインに適合するデバイス集積度を探します。コーディング・スタイル、デバイス・アーキテクチャ、および Quartus II ソフトウェアで使用される最適化オプションが、デザインのリソース使用率に大きく影響する可能性があることに留意してください。

アルテラの IP (Intellectual Property) デザインの特定のコンフィギュレーションに対するリソース使用率を見積もるには、アルテラ・ウェブサイト ([www.altera.co.jp/literature/lit-ip.jsp](http://www.altera.co.jp/literature/lit-ip.jsp)) の IP メガファンクション・ページのアルテラ・メガファンクションおよび IP MegaCores® のユーザーガイドを参照してください。ユーザーガイドに記載された値は、デザインのリソース使用率を見積もるのに役立ちます。

## デバイスの移行プランニング

デザインの完成が近づくと、デザインを他のデバイス集積度に移行し柔軟性をもたせるかどうか、あるいはデザインが量産レベルに達したときに HardCopy® ASIC に移行するかどうかを決定します。場合により、設計者は小型（かつ安価な）デバイスをターゲットにし、後でデザインに適合させるために必要であれば大型デバイスに移行することができます。また、大型デバイスでデザインのプロトタイプを作成し、最適化時間を短縮し、タイミング・クロージャをより短時間で達成した後、プロトタイプ作成後に最終的な小型デバイスに移行することも可能です。同様に、多くの設計者はデザインが完成し、量産の準備が整うと、デザインを FPGA デバイス向けにコンパイルし、最適化してから HardCopy ASIC に移行します。この柔軟性が必要な場合は、デザイン・サイクルの初期段階で、Quartus II ソフトウェアでこれらのマイグレーション・オプションを指定する必要があります。ターゲット・マイグレーション・デバイスは、Settings ダイアログ・ボックスの Device ページにある Migration compatibility セクションまたは Companion device セクションで指定します。

デバイス集積度またはパッケージ・サイズに応じて、ピンの果たす機能が異なる場合があるため、マイグレーション・デバイスの選択はピン配置に影響します。Quartus II ソフトウェアでピン・アサインメントを行う場合、Pin Planner の Pin Migration View で、マイグレーション・デバイス間で機能が変更されるピンが強調表示されます。（詳細は、1-7 ページの「[早期ピン・プランニングおよび I/O 解析](#)」を参照してください。）コンパニオン・デバイスを選択した場合、デザインが選択した HardCopy デバイスに確実に適合するようロジック使用率が制限される場合があります。マイグレーション・デバイスまたはコンパニオン・デバイスをデザイン・サイクルの後半で追加することができますが、ピン・アサインメントのチェックという余分な作業が必要になり、さらに新しいターゲット・デバイスに適合するようにデザインの変更が必要になる場合もあります。デザインがほぼ完成しマイグレーションが可能な状態となるデザイン・サイクルの後半よりも前半の方がこれらの問題に容易に対応できます。

さらに、HardCopy ASIC の使用を予定している場合、デザイン・サイクルの早い段階で HardCopy のガイドラインを調査して、どの Quartus II 設定を使用すべきか、あるいは検討すべき他の制約がないか確認してください。ASIC の検証要件が厳しいため、HardCopy デバイスへの移行する場合、完全なタイミング制約を使用することが特に重要です。



HardCopy デザインのタイミング要件と解析について詳しくは、「[HardCopy シリーズ・ハンドブック](#)」を参照してください。

## デバイス・プログラミング / コンフィギュレーションの プランニング

デバイス・プランニングのもう一つの重要な要素は、システム内のデバイスのプログラムまたはコンフィギュレーション方法を決定することです。プログラミング方法またはコンフィギュレーション方法を事前に選択することにより、システム設計者やボード設計者は、システムにコンパニオン・デバイスが必要かどうか、また必要な場合はどのコンパニオン・デバイスを使用するか決定できます。また、ボード・レイアウトも、プログラマブル・デバイスに使用する予定のプログラミングまたはコンフィギュレーション方法の種類によって異なります。多くのプログラミング・オプションでは、JTAG インタフェースを使用してデバイスに接続するため、ボード上で JTAG チェインをセットアップする必要がある場合があります。

デバイス・ファミリ・ハンドブックでは、特定のデバイス・ファミリで使用可能なコンフィギュレーション・オプションを説明しています。コンフィギュレーション・オプションについては、「[コンフィギュレーション・ハンドブック](#)」を参照してください。CPLD デバイスのプログラミングについては、デバイスのデータシートまたはハンドブックを参照してください。アルテラ・デバイスのプログラミングとコンフィギュレーションには、以下のオプションが含まれています。

- エンハンスド・コンフィギュレーション・デバイスの使用 — これらのデバイスは、業界標準フラッシュ・メモリを、同時およびダイナミック・コンフィギュレーション、データ圧縮、クロック分周、外部フラッシュ・メモリ・インタフェースなどのデバイス機能を含む、豊富な機能を備えたコンフィギュレーション・コントローラと組み合わせます。またエンハンスド・コンフィギュレーション・デバイスで、リモートおよびローカル・システム・アップデートを実装することもできます。
- アルテラ MAX<sup>®</sup> デバイスなどのメモリ・コントローラとフラッシュ・メモリ・デバイスの使用 — このフラッシュ・メモリ・コントローラは、PC またはマイクロプロセッサとインタフェースし、パラレル・ポート経由でコンフィギュレーション・データを受信できます。
- Quartus II シリアル・フラッシュ・ローダ (SFL) の使用 — この手法では、同じ JTAG インタフェースを使用して、FPGA をコンフィギュレーションし、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できます。
- Quartus II パラレル・フラッシュ・ローダ (PFL) の使用 — このソリューションでは、JTAG インタフェースから迅速にデータを受信して、受信側ターゲットのフラッシュ・デバイス向けにフォーマットされたデータを生成し、フラッシュ・デバイスのプログラミング時間を大幅に短縮します。システムに既にコモン・フラッシュ・インタフェース (CFI) 対応フラッシュ・メモリが内蔵されている場合は、それを FPGA コンフィギュレーション・ストレージにも利用できます。これは PFL 機能が、多数の一般的な業界標準フラッシュ・デバイスをサポートしているためです。この方法を選択する場合、システム・デザイン・サイクルの初期段階でサポートされているフ

ラッシュ・デバイスのリストをチェックし、それに応じてプランニングします。サポートされるフラッシュ・デバイスのリストについては、「AN 386: Using the MAX II Parallel Flash Loader with the Quartus II Software」を参照してください。

## 早期消費電力見積もり

Quartus II 消費電力見積もりおよび解析ツールを使用して、PCB ボード設計者とシステム設計者に情報を提供できます。ソース・コードを作成する前、またはデザインの暫定版を作成したときに、早期消費電力見積もりを実行し、ついでデザインの完成時に最も正確な解析を実行することができます。

適切な電力供給量を把握し、電源、電圧レギュレータ、ヒート・シンク、および冷却システムを設計するには、デバイスの消費電力を正確に見積もる必要があります。消費電力の見積もりと解析には、以下の2つの重要なプランニング要件があります。

- 熱プランニング — 冷却ソリューションによって、デバイスで発生した熱を十分に放逸できるようにします。特に、計算されたジャンクション温度がデバイスの標準仕様の範囲内に収まる必要があります。
- 電源プランニング — 電源は十分な電流を供給することによって、デバイスの動作をサポートしなければなりません。

FPGA デバイスの消費電力はデザインによって異なるため、早期段階でのボードの仕様およびレイアウト作成時に考慮する必要があります。アルテラの PowerPlay Early Power Estimator スプレッドシートでは、デザインで使用されるデバイス・リソース、動作周波数、トグル・レート、および環境面への配慮などに関する情報を処理して、デザインが完成する前に消費電力を見積もることができます。

既存のデザインがある場合、またはデザインが部分的に完成している場合は、Quartus II ソフトウェアで生成される Power Estimator ファイルから現行デザインのスプレッドシートに情報を渡すことができます（「Early Power Estimator ファイル」を参照）。

デザインが完成したら、Quartus II ソフトウェアの PowerPlay Power Analyzer ツールを使用して、デザインの消費電力を正確に見積もり、熱バジェットおよび電源バジェットに違反していないことを確認します。

サポートされている各デバイス・ファミリの PowerPlay Early Power Estimator スプレッドシートは、アルテラ・ウェブサイト ([www.altera.co.jp/support/devices/estimator/pow-powerplay.jsp](http://www.altera.co.jp/support/devices/estimator/pow-powerplay.jsp)) を参照してください。

デザイン・サイクルの初期段階での消費電力の見積もりにより、電力バジェットをプランニングできるため、設計者は電力量を把握しながら PCB を開発できます。



消費電力の見積もりおよび解析について詳しくは、「Quartus II ハンドブック Volume 3」の「PowerPlay による電力解析」の章を参照してください。

## Early Power Estimator ファイル

Early Power Estimator スプレッドシートにデータを入力するときは、デバイス・リソース、動作周波数、トグル・レート、その他のパラメータを含める必要があります。これらの値を指定するには、デザインに精通する必要があります。既存のデザインがない場合は、デザインで使用するデバイス・リソース数を見積もって、その情報を手動で入力します。既存のデザインがある場合、またはデザインが部分的に完成している場合、Power Estimator ファイルを生成できます。

まず、Quartus II ソフトウェアでデザインをコンパイルします。コンパイルの終了後、Project メニューの **Generate PowerPlay Early Power Estimator File** をクリックします。このコマンドを使用すると、Quartus II ソフトウェアは、Power Estimator カンマ区切り値ファイル (.csv) (または、旧デバイス・ファミリの場合はテキスト・ファイル [.txt]) を書き出します。

PowerPlay Early Power Estimator スプレッドシートには、消費電力見積もりファイルの情報を分解しスプレッドシートに転送する Import Data マクロがあります。マクロを使用しない場合は、データを Early Power Estimator スプレッドシートに手動で転送します。

既存の Quartus II プロジェクトがフル・デザインの一部のみをカバーしている場合は、最終デザインで使用する追加リソースを手動で入力する必要があります。消費電力見積もりファイル情報をインポートした後、スプレッドシートを編集し、追加デバイス・リソースを追加できます。

## 早期ピン・ プランニング および I/O 解析

トップレベルの FPGA I/O ピンを早期にプランニングして、ボード設計者が PCB のデザインおよびレイアウトの開発を開始できるようにすることが重要です。FPGA デバイスの I/O 機能は、ピン位置およびその他のタイプのアサインメントに影響を及ぼします。ボード・デザイン・チームが FPGA のピン配置を指定する場合は、できるだけ早期に FPGA 配置配線ソフトウェアでピン位置を確認して、ボード・デザインを変更する必要が生じないようにすることが非常に重要です。

従来、設計者とシステム設計者は、デザインが完成するまで、FPGA ピン・アサインメントの有効性を確認できませんでした。現在は、Quartus II Pin Planner を使用してアルテラの FPGA の暫定ピン配置を作成してから、標準 I/O インタフェース（メモリとバス・インタフェースなど）、およびシステム要件で定義されるその他の I/O 関連のアサインメントに基づいてソース・コードを設計できます。1-9 ページの「I/O 解析のためのトップレベルのデザイン・ファイルの作成」を参照してください。Quartus II の I/O Assignment Analysis は、ターゲットの FPGA アーキテクチャでピン位置とピン・アサインメントがサポートされていることをチェックします。I/O Assignment Analysis を使用して、デザイン・プロセスを通じて作成または変更した I/O 関連アサインメントを検証することができます。

Pin Planner により、I/O ピン・アサインメントのプランニング、アサインメント、および検証を簡単に行うことができます。Pin Planner Package View により、ピン番号ではなくデバイス・パッケージ・ビューを使用して、ピン位置およびその他のアサインメントを作成します。Pads View には、シリコン・ダイの周囲の I/O パッドが順番に表示され、パッド距離とピン配置のガイドラインに沿って操作することができます。Pin Planner を使用すると、I/O バンク、電圧リファレンス (VREF) グループ、および差動ピンの組み合わせを識別でき、I/O プランニング・プロセスを円滑に進められます。1-4 ページの「デバイスの移行プランニング」の説明に従ってマイグレーション・デバイス (HardCopy デバイスを含む) を選択すると、Pin Migration View に、現在選択しているデバイスと比較したときに、マイグレーション・デバイスの機能が変化するピンがハイライトされます。Device Migration View でピンを選択すると、Pin Planner の他のピンに対してクロスプローブが行われるため、デバイスのマイグレーション情報を使用してピン・アサインメントをプランニングできます。また **Enable Advanced I/O Timing** オプションで生成される“ボードを考慮した”シグナル・インテグリティ・レポートで使用できるように、選択したピンのボード・トレース・モデルをコンフィギュレーションすることもできます。このオプションにより、非常に正確な I/O タイミング解析を実行できます。デザイン・フローで通常スプレッドシートを使用している場合、Microsoft Excel スプレッドシートを使用して I/O プランニング・プロセスを開始する選択肢があるため、すべてのピンが割り当てられているときに、スプレッドシート用の I/O アサインメントを含むコンマ区切り値 (.csv) ファイルをエクスポートできます。

プランニングが完了したら、ピン位置に関する情報を PCB 設計者に渡すことができます。Pin Planner は、特定の PCB デザイン EDA ツールと密接に統合され、これらのツールからピン位置の変更を読み出して提案された変更をチェックすることができます。特にピン配置を変更する必要がある場合は、デザインが配置されたボード上で正しく機能するように、

Quartus II ソフトウェアと回路図ツールおよびボード・レイアウト・ツールとの間でピン・アサインメントを一致させることが重要です。システム設計者は Quartus II ソフトウェアを使用して、個々のロジック・ブロックをデザインしているチーム・メンバーにピン情報を渡すことができるため、メンバーがデザインをコンパイルするときにタイミング・クロージャが改善されます。デザインが完成したら、Quartus II Fitter のレポートを使用して、ピン・アサインメントの最終的なサインオフを行います。

FPGA のピン・プランニングを早期に (HDL デザインが完成する前に) 開始することにより、早期ボード・レイアウトに対する信頼が高まり、エラーが発生する可能性が低くなり、デザインの全体的な「Time-To-Market」が短縮されます。



I/O アサインメントおよび解析について詳しくは、「Quartus II ハンドブック Volume 2」の「I/O 管理」の章を参照してください。Quartus II ソフトウェアとサードパーティ EDA ツール間での I/O 情報の受け渡しについて詳しくは、「Quartus II ハンドブック Volume 2」の「I/O および PCB ツール」セクションの「Mentor Graphics PCB Design Tools Support」および「Cadence PCB Design Tools Support」の章を参照してください。

### I/O 解析のためのトップレベルのデザイン・ファイルの作成

デザイン・プロセスの早期 (ソース・コードが作成される前) に、システム設計者は通常、デザインで使用される I/O インタフェースと IP コアに関する情報を持っています。この情報を Pin Planner の **Create/Import Megafunction** 機能で使用して、デザイン I/O インタフェースに関する詳細を指定できます。これらの詳細の指定により、すべての I/O 情報を含むトップレベルのデザイン・ファイルを作成できるため、Quartus II ソフトウェアで I/O アサインメントを解析できます。

Pin Planner は MegaWizard® Plug-In Manager とインタフェースし、I/O インタフェースを使用するカスタム・メガファンクションおよび IP コアの作成またはインポートを可能にします。**Set Up Top-Level Design File** ダイアログ・ボックスで、選択したポートの一致するノード名を指定して、これらの相互接続方法をコンフィギュレーションします。Pin Planner で、これらのインタフェースまたは他のデザイン I/O ピンのその他の I/O 関連アサインメントを行います。

できるだけ多くの情報を入力したら、**Create Top-Level Design File** コマンドを使用して、トップレベル・デザイン・ファイルを生成します。Pin Planner は、内部ノードに対して仮想ピン・アサインメントを作成する

ため、コンパイル中、内部ノードはデバイス・ピンに割り当てられません。新規に生成されたトップレベルのラッパー・ファイルの解析と合成の後で、生成されたネットリストを使用して、**Start I/O Assignment Analysis** コマンドで I/O 解析を実行します。

I/O 解析の結果を使用して、ピン・アサインメントまたは IP パラメータを変更し、I/O インタフェースがデザイン要件を満たし、Quartus II ソフトウェアのピン・チェックに合格するまで、チェック・プロセスを繰り返すことができます。この初期ピン・プランニングが終了したら、Quartus II で生成されたネットリストに基づいて、Quartus II のリビジョンを作成できます。ここで、作業の進め方を選択できます。すなわち、生成されたネットリストを使用して実際のデザイン用のトップレベル・ファイルを開発するか、生成されたネットリストを無視して実際のデザインでは生成された Quartus II 設定ファイル (.qsf) を使用することができます。

## サードパーティ EDA ツールの フローの選択

完全な FPGA デザイン・フローには、Quartus II ソフトウェアの他にサードパーティ EDA ツールを含めることができます。Quartus II ソフトウェアで使用するツールを決定し、それらのツールがサポートされ正しく設定されているか、また有用な機能または不必要な制限を認識しているかどうかを確認します。

### 合成ツール

Quartus II ソフトウェアの合成ツール、または好みのサードパーティ合成ツールを使用して、デザインを合成することができます。合成ツールが異なると、得られる結果が異なることがあります。アプリケーションに対して最高の性能を発揮するツールを選択したい場合は、アプリケーションおよびコーディング・スタイルに対応した標準的なデザインを合成し、結果を比較することによってツールを実験することができます。正確なタイミング解析およびロジック使用率の結果を得るために、必ず Quartus II ソフトウェアで配置配線を実行してください。合成の結果は、配置配線前の見積もりであるため、合成のためのブラック・ボックスとして扱われるロジック（一部の合成ツールのメガファンクションやアルテラの IP コアなど）は含まれていません。また、これらの見積もりでは、レジスタ・パッキングまたはタイミングとリソース使用率のいずれの結果にも影響するフィジカル・シンセシスなど、Quartus II のその他の最適化を通じて、Quartus II Fitter で達成されるロジック使用の低下は考慮されていません。

ツール・ベンダは絶えず新機能を追加し、ツールの問題を修正し、アルテラ・デバイスに対する性能を向上させているため、最新バージョンのサードパーティ合成ツールを使用するようにしてください。「[Quartus II ソフトウェア・リリース・ノート](#)」には、Quartus II ソフトウェアの当該バージョンで正式にサポートされている各合成ツールのバージョンが記載されています。

New Project Wizard または **Settings** ダイアログ・ボックスの **EDA Tools Settings** ページで合成ツールを指定して、合成ネットリストに適した Library Mapping ファイルを使用します。

合成ツールによっては、Quartus II プロジェクトを作成し、EDA ツールの設定、デバイスの選択、および合成プロジェクトで指定したタイミング要求などの制約を渡す機能を提供するものもあります。配置配線のために Quartus II プロジェクトを設定するときに、この機能を使用して時間を節約することができます。

インクリメンタル・コンパイル手法を活用する場合、合成するデザインをパーティションに分割し、複数の出力ネットリスト・ファイルを生成しなければなりません。詳細は、1-18 ページの「**デザイン・パーティションを含むインクリメンタル・コンパイル**」を参照してください。



合成ツールについて詳しくは、「Quartus II ハンドブック Volume 1」の「**合成**」セクションの該当する章を参照してください。

## シミュレーション・ツール

アルテラでは、Quartus II ライセンス・サブスクリプション付きの ModelSim Altera シミュレータを提供しています。また、Quartus II ソフトウェアは、他のサードパーティ・シミュレーション・ツールをサポートするタイミング・ネットリスト・ファイルを生成できます。

サードパーティ製シミュレーション・ツールを使用する場合は、Quartus II バージョンでサポートされているソフトウェア・バージョンを使用してください。Quartus II ソフトウェア・リリース・ノートには、Quartus II ソフトウェアの特定のバージョンで正式にサポートされている各シミュレーション・ツールのバージョンが記載されています。また、必ず使用中の Quartus II ソフトウェア・バージョンに付属のモデル・ライブラリを使用してください。ライブラリはバージョン間で変更されている可能性があり、その場合は使用中のシミュレーション・ネットリストとの不一致が生じます。

**Settings** ダイアログ・ボックスの **EDA Tools Settings** ページでシミュレーション・ツールを指定して、適切な出力シミュレーション・ネットリストを生成します。



シミュレーション・ツールのフローについて詳しくは、「Quartus II ハンドブック Volume 3」の「**シミュレーション**」セクションの該当する章を参照してください。

## フォーマル検証ツール

Quartus II ソフトウェアは、フォーマル検証フローをサポートしています。必要とするフォーマル検証フローがデザインおよびデザインのコンパイル・ステージに影響を及ぼすか否かについて検討してください。

フォーマル検証フローを使用すると、レジスタのリタイミングなどの特定のロジック最適化をオフにし、階層ブロックが強制的に維持され、それによって最適化が制約されるので、性能結果に影響を及ぼす可能性があります。フォーマル検証ではメモリ・ブロックをブラック・ボックスとして扱います。したがって、メモリを別の階層ブロックに保持して、他のロジックが検証のためにブラック・ボックスに取り込まれないようにすることが最良の策です。デザインが制限されるその他の制約もあります。詳しくはドキュメントを参照してください。デザインにフォーマル検証が重要な場合は、デザイン・フローの初期段階で制限と制約をプランニングする方が、後半で変更を行うよりも簡単です。

**Settings** ダイアログ・ボックスの **EDA Tools Settings** ページでフォーマル検証ツールを指定して、適切な出力ネットリストを生成します。



フォーマル検証のフローについて詳しくは、「**Quartus II ハンドブック Volume 3**」の「**フォーマル検証**」セクションの該当する章を参照してください。

## オンチップ・ デバッグ・ プランニング のオプション

アルテラのインシステム・デバッグ・ツールは、さまざまな利点およびトレードオフを提供するため、システムおよび設計者によっては特定のデバッグ・ツールがより高い性能を発揮することがあります。システム・ボード、Quartus II プロジェクト、およびデザインがすべて適切なオプションに対応して設定されるように、デザイン・プロセスの早期段階でオンチップ・デバッグ・オプションを評価すると効果的です。プランニングによってデバッグに費やす時間を短縮でき、後で使用するデバッグ手法に合わせて変更を加える手間を省くことができます。

検証ツールの Quartus II ポートフォリオには、以下のシステム・デバッグ機能が含まれます。

- **SignalProbe** インクリメンタル配線—この機能は、デザインに影響を与えることなく、内部信号をすばやく I/O ピンに配線することによってデザイン検証の効率を高めます。完全に配線されたデザインから開始して、デバッグ用の信号を選択し、以前予約した I/O ピンまたは現在未使用の I/O ピンに配線することができます。

- **SignalTap® II エンベデッド・ロジック・アナライザ** — このロジック・アナライザは、FPGA デバイスでデザインをフル・スピードで実行しながら、外部装置や余分な I/O ピンを使用しないで、デザインの内部信号の状態をプローブすることによって、FPGA デザインをデバッグするのに役立ちます。カスタム・トリガ条件ロジックを定義して、精度を向上させ、問題を特定する能力を改善します。**SignalTap II エンベデッド・ロジック・アナライザ**は、デザインの内部ノードまたは I/O ピンの状態をキャプチャするために外部プローブやデザイン・ファイルへの変更を必要としません。キャプチャしたすべての信号データは、ユーザーがデータを読み出して解析できるようになるまでデバイス・メモリに便宜的に保存されます。
- **ロジック・アナライザ・インタフェース (LAI)** — このインタフェースにより、外部ロジック・アナライザを接続し、FPGA の内部信号をこのアナライザに送信して解析することができます。この機能を使用して、多数の内部デバイス信号をデバッグのために少数の出力ピンに接続することができます。また、外部ロジック・アナライザまたはミックスド・シグナル・オシロスコープの最新機能を活用できます。
- **In-System Memory Content Editor** — この機能は、JTAG インタフェースを介してイン・システム FPGA メモリおよび定数への読み出しおよび書き込みアクセスを提供し、システム内でデバイスが動作している間に、FPGA のメモリ内容および定数値への変更をより簡単にテストできるようにします。
- **In-System Sources and Probes** — この機能は、カスタマイズされたレジスタ・チェーンを設定して、ロジック・デザインに組み込まれたノードをドライブまたはサンプリングし、シンプルな仮想ステイミュラスを提供し、組み込まれたノードの現在の値をキャプチャします。**SignalTap II** ロジック・アナライザを使用してトリガ条件を設定し、外部テスト装置を使用しないでデザインをエキササイズするための単純なテスト・ベクタを作成し、JTAG チェーンを使用してランタイム・コントロール信号をダイナミックに制御することができます。
- **Virtual JTAG メガファンクション** — `sld_virtual_jtag` メガファンクションを使用すれば、システム・レベルのデバッグのためのソフトウェアにおけるプロセッサ・ベースのデバッグ・ソリューションおよびデバッグ・ツールなど、独自のシステム・レベルのデバッグ基盤を構築することができます。`sld_virtual_jtag` メガファンクションは、HDL コードで直接インスタンス化し、デバイスの JTAG インタフェースを使用して FPGA デザインの一部にアクセスするために、1本または複数の透過通信チャネルを提供することができます。



デバッグ・ツールについて詳しくは、1-25 ページの「参考資料」を参照してください。使用するオプションを決定するのに役立つデバッグ・オプションの概要については、「Quartus II ハンドブック Volume 3」の「Section V. In-System Design Debugging」の「はじめに」を参照してください。

これらの機能のいずれかを使用する場合、システム・ボード、Quartus II プロジェクト、およびデザインを開発するときに機能のプランニングが必要なこともあります。以下、デザイン・プランニング・ステージで考慮すべきさまざまな要素について説明します。

SignalTap II エンベデッド・ロジック・アナライザ、ロジック・アナライザ・インタフェース、In-System Memory Content Editor、In-System Sources and Probes、および Virtual JTAG メガファンクションはすべて、イン・システム・デバッグを実行するのに JTAG 接続を必要とします。デバッグに使用できる JTAG ポート付きのシステムおよびボードをプランします。

また、JTAG デバッグ機能では、JTAG ハブ・ロジックを実装するために少量の追加ロジック・リソースも必要になります。デザイン・サイクルの早期段階で適切な機能を設定すると、これらのデバイス・リソースを早期リソース見積りに含めて、ロジックを過剰に使用することを防ぐことができます。

SignalTap II エンベデッド・ロジック・アナライザは、システム動作中にデバイス・メモリを使用してデータをキャプチャします。このデバッグ手法を活用するのに十分なメモリ・リソースを確保するために、デバッグ時に使用するデバイス・メモリを予約することを検討します。

SignalTap II エンベデッド・ロジック・アナライザでインクリメンタル・デバッグを使用するには、**Full incremental compilation** オプションがオンになっていなければなりません。このオプションは、Quartus II ソフトウェア v6.1 以降で作成したプロジェクトに対しては、デフォルトでオンに設定されますが、既存のプロジェクトでは自動的にオンにはなりません。インクリメンタル・コンパイルがイネーブルされていない場合は、デバッグ機能を追加するとき、または SignalTap II 設定に特定の変更を行うときに、デザイン全体を再コンパイルする必要があります。SignalTap II エンベデッド・ロジック・アナライザでインクリメンタル・コンパイルを使用することによって、デバッグに必要なコンパイル時間が大幅に短縮されます。

SignalProbe およびロジック・アナライザ・インタフェースには、デバッグ用の I/O ピンが必要です。後でデバッグ信号に対応するためにデザインやボードを変更しなくてすむように、デバッグ用の I/O ピンを予約しておくことを検討してください。ロジック・アナライザ・インタフェー

スでは、必要に応じて信号をデザイン I/O ピンでマルチプレクス化できることに注意してください。対象のボードで、デバッグ信号がシステムの動作に影響を与えないデバッグ・モードがサポートされていることを確認します。

カスタム・デバッグ・アプリケーションに Virtual JTAG メガファンクションを使用する場合は、これをインスタンス化し、デザイン・プロセスの一部として組み込む必要があります。

In-System Sources and Probes 機能では、HDL コードでメガファンクションをインスタンス化することも必要です。さらに、オプションで SignalTap II エンベデッド・ロジック・アナライザをメガファンクションとしてインスタンス化し、それをデザインのノードに手動で接続して、タップされたノード名が合成時に変更されないようにすることができます。デバッグ・ブロックをインクリメンタル・コンパイル用の個別デザイン・パーティションとして追加して、再コンパイル時間を短縮することができます。

RAM や ROM ブロック、あるいは lpm\_constant メガファンクションに In-System Memory Content Editor を使用するには、MegaWizard Plug-In Manager でメモリ・ブロックを作成するときに、**Allow In-System Memory Content Editor to capture and update content independently of the system clock** オプションを必ずオンにします。

## デザイン 手法と HDL コーディング・ スタイル

複雑な FPGA デザインの開発では、適切なデザイン手法およびコーディング・スタイルが、デバイスのタイミング性能、ロジック使用率、およびシステムの信頼性にきわめて大きな影響を与えます。最良の合成結果およびフィージビリティ結果が得られるよう、アルテラの推奨事項を順守してください。

### デザインの推奨事項

常にデザイン目標を達成できるように、同期デザイン方式を使用します。その他のデザイン手法に関する問題としては、デバイスにおける伝播遅延への依存、不完全なタイミング解析、およびグリッチが発生する可能性があります。同期デザインでは、クロック信号がすべてのイベントをトリガします。同期デザインは、すべてのレジスタのタイミング要求が満たされている限り、すべてのプロセス、電圧、および温度 (PVT) 条件に対して、予測可能かつ信頼性を確保した形で動作します。同期デザインであれば、異なるデバイス・ファミリーまたはスピード・グレードをターゲットに変更するのが容易になります。

クロック信号は、デザインのタイミング精度、性能、および信頼性に大きな影響を及ぼすので、特に注意してください。クロック信号に関連する問題は、デザインにおいて機能上の問題およびタイミング問題を引き起こす可能性があります。最良の結果を得るために、専用のクロック・ピンとク

ロック配線を使用し、ターゲット・デバイスで PLL が利用できる場合は、クロック反転、逡倍、分周に PLL を使用します。クロックの多重化およびゲーティングには、デバイスで使用可能な場合には、組み合わせロジックの代わりに専用のクロック・コントロール・ブロック、または PLL クロック・スイッチオーバー機能を使用します。内部生成クロック信号を使用する必要がある場合は、グリッチを低減するためにコントロール信号として使用される組み合わせロジックの出力をラッチします。

Quartus II ソフトウェアのデザイン・アシスタントは、デザイン・フローの早期におけるデザイン問題のチェックを可能にするデザイン・ルール・チェック・ツールです。デザイン・アシスタントは、デザインがアルテラ推奨のデザイン・ガイドラインまたはデザイン・ルールを順守しているかどうかをチェックします。デザイン・アシスタントを実行するには、Processing メニューで **Start** をポイントして、**Start Design Assistant** をクリックします。デザイン・アシスタントがコンパイル時に自動的に実行されるように設定するには、**Settings** ダイアログ・ボックスで **Run Design Assistant during compilation** をオンにします。また、サードパーティの「リント」ツールを使用してコーディング・スタイルをチェックすることもできます。

また、デバイスのターゲット・アーキテクチャを理解することも設計に役立ち、それらの機能を活用するようにデザインを最適化することができます。例えば、コントロール信号がデバイス・アーキテクチャの専用コントロール信号を使用することが重要です。したがって、最良の結果を得るために、デザインで使用する異なるコントロール信号の数を制限する場合があります。



デザインの推奨事項および Design Assistant の使用について詳しくは、「Quartus II ハンドブック Volume 1」の「[Design Recommendations for Altera Devices](#)」および「[Quartus II Design Assistant](#)」の章を参照してください。業界紙を参照して、複数のクロック・デザインに関する詳しい情報を入手することもできます。適切な解析については、[www.sunburst-design.com](http://www.sunburst-design.com) のホワイトペーパー内の [CummingsSNUG2001SJ\\_AsyncClk.pdf](#) ファイルを参照してください。

## 推奨 HDL コーディング・スタイル

HDL コーディング・スタイルは、プログラマブル・ロジック・デザインの結果の品質 (QoR) に大きな影響を与える可能性があります。最適な合成結果が得られるように、アルテラの推奨コーディング・スタイルを使用してください。メモリ・ファンクションおよびデジタル・システム処理 (DSP) ファンクションを設計する場合、デバイスのターゲット・アーキテクチャを理解すれば、専用のロジック・ブロックのサイズやコンフィギュレーションを活用できます。メガファンクションの推測、お

よびメモリ・ブロックや DSP ブロックなどの専用デバイス・ハードウェアをターゲット化するためのコーディング・ガイドラインに従ってください。



具体的な HDL コーディング例と推奨事項については、「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。その他のツール特有のガイドラインについては、合成ツールのドキュメントを参照してください。Quartus II ソフトウェアでは、テキスト・エディタの右クリック・メニューから選択できる Language Templates の HDL の例を使用できます。

## 階層および チーム・ ベース・ デザインの プランニング

Quartus II インクリメンタル・コンパイルでのコンパイル時間の節約と性能の維持を活用できる、階層デザインを作成する場合は、デザイン・サイクルの最初からインクリメンタル・コンパイル・フローをプランします。以下の項では、デザインのパーティションを使用しないでデザイン階層がフラット化されるフラット・コンパイル・フローを説明した後、トップダウン、ボトムアップ、およびそれらの混在したデザイン手法でデザイン・パーティションを使用するインクリメンタル・コンパイル・フローを説明します。インクリメンタル・コンパイル・フローにはいくつかの利点がありますが、良質な結果を得るために多くのデザイン・プランニングが必要です。最後の項では、インクリメンタル・コンパイル・フローをプランニングする際に考慮すべき要素として、デザイン・パーティションのプランニングとデザイン・フロアプランの作成について説明します。



Quartus II ソフトウェアでのインクリメンタル・コンパイル・フローの使用について詳しくは、「Quartus II ハンドブック Volume 1」の「階層およびチーム・ベース・デザインのための Quartus II インクリメンタル・コンパイル」の章を参照してください。

### デザイン・パーティションを含まないフラット・コンパイル・フロー

Quartus II ソフトウェアのこのコンパイル・フローでは、デザイン全体が「フラット」ネットリストとしてコンパイルされます。このフローはデザイン・パーティションを作成しない場合に使用します。ソース・コードを階層化することができますが、デザインはコンパイル時にフラット化され、デザインが変更された後で再コンパイルされるたびに、デザインのソース・コードはすべて合成されて、ターゲット・デバイスに組み込まれます。デザイン全体を処理することにより、デザイン全体で利用可能なすべてのロジックおよび配置の最適化が実行され、面積と性能が改善されます。インクリメンタル・デザイン・フローでは、SignalTap II Logic Analyzer などのデバッグ・ツールを使用できますが、コンパイル中にデザイン階層を維持するためにデザイン・パーティションを指定することはできません。

フラット・コンパイル・フローは使いやすく、デザイン・パーティションのプランニングは必要ありません。ただし、デザインが変更されるたびに、デザイン全体が再コンパイルされるため、大型デバイスではコンパイル時間が比較的長くなることがあります。また、デザインのある部分を変更した場合、結果的に別の部分に変更されることがあります。



Quartus II ソフトウェアでは、フル・インクリメンタル・コンパイル・オプションはデフォルトでオンになっているため、プロジェクトではインクリメンタル・コンパイルのためのデザイン・パーティションをすぐに作成できます。下位のデザイン・パーティションを作成しない場合、デザイン全体が1つのデザイン・パーティションと見なされ、ソフトウェアはフラット・コンパイル・フローを使用します。

## デザイン・パーティションを含むインクリメンタル・コンパイル

インクリメンタル・コンパイル・フローでは、システム開発者は大規模なデザインを個別に設計可能な小さなパーティションに分割します。チーム・メンバーは個々にパーティションで作業できるため、デザイン・プロセスが簡素化され、コンパイル時間が短くなります。

階層化されたデザイン・パーティションが正しく選択され、デザイン・フロアプランに正しく配置されている場合、結果の品質を維持あるいは改善しながらデザインのコンパイル時間をスピード・アップすることができます。

デザインのほとんどの部分でそれ以上改善する必要がなく、特定の1つのブロックを変更または最適化する場合は、デザイン・サイクルの後の方でインクリメンタル・コンパイルを使用できます。この場合、変更されていないモジュールの性能を維持し、以降の繰り返しに要するコンパイル時間を短縮することができます。

インクリメンタル・コンパイルは、コンパイル時間の短縮とタイミング・クロージャの達成の両方に有効なこともあります。例えば、以降のインクリメンタル・コンパイルで維持する必要があるパーティションを指定し、高度な最適化を有効にした状態で他のパーティションを再コンパイルすることができます。

デザインの一部が完成していない場合、未完成の部分には空のパーティションを作成しておき、完成したパーティションをコンパイルすることができます。次に、完成したパーティションの結果を保存しながら、デザインの新しい部分での作業を進めます。

あるいは、別々の設計者や IP プロバイダがチーム・ベース手法を使用し、デザインの異なるブロックで作業して、それらをボトムアップ・コンパイル・フローで結合することができます。

デザイン・コードと階層をプランニングする場合、各デザイン・エンティティが別のファイルで作成され、ファイルのソース・コードを変更するときに、エンティティの独立性が維持されるようにしてください。サードパーティ合成ツールを使用する場合は、合成ツールでデザイン・パーティションごとに個別の Verilog Quartus Mapping (VQM) または EDIF ネットリストを作成します。また、合成ツール内で個別にプロジェクトを作成して、ツールが各パーティションを個々に合成し、個別の出力ネットリスト・ファイルを生成しなければならないことがあります。Quartus II インクリメンタル・コンパイルのサポートについては、合成ツールのドキュメントを参照してください。この場合、ネットリストはインクリメンタル・コンパイルのソース・ファイルと見なされます。

## トップダウンとボトムアップのインクリメンタル・フロー

Quartus II のインクリメンタル・コンパイル機能は、異なるデザイン方法に適した、トップダウンとボトムアップ両方のコンパイル・フローをサポートします。また、これらのフローを混在させたコンパイル・フローを使用することもできます。以下の項では、ユーザーのデザイン・ニーズに最も適したフローを選択できるように、コンパイル・フローのそれぞれを簡単に説明します。

### トップダウン・インクリメンタル・コンパイル・フロー

トップダウン・コンパイルでは、1人の設計者またはプロジェクト・リーダーが、デザイン全体をコンパイルします。複数の設計者または IP プロバイダが、デザインの異なる部分を設計および検証し、作業の完了時にプロジェクト・リーダーがプロジェクトにデザイン・エンティティを追加することができます。また、デザインの一部を最適化のターゲットとしながら、残りの部分を“エンティティ”として指定することもできます。すべてのデザイン・ロジックのソースかどうかに関係なく、プロジェクト・リードはトップレベルのプロジェクトを全体としてコンパイルおよび最適化します。

インクリメンタル・コンパイルでは、デザインで変更されていないパーティションのコンパイル結果と性能が維持され、新しいコンパイルを変更されたデザイン・パーティションにのみ集中させることによって、デザインのイタレーション時間を大幅に短縮します。新しいコンパイル結果は、変更されていないデザイン・パーティションの前のコンパイル結果と併合されます。さらに、フィジカル・シンセシスなどの最適化手法のターゲットを特定のデザイン・パーティションに絞り、他のパーティ

ションは変更しないでおくことも可能です。またデザインの一部が未完または欠落している場合にも、エンプティ・パーティションでこのフローを使用できます。

### ボトムアップおよびチーム・ベース・インクリメンタル・フロー

ボトムアップ・デザイン・フローにより、個々の設計者は個別のプロジェクトでのそれぞれのデザインの最適化を完了し、下位レベルの各プロジェクトを1つのトップレベルのプロジェクトに統合できます。ボトムアップ手法には、デザイン・パーティションが別の場所のチーム・メンバ、またはサードパーティ IP プロバイダによって作成されるチーム・ベースのデザイン・フローも含まれます。

インクリメンタル・コンパイルには、ボトムアップ設計手法を可能にするエクスポート機能とインポート機能があります。下位レベル・ブロックの設計者は、LogicLock™ 領域などの一連のアサインメントとともに、自分のデザインのために最適化したネットリストをエクスポートできます。次に、システム設計者はトップレベル・プロジェクトで、各デザイン・ブロックをデザイン・パーティションとしてインポートします。

ボトムアップ・デザイン・フローでは、システム開発者が下位レベルのブロックの設計者に対して、各パーティションが適切なデバイス・リソースを使用するように指導することが重要です。デザインが個別に開発されるため、各下位レベルの設計者は、全体的なデザインまたは各自のパーティションが他のパーティションとどのように接続されるのかについて情報を持っていません。このような情報の欠如は、システム統合時に問題を引き起こす可能性があります。下位レベル・パーティションの設計者がデザインに着手する前に、ピン位置、物理的制約、およびタイミング要求を含むトップレベル・プロジェクト情報を設計者に伝達する必要があります。

システム開発者は、トップレベルでデザイン・パーティションをプランニングし、Quartus II インクリメンタル・コンパイルを使用して、自動的に生成されたスクリプトを介して下位レベルの設計者に情報を伝達することができます。 **Generate bottom-up design partition scripts** オプションは、トップレベル・プロジェクト情報の下位レベル・モジュールへの転送プロセスを自動化します。このソフトウェアには、トップレベル・デザインでプロジェクト情報を管理するための、プロジェクト・マネージャ・インタフェースが備わっています。

このスクリプトは、すべての下位デザイン・ブロックのための Quartus II プロジェクトを作成し、関連するすべてのプロジェクト・アサインメントを渡すことができます。これらのスクリプトを使用すると、下位レベル・モジュールの設計者は、容易にプロジェクト・リーダーからの指示を実装でき、またプロジェクトをトップレベル・デザインにインポートおよび組み込むときに、プロジェクト間の衝突を回避できるようになり

ます。この方法を使用すると、統合後にデザインをさらに最適化する必要性が少なくなり、設計者の生産性とチーム・コラボレーションが全体的に改善されます。

### インクリメンタル・コンパイル・フローの組み合わせ

トップダウンとボトムアップのコンパイル・フローを組み合わせ、デザインの一部にトップダウン・フローを活用しながら、独自に開発されたデザイン部分をインポートすることができます。

トップダウン・フローは、一般にボトムアップ・フローを実行するよりも簡単です。例えば、下位レベルのデザインをエクスポートおよびインポートする必要はありません。トップダウン手法では、デザイン・ソフトウェアにデザイン全体に関する情報も提供されるため、特定の位置にロックダウンされる部分がデザインにない場合に、グローバルな配置の最適化を実行できます。

ボトムアップ・デザイン手法では、ソフトウェアが下位レベルの個々のパーティションをコンパイルするとき、トップレベル・デザインでの他のパーティションに関する情報がまったく提供されないため、ユーザーが注意深くリソース・バランシングとタイム・バジェットを実行する必要があります。必要な箇所、ボトムアップ・コンパイル・フローをトップダウン・コンパイル・フローと組み合わせ、コンパイル時間を短縮し、デザインの他の部分の結果を維持する方法は、生産性を向上させるのに効果的です。

### デザイン・パーティションのプランニング

FPGA のデザインを分割するには、パーティションが統合されたときに最良の結果が得られるよう、また各パーティションがデバイス内の他のパーティションに対して適切に配置されるようにプランニングする必要があります。アルテラの推奨事項を順守してデザイン・パーティションを作成すると、結果の全体的な品質が改善されます。例えば、パーティションの I/O 境界をレジスタで受けることにより、クリティカル・タイミング・パスを個別に最適化可能な 1 つのパーティション内に維持します。デザイン・パーティションが指定された場合は、**Incremental Compilation Advisor** を使用して、パーティションがアルテラの推奨事項を確実に満たすようにします。

設計者が個々のブロックを開発する前にタイミング・バジェットを決定することにより、システム統合時にタイミング問題が発生する可能性が低減されます。下位レベルのパーティションを個別に最適化する場合、パーティション間にまたがるレジスタで受けられないパスは完全なパスとして最適化されません。ソフトウェアが各パーティションの入力および出力ロジックを正しく最適化するように、手動で何らかのタイミング・バジェットを実行する必要があります。パーティションをまたぐそれぞれ

のレジスタで受けないタイミング・パスについて、各パーティションの対応する I/O パスでタイミング・アサインメントを行い、パスの両端を予定タイミング遅延に制限します。接続の各部分にタイミング・バジェットを割り当てることによって、パスが適切に最適化され、トップレベルのデザイン要件に適合します。

リソース使用率をプランニングし、バランスを取ることが重要です。インクリメンタル・コンパイルを実行する場合、他のパーティションで使用されるリソースに関するデータなしに、各パーティションは別々に合成されます。したがって合成時に、個々のパーティションでデバイス・リソースが過剰に使用され、パーティションがマージされたときにデザインがターゲット・デバイスに適合しなくなる場合があります。

設計者が各自の下位レベルのデザインを最適化し、それらをトップレベルのデザインにエクスポートするボトムアップ・デザイン・フローでは、各パーティションの配置配線も個別に実行されます。パーティションがトップレベルで結合される際には、競合するリソースを使用してしまう場合もあります。デザイン・パーティション間でリソース使用率のバランスを取ることによって、すべてのパーティションが統合されるときにリソース競合の問題を回避することができます。



デザイン・パーティションの作成と、ソース・コードの構成のガイドラインについては、「[Quartus II ハンドブック Volume 1](#)」の「[Best Practices for Incremental Compilation Partitions and Floorplan](#)」の章を参照してください。

## デザイン・フロアプランの作成

インクリメンタル・コンパイルをフルに活用するために、デザイン・フロアプランを作成してデザイン・パーティション間の競合を回避し、各パーティションが他のパーティションに対して適切に配置されるようにすることが要求されます。各パーティションのロケーション・アサインメントを作成することにより、異なるパーティション間での配置の衝突を防ぐことができます。また、デザイン・フロアプランは、ほとんどのリソースが使用済みであるデバイス・エリアで、Fitter にデザインの一部を配置または再配置するよう指示する状況を回避するのに役立ちます。フロアプラン・アサインメントが実行されない場合、コンパイル時間の増大や結果の品質低下につながる場合があります。

ターゲット・デバイスに応じて、Quartus II の **Timing Closure Floorplan** または **Chip Planner** を使って、各デザイン・パーティションに対する **LogicLock** 領域アサインメントを使用したデザイン・フロアプランを作成することができます。トップレベル・デザインの基本的なデザインの枠組みがあれば、これらのフロアプラン・エディタを使用して、領域間の接続を表示し、チップ上のフィジカル・タイミング遅延を見積もり、

デザイン・フロアプランで領域を移動させることができます。完全なデザインをコンパイル済みの場合は、ロジック配置を表示し、配線が密集する領域を特定してフロアプラン・アサインメントを改善することもできます。

適切なパーティションとフロアプラン・デザインは、下位レベルのデザインがデザインの他の部分との統合時に、トップレベルのデザイン要件を満たすのに役立ち、トップレベル・デザインの統合とタイミングの検証に費やされる時間を短縮します。



デザイン・フロアプランでの配置アサインメントの作成については、「Quartus II ハンドブック Volume 2」の「[デザイン・フロアプランの解析および最適化](#)」の章を参照してください。インクリメンタル・コンパイルのデザイン・パーティションの作成のガイドラインについては、「Quartus II ハンドブック Volume 1」の「[Best Practices for Incremental Compilation Partitions and Floorplan Assignment](#)」の章を参照してください。

## 高速合成と 早期タイミング 見積もり

最終的なタイミング・クロージャ段階で問題を見つけるよりも、デザイン・サイクルの早期段階でデザイン問題を見つける方が、はるかにコストを抑えられます。デザイン・ソース・コードの最初のバージョンが完成したら、クイック・コンパイルを実行して、一種のシリコン仮想プロトタイプ (SVP) を作成してタイミング解析に使用できます。

Quartus II ソフトウェアで合成する場合、結果の品質は低下するおそれがあるが、コンパイル時間を短縮できる高速合成を選択できます。Assignments メニューの **Settings** をクリックします。**Analysis & Synthesis Settings** タブで、**More Settings** をクリックし、**Synthesis Effort** を設定します。

コンパイル・フローに関係なく、**Early Timing Estimate** を使用して、デザインの迅速な配置配線、およびタイミング解析を実行できます。Processing メニューの **Start** をポイントして、**Start Early Timing Estimate** をクリックします。ソフトウェアで必要に応じて自動的にデバイスが選択され、フロアプランの作成に使用される **LogicLock** 領域が配置され、すべてのデザイン・ロジックで迅速な初期配置が検索され、最終的なデザイン性能の有効な見積もりが提供されます。タイミング制約を入力した場合、タイミング解析によりこれらの制約がレポートされます。



Early Timing Estimation は、TimeQuest タイミング・アナライザとクラシック・タイミング・アナライザの両方でサポートされます。Synopsys Design Constraint (SDC) フォーマット制約を適用して TimeQuest タイミング・アナライザを使用し、クラシック・タイミング・アナライザでは使用できない Advanced Timing Analysis 機能を有効にできます。

ボトムアップ・デザイン・フローで個々のブロックを設計する場合は、デザインの開発時にこれらの機能を使用できます。下位レベルのデザイン・ブロックでハイライトされる問題は、システム設計者に伝達できます。これらの問題を解決するには、個々のブロックにデバイス・リソースを追加割り当てするか、タイミング・バジェットを変更する必要があります。

トップレベルの設計者は、デザイン全体のプロトタイプ作成にも高速合成と早期タイミング見積もりを使用できます。インクリメンタル・コンパイル・フローでは、未完成のパーティションは空としてマークし、デザインの他の部分はコンパイルして早期タイミング見積もりを実施し、デザイン統合での問題を検出できます。

システム設計者は、早期タイミング見積もりをデザイン・パーティション・スクリプトとともに使用し (1-20 ページの「ボトムアップおよびチーム・ベース・インクリメンタル・フロー」で説明)、下位レベルの設計者に追加制約を渡し、デザインの他のパーティションに関する詳細情報を提供できます。この情報は、パーティション間パスを最適化するのに特に有用となります。早期タイミング見積もりを実行すると、設計者が早期デザイン段階でデザイン問題を見つけて解決するのに役立ちます。

## まとめ

最近の FPGA は、高速タイミング性能により大規模で複雑なデザインをサポートしています。プロセスの早期段階でデザインのいくつかの側面をプランニングすることによって、プロセス後半の段階で問題処理に費やす不必要な時間を短縮できます。Quartus II ソフトウェアのさまざまな機能を使用すると、デザインを迅速にプランニングし、可能な最良の結果を達成できます。この章で提示したガイドラインを順守することにより、生産性が向上し、デザイン・コストが削減され、最終製品の「Time-to-Market」が改善されます。

## 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。

- 「Quartus II ハンドブック Volume 2」の「Analyzing and Optimizing the Design Floorplan」の章
- 「AN 386: Using the MAX II Parallel Flash Loader with the Quartus II Software」
- 「Quartus II ハンドブック Volume 1」の「Best Practices for Incremental Compilation Partitions and Floorplan Assignments」の章
- 「Quartus II ハンドブック Volume 2」の「Cadence PCB Design Tools」の章
- 「コンフィギュレーション・ハンドブック」
- 「Quartus II ハンドブック Volume 3」の「Design Debugging Using the SignalTap II Embedded Logic Analyzer」の章
- 「Quartus II ハンドブック Volume 3」の「Design Debugging Using In-System Sources and Probes」の章
- 「Quartus II ハンドブック Volume 1」の「Design Recommendations for Altera Devices」および「Quartus II Design Assistant」の章
- 「Quartus II ハンドブック Volume 3」の「フォーマル検証」セクション
- 「Quartus II ハンドブック Volume 2」の「I/O管理」の章
- 「Quartus II ハンドブック Volume 3」の「外部ロジック・アナライザを使用したイン・システム・デバッグ」の章
- 「Quartus II ハンドブック Volume 3」の「In-System Updating of Memory and Constants」の章
- 「Introduction to Quartus II Software」
- 「Quartus II ハンドブック Volume 2」の「Mentor Graphics PCB Design Tools Support」の章
- 「Quartus II ハンドブック Volume 3」の「PowerPlay による電力解析」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
- 「Quartus II ハンドブック Volume 3」の「Quick Design Debugging Using SignalProbe」の章
- 「Quartus II ハンドブック Volume 3」の「シミュレーション」セクション
- 「sld\_virtual\_jtag Megafunction User Guide」
- 「Quartus II ハンドブック Volume 1」の「合成」セクション

## 改訂履歴

表 1-1 に、本資料の改訂履歴を示します。

表 1-1. 改訂履歴		
日付およびドキュメント・バージョン	変更内容	概要
2008 年 5 月 v 8.0.0	<ul style="list-style-type: none"> <li>● 編成を変更。</li> <li>● 「デザイン仕様の作成」の項を追加。</li> <li>● Volume 3 「In-System Design Debugging」への参照を追加。</li> <li>● 「デザイン手法と HDL コーディング・スタイル」の項に詳細を追加。</li> <li>● 新しい「Best Practices for Incremental Compilation Partition and Floorplan Assignments」の章への参照を追加。</li> <li>● Quartus II Language Templates への参照を追加</li> </ul>	Quartus II ソフトウェア v8.0 のリリースによるトピックの追加と更新。
2007 年 10 月 v7.2.0	1-22 ページの「参考資料」を再編成。	Quartus II ソフトウェア v7.2 のための更新。
2007 年 5 月 v7.1.0	<p>Quartus II ソフトウェア v7.1 のリリースにより以下を更新。</p> <ul style="list-style-type: none"> <li>● 「はじめに」、「デバイス・マイグレーション・プランニング」、「早期ピン・プランニングと解析」の項を拡張。</li> <li>● 新しい項として、「サードパーティ EDA ツールのフローの選択」と「デバッグ・オプションのプランニング」を追加。</li> <li>● その他のマイナー・チェンジおよび編成。</li> <li>● 参考資料の項を追加。</li> </ul>	Quartus II ソフトウェア v7.1 のためのトピックの追加と更新。
2007 年 3 月 v7.0.0	Quartus II ソフトウェア v7.0 のリビジョンおよび日付のみ更新。その他の変更はありません。	—
2006 年 11 月 v6.1.0	初版	—