

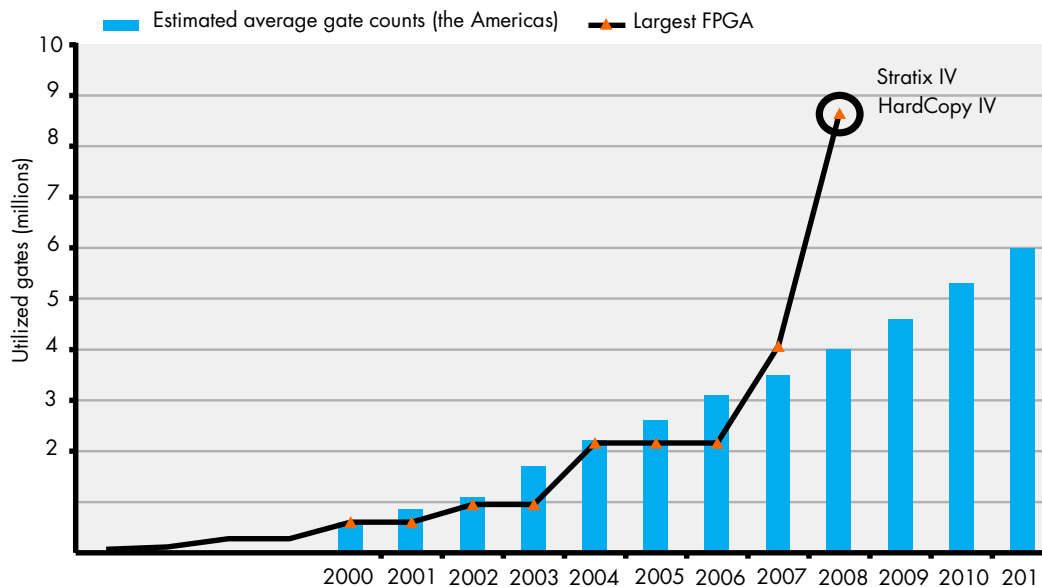
Quartus II インクリメンタル・コンパイルによる生産性の向上

はじめに

FPGA のロジック集積度と性能が向上する一方、設計者への「Time-to-Market」の圧力はますます大きくなっています。演算能力は、合成、配置、および配線のためのコンパイル時間を維持できるほどには伸びていないのが現状です。例えば、アルテラ・デバイスは過去 10 年で、ロジック・セル数が 35 倍、メモリ・ビット数が 100 倍に増加していますが、演算能力の伸びは 10 倍にとどまります。高集積 FPGA でデザインが大規模になり性能が高くなるほど、コンパイル時間が長くなります。

今日の FPGA 設計者は、従来は ASIC デザインにのみ関係していたコンパイル時間とタイミング・クロージャの達成に関して懸念を募らせています。2008 年には、アルテラ最大集積度の Stratix® IV および HardCopy® IV デザインは、ゲート数が ASIC デザインの平均ゲート数の 2 倍超になると予想されています (図 1 を参照)。コンパイル時間を短縮するアルテラの努力がなければ、これらの大規模なデザインでは、デザインが変更されるたびにコンパイルに丸一日以上かかり、非効率でデザイン時間が無駄になります。

図 1. 平均的な ASIC デザインのサイズを上回る FPGA (1)



注:

(1) データは Gartner Dataquest 社のレポート (2007 年 11 月 21 日付) に基づくものです。

FPGA のデザインおよびデバッグ・ステージにおいて迅速にコンパイルを繰り返し実行できることが成功には不可欠です。設計者は、各デザイン・イタレーションでコンパイル時間を短縮することを望んでおり、タイミング・クロージャ結果を保持することで、デザインの完成に必要なデザイン・イタレーション回数を少なくできると確信しています。これらの懸案事項に対応するために、ASIC デザインで実証済みのアプローチが FPGA にも定着してきています。FPGA および EDA ベンダは、従来は ASIC デザイン・ツールでしか使用できなかったインクリメンタル・デザインおよびコンパイル機能を提供しています。これらの機能には、デザインの再利用、デザインの進化、および ECO (Engineering Change Order) をサポートするトップダウン手法と、チーム・ベースのデザイン・フローを含むボトムアップ・デザイン手法があります。

アルテラの Quartus® II デザイン・ソフトウェアは、今日の FPGA 設計者が求める生産性の向上を実現するインクリメンタル・コンパイルに対応しています。このホワイトペーパーでは、インクリメンタル・コンパイル・フローがどのように高集積、高性能 FPGA のデザインの生産性を向上するかを説明します。

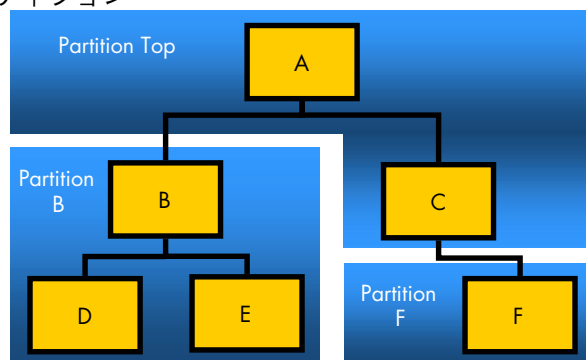
従来のフラット・コンパイルとインクリメンタル・コンパイルの比較

これまでは、ロジック・シンセシスとフィッティング（または配置配線）の前に、階層構造デザインは単一のネットリストにフラット化されていました。デザインが変更されるたびに、デザイン全体がリコンパイルされます。その理由の1つは、コンパイラはデザイン全体を処理することによって、グローバルな最適化を実行し、リソース利用率とタイミング性能を改善できることです。しかし、この方法では、わずかなデザイン変更に対して、予想していたより長くコンパイル時間がかかることがあります。また、デザインの一部に行う変更が、デザインの他の部分のタイミング・クロージャに影響することがわかると、不満を感じるでしょう。プッシュ・ボタン式コンパイルによりタイミング要求を容易に満たすことができる場合は、CPLDまたは低集積度FPGAでのデザインなど、小さなデザインに対してフラット・コンパイル・フローを正常に使用できます。コンパイル時間およびタイミング・クロージャの結果の維持が問題でない場合、フラット・デザインは満足できるものです。

インクリメンタル・フローを使用することによって、デザインに変更を加えても、変更されないロジックの結果と性能は維持されます。これにより、デザインのコンパイル時間が大幅に短縮されるため、1日あたりのデザインのイタレーション回数を増やし、タイミング・クロージャをより効率的に達成できます。また、インクリメンタル・デザイン・フローは、デザインの簡単な再利用やチーム・ベースのデザイン手法をサポートしており、今日の地理的に散在するデザイン環境でデザイン・ブロックを個別に作成し、最適化することができます。大規模なデザイン、高集積デバイス、およびデバイス・アーキテクチャの速度に対して高性能が要求されるデザインでは、インクリメンタル・コンパイルが推奨されます。

インクリメンタル・コンパイルを使用する場合、インクリメンタル機能を可能にするため、デザイン階層はデザイン・パーティションにマップされ、コンパイル時に個別に処理されます。デザインの各エンティティまたはインスタンスは、自動的にデザイン・パーティションとして考慮されないため、トップレベルの下にある1つまたは複数のデザイン階層を、インクリメンタル・コンパイル用のデザイン・パーティションとして指定する必要があります。パーティションが宣言されると、そのパーティション内のすべての階層は、同じパーティションに含まれるようになります。既存のパーティション内の階層に新しいパーティションを作成すると、新しい下位レベルのパーティション内のロジックは、上位レベルのパーティションの一部ではなくなります。図2に、インスタンスBおよびFがデザイン・パーティションとして指定されたデザイン階層の例を示します。パーティションBには、サブインスタンスDおよびEが含まれています。デフォルトのパーティション「Top」には、トップレベルのブロックAと、他のどのパーティションにも割り当てられていないインスタンスCが含まれます。

図2. デザイン階層のパーティション



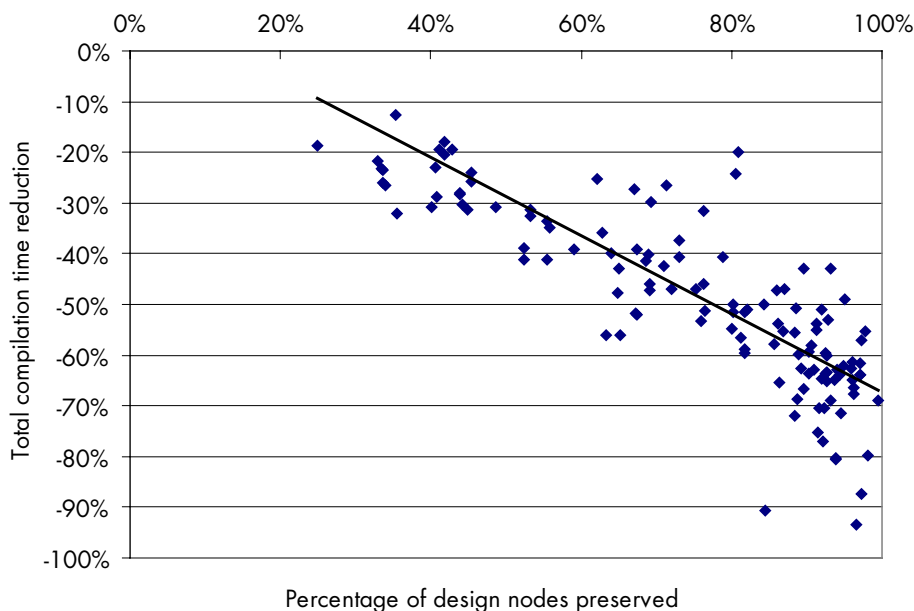
パーティションを作成すると、ソフトウェアは合成または配置時に、パーティション境界を越えてロジックの最適化を行わないため、各パーティションをいつでも個別に合成または配置配線できるようにすることによって、インクリメンタル動作も可能になります。

インクリメンタル・コンパイルによる生産性の向上

インクリメンタル・コンパイルでは、前項で説明したように、デザイン階層を合成とフィッティングのためのロジカル・パーティションに分割できます。デザインをリコンパイルするときに、ソフトウェアは更新された HDL コードと設定を使用するか、各パーティションのコンパイル結果を再利用して維持することができます。コンパイル作業を特定のパーティションに限定して行い、変更されないパーティションはリコンパイルしないことにより、デザインのイタレーション回数が減ります。フィジカル・シンセシスなどの最適化手法を特定のデザイン・パーティションに対してのみ実行し、他のロジック・ブロックはそのままにしておくこともできます。

図 3 のベンチマーク・テストによると、デザイン変更後に、デザイン・ノードの 80% 以上を維持した状態でリコンパイルすると、平均コンパイル時間は約 50 ~ 70% 短縮されます。これは、1 回のフラット・コンパイルと同じ時間で、2 回または 3 回のデザイン・イタレーションを実行できることを意味します。

図 3. デザインの変更されない部分を維持した場合のフィタ時間の短縮



性能が維持されることも、インクリメンタル・コンパイル手法の大きな利点です。デザイン内の特定のパーティションのみをコンパイルするため、残りのパーティションのタイミング性能は変更なく維持されます。性能が維持されるため、少ないデザインのイタレーション回数で効率的にタイミング・クロージャに到達できます。

インクリメンタル・コンパイルは、トップダウンとボトムアップの両方のデザイン手法をサポートしています。トップダウン・デザイン・フローでは、さまざまな設計者または IP (Intellectual Property) プロバイダが HDL コードを個別に作成および検証し、1 名の設計者でデザイン全体の最終的な Quartus II プロジェクトを管理できます。ボトムアップ・デザイン・フローにより、個々の設計者または IP プロバイダは個別のプロジェクトでのそれぞれのデザインの配置配線の最適化を完了し、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エクスポートできます。これらの手法では、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企業によって作成されるチーム・ベースのデザイン・フローをサポートします。個々のパーティションは、パーティション間で影響を与えることなく統合できます。また、インクリメンタル・フローでは、一部のデザイン・パーティションが欠けていたり不完全な場合でも、それ以外のパーティションをコンパイルして最適化できます。

インクリメンタル・フローが有効なデザイン・シナリオ

インクリメンタル・コンパイルは、デザイン・フローのさまざまなステージで役立ちます。これから紹介するシナリオでは、インクリメンタル・コンパイルがさまざまな局面で、どのようにデザインの生産性向上に役立つかを説明します。

ソース・ファイル変更時のコンパイル時間の短縮

インクリメンタル・コンパイルでは、デザインの他の部分をリコンパイルすることなく、1 つのパーティションのソース・コードを更新できます。この機能は、コンパイル後に HDL ソース・ファイルでエラーが見つかった場合などに効果的です。修正がデザインの一部にのみ限定され、他のデザイン・ブロックのタイミング性能に影響しないと考えられる場合は、影響を受けるコードのみコンパイルし、デザインの残りの部分は維持することが望ましいといえます。

インクリメンタル・コンパイルのためにデザインがいくつかのパーティションに分割されている場合は、次のように Quartus II ソフトウェアで 1 つのソース・ファイルを更新するだけですみます。まず、HDL ソース・ファイルを修正し、保存します。変更されなかったパーティションのフィッティング後のネットリストを再利用するようにソフトウェアを設定し、ソース・ファイルで変更が検出されたパーティションをソフトウェアが自動的にリコンパイルできるよ

うにします。これにより、コンパイル時間はデザイン全体を再処理するよりも大幅に短縮され、また変更されていないブロックの性能が維持されるため、追加タイミング・クロージャ作業の必要性も少なくなります。

他のロジックを追加する前にデザインの一部の結果を最適化

インクリメンタル・コンパイルでは、1つのパーティション・セットを単独で最適化し、デザインの残りの部分を完成させる間に、配置を固定して結果を維持することもできます。例えば、カスタム・ロジックの残りを組み込む前に、最適化を実行する命令とセットになったIPのためにパーティションを作成できます。このフローは、各クリティカル・パスが1つのパーティション内に含まれ、パーティション間に依存関係がない場合に最も効果的です。

このシナリオでは、追加が必要となる最適化をすべてオンにした状態で、タイミング・クリティカルIPブロックが設定されたパーティションのみコンパイルされます。他のデザイン・ブロックは「エンプティ」パーティションとして割り当てて、これらのロジックがコンパイルされないようにします。クリティカル・パーティションに対してタイミング・クロージャが達成された後、フィッティング後のネットリストを使用してその内容と配置を維持します。これで、このロジック・ブロックのタイミング・クロージャを変更する必要はなく、このロジックを残りのパーティションのためにコンパイルできます。

インシステム・ロジック・アナライザによるインクリメンタル・デバッグ

インクリメンタル・デバッグは、インクリメンタル・デザイン・フローの強力な機能です。内部ロジック・アナライザを追加してデザインをデバッグする場合、またはロジック・デザインやその配置を修正することなく、ロジック・アナライザのコンフィギュレーションを修正する場合は、このフローでコンパイル時間を短縮することができます。Quartus II インクリメンタル・コンパイルでは、オリジナル・デザインの結果を保持し、オリジナル・ソース・コードをリコンパイルすることなく、アルテラの SignalTap® II ロジック・アナライザを追加します。

SignalTap II ロジック・アナライザでインクリメンタル・デバッグを実行するのに、デザイン・パーティションを作成する必要はありません。このアナライザは、インクリメンタル・コンパイルのための独自のデザイン・パーティションとして自動的に機能します。デフォルトの Top パーティションを含むすべてのデザイン・パーティションのフィッティング後の結果を保持するようソフトウェアを設定します。デザイン・ネットリスト内のフィッティング後のノード名をブロープするようにアナライザを設定します。この設定によって、Quartus II Fitter は次回のコンパイル時に、既存のデザイン結果を修正することなく、フィッティング後のネットリストに SignalTap II ロジックを追加することができます。

チーム・ベースのデザイン環境とボトムアップ・デザイン・フローの実装

複数の下位レベル・デザインで構成されるプロジェクトは、異なる設計者が別々に実装する複数のプロジェクトに分割できます。このシナリオでは、トップレベルのプロジェクトによって下位レベルの各デザインがインスタンス化され、各ブロックの設計者は各自のデザインを個々に最適化してタイミング・クロージャを確実にした後で、プロジェクト・リーダーにコンパイル結果を渡すことができます。

このシナリオでは、プロジェクト・リーダーは最初にプロジェクトを作成します（このプロジェクトには最終的にデザイン全体が含まれます）。次に、各サブデザインのために空のデザイン・パーティションを作成し、下位レベル・デザインを担当する設計者にデザインの制約事項を渡します（一般にリソース位置の競合を避けるためのフロアプラン制約が含まれます）。チーム・メンバは各自のデザインを個々に最適化し、タイミング要求を検証して、それぞれのパーティションを配置情報とともにエクスポートします。プロジェクト・リーダーは各デザイン・ブロックの情報をトップレベル・プロジェクトにインポートして、デザイン全体をコンパイルします。

すべてのタイミング・クリティカルなパスが1つのパーティション内で分離されている場合、トップレベル・デザインは追加作業なしでタイミング要求を満たします。境界間パスがタイミング要求を満たしていない場合、プロジェクト・リーダーは下位レベルの設計者に追加情報を提供して各デザインの最適化を支援することができます。Quartus II ソフトウェアには、デザイン全体に関する情報（パーティションを接続するポートの位置など）を下位レベル・ブロックの各設計者に渡すスクリプトを自動的に生成する機能が含まれています。

インクリメンタル・デザインの設定ガイドラインの遵守

インクリメンタル・コンパイル・フローでは、フル・フラット・コンパイルよりも綿密な先行計画が必要になる場合があります。例えば、ロジックが最適化のために正しくグループ化されるには、ソース・コードやデザイン階層が構造化されていなければならない場合があります。デザイン・サイクルの早い時点で正しいロジック・グループ化を実装するほうが、後でコードを再構成するよりも簡単です。デザインがパーティションに分割されると、設計者は各パーティションをデバイス上の物理的位置に割り当てて、デザインのフロアプランを作成したい場合があります。パーティションまたはフロアプランの割り当てが不適切な場合は、デザイン領域の利用率や性能が低下し、タイミング・クロ

ジャがより困難になる可能性があります。インクリメンタル・コンパイルでは、一般にフラット・コンパイルの場合よりも最良のデザイン手法に厳密に従う必要があります。

デザインを計画する場合は、各パーティションのサイズとスコープを考慮し、またデザインの進展に従って、デザインの異なる部分に変化する可能性を検討してください。頻繁に変更されるロジックは、デザインの固定部分から分離しておかなければなりません。デザイン階層では、1つのパーティション内のタイミング・クリティカルなロジックを（理想的にはレジスタで受けたポート境界で）分離して、ソフトウェアが各パーティションを個別に効果的に最適化できるようにする必要があります。

デザインを計画するときに、アルテラのドキュメントに記載されているガイドラインに従うと、良好な結果を得ることができます。FPGA が大規模かつ複雑になるほど、すべてのデザイン・フローで最良のデザイン手法に従うことがますます重要になります。推奨される階層同期デザイン手法に準拠すれば、デザインは信頼性が高くデバッグも容易なものになります。インクリメンタル・コンパイル・フローを使用すると、プロジェクトに必要なステップや要件が増えることもありますが、クリティカルなブロックの性能を維持し、コンパイル時間を短縮することによって、デザイン生産性において大きなメリットが得られます。

まとめ

FPGA デザインのサイズが大きくなり高性能化が進むのに伴い、FPGA デザイン・コミュニティではインクリメンタル・コンパイル手法が定着してきています。アルテラの Quartus II インクリメンタル・コンパイル手法は、デザインのコンパイル時間を平均で 60% 短縮し、デザイン性能を維持し、チーム・ベースのデザイン・フローをサポートすることによって、設計者が必要とする生産性の向上を達成します。デザインのプランとパーティション分割によっては、デザインのイタレーション時間が短縮され、開発時間が極端に短くなることもあります。性能維持とチーム・ベースのフローがインクリメンタル・コンパイルを通じてサポートされているため、少ないデザインのイタレーション回数で効率的にタイミング・クロージャに到達し、迅速な「Time-to-Market」を達成できます。

詳細情報について

- 「Quartus II ハンドブック Volume 1」の「Quartus II Incremental Compilation for Hierarchical and Team-Based Design」：
www.altera.co.jp/literature/hb/qts/qts_qii51015_j.pdf
- 「Quartus II ハンドブック Volume 1」の「Best Practices for Incremental Compilation Partitions and Floorplan Assignments」：
www.altera.co.jp/literature/hb/qts/qts_qii51017_j.pdf
- 「Leveraging the 40-nm Process Node to Deliver the World's Most Advanced Custom Logic Devices」：
www.altera.co.jp/literature/wp/wp-01058-stratix-iv-40nm-process-node-custom-logic-devices_j.pdf

謝辞

- Jennifer Stephenson, MTS Applications Engineer, Software Applications Engineering, Altera Corporation
- Scott Brissenden, Advanced Software Engineer, Performance Analysis Group, Software and IP Engineering, Altera Corporation
- Albert Chang, Senior Product Marketing Engineer, Software Marketing, Altera Corporation

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。



101 Innovation Drive
San Jose, CA 95134
www.altera.com

Copyright © 2008 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.