ISSUE 55 2005

Xcell journal

THE AUTHORITATIVE JOURNAL FOR PROGRAMABLE LOGIC USERS

新しい ISE 8.1i ソフトウェアによる 高速タイミング クロージャ

POWERED BY FMAX TECHNOLOGY

DESIGN PERFORMANCE
ISE ソフトウェアによる
物理合成と最適化

VERIFICATION
ABV を活用する
初期欠陥の発見

PRODUCTIVITY
ISE Foundation の
アーキテクチャ
ウィザード活用法

PARTIAL RECONFIGURATION
パーシャル
リコンフィギュレーションの
プラットフォームとして
活用する設計・分析ツール
PlanAhead ソフトウェア

VOTED #1 ISE SOFTWARE

XILINX®

Xcell journal

TABLE OF CONTENTS 2005, ISSUE 55

VIEWPOINT

統合開発環境 ISE ソフトウェアを使った タイミング クロージャの革新的な向上 …… 2

DESIGN PERFORMANCE

ISE ソフトウェアによる物理合成と最適化 …… 6

設計・分析ツール PlanAhead を使った デザイン パフォーマンスの向上 …… 10

デザイン パフォーマンス向上のための HDL コーディング法 …… 14

XST 8.1i を用いた Virtex-4 DSP48 ブロック向け RTL コーディング術 …… 19

VERIFICATION

1 回で成功する論理設計の検証法 …… 24

ABV を活用する初期欠陥の発見 …… 28

PRODUCTIVITY

FPGA 設計におけるピン配置の簡素化 …… 33

90 nm FPGA のデザインにおける消費電力の考察 … 36

ISE Foundation のアーキテクチャウィザード活用法 … 41

PARTIAL RECONFIGURATION

パーシャル リコンフィギュレーションの プラットフォームとして活用する 設計・分析ツール PlanAhead ソフトウェア …… 45

GENERAL

DSP デザインにおけるリアルタイム解析法 …… 50

ザイリンクスの FPGA と CPLD による SPI シリアル フラッシュのプログラミング …… 54

LETTER FROM THE EDITOR

生産性へのロードマップ …… 表2

INFORMATION

ザイリンクストレーニング スケジュール …… 57

ザイリンクス イベント カレンダー …… 58

広告索引

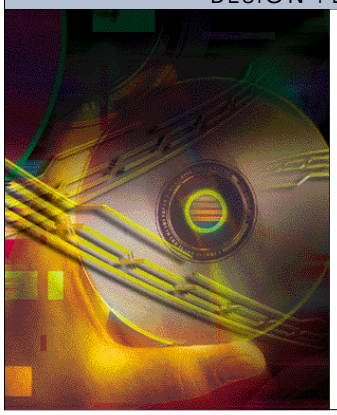
有限会社ヒューマンデータ …… 5

株式会社ミッシュインターナショナル …… 23

株式会社コンピューテックス …… 32

Xcell Journalのご送付先住所等の変更は：
http://www.xilinx.co.jp/xcell/henko/
Xcell Journalの新規定期購読のお申込みは：
http://www.xilinx.co.jp/xcell/toroku/

DESIGN PERFORMANCE

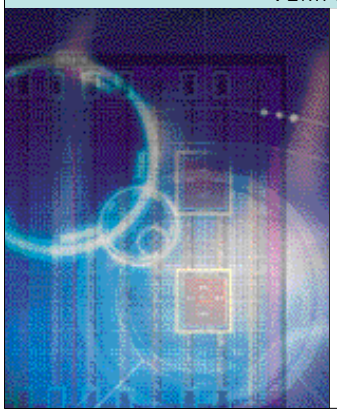


ISE ソフトウェアによる物理合成と最適化

インプリメンテーションツールを有効活用するためのヒント

6

VERIFICATION

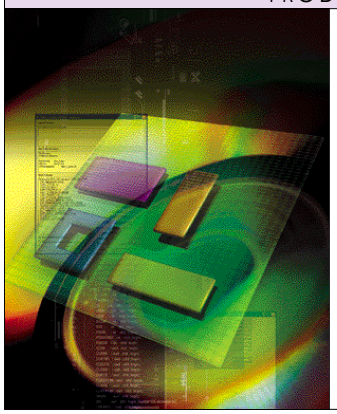


ABV を活用する初期欠陥の発見

デザイン、合成、検証の統合によるアサーション ベースのバグ発見法

28

PRODUCTIVITY

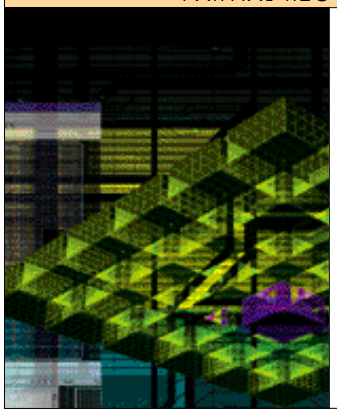


ISE Foundation のアーキテクチャウィザード活用法

ザイリンクスのデバイスにおける複合ブロックの設定とカスタマイズ プロセスの合理化法について

41

PARTIAL RECONFIGURATION



パーシャル リコンフィギュレーションのプラットフォームとして活用する設計・分析ツール PlanAhead ソフトウェア

合理的なスペース、重量、電力、コスト削減環境を提供する PlanAhead

45

Improving Time to Design Closure with ISE Software

統合開発環境 ISE ソフトウェアを使ったタイミング クロージャの革新的な向上

デザイン目的を達成するためのヒントと戦略

Steve Lass
Director, Software Product Marketing
Xilinx, Inc.
steve.lass@xilinx.com

今日の設計者が直面しているデザイン上の最重要課題は、タイミング クロージャではないでしょうか。FPGA やその他のディープサブミクロン IC では、一般に配線遅延がロジック遅延の大部分を占めます。エンベデッド PowerPC™ プロセッサをはじめとする高速コアを使うなど、タイミングを改善する方法はたくさんありますが、ここではデザイン中のロジックと配線部分のパフォーマンスを改善することに焦点を当てます。本稿では、ザイリンクスのデザインに対してタイミング クロージャを達成する方法について解説していきます。

タイミング クロージャのデザイン フロー

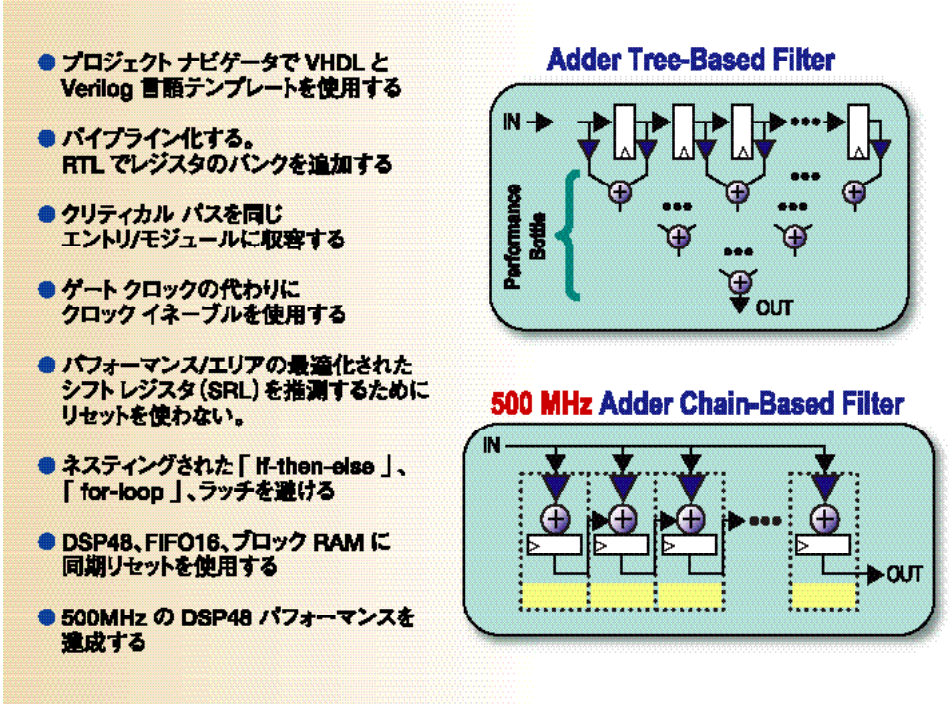
図 1 は、FPGA アーキテクチャ向けに最適化され、効率よく書かれた HDL から始まる典型的なフローです。ザイリンクスの ISE™ ソフトウェアは Verilog および VHDL テンプレートを備えているので、効率的な HDL コードを書くことができます。FPGA にはたくさんのレジスタがあるため、パイプライン ステージを追加すればタイミングを大幅に改善できるうえ、エリアに影響を及ぼすこ

ザイリンクスが最近発表した新しいユーティリティ「 Xplorer 」を使うことにより、インプリメンテーション ツールの各種オプションを自動的に試したり、そのデザインの達成 可能な最高スピードを見つけるために異なるクロック周波数を試したりできます。

とはほとんどありません。よく使われるベスト プラクティスとしては、クリティカル パスを同一のエンティティ/モジュールに収容する、ゲート クロックの代わりにクロック イネーブルを使う、HDL コードでラッチ、ネスティングされた for-loop、if-then-else 文の使用を避けるといったことが挙げられます。

ザイリンクスは、DSP48、FIFO16、ブロック RAM などのモジュールに同期リセットを使うこと、また 500 MHz の DSP48 パフォーマンスを達成するため、加算器ツリーでなく加算器チェーンを使うことを推奨しています（図 1 ）。コーディング スタイルの詳細は、本号 14 ページの「 デザイン パフォーマンス向上のための HDL コーディング法 」をご覧ください。

図 1　コーディング スタイル

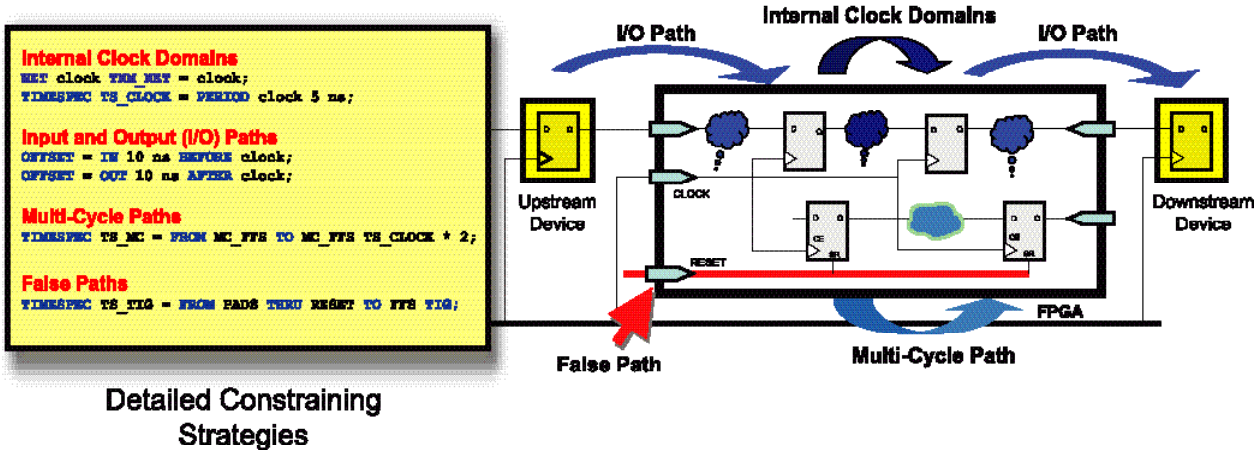


合成

次の段階は合成です。合成により、記述した HDL がタイミングとエリアの要件をうまく満たすかどうか、デザイン フローの初期段階で判断できます。合成ツールは、任意の指示を与えない限り、タイミングを犠牲にしてもエリアの最小化を優先する傾向があるため、そうならないよう必ずタイミングを制約してください。最低限、クロックと I/O パスだけでも制約する必要があります。

最適化に努めることで、合成ツールをさらに活用できます。タイミングの要件を満たすためによく使われるもう 1 つのオプションとして、ロジックを通してレジスタを前後に移動してクロック周波数を高めるレジスタ バランシングがあります。合成ツールを使用してもタイミングを満たすことができない場合

図 2　インプリメンテーション制約



や、タイミングが非常に厳しい場合、前述のコーディング テクニックをどれか 1 つ、あるいは複数組み合わせて使うことで、HDL コードをさらに最適化できるでしょう。

インプリメンテーション

合成ツールから許容し得るタイミング見積もりを得たら、デザインの実際のタイミングを決定するため、インプリメンテーション ツール（ マップ、配置・配線、タイミング解析 ）を使います。Fmax Technology を搭載するザイリンクスの ISE ツールは、最高のパフォーマンスの達成を試みますが、そのためには制約が既に完成していることが条件です。タイミング制約（ 図 2 ）には、クロック ピリオド、I/O オフセット、マルチサイクル パスの指定、および False パスを無視するためのタイミング無視（ TIG ）を含めることを推奨します。タイミングが 20 % 以上不足している場合は、特にワースト ケース パスを含むモジュール内の HDL をさらに最適化する必要があるでしょう。

それでもインプリメンテーション段階でタイミング要件を満たしきれなかった場合、劇的に改善できるツール オプションが数多くあります。まず、合成ツールでリタイミングを使うことです。この他、ISE Mapper のリタイミングおよびグローバル最適化を有効にするという方法もあります。

Xplorerユーティリティ

ザイリンクスは最近、インテリジェントな制約テクニックとさまざまな物理的最適化戦略を採用することで、最適なデザインを実現する Xplorerという新しいユーティリティを発表しました。Xplorer を使うと、インプリメンテーション ツールの各種オプションを自動的に試したり、そのデザインの達成可能な最高スピードを見つけるために異なるクロック周波数を試したりできます。Xplorer が最適なツール オプションを探し出したら、次回にインプリメンテーション ツールを実行するときは、時間のかかる Xplorer の使用を避け、それらオプションを使うようにしてください。Xplorer で最高のパフォーマンスを達成する方法については、英語版 Xcell Journal Issue 55 の「 Accelerate Design Performance Using Xplorer 」(http://www.xilinx.com/publications/xcellonline/index.htm) をご覧ください。

PlanAhead 設計ツール

あらゆる方法を試し、それでもタイミング クロージャの目標に到達できない場合にも、まだ希望はあります。ザイリンクスの PlanAhead™ ツールを使えば、デザインを解析し、必要に応じてフロアプランを作成することで、より高いパフォーマンスを達成できます（ 平均 15 % 、最高 2 倍まで改善 ）。PlanAhead 設計ツールでは、配置・配線プロセスを詳しく調べたり、「 what if 」シナリオをすばやく解析して、早い段階で潜在的な問題点を識別・修正したりできます。また、コネクティビティ解析とユーティリゼーションの制御を通して配線効率を高めるため、クリティカル パスとモジュールをグループ化できます。

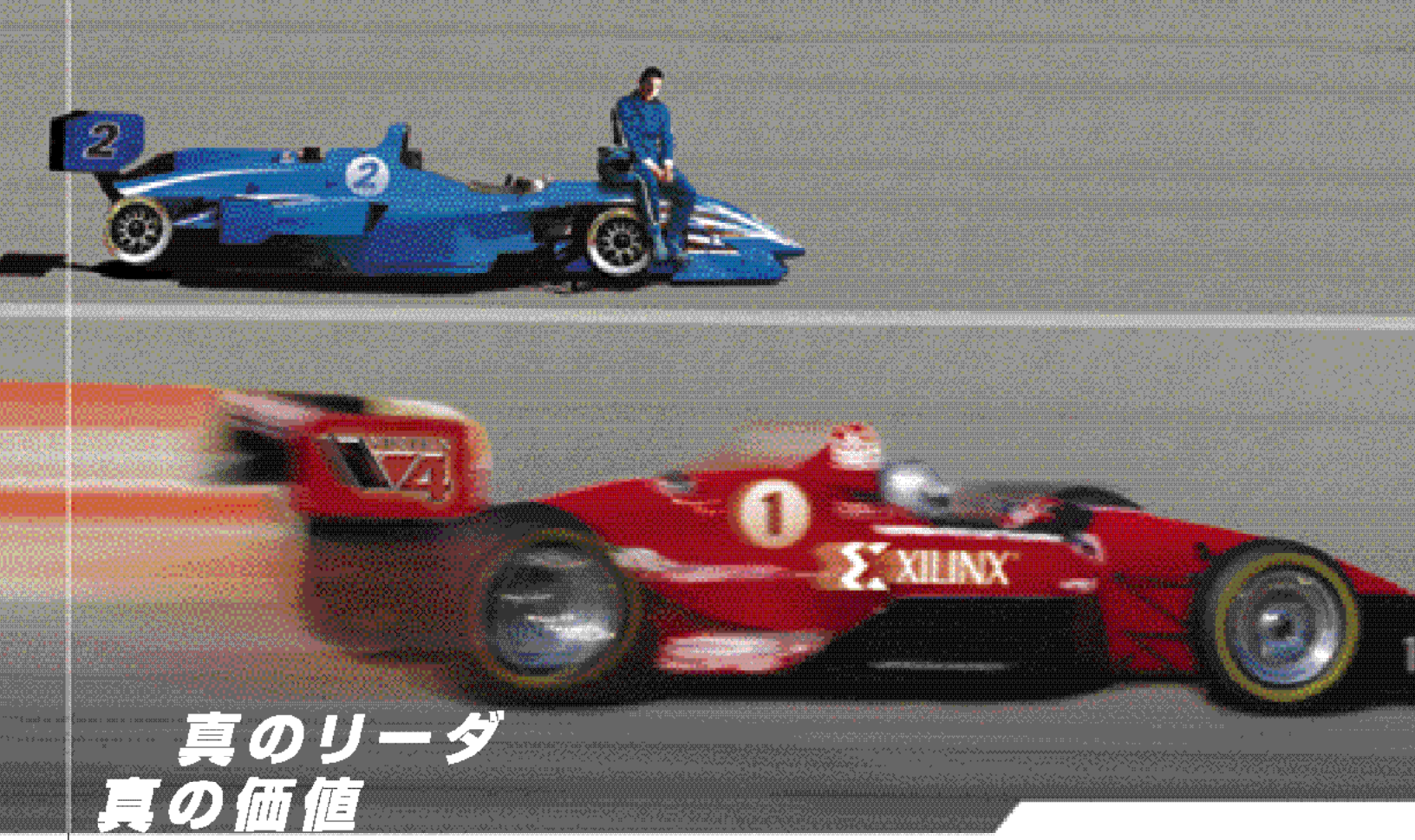
この優れたツールの各種機能については、本号 10 ページの「 設計・分析ツール PlanAhead ソフトウェアを使ったデザイン パフォーマンスの向上 」をご覧ください。詳細なフロアプランの作成や RPM（ 相対配置マクロ ）など、他にも高度なテクニックはたくさんありますが、まず本稿の方法を試してみてください。

結論

ザイリンクスは、デザイン パフォーマンスの改善に役立つ、ISE Fmax Technology 採用の包括的なソフトウェア ツールを提供しています。ISE ソフトウェアと本稿のヒントやテクニックを使うことで、タイミング クロージャをすばやく達成できます。

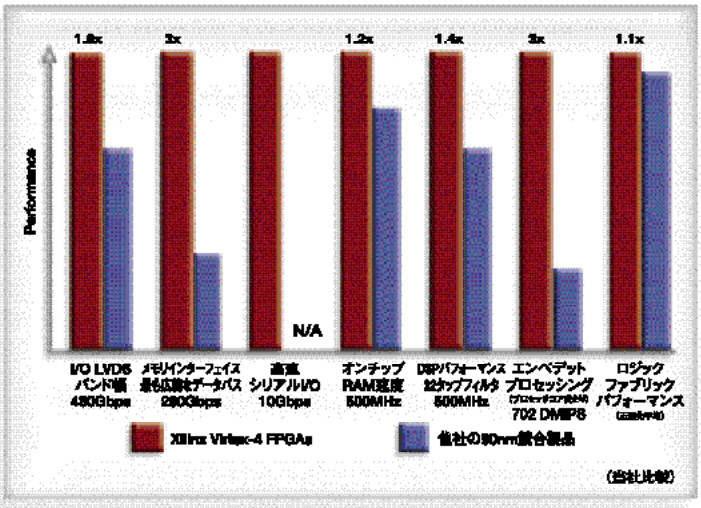
さらに、大手サードパーティ ベンダとも緊密に協力し、各社の合成ツールからザイリンクスのデバイスに対するデザインの最適化や、デザイン パフォーマンスの改善を図っています。

ザイリンクスのソフトウェア ツールを使用してデザイン パフォーマンスを達成する方法の詳細については、本誌に掲載の関連記事をご覧ください。



真のリーダ
真の価値
真の性能

LX ロジックに最適　SX DSPに最適　FX プロセッシング/コネクティビティに最適



VIRTEX V4

Virtex-4 FPGAは、どの競合90nm FPGAと比較しても3倍のパフォーマンスと集積度、1/10の消費電力、最高の「シグナル+パワー」インテグリティをお届けします。

Virtex-4™プラットフォームFPGAは、システムのあらゆる局面で優れた性能を発揮します。他のどんなFPGAも、I/Oバンド幅、オンチップRAM速度、DSPおよびエンベデット プロセッシング、さらにロジック・ファブリックの性能すべてにおいてVirtex-4の敵ではありません。的確な機能すべてを装備していることが、最少コストで飛躍的な性能を達成する唯一の方法です。

世界最高最速クラスと低消費電力を両立させたFPGA!

Virtex-4 FPGAは、他の90nm FPGAに比べ1/10以下の電力消費によりインラッシュ消費電力（電源投入時にかかる消費電力）で最高94%、スタティック消費電力で最高78%、ダイナミック消費電力で86%もの低消費電力化を達成することにより設計者の方々に業界最高レベルの性能を維持しながらこの低消費電力のメリットを充分に活かすことが可能となりました。

www.xilinx.co.jp/virtex4/

すぐに使いこなせるソフトウェア

最少のシステムコストで、最高のシステム・パフォーマンスを。

XILINX®

The Programmable Logic Company℠

Physical Synthesis and Optimization with ISE Software

ISE ソフトウェアによる物理合成と最適化

インプリメンテーション ツールを有効活用するためのヒント

Kevin Bixler
Manager, ISE Technical Marketing
Xilinx, Inc.
kevin.bixler@xilinx.com

David Dye
Senior Technical Marketing Engineer
Xilinx, Inc.
david.dye@xilinx.com

プロセス技術の進歩により、FPGA デバイスは劇的に高密度化しました。ザイリンクスの Virtex™ ファミリの中には、100 万システム ゲートを超えるデバイスさえあります。こうしたデバイスの高密度化と 300 mm ウエハの採用により、FPGA を経済的に量産化に使用できるようになりました。

かつて ASIC だけをターゲットにしていたデザインは、今やプログラマブル デバイスに実装されつつあります。最も大型の 90 nm Virtex-4 デバイスは、20 万以上のロジック セル、6 MB のブロック RAM、100 近くの DSP ブロックを備えています。これらデバイスのリソースを有効に利用し、かつパフォーマンスの要件を満たすデザインを作成するのは、そう容易ではありません。幸いなことに、今日の EDA ソフトウェア ツールは、こうした課題を克服できるまでに進化しました。

ロジックの最適化、ロジックの配置、インターコネクト（相互接続）遅延の最小化は、どれもパフォーマンスを最大限引き出すために重要なことです。タイミングドリブンな合成技術は、デザイン パフォーマンスを飛躍的に改善しました。ですが、タイミングドリブン合成の効能は、配線遅延をいかに正確に見積もるかによって変わってくるのです。

こうした問題に効果的に対処するため、合成時に物理的な配置・配線情報を使用する物理合成という方法が主流として使われてきました。また、ネットリストの生成後に実装の決定に合成を含めることで、物理合成と最適化はさらなる正確性を実現しています。これにより、実装時に実際の配置・配線情報に基

マッピング段階では、個々のタイミング パスの緊急度に基づいてネットリストを再最適化し、パッキングと配置が可能です。このアプローチは、タイミング クロージャに必要なインプリメンテーション サイクルの数を減らします。

づき、合成のマッピングとパッキングに関する決定を動的に再検討できるのです。

物理合成と最適化の利点

ロジック レベル間の相互接続遅延は、ロジック エレメントの配置の近さ、配線の密集度、そして最も速い配線リソースを必要とするネット間のローカルな競合により影響されます。

この問題の解決策は、マッピングと配置・配線時に合成の結果を見直すことです。マッピング段階では、個々のタイミング パスの緊急度に基づいてネットリストを再最適化し、パッキングと配置が可能です。このアプローチは、タイミング クロージャに必要なインプリメンテーション サイクルの数を減らします。

物理合成と最適化のフロー

ザイリンクスの ISE ソフトウェアは、物理合成と最適化を可能にするいくつかのソフトウェア オプションを備えています。これらのオプションは、デザインの具体的なニーズに応じて、個別に使うことも、また複数のオプションを組み合わせて使うこともできます。

タイミングの要件を定義する

効果的な物理合成に最も重要なステップは、正確かつ包括的なタイミング制約を確立することです。この点に留意してタイミング制約を作成すれば、インプリメンテーション ツールはより詳しい情報に基づいて決定を下すことができ、全体的な結果が改善されます。要件の固まっているクロックおよび I/O ピンを制約することで、デザインの他の部分に余裕を持たせてください。

タイミング制約を定義するための一番簡単な方法は、制約エディタ（ Constraints Editor ）を使用することです。このグラフィカル ツールを使えば、クロック周波数、マルチサイクル パスと False パスの制約、I/O のタイミング要件など、明確にすべき多くの要件を入力できます。制約はユーザー制約ファイル（ UCF ）に書き込まれ、任意のテキスト エディタで編集することもできます。

ユーザー定義のタイミング制約を作成しない場合、ISE™ 8.1i ソフトウェアの新機能が各内部クロックに対して自動的にタイミング制約を生成します。性能評価モード（ PEM ）を使用すると、タイミングの目標値を指定しなくても物理合成と最適化の高性能な結果を得ることができます。

グローバルな最適化を実行する

IP コアやその他のネットリストを含むデザインについては、インプリメンテーションの変換（ NGDBuild ）後に NGD ファイルが作成されます。これはデザイン全体が初めて完全にアセンブルされたことを意味します。Map のバージョン 7.1.01i に加えられた新機能、グローバル最適化は、この完全にアセンブルされたデザインを使い、組み合わせロジックとレジスタ ロジックを再最適化することでデザイン パフォーマンスを改善しようとします。グローバル最適化（ コマンドラインで「 map -global_opt 」と入力 ）は、デザインのクロック周波数を平均 7 % 高めることがわかっています。

この段階で完了した最適化をさらに制御するオプションが 2 つあります。1 つは、組み合わせロジックの遅延のバランスを取るためレジスタを前後に移動するリタイミング（ map -retiming ）、もう 1 つは、機能的に重複するレジスタを削除する等価レジスタの削除（ map -equivaent_register_removal ）です。

タイミング ドリブンなパッキングと配置を有効にする

インプリメンテーション フローで利用できる物理合成機能の中核を担うのが、タイミングドリブンなパッキングと配置です。このオプションを有効にすると（ map -timing ）、Map 内で配置・配線のうち配置段階が実行され、最初の結果が最適とはいえない場合にパッキングの決定を再検討できます。この反復フローでは、無関係なロジック パッキングは省かれます。

ザイリンクスの物理合成と最適化には、さまざまなレベルの最適化があります。最初のレベルは ISE 6.1i ソフトウェアに採用され、ファンアウトの制御、ロジックの複製、混雑の制御、遅延見積もりの改善といったロジック変換からスタートしました。これらルーチンにより、デザインのパッキングと配置をより効率的に行えるようになり、クロック周波数の高速化とロジック ユーティリゼーションの高密度化を実現したのです。

次のレベルでは、ロジックとレジスタの最適化を追加し、Map がクリティカル パスの遅延を改善するためエレメントを再配置できるようにしました。これらの変換は、デザインのタイミング要件を満たすうえで大きな柔軟性を提供します。物理エレメントを、形は異なるものの論理的には同じ構造に変えてデザイン要件を満たすには、ピン スワッピング、基本エレメント スイッチング、ロジックの再組み合わせといったテクニックを使います。

ISE 8.1i ソフトウェアには、さらにもう 1 レベルの物理合成、すなわち組み合わせロジックの最適化もあります。「 -logic_opt 」スイッチは、デザイン内のすべての組み合わせロジックを検査するフローを有効にします。あとは配置とタイミングに関する情報を基に、全体的なデザインを改善するため LUT 構造をどう最適化するかについて、より的確

な決定を下すことができるのです。

物理合成と最適化の例

・ロジックの複製：
LUT やフリップフロップが複数の負荷を駆動し、それら負荷の1つ以上がソースから遠すぎてタイミングを満たせない場合、LUT やフリップフロップを複製し、その負荷グループに近接して配置します。これにより配線遅延を減らすことができます（図 1）。

・ロジックの再組み合わせ：
クリティカル パスが複数のスライスを通して複数の LUT を横断する場合、LUT とマルチプレクサのタイミング効率のより高い組み合わせを使用してそのパスに必要な配線リソースを減らします。そうすることで、より少ないスライスを利用するようロジックを再アセンブルすることができます（図 2）。

・基本エレメント スイッチング：
ファンクションが 1 枚のスライス内に LUT とマルチプレクサを使って構築されている場合、物理合成と最適化によりそのファンクションを再配置すると、通常はマルチプレクサの選択ピンを通して最もクリティカルな信号までの最速パスを実現できます（図 3）。

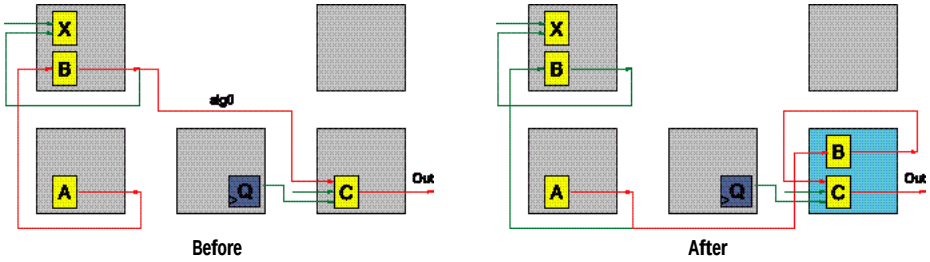
・ピン スワッピング：
LUT の入力ピンはそれぞれ遅延が異なることがあるため、Map は最もクリティカルな信号を最速のピンに配置するよう、ピンとその関連 LUT 式をスワップする機能を備えています（図 4）。

結論

ザイリンクスのデザイン ツールで提供する物理合成と最適化の機能は、ソフトウェアのリリースが新しくなるたびにますます成熟し、拡張されています。デザイン品質の改善に加え、各種の最適化機能をいっそう自由に制御できるようになります。この他にも、I/O ブロックの内外に移動できるレジスタや、ブロック RAM と DSP ブロックのような専用のファンクションなど、再最適化段階にデザイン エレメントを追加したり、反復可能な物理合成と最適化システムに配線段階を含めたりすることも検討中です。

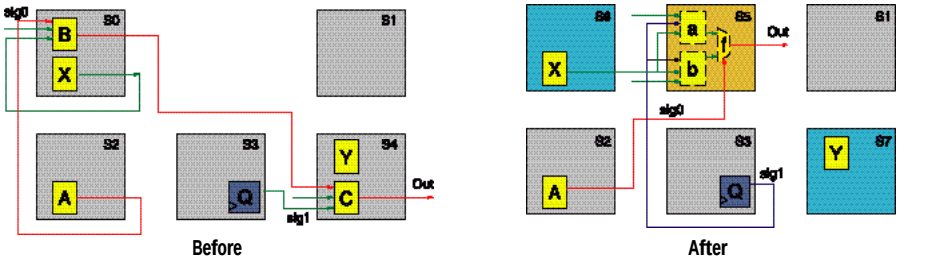
ザイリンクスの ISE ソフトウェアに用意されている物理合成と最適化ツールは、インプリメンテーションのパッキングおよび配置段階で FPGA デザインの構造を見直すことを目的として作成されました。タイミング制約と物理レイアウトの知識を利用して、マップおよび配置・配線時に合成の決定を最適化することにより、デザイン結果を飛躍的に改善できるのです。

図 1　ロジックの複製



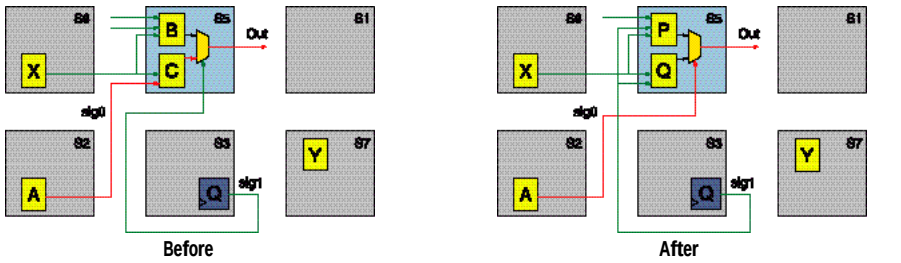
LUT A → LUT B → LUT C → Out のパスがクリティカルです。LUT B は 2 個のクリティカルな負荷を駆動していますので、パス遅延を減らすために複製することができます。

図 2　ロジックの再組み合わせ



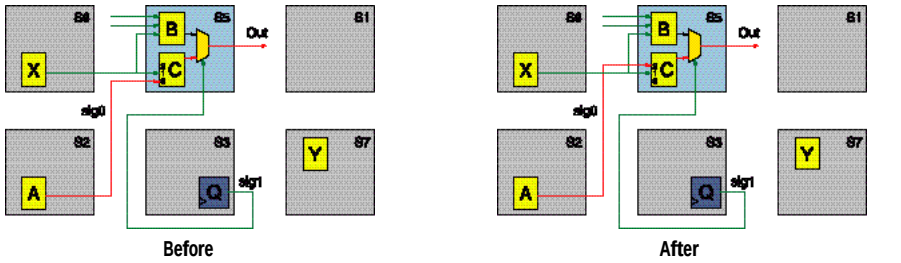
LUT A → LUT B → LUT C → Out のパスがクリティカルです。LUT B と C を組み合わせて、LUT a、LUT b、とF5Mux f で置き換えることができます。

図 3　基本エレメント スイッチング



LUT A → LUT C → MuxF5 → Out のパスがクリティカルです。マルチプレクサの選択ピンを通るパスのほうが、LUT を通るパスよりも高速です。

図 4　ピン スワッピング



LUT A → LUT C → MuxF5 → Out のパスがクリティカルです。LUT C の Pin 2 がPin 0 よりも高速です。LUT C のピン 0とピン 2をスワップします。

Improve Design Performance Using PlanAhead Design Tools

設計・分析ツール PlanAhead を使ったデザイン パフォーマンスの向上

PlanAhead はスピードに対する要求に応える

Mark Goosman
Marketing Product Manager
Xilinx, Inc.
mark.goosman@xilinx.com

デザインの問題、特に大型で高性能なデザインに特有の問題に直面したときは、最初に何が問題なのかを調査し、その大きな問題をより小さな扱いやすい単位に切り分けるのが最も効果的な対処法です。ここ数年におけるプログラマブル デバイスの進化を振り返ると、FPGA は飛躍的に大型化、複雑化したものの、PLD の EDA ツール フローはほとんど変わっていません。

従来のフラットなデザイン フローでは、デザインを変更するたびにデザイン全体を再合成・再実装する必要がありました。数百万ゲートのデバイスに複雑なデザインを開発してある場合、小さな変更でも配置・配線 （ PAR ）に時間がかかりすぎ、一貫性に欠ける結果をもたらすことがひんぱんに発生します。もちろん、RTL から PAR までの反復によって貴重な時間が失われることは言うまでもありません。

デザインの完成まで、イライラやストレスを抱えながら予想以上に長い時間を費やしたのにパフォーマンスが予想よりずっと低い結果になるのは、設計チームにとって耐えがたいことでしょう。また、これでは FPGA のユーティリゼーションが低くなりますし、Time-to-Market の機会さえ失いかねません。

解決策は PlanAhead ソフトウェア

最近では、ザイリンクスの設計・解析ツール、PlanAhead™ に提供されている階層型の設計手法で解決を図るユーザーが増えています。PlanAhead は、FPGA デザイン フローに可視性と制御を追加するソフトウェアです。ロジック合成とインプリメンテーション プロセス間の物理サイドで問題に対処することで、デザインのパフォーマンスを改善できます。

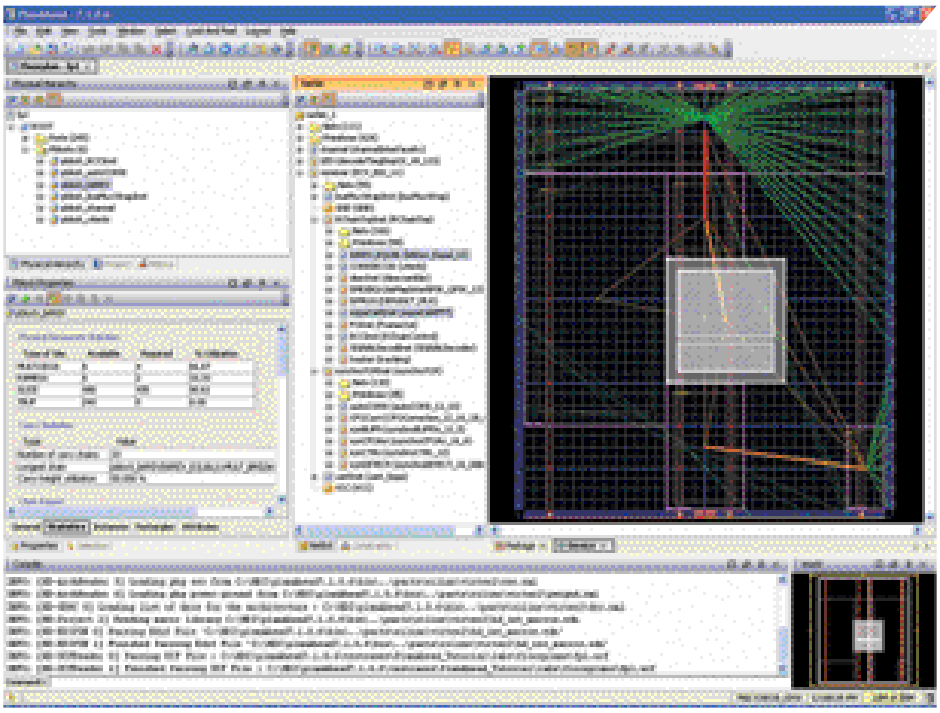
先進の FPGA 合成ソフトウェアを使えば、数百万ゲートのデザインに対してかなりのレベルまで自動的に最適化できますが、多くの設計者は最適なパフォーマンス目標を達成するためのよりヒューリスティックなテクニックを求めています。PlanAhead 設計ツールは、初期段階での解析とフロアプランニングを通して、最初のデザイン インプリメンテーションを制御するための物理的な制約を適用できます。インプリメンテーション後は、デザインを完成するために使われるフロアプランの改善に向け、配置とタイミングの結果を解析できます。インポートした結果から取り出された物理的制約は、その後のインプリメンテーションで配置をロックするために使用できます。これらの制約は、他のデザインで再利用が可能な配置をロックした IP を作成するために利用できます。

PlanAhead の設計手法は、パフォーマンスと生産性、結果の再利用性を高めます。階層型のデザイン フローにより、PAR を実行して RTL と合成に戻るという反復回数を減らすことができます。インプリメンテーション前にデザインを解析することにより、物理サイドの問題に対処できるのです。

より短時間でより高速なデザイン

PlanAhead のユーザーはたいていパフォーマンスを 10 ～ 15 % 改善しており、中にはそれ以上の改善を達成しているユーザーもいます。また、設計者は、タイトなデバイスにさらに 10 % のロジックを詰め込んでいます。より高速なパフォーマンスとより高いユーティリゼーションの組み合わせにより、いっそう小型で安価なデバイスを開発し、より遅いスピード グレードでデザイン目標を達成できることになります。

図 1　PlanAhead ソフトウェアは、デザインをさまざまなビューに表示し、物理的な階層やプロパティ、ネットリスト、各種制約、デバイスのパッケージピン、図などを明らかにします



PlanAhead 設計ツールを使えば、デザインにかかる総時間を短縮できるうえ、結果に一定レベルの一貫性を持たすことができます。過去のフロアプランやインクリメンタルなデザイン テクニックを活用することで、デザインの反復をはるかに短時間で実行でき、反復可能な結果が得られます。また、成功した結果は、ロックしたり他のデザインで再使用したりすることで、自由に流用できます。

極めて困難なパフォーマンスの問題に対処するには、新しいメニュー アイテムやスクリプト記述機能を追加するだけでは十分でありません。PlanAhead ソフトウェアは、デザイン データをさまざまなビューに表示することで、この階層型の設計手法を使いやすいインタラクティブな完全環境として提供しています（図 1 ）。これら独立したビューはお互いに連携して動作するため、クリティカルなデザイン オブジェクトと情報をすばやく判別してナビゲートできます。

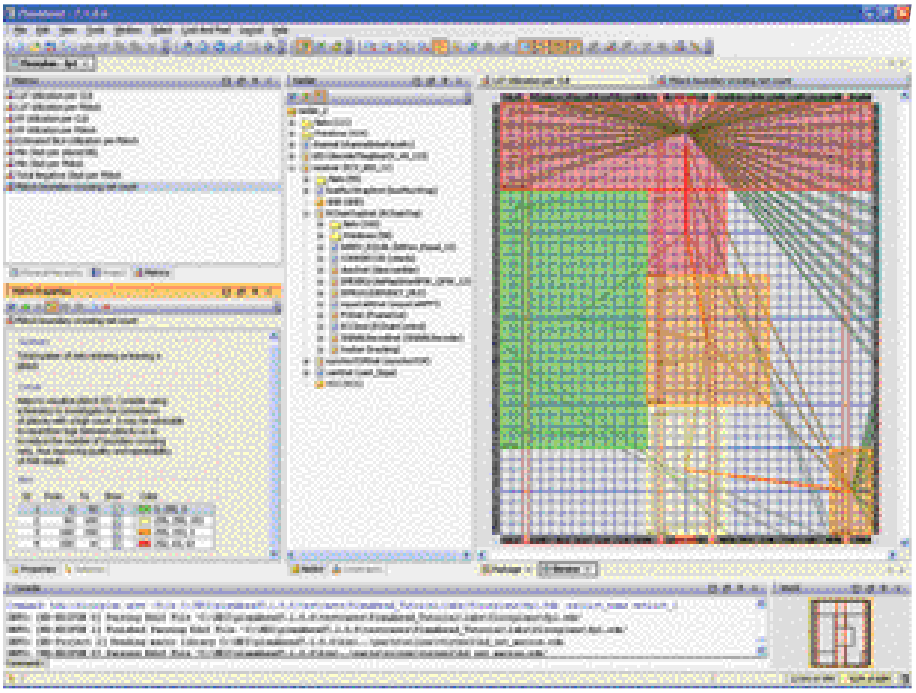
パフォーマンスのボトルネックを視覚的に識別

PlanAhead の環境では、I/O インターコネクトと物理ブロック（ PBlock ）のネット バンドルを表示することで、デザインのデータ フローを詳しく見ることができます。バンドルの色と線の太さは、信号の数に応じてコントロールできます。これにより、デザインを通して全体的なデータ フローの中で大量に接続されている PBlock を容易に識別できます。識別したら、配線の混雑するトラブル スポットを避け、大量に接続されている PBlock を近接して配置するかマージするよう修正すればよいのです。

また、クロック領域も表示されますので、フロアプランの作成中に、さまざまなクロックを最適化したりデバイス内の消費電力を最小限に抑えたりできます。クロックを個々のクロック領域に隔離すれば、隔離したクロックはより高速に動作し、他のクロック領域を立ち上げる必要がなくなります。

PlanAhead 設計ツールの解析・探索環境は、デザイン プロセスのさまざまな段階で使用できます。まず、インプリメンテーション前にデザインを解析できます。PlanAhead ソフトウェアは、タイミングの観点からデザインの実現性を調査するスタティック タイミング エンジン、TimeAhead を搭載しています。また、インターコネクトされてない純

図 2　メトリック マップは、デザイン中の問題がありそうなさまざまなエリアの熱メトリック表示を提供します。現行の メトリックは、Pblock とデザインレベルの両方に対してユーティリゼーションとタイミング チェックを行います



粋なロジック遅延を考慮して見積もった配線遅延を使って解析することもできます。これにより、デザインにどれだけのタイミングの余裕が組み込まれているかを確認できます。

その後、PlanAhead 環境内でタイミング制約を編集、微調整できます。こうして解析した結果は、どのロジックをグループ化してフロアプランを作成すべきかを判断する目安になります。フロアプランニングに際して、パスを論理的にソート、グループ化、選択できます。また、これと同じ TimeAhead 環境を、ザイリンクスの ISE™ ソフトウェア内のタイミング評価ツール、TRCE からインポートしたタイミング結果を使って使用することも可能です。

デザインに割り当てられたタイミング制約は、表示して変更することが可能です。ISE のタイミング制約は、どれでもエディタ内で新規制約として定義できます。これにより個々の制約フォーマットを覚えなくてすむため、制約を簡単に割り当てられます。TimeAhead でこれを使えば、ISE インプリメンテーション ツールを実行する前に制約セットを検証、最適化できます。

PlanAhead 設計ツールは、物理的なインプリメンテーション結果をすぐ理解できるよう、視覚的に表示します。エラーを早期に検出するためのデザイン ルール チェック（ DRC ）が用意されています。また、XtremeDSP™ スライスの専用レジスタや Virtex™-4 FPGA 内の RAM など、特定のデバイス リソースを正しく利用していないデザインにはフラグを付けます。

問題のエリアを視覚化することで、設計者は RTL と合成を繰り返すことなく、RTL または物理的なインプリメンテーション サイドのいずれかで問題にすばやく対処できます。ロジック モジュールをどこに配置したかや、ロジックが最も集中する部分に作成された PBlock をよく理解できるよう、さまざまなロジック モジュールを選択的にハイライト表示することができます。また、障害のあるタイミング パスをハイライト表示することで、デザイン内で物理的に何が起こっているのかを視覚的に理解できます。

PlanAhead ソフトウェアは、デザインの問題エリアをすばやく識別できるメトリック マップを搭載しています（ 図 2 ）。メトリック マップはタイミングやリソースの使用効率に関連付けることができます。この機能は、ロジック圧縮やタイミング コネクティビティに専念すべきデザインの特定エリアを識別するときに便利です。

PlanAhead 設計ツールでは、デザイン内のコネクティビティを探索できます。デザイン内の特定のネットや PBlock 、インスタンスを選択したら、マウスを 1 回クリックするだけで選択したエレメントに接続されているネットをすべてハイライト表示できます。

インスタンスや PBlock の選択後、そのエレメントに接続されているすべてのネットがハイライト表示されます。このプロセスを続けて、ロジック コーンを選択、展開していくことができます。「 コネクティビティを表示（ Show Connectivity ）」を実行すると、選択したインスタンスに接続されている次のレベルのネットがハイライト表示されます。これにより、特定のインスタンスや I/O ポートから始まるロジック コーンを容易に選択でき、デザイン階層を有効に活かせます。

パフォーマンスの問題に対処

タイミングの問題を解析し、そうした問題を回避もしくは修正するため、そのロジックを容易に制約できる包括的な環境が必要です。TimeAhead または TRCE からのタイミング結果を使えば、どのロジックをグループ化してフロアプランを作成すればいいかの判断の手助けとなり、よりパフォーマンスの高いデザインを実現するフロアプランを作成することができます。

クリティカル パスがロジック階層を横断することがよくあります。PlanAhead ソフトウェアは、物理階層をロジック階層から独立させることができるので、デザイン内のどこにあるロジックでもグループ化して効率的にフロアプランを作成できます。

PlanAhead はまた、リソースの使用効率の見積もりを提供し、PBlock のサイズと形状を決めるうえで役立ちます。この使用効率の統計は、クロック情報、キャリー チェーン、最適な RPM サイズなど、さまざまな有益な情報を報告します。

PlanAhead 設計ツールは、ロジック階層に基づく自動的なパーティショニングや PBlock の自動的なサイズ設定/配置など、自動フロアプランニング機能を搭載しています。1 個の PBlock の矩形内で必要なデバイス リソースを網羅するのは困難なことがありますが、

必要に応じて、PBlock をネスティングすることで子/親階層を作成し、サブ モジュールをさらに制約してパフォーマンス ゲインをいっそう高めることも可能です。

その場合は複数の矩形を使って非線形（non-recta-linear ）の形状を作成できます。さらに、デザイン階層の維持を容易にするため、PBlock 内に別の PBlock を作成することも可能です（「子」ブロック）。

デバイスの容量はロジックを PBlock で圧縮することで改善できます。方法は 2 通りあります。1 つは COMPRESSION というザイリンクスの AREA_GROUP 属性を使う方法です。AREA_GROUP は、マッピングやパッキング、配置・配線のために、デザインを複数の物理領域にパーティショニングできるデザイン インプリメンテーション制約です。COMPRESSION 属性を使うと、ISE Mapper は無関連のロジックを未使用の CLB サイトにパッキングします。ただし、これはタイミングに悪影響を与えることがあるため、使用には注意が必要です。

パフォーマンスを改善するための最良の戦略は、タイミング クリティカルでないロジックを圧縮し、デバイス内にタイミング クリティカルなロジック用のスペースを空けることです。もう 1 つのオプションは、PlanAhead の機能を使い、PBlock に個別に PAR を実行することです。PBlock のサイズは、PAR が失敗するまでどこまでも縮小できます。これにより、ロジックをブロック内にできる限りタイトにパッキングし、デバイスのスペースを開放できます。

Virtex-4 のフロアプラン作成例

PlanAhead 設計ツールでは、配置とタイミングの結果を容易にインポートできます。この情報を使えば、タイミング レポートからクリティカル パスを表示してソートし、回路図ビューまたはデバイス ビューを使用してパスを視覚化できます。障害のあるパスを識別したら、回路図ビューにあるすべてのパス インスタンスを識別するため、フロアプラン上のパス インスタンスをすべてハイライト表示できます。

図 3 は、Vrtex-4 FX140 デバイスをターゲットにしたデザインのフロアプランです。画面のハイライト表示されている部分は、タイミングを満たせなかったパスにあるフリップフロップです。それらのフリップフロップはデバイス全体に広く分散されているため、結果的にそのデザイン インプリメンテーションは許容できないほど長い遅延になります。Virtex-4 FPGA にはクロック ドメインが数多くありますので、こうした状況はよく起こりえます。

これらの各フリップフロップを選択して 1 個の PBlock に制限することで、PBlock のサイズとロケーションを調整して最適化し、クリティカル パスにおける遅延を短縮できます（図 4）。必要に応じて、PBlock をネスティングすることで子/親階層を作成し、サブ モジュールをさらに制約してパフォーマンス ゲインをいっそう高めることも可能です。キャプチャしたロジックのリソース要件に応じて、必要なリソースに最も効率よくアクセスできるロケーションに、クリティカル ロジックをロック ダウンできます。

結論

PlanAhead ソフトウェアの 30 日間無償で使える評価版は、http://www.xilinx.co.jp/planahead/ からダウンロードが可能です。この評価版では、PlanAhead のすべての特徴と機能をフルに試すことができます。また、同サイトには製品デモやダウンロード可能なホワイトペーパー、詳細な製品情報も用意されています。

図 3　タイミングの合っていないパスをハイライト表示する Virtex-4 FPGA のフロアプラン

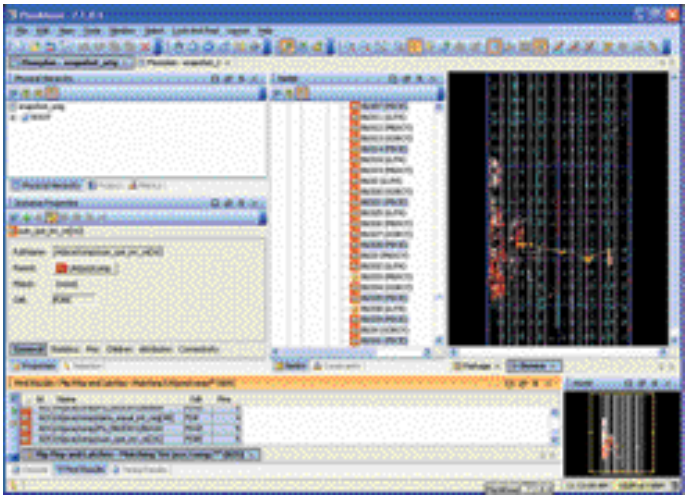
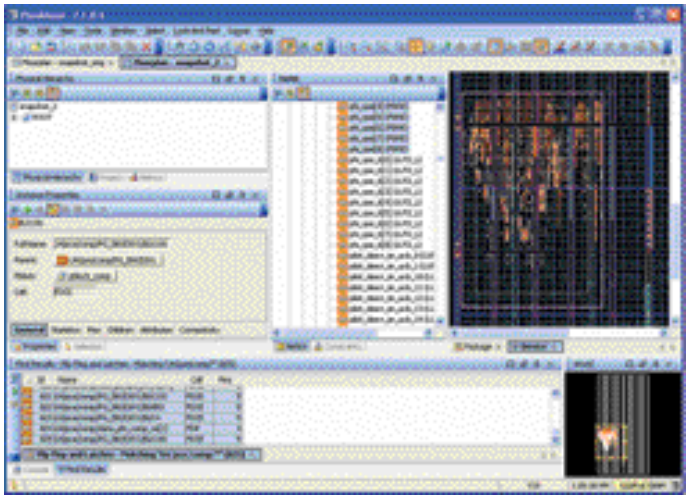


図 4　パスに関連するすべての問題を制限するすることによって、そのパスが必要なタイミングがとれるように Pblock を最適化できます。



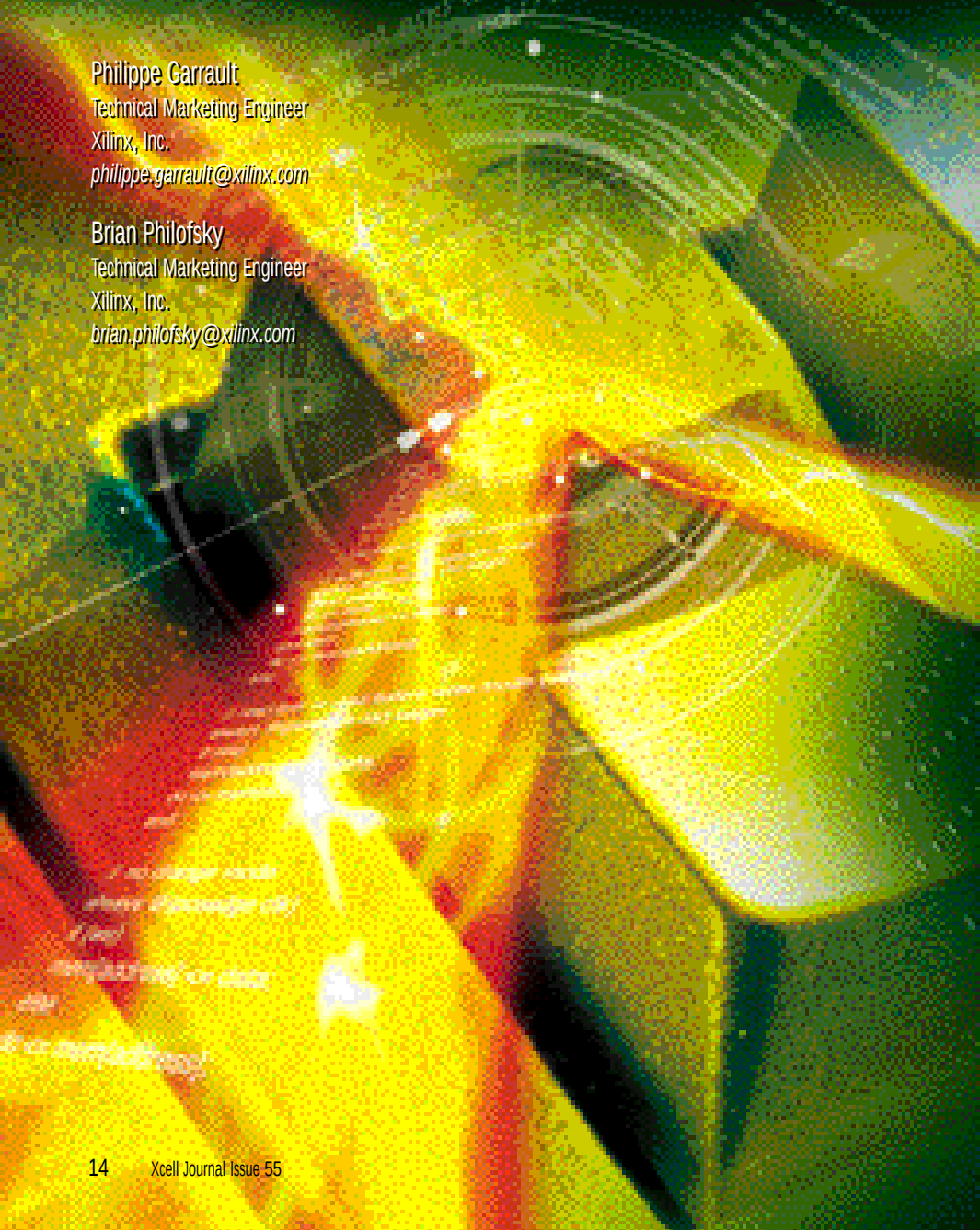
HDL Coding Practices to Accelerate Design Performance

デザイン パフォーマンス向上のための HDL コーディング法

コーディングの小さな工夫が、パフォーマンス面で大きな違いを生む

Philippe Garrault
Technical Marketing Engineer
Xilinx, Inc.
philippe.garrault@xilinx.com

Brian Philofsky
Technical Marketing Engineer
Xilinx, Inc.
brian.philofsky@xilinx.com



デザイン パフォーマンスを高める有効な方法としては、適切なハードウェア プラットフォームとシリコン機能を選択する、デバイス アーキテクチャに精通する、インプリメンテーション ツールで最適な設定と機能を使用するなど、さまざまな方法があります。しかし、ターゲット デバイスに対する非常に効率的な HDL のコードを書く方法は、デザイン パフォーマンスを高めるうえで最も見過ごされがちです。本稿では、デザイン パフォーマンス向上のためのコーディング スタイルについて、ヒントやテクニックを紹介していきます。

リセットの使用とパフォーマンス

システム規模のオプションのうち、パフォーマンスやエリア、消費電力に大きな影響を及ぼすオプションは、リセットです。システム設計者の中には、システムに対してグローバルな非同期リセットを使うよう指定する場合があります。本当に必要かどうかは別として、このオプションの効果を正しく理解していない設計者もいるようです。ザイリンクスの FPGA アーキテクチャでは、リセットの使用とそのタイプがコードのパフォーマンスに重大な影響を及ぼすことがあります。

SRL

現在、ザイリンクスのすべての FPGA アー

キテクチャでは、LUT（ ルックアップ テーブル）エレメントをロジック、ROM/RAM もしくはシフト レジスタ（ シフト レジスタ LUT、略称 SRL ）としてコンフィギュレーションすることが可能です。合成ツールは RTL コードからどの構造が使われているかを推論できます。ただし、SRL はリセット機能を持たないため、コードにリセットを記述するとシフト レジスタとして使われている LUT を認識できなくなります。つまり、リセットをコーディングしていないシフト レジスタは SRL を駆使した高速かつコンパクトなインプリメンテーションになるのに対し、リセットをコーディングした場合は複数のフリップフロップとそれらを結ぶ配線が必要となり、効率の悪いインプリメンテーションになってしまうのです。

この 2 つのケースを比較すると、エリアと消費電力に及ぼす影響は明らかですが、パフォーマンスへの影響はすぐには認識できません。通常、レジスタ間のタイミング パスはデザイン内の最長パスにならないため、フリップフロップで構築したシフト レジスタがデザイン内のクリティカル パスになることは一般的にありません。デザインの他の部分に対するリソースの追加消費（ フリップフロップと配線 ）が配置・配線の選択にマイナスの影響を与えることにより、配線パスが長くなる場合があります。

専用の乗算器と RAM ブロック

乗算器は一般に DSP デザイン向けと考えられています。しかし、ザイリンクスの FPGA は、アーキテクチャに乗算のための専用リソースを用意しているため、乗算器は、乗算だけでなく他のファンクションも発行できるように設計されています。同様に、アプリケーションにかかわらず、ほぼすべての FPGA デザインにおいてさまざまなサイズの RAM が使われています。

ザイリンクスの FPGA には、デザインで RAM、ROM、大規模 LUT、さらには汎用ロジックとして使えるいくつかのブロック RAM エレメントがあります。乗算器と RAM リソースを併用すれば、よりコンパクトで高性能なデザインを実現できますが、リセットをどのように使用するかによってパフォーマンスにプラス、またはマイナスの影響を及ぼします。RAM と乗算器のブロックは両方とも同期リセットしか持たず、これらのファンクションに非同期リセットをコーディングすると、ブロック内のレジスタが使用されなくなります。これがパフォーマンスに及ぼす影響は甚大です。たとえば、非同期リセットを設定して Virtex™-4 デバイスを対象にフルにパイプライン化した乗算器を使うと、パフォーマンスは 200 MHz にしかなりません。コードを同期リセットに変更すれば、デザイン パフォーマンスを 2 倍以上の 500 MHz まで改善できます。

RAM の問題は 2 つあります。Virtex-4 のブロック RAM は、乗算器と同様、オプションとして出力レジスタを持っており、これを使うと RAM の clock-to-out（ クロックから出力までの ）時間が短縮され、全体的なデザイン スピードが向上します。これらの出力レジスタは、非同期リセットではなく同期リセットのみを提供するため、コード内のレジスタが非同期リセットに記述されている場合は使用されません。

2 番目の問題は、RAM を LUT または汎用ロジックとして使用する場合です。エリアとパフォーマンスの理由から、ROM や汎用ロジックとしてコンフィギュレーションされているいくつかの LUT を、1 個のブロック RAM に凝縮することが有効な場合があります。方法としては、これらの構造をマニュアルで指定する、もしくは、ロジック デザインのその部分を未使用のブロック RAM リソースに自動的にマッピングするかのいずれかです。ブロック RAM は同期リセットを持っているため、同期リセットを使う場合や、リセットをまったく使わない場合、デザインの指定した機能に影響を及ぼすことなく汎用ロジックをマッピングできます。非同期リセットが記述されている場合は、これが不可能です。

汎用ロジック

おそらく、非同期リセットの影響で最も知られていないのは、汎用ロジックの構造に対する影響でしょう。ザイリンクス FPGA のすべての汎用レジスタには、セット/リセットを非同期または同期のいずれでもプログラミングできる機能が搭載されているため、非同期リセットを使っても特に問題ないと考えるようですが、それは得てして間違いです。非同期リセットを使わない場合、セット/リセット ロジックを同期ロジックとして構成することが可能です。その場合、ロジック最適化のための追加リソースが開放されることになります。非同期リセットがどのように最適化を妨げるかを理解するため、次の不適切なサンプル コードを見てみましょう。

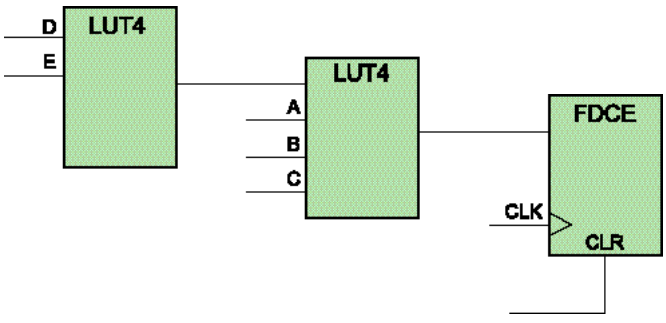
VHDL Example #1

```
process (CLK, RST)
begin
 if (RST = '1') then
  Q <= '0';
 elsif (CLK'event and CLK = '1') then
  Q <= A or (B and C and D and E);
 end if;
end process;
```

Verilog Example #1

```
always @(posedge CLK, posedge RST)
 if (RESET)
  Q <= 1'b0;
 else
  Q <= A | (B & C & D & E);
```

図 1　合成ツールは 2 つの LUT を推論する



このコードをインプリメントするには、合成ツールは、データ パスに対する 2 つの LUT を推論する以外に選択の余地がありません。このロジックを作成するため 5 つの信号が使われているからです。前述のコードは図 1 のようにインプリメントされます。

では、このコードを同期リセット用に書き直し、次のサンプルコードのように、エリアの縮小とパフォーマンスの改善を加えてみましょう。

VHDL Example #2

```
process (CLK)
begin
 if (CLK'event and CLK = '1') then
  if (RST = '1') then
   Q <= '0';
  else
   Q <= A or (B and C and D and E);
  end if;
 end if;
end process;
```

Verilog Example #2

```
always @(posedge CLK)
 if (RESET)
  Q <= 1'b0;
 else
  Q <= A | (B&C&D&E);
```

合成ツールは、柔軟にこのファンクションの構築方法を選択できます。このコードは図 2 のようにインプリメントされます。

このインプリメンテーションでは、合成ツールは、A がアクティブ High のとき Q は必ずロジック 1 であると識別できます（ OR ファンクション）。レジスタはセット/リセットを用いて同期オペレーションとしてコンフィギュレーションされているため、セットは同期データ パスの一部として自由に使用できます。これにより、ファンクションのインプリメントに必要なロジックの量を削減するとともに、前の例の D および E 信号に起因するデータ パス遅延も減少します。コードがさらに有益なインプリメンテーションを実行するように書かれていたら、ロジックはリセット側にもシフトしていたかもしれません。

これまでのサンプルコードに次のコードを追加してみましょう。

VHDL Example #3

```
process (CLK, RST)
begin
 if (RST = '1') then
  Q <= '0';
 elsif (CLK'event and CLK = '1') then
  Q <= (F or G or H) and (A or (B and C and D and
  E));
 end if;
end process;
```

Verilog Example #3

```
always @(posedge CLK, posedge RST)
 if (RESET)
  Q <= 1'b0;
 else
  Q <= (F|G|H) & (A | (B&C&D&E));
```

ロジック ファンクションには 8 つの信号が関わっているため、このファンクションをインプリメントするには最低でも 3 つの LUT が必要です。上記コードは図 3 のようにインプリメントされます。

同じコードを同期リセットで書くと、次のようになります。

VHDL Example #4

```
process (CLK)
begin
 if (CLK'event and CLK = '1') then
  if (RST = '1') then
   Q <= '0';
  else
   Q <= (F or G or H) and (A or (B and C and D and
   E));
  end if;
 end if;
end process;
```

Verilog Example #4

```
always @(posedge CLK)
 if (RESET)
  Q <= 1'b0;
 else
  Q <= (F|G|H) & (A | (B&C&D&E));
```

上記コードは図 4 のようにインプリメントされます。この場合も、同じロジック ファンクションをインプリメントするのに使う LUT が少なくて済むうえ、このファンクションを生成するほぼすべての信号についてロジック レベルが減少するため、より高速なデザインにできる可能性があります。

ここまで紹介してきた例はシンプルですが、非同期リセットがデータ入力上のすべての同期データ信号をレジスタに送り込んで、ロジック レベルの増加とインプリメンテーションの劣化を招きかねないことは十分理解できたことと思います。一般に、ロジック ファンクションに入り込む信号が増えるほど、同期セット/リセットを使う（ もしくはリセットをまったく使わない ）ことで、ロジック リソースを最小限に抑えたり、デザイン パフォーマンスを最大限に高めたりすることができます。

加算器ツリーでなく 加算器チェーンを使用する

多くの信号処理アルゴリズムは、サンプルの入力ストリームに算術演算を実行した後、

図 2　より柔軟な LUT の推論

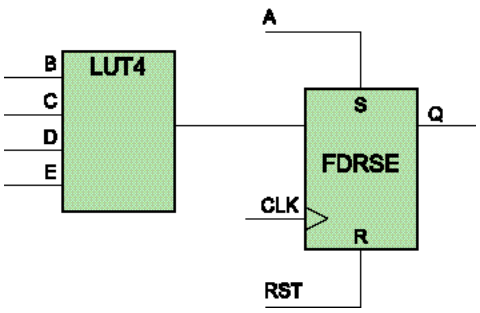


図 3　3つの LUT を推論する合成

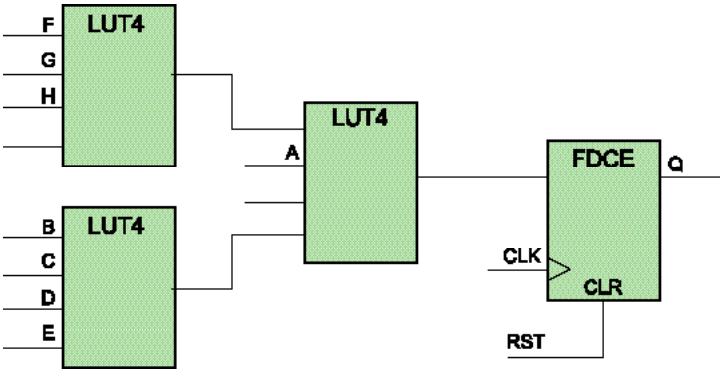
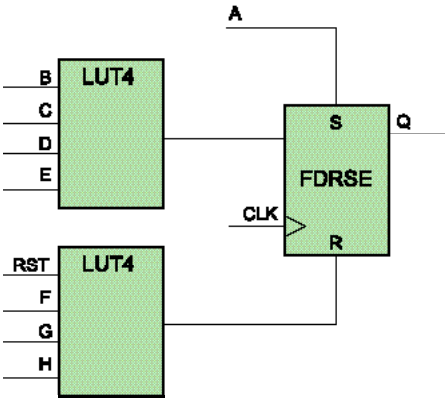


図 4　同期リセットを持つ推論



この算術演算のすべての出力の和を求めます。FPGA のようなパラレル アーキテクチャでは、和をインプリメントするために通常は加算器ツリーが使われます。

加算器ツリーの難しいところは、サイズが変わることです。加算器の数は、加算器ツリーにある入力の数で決まります。加算器ツリーの入力数が多いほど、必要な加算器の数が増え、ロジック リソースの数や消費電力も増加します。また、大きなツリーほどツリーの最終段階に進むにしたがい加算器の規模が大型化し、システム パフォーマンスのさらなる悪化を招きます。

消費電力を減らし高いパフォーマンスを維持するには、加算器ツリーを専用のシリコン リソースとしてインプリメントする必要があります。しかし、固定サイズの加算器ツリー コンポーネントをたくさん配置するのは効率的ではありません。というのも、加算の固定数を超えるとロジック リソースを使う必要が生じ、あるいはより大規模な FPGA に移行すると、デバイスのコストが増えるからです。

Virtex-4 デバイス ファミリは、DSP48 専用シリコンのカラムを使うことで、和をインプリメントするのに異なるアプローチを取ります。加算器ツリーではなく、チェーン型の加算器を使って和をインクリメンタルに算出します。この方法は従来の FPGA とは一線を画し、ロジックとインターコネクトが両方とも完全に専用シリコン内にあることから、DSP アルゴリズムのパフォーマンスの最大化と消費電力の低減につながります。

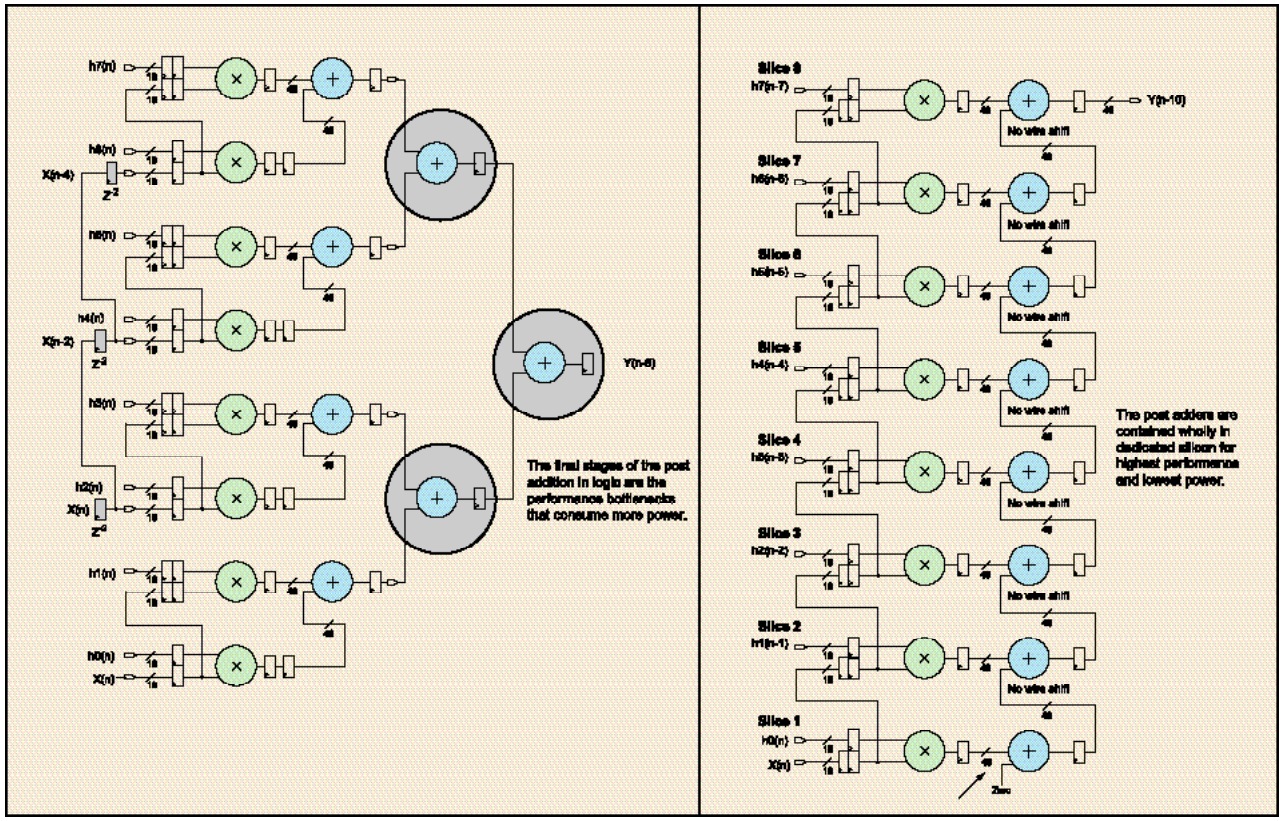
パイプライン化した場合、DSP48 ブロックのパフォーマンスは、加算器の数にかかわらず 500 MHz です。図 5 に示すとおり、48 ビット加算器/アキュムレータと組み合わされたポートをカスケード接続することで、現在のサンプルの算出に加え、これまで算出したサンプルの和も計算することができます。

RTL で Virtex-4 加算器チェーンの構造を利用するには、加算器ツリーの記述を加算器チェーンの記述に置き換えるのみです。ダイレクト フォーム フィルタから記述を置き換えたシストリック フォームに変換するプロセスについては、「 XtremeDSP Design Considerations User Guide 」を参照してください。

変換が完了すると、アルゴリズムが、アプリケーションが必要とする以上に高速に実行されることに気付くでしょう。その場合、フォールディングまたはマルチチャネリングのいずれかのテクニックを使うことで、デバイスの使用効率と消費電力をさらに低減させることができます。いずれのテクニックも、デザインをより小型のデバイスにインプリメントしたり、開放されたリソースを使ってデザインに機能を追加したりするのに役立ちます。

マルチチャネリングとは、非常に高速な演算エレメントを使って、はるかに低いサンプ

図 5　加算器をカスケード接続することにより、パフォーマンスを予測できる



ル レートの複数の入力ストリーム（チャネル）を処理することです。このテクニックは、シリコン効率をほぼチャネルの数と同倍に高めます。マルチチャネル フィルタリングは、時分割多重化単一チャネル フィルタとして見ることができます。たとえば、一般的なマルチチャネル フィルタリングでは、各チャネルに別々のデジタル フィルタを使って複数の入力チャネルがフィルタリングされます。Virtex-4 DSP48 ブロックを利用すれば、8倍クロックで単一のデジタル フィルタをクロッキングし、単一のフィルタで 8 つの入力チャネルをすべてフィルタリングすることができます。これにより、必要な FPGA リソースの数は約 1/8 に減少します。

ブロック RAM のパフォーマンスを最大化

メモリ エレメントを実装する際、パフォーマンスに影響を与える要因は次のとおりです。

- 専用ブロック RAM または分散 RAM を使う
- 出力パイプライン レジスタを使う
- 非同期リセットを使わない

また、あまり知られてないことですが、HDL のコーディング スタイルと合成ツールの設定もメモリ パフォーマンスに大きな影響を与えます。

HDL のコーディング スタイル

デュアルポートのブロック メモリを実装するとき、両方のポートが同時に同じメモリ セルにアクセスしようとすることが考えられます。両方のポートが同じメモリ セルに同時に異なる値を書き込んでいる場合、それぞれが衝突し、メモリ セルの内容が保証されません。しかし、一方のポートが読み出している間、もう一方のポートが同じアドレスに書き込んでいるとしたらどうでしょう。これはターゲットとするデバイスによって異なります。最新の Virtex および Spartan™ ファミリには、書き込み操作が行われている間、メモリ出力を制御するための 3 つのプログラマブルな動作モードがあります。各動作モードの詳細は、デバイスのユーザー ガイドをご覧ください。

メモリ出力のビヘイビアとメモリのパフォーマンスは、それぞれのモードで異なることに注意してください。次の例に示すとおり、メモリがどのモードで動作するかはコーディング スタイルで決まります。

```
// Inference of Virtex-4 memory blocks
//
// 'write first' or transparent mode
always @(posedge clk) begin
 if(we) begin
  do <= data;
  mem[address] <= data;
 end else
  do <= mem[address];
end

// 'read first' or read before write mode (slower)
always @(posedge clk) begin
 if (we)
  mem[address] <= data;
 do <= mem[address];
end

// 'no change' mode
always @(posedge clk)
 if (we)
  mem[address] <= data;
 else
  do <= mem[address];
end
```

パイプライン レベルの追加

パフォーマンスを改善するためのもう 1 つの方法は、いくつかのロジック レベルからなる長いデータ パスを、複数のクロック サイクルに分割して再構築することです。この方法は、待ち時間とパイプライン管理のオーバヘッド ロジックを犠牲にして、クロック サイクルの高速化とデータ スループットの向上を図ろうというものです。FPGA はレジスタを多数搭載しているため、通常はレジスタとオーバヘッド ロジックをさらに追加することが問題になることはありません。

データはマルチサイクル パス上にあるため、デザインの残りの部分に対してはこの追加の待ち時間を考慮する必要があります。次の例は、32 x 32 乗算器の出力に 5 レベルのレジスタを追加するためのコーディング スタイルです。合成ツールは、データ スループットを最大化するため、Virtex-4 DSP48 ブロックにあるレジスタにこれらのレジスタをパイプライン化します。

```
// 32x32 multiplier with 4 DSP48 (PIPE=5)
always @(posedge clk) begin
  prod[0] <= a * b;
  for (i=1; i<=PIPE-1; i=i+1)
          prod[i] <= prod[i-1];
end
```

コード内のネスト

ネストされた if 文や case 文など、コードにあまり多くのネストを入れすぎないよう注意してください。if 文の中に他の if 文をたくさん入れすぎると、行が長くなりすぎることがあり、合成の最適化に支障をきたします。この点に注意すれば、コードをより読みやすく、移植しやすくできます。

HDL で「 for ループ 」を記述する際、特に算術演算などのロジックに依存度の高い演算がある場合には、データ パスに少なくとも 1 個のレジスタを配置するとよいでしょう。コンパイル中、合成ツールはループをアンロール（ unroll : とき解く ）します。これらの同期エレメントがないと、合成ツールはループを反復するたびに生成されたロジックを連結し、非常に長い組み合わせパスとなり、デザイン パフォーマンスを制限する可能性があるからです。

結論

近年における合成および配置・配線アルゴリズムの進歩により、特定のデバイスからベスト パフォーマンスを引き出すのがはるかに容易になりました。合成ツールは、複雑な算術演算とメモリ記述を推論して専用のハードウェア ブロックにマッピングできます。また、合成ツールはリタイミングなどの最適化と、ロジックおよびレジスタの複製も行います。配置・配線ツールは、配置・配線の混雑を最小限に抑えるため、タイミング制約に基づいてネットリストを再構築し、タイミング ドリブンなパッキングと配置を実行できるようになりました。

とはいえ、以前と同様に今日でもパフォーマンスを最大化できるツールは限られています。デザインのパフォーマンスをさらに改善する必要がある場合は、ターゲット デバイスと合成ツールにいっそう精通し、本稿で説明したコーディング ガイドラインに従うのが効率的です。

Writing RTL Code for Virtex-4 DSP48 Blocks with XST 8.1i

XST 8.1i を用いた Virtex-4 DSP48 ブロック向け RTL コーディング術

シンプルで効率的な DSP アプリケーション向け RTL コードの書き方

Edgard Garcia
Xilinx Consultant / Designer
Multi Video Designs
edgard.garcia@mvd-fpga.com

ザイリンクスの Virtex™-4 ファミリは、複雑な DSP アルゴリズムを素早くインプリメントするための高性能な新コンセプトを導入しました。ザイリンクスの Web サイト http://www.xilinx.co.jp/bvdocs/userguides/ug073.pdf に掲載されている「 Xtreme DSP™ for Virtex-4 FPGAs User Guide 」（以下、「 XtremeDSP User Guide 」）に、DSP48 アーキテクチャの活用法を解説するとともに、いくつかの例を紹介しています。

もちろん、実際の DSP アプリケーションを開発するときは、各 DSP ブロックをインスタンシエートしてそれぞれの属性値を割り当てれば正しいビヘイビアを得られます。しかし、ごくシンプルな RTL コードを書くことで、効率的な DSP48 コンフィギュレーションのほとんどを実装することができることはご存じでしょうか。

VHDL や Verilog で DSP アルゴリズムを開発するのは、長期にわたってデザインを維持するうえで優れた方法ですが、合成結果がパフォーマンスの要件を満たしている必要があります。本稿では、Virtex-4 DSP48 ブロックをフルに活かすための RTL コードの書き方を解説します。

DSP48 のアーキテクチャ

Virtex-4 DSP48 のアーキテクチャについては、「 XtremeDSP User Guide 」に詳しく説明されていますので、ここでは DSP48 ブロックのいくつかの重要点だけまとめておきます。

- DSP48 ブロックは乗算器に送る 18 ビットの入力を 2 個持っています。符号なしのデータを扱う場合、乗算器の入力の最大幅は 17 ビットです。最上位ビット（ MSB ）に 1 個以上の「 0 」を連結して、必ず符号なしのデータ/係数を拡張してください。同様に、加算器/減算器を使う場合、入力と出力は符号付き演算には 48 ビット以下、符号なしには 47 ビット以下でなくてはなりません。

本稿で紹介する例には、符号付きデータを使います。IEEE.STD_LOGIC_SIGNED パッケージを使用する必要があります。

- Virtex-4 DSP48 ブロック向けに DSP のビヘイビアを記述する際、もう 1 つ重要な条件は、DSP48 のすべての内部レジスタが同期リセットを持つことです。非同期リセットを使うと、合成ツールが DSP48 の内部レジスタを使用できなくなります。OpCode やその他のコントロール入力にかかわらず、リセット機能が優先されます。

- 加算器/減算器のステージは、48 ビットの入力（ 加算器/減算器の出力からのフィードバック、もしくは C 、Pcin からの入力 ）と、36 ビットの入力（ 主に、DSP48 の乗算器からの出力 ）を演算させることができます。

基本的な例

1. Multiplier_accumulator：
   最初に、FIR フィルタやその他の DSP ファンクションに便利な、この一般的な関数を例に取ります。

```
library IEEE;              use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;                  — Signed arithmetic is used

entity MULT_ACC is
   Port ( CK : in std_logic;
          RST : in std_logic;                          — Synchronous reset
          Ain, Bin : in std_logic_vector(17 downto 0);  — A and B inputs of the multiplier
          S : out std_logic_vector(47 downto 0)        );  — Accumulator output
end MULT_ACC;

architecture Behavioral of MULT_ACC is

signal ACC : std_logic_vector(47 downto 0);          — Accumulator output

begin

process(CK)  begin
          if CK'event and CK = '1'  then
                    if RST = '1'  then          ACC <= (others => '0');
                    else                        ACC <= ACC + (AIN * BIN);
                    end if;
          end if;
end process;

S <= ACC;

end Behavioral;
```

この例は 1 個の DSP48 ブロックに合成されるため、他のロジック リソースは必要ありません。パフォーマンスは配置・配線に依存しますが、だいたい 180 ～ 200 MHz です。

2. フルにパイプライン化された Multiplier_accumulator：
   より高いパフォーマンスが必要だが、配置・配線ツールにはあまり頼りたくないという場合でも、Multiplier_accumulator のパフォーマンスの改善はできます。DSP48 ブロックは、内部の入力レジスタ（A と B の入力に対して 0、1、または 2 つのステージ）と、1 つの選択可能な乗算器出力レジスタを持っています。次の RTL コードは、A と B の入力で 1 レベルのレジスタを使うとともに、乗算器の出力レジスタも使用します。

```
library IEEE;              use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;                  — Signed arithmetic is used

entity MULT_ACC is
   Port ( CK : in std_logic;
          RST : in std_logic;                          — Synchronous reset
          Ain, Bin : in std_logic_vector(17 downto 0);  — A and B inputs of the multiplier
          S : out std_logic_vector(47 downto 0)        ); — Accumulator output
end MULT_ACC;

architecture Behavioral of MULT_ACC is


signal AinR, BinR : std_logic_vector(17 downto 0);     — Registered Ain and Bin
signal MULTR : std_logic_vector(35 downto 0);          — Registered multiplier output
signal ACC : std_logic_vector(47 downto 0);            — Accumulator output

begin

process(CK)  begin
          if CK'event and CK = '1'  then
                    if RST = '1'  then          AinR <= (others => '0');
                                                BinR <= (others => '0');
                                                MULTR <= (others => '0');
                                                ACC <= (others => '0');
                    else                        AinR <= Ain;
                                                BinR <= Bin;
                                                MULTR <= AinR * BinR;
                                                ACC <= ACC + MULTR;
                    end if;
          end if;
end process;

S <= ACC;

end Behavioral;
```

この例も、1 個の DSP ブロックだけを使って合成されます。内部レジスタを活用すれば、インプリメンテーション（配置・配線）ツールを使わなくても、最も低速な Virtex-4 スピード グレードに対するパフォーマンスを 400 MHz 以上まで大幅に改善できます。

3. フルにパイプライン化された Loadable_Multiplier_accumulator：
   ロード可能な乗算アキュムレータを使うことで、デザインをさらに改善できます。詳細は、ザイリンクスの DSP デザイン コースのクラスで使用する教材「DSP Implementation Techniques for Xilinx FPGAs」（http://www.xilinx.co.jp/support/training/abstracts/dspimplementation.htm）をご覧ください。では、先ほどのコードに変更を加えてロード機能を実現しましょう。

```
library IEEE;              use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;                  — Signed arithmetic is used

entity MULT_ACC_LD is
   Port ( CK : in std_logic;
          RST : in std_logic;                          — Synchronous reset
          Ain, Bin : in std_logic_vector(17 downto 0);  — A and B inputs of the multiplier
          LOAD : in std_logic;                         — Active high LOAD command
          S : out std_logic_vector(47 downto 0)        ); — Accumulator output
end MULT_ACC_LD;

architecture Behavioral of MULT_ACC_LD is

signal AinR, BinR : std_logic_vector(17 downto 0);     — Registered Ain and Bin
signal MULTR : std_logic_vector(35 downto 0);          — Registered multiplier output
signal ACC : std_logic_vector(47 downto 0);            — Accumulator output

— 48 bit "ZERO" constant used for MULTR sign extension to 48 bits
constant ZERO : std_logic_vector(47 downto 0) := (others => '0');

begin

process(CK)  begin
          if CK'event and CK = '1'  then
                    if RST = '1'  then    AinR <= (others => '0');
                                          BinR <= (others => '0');
                                          MULTR <= (others => '0');
                                          ACC <= (others => '0');
                    else                  AinR <= Ain;
                                          BinR <= Bin;
                                          MULTR <= AinR * BinR;
                              if LOAD = '1'  then
                                    ACC <= ZERO + MULTR; — OpCode = x05
                              else
                                     ACC <= ACC + MULTR;   — OpCode = x25
                              end if;
                    end if;
          end if;
end process;

S <= ACC;

end Behavioral;
```

4. Multiplier_accumulator_or_adder：
これも乗算アキュムレータの便利なバージョンであり、18 ビットを超えるデータ バスの乗算に役立ちます（「 XtremeDSP User Guide 」の図 1-18 を参照)。RTL コードは次のとおりです。

```
library IEEE;                use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;                — Signed arithmetic is used

entity MULT_ACC_ADD is
   Port ( CK : in std_logic;
          RST : in std_logic;
          SEL : in std_logic;
          A_in, B_in : in std_logic_vector(17 downto 0);
          C_in : in std_logic_vector(47 downto 0);
          S : out std_logic_vector(47 downto 0));
end MULT_ACC_ADD;

architecture Behavioral of MULT_ACC_ADD is

constant ZERO : std_logic_vector(47 downto 0) := (others => '0');

signal AR, BR : std_logic_vector(17 downto 0);
signal MULT : std_logic_vector(35 downto 0);
signal Pout : std_logic_vector(47 downto 0);

begin

process(CK)  begin
    if CK'event and CK = '1' then
               if RST = '1'  then     AR <= (others => '0');
                                      BR <= (others => '0');
                                      MULT <= (others => '0');
                                    Pout <= (others => '0');
               else
                      AR <= A_in;
                      BR <= B_in;
                      MULT <= AR * BR;

                      if SEL = '0' then Pout <= C_in + MULT;  — Opcode = 0x35 for C input
                                                              —  0x15 for PCIN input
—                     if SEL = '0' then Pout <= ZERO + MULT; — Opcode =     0x05 for ZERO
                                                              — constant as input (Note 1)
                     else    Pout <= Pout + MULT;  — Opcode = 0x25 (Notes 2, 3)
                     end if;

                end if;
   end if;
end process;

S <= Pout;

end Behavioral;
```

今のところ、XST 8.1 で期待されるほど合成結果は最適化されていません。DSP48 ブロック内に同じ関数が用意されていましたが、C_in と Pout 間にマルチプレクサをインプリメントするためいくつかの組み合わせロジックが使われます。パフォーマンスはまだ、－10 スピード グレードで 220 MHz、－12 で 270 MHz 強にすぎません。とはいえ、Synplify Pro 8.2 を使用すると、同じ RTL コードで理想的なインプリメンテーションを実現します。

注 1： Pout に ZERO を追加するのは、先ほどのロード機能に相当します。

注 2： この行を次のように変更することで、Pout に 17ビットの右シフトを使うことも可能です（ 現時点では、この機能は Synplicity社の Synplify Pro 8.2 でしかサポートされていません)。

```
else Pout <= ZERO + Pout(47 downto 17) + MULT;
```

注 3：もし何かの理由で乗算器の出力レジスタを使いたくない場合は、組み合わせ乗算器の出力を宣言する代わりに、次のように書くことができます。

```
Pout <= Pout + (AR * BR);
```

その結果、RTL コードはよりコンパクトになります。

5. 対称的な四捨五入：
もう 1 つのシンプルで便利な例は、対称的な四捨五入を用いた乗算器です（「 XtremeDSP User Guide 」の表 1 - 9 を参照)。Ain × Bin の乗算結果を 20 ビットに四捨五入する場合、1 個の DSP48 ブロックと 1 枚のスライスに次の RTL コードが合成されます。

```
library IEEE;                use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity ROUNDING is
   Port ( CK : in std_logic;
          RST : in std_logic;
          Ain, Bin : in std_logic_vector(17 downto 0);
          P : out std_logic_vector(19 downto 0));
end ROUNDING;

architecture Behavioral of ROUNDING is

constant ZERO : std_logic_vector(47 downto 0) := (others => '0');

signal AR, BR : std_logic_vector(17 downto 0);
signal MULTR : std_logic_vector(35 downto 0);
signal Pout : std_logic_vector(47 downto 0);

signal Carry_in, Carry_inR : std_logic;

begin

process(CK)  begin
               if CK'event and CK = '1'  then
                              Carry_in <= not(Ain(17) xor Bin(17));
                              Carry_inR <= Carry_in;
                              if RST = '1'  then          AR <= (others => '0');
                                                          BR <= (others => '0');
                                                          MULTR <= (others => '0');
                                                          Pout <= (others => '0');
                              else
                                                  AR <= Ain;
                                                  BR <= Bin;
                                                  MULTR <= AR * BR;
— Note that the following 4 operands adder will be implemented as a 3 operand one :
— ZERO is a constant that allows easy sign extension for the VHDL syntax
                                                  Pout <= ZERO + MULTR + x"7FFF" + Carry_inR;
                              end if;
               end if;
end process;

P <= Pout(35 downto 16);

end Behavioral;
```

また、この例は Virtex-4 の－10 スピード グレードのデバイスでは 400 MHz、－12 スピード グレードのデバイスでは 500 MHz で動作します。2 番目のフリップフロップはキャリー入力用として DSP48 ブロック内にプッシュされるため、使用されるのは 1 個の LUT と、同

じスライスの中にあるスライス フリップフロップだけです。

ここで紹介した例は、どれも幅広いアプリケーションで利用できます。非常に効率よく合成され、すべてのロジックが DSP48 ブロックにマッピングされることがわかるはずです。各 DSP ファンクションのパフォーマンスは、配置・配線ツールから独立しています。

合成ツールが DSP48 の構造を容易に認識できるようにするには、コードを単純な方法で書き、ツールには各 DSP48 ブロックに必要なファンクションをパッキングするためのベスト オプションを与えることが重要です。各コードを 1 個のプロセスで書いたのは、そうした理由からです。

RTL コードがシンプルでコンパクトなほど、より効率的な合成結果になります。もちろん、合成ツールによっては他の方法を用いて卓越した結果を出すこともできるでしょうが、その場合は合成ツールへの依存度が高まります。

より複雑なデザイン

もっと複雑な DSP ファンクションが必要なときはどうでしょうか。その場合は、最適な合成結果を得るため、各ブロックを別々に記述することで、たくさんの複雑な DSP アルゴリズムのインプリメンテーションに同様のアプローチを使用できます。

「 XtremeDSP User Guide 」には、これ以外の例もたくさん掲載されています。そのほとんどは、アルゴリズムと回路図で説明されているものに直接関係しています。

結論

本稿は、アプリケーション ノート「 Virtex-4 DSP48 Inference 」(http://www.mvd-fpga.com/en/publi_V4_DSP48.html) からの抜粋です。

アプリケーション ノートには、次のような例も紹介されています。

- シングル DSP スライス 35 × 18 乗算器 (「 XtremeDSP User Guide 」の図 1-18)。
- シングル DSP スライス 35 × 35 乗算器 (「 XtremeDSP User Guide 」の図 1-19)。
- フルにパイプライン化された複素数用 18 × 18 乗算器 (「 XtremeDSP User Guide 」の図 1-22)。
- 高速 FIR フィルタ (「 XtremeDSP User Guide 」の図 1-17)。

アプリケーション ノートでは、XST 8.1i と Synplify Pro 8.2 がサポートする DSP48 ブロックのたくさんの重要な機能についてもご紹介しています。Map レポート、Timing Analyzer レポート、FPGA Editor の詳細なビューには、合成ツールとインプリメンテーション ツールの効率が示されます。パフォーマンスと消費電力を改善するため、隣接する DSP48 スライス間のカスケード チェーンがどう使われているかもわかります。これらの広く使われているコンフィギュレーションのほとんどは、使用するリソースとパフォーマンスの点から、最高のインプリメンテーション結果をもたらします。とはいえ、まだいくつかの問題点が残っており、そうした問題点は将来のリリースで解決していく予定です。

詳細は 、「 XtremeDSP for Virtex-4 FPGAs User Guide 」(http://www.xilinx.com/bvdocs/userguides/ug073.pdf)をご覧ください。Timing Analyzer や FPGA Editor などの ISE™ ソフトウェア ツールを用いて、デザイン メソッドをわかりやすく説明するとともに、インプリメンテーション結果を詳しく解析しています。

マルチ ビデオ デザイン (MVD) 社は、FPGA デザイン、PowerPC™ プロセッサ、エンベデッド/リアルタイム アプリケーション向けの RTOS、および PCI Express や RapidIO といった高速バスを専門とするトレーニング & デザイン センターです。MVD 社は正規のトレーニング パートナーでザイリンクスの XPERTS プログラムのメンバーでもあり、フランス、スペイン、南米にオフィスを持っています。

DSP48 マクロを実装するための XST サポート

ザイリンクスの ISE™ ツールセットに搭載されている合成エンジン、XST には、DSP48 のマクロを実装するための広範なサポートが用意されています。これにより、多数のマクロ関数が認識され、加算器、減算器、乗算器、アキュムレータ、さらに乗算加算や乗算アキュムレート (MAC) といった組み合わせなどが、専用のリソースにマッピングされます。レジスタ ステージは DSP48 ブロックに吸収でき、大きな関数や複数の関数をカスケード接続するにはダイレクト コネクト リソースが使われます。

DSP48 ブロックにおけるマクロのインプリメンテーションは、USE_DSP48 制約により制御され、デフォルト値は「 auto 」です。auto モードでは、XST は DSP48 のリソース上に加算器や減算器を除く、前述のマクロをすべてインプリメントしようとします。DSP48 に加算器や減算器を押し込むには、USE_DSP48 制約値を「 yes 」に設定します。

auto モードの場合、XST は加算器と減算器を除くすべてのマクロに対してリソースの自動制御を実行します。このモードでは、DSP_UTILIZATION_RATIO の制約を使い、合成に利用する DSP48 リソースの数をパーセントまたは絶対数で指定できます。デフォルトの場合、XST は指定されたデバイス内に用意されている DSP48 リソースをできる限り利用しようとします。

ISE ソフトウェアの 8.1i リリースでは、XST は DSP のサポートにさらなる拡張機能を投入しました。これにより、XST はフィルタ アプリケーションに欠かせないロード可能なアキュムレータと MAC を実装できるようになりました。XST は階層構造の境界にまたがる複雑なフィルタや乗算器のチェーンでさえ認識でき、これら DSP48 チェーンを構築するために専用の高速コネクションを使います。Register Balancing (レジスタ調整) 最適化機能は、クロック周波数を最適化する際、DSP48 ブロックに対してレジスタを考慮します。コーディング スタイルの詳細は、「 XST User Guide 」(http://toolbox.xilinx.com/docsan/xilinx7/books/docs/xst/xst.pdf) をご覧ください。また、Virtex™-4 デザインに対する個別のインプリメンテーション結果は、合成レポートを参照してください。

David Dye
Senior Technical Marketing Engineer
Xilinx, Inc.

Verifying Your Logic Design for First-Time Success

1 回で成功する論理設計の検証法

ザイリンクスとアライアンス各社が提供するユーザーの検証要求を支援する最新ツールと方法論

Hamid Agah
Senior Technical Marketing Manager
Design Software Division
Xilinx, Inc.
hamid.agah@xilinx.com

Howard Walker
Technical Marketing Engineer
Design Software Division
Xilinx, Inc.
howard.walker@xilinx.com

Scott Campbell
Technical Marketing Engineer
Design Software Division
Xilinx, Inc.
scott.campbell@xilinx.com

ザイリンクスが 1980 年代中旬に FPGA を発明した当時、デザインを検証するには実際のシステムで FPGA をプログラミングし、タイミングとファンクションの観点から正しく動作するか確認するというのが一般的でした。

こうしたやり方はとっくの昔に消えています。CMP Media 社の 2005 年 EDA 調査によれば、FPGA 設計者にとって最も頭の痛い問題のトップ 3 の 1 つは検証です。製品を予定どおりに市場に投入するには、検証をできるだけ早く成功させることが不可欠です。では、今日の高密度な FPGA を考えた場合、現行のフローがベストな選択肢かどうかを知るにはどうすればよいのでしょうか。有効な検証戦略には、少なくともスタティック/ダイナミック タイミング検証とダイナミック シミュレーションを含めるべきです。現在、ザイリンクスの FPGA ユーザーには、等価チェックとアサーション ベース検証など、高度なメソッドをオプションとして用意しています（図 1）。

本稿では、ザイリンクスとアライアンス プログラム メンバ各社から提供されている検証ソリューションの改善点と追加機能についてご紹介します。

ISE 8.1i ソフトウェアにおけるタイミング解析の改善

タイミング検証を完璧に行うには、ベスト ケースとワースト ケースの両方の動作条件で FPGA デザインをチェックする必要が

図 1　ザイリンクス ユーザーが利用可能な検証ソリューション

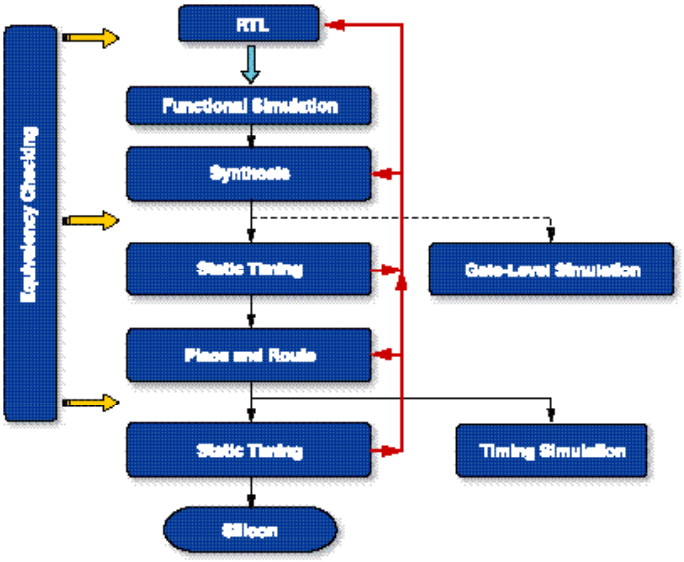
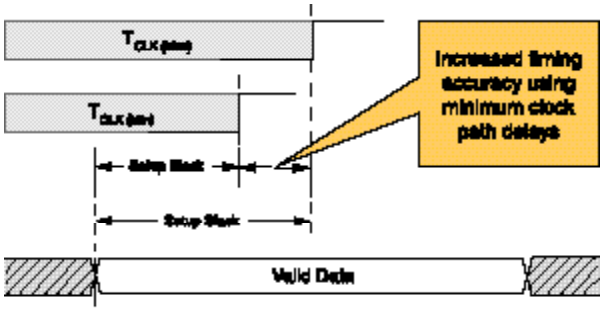


図 2　信頼性が向上するスタティックタイミング分析



あります。ワースト ケースの条件は、FPGA への供給電圧が最小で、FPGA の温度が最大のときに起こります。このような条件下ではデバイスの内部遅延の増加が発生するため、セットアップ タイムの違反を招く危険性が高まります。ベスト ケースの条件は、FPGA への供給電圧が最大で、かつ FPGA の温度が最小のときに起こります。この場合、デバイスの内部遅延は減少し、ホールド タイムの違反を招く危険が高まります。

電圧と温度の変動に加え、システム全体に存在するさまざまなジッタの発生源により、クロック システムが不確実になりがちです。ジッタによりクロック エッジの到着が早くなったり遅くなったりするため、セットアップ タイムやホールド タイムのエラー発生率が高まります。

従来、FPGA ベースのスタティックなタイミング解析ソフトウェアは、クロックの不確実性を考慮せず、ワースト ケースの温度および電圧条件下でデバイスの動作を解析することしかできませんでした。今より遅いシステム レベルのインターフェイス規格にはこの方法で十分でしたが、今日の高速なデュアル データ レート（ DDR ）ソース同期インターフェイス規格では、より完全な検証ソリューションが求められます。

現実の条件に即した STA 解析

タイミング検証を最も正確に行うため、ザイリンクスのスタティックなタイミング解析ソフトウェアは、自動的にワースト ケースとベスト ケースの動作条件でデザインを解析します。この解析は、デザインのシステム I/O インターフェイスと内部ロジックの両方に対して同時に実行されます。また、この方法はセットアップ タイムとホールド タイムの解析に最小遅延と最大遅延を両方使います。セットアップ タイムの解析では、データ パス遅延が最大、クロック パス遅延が最小のときにワースト ケースが起こります。反対に、データ パス遅延が最小、クロック パス遅延が最大のときは、ワースト ケースのホールド タイム解析条件となります。

デザインにおけるクロックの不確実性を考慮に入れるため、ザイリンクスのソフトウェア システムでは、各クロックに対して入力ジッタを指定できるようになっています。入ってくるクロック ジッタに加え、システム ジッタとクロッキング システムのデザインに起因するクロックの不確実性には、タイミング解析のチェック項目のすべてが自動的に考慮されます。クロックの不確実性により実質的にクロック パス遅延が減ると、セットアップ タイムの違反を招く危険性が高まります。同様に、クロックの不確実性によりクロック パス遅延が増えると、ホールド タイムの違反が起こりやすくなります。

ザイリンクスのタイミング解析ソフトウェアは、クロック パスとデータ パスの最小遅延と最大遅延を同時にモデル化する機能を備え、すべての動作条件にわたってシステム デザインの信頼性を最大限高めます（ 図 2 ）。

ISE 8.1i ソフトウェアによる正確なシミュレーション

スタティック タイミング解析は、FPGA デザインにおけるデータ パスとクロック パスの物理的遅延がセットアップ タイム違反やホールド タイム違反を起こさないよう検証しますが、デザインのファンクションの動作を検証する必要もあります。デザインのダイナミックな特性により、ファンクションのタイミング動作をシステム レベルのスピードでテストする必要があるためです。正確なタイミング シミュレーションを行うには、スタティック タイミング解析のようにプロセス、電圧、温度、クロックの不確実性を原因とするベスト ケースとワースト ケースの条件を考慮しなければなりません。ザイリンクスの ISE™ 8.1i ソフトウェアは、最小および最大のクロック遅延とパス遅延を同時にシミュレーションできるようにすることで、極めて正確なダイナミック タイミング シミュレーションを実現しました。このユニークな機能は、セットアップとホールド タイムの違反を正確に考慮に入れ、すべてのシミュレータで自動的に働きます。

ザイリンクスの使いやすいシミュレータ

ModelSim Xilinx Edition III

ザイリンクスは、Mentor Graphics 社の Model Technology 事業部と共同で、ポピュラーな「 ModelSim PE 」シミュレータをカスタマイズした廉価版の 「 ModelSim Xilinx Edition III（ MXE III ）」を開発しまし

図 3　ザイリンクスの使いやすい「 ModelSim Xilinx Edition III (MXE III)」

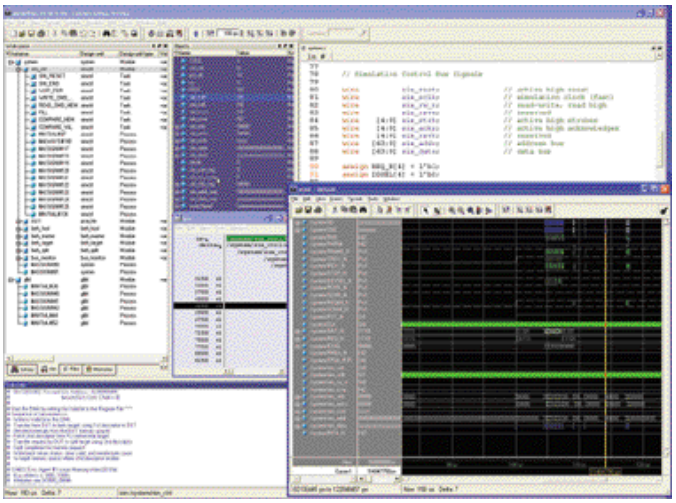
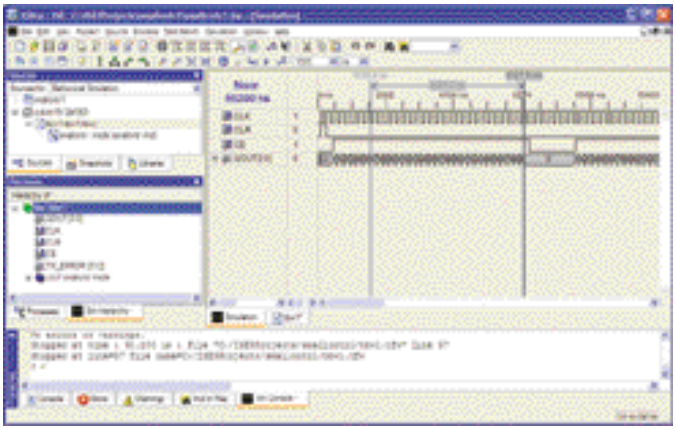


図 4　シームレスなインテグレーションを実現するザイリンクスの ISE シミュレータ



た（図 3 ）。MXE III は、Spartan™-3E FPGA ファミリなど、容量 200 万システム ゲートまでの中密度 FPGA に理想的です。このシミュレータを使えば、デザインと HDL ソースコードのファンクションとタイミング モデルを検証することが可能です（ 詳細は http://www.xilinx.co.jp/ise/ を参照 ）。MXE III の低パフォーマンス バージョン「 ModelSim Starter 」は、ISE Foundation™ ツールセットとして無償で提供される機能です。

MXE III の特徴と機能は次のとおりです。

- ISE ソフトウェアとのシームレスな統合が可能で、グラフィカルなテスト ベンチの自動生成を通してよりダイナミックな検証を実現し、また、視覚性の高い Project Navigator プロセス ウインドウを提供
- MXE-II より大容量かつ高速
- システム Verilog と Verilog PLI/VPI をサポート
- 卓越したデバッグ環境
- 波形管理ツール
- カスタマイズ可能なユーザー インターフェイス
- バッチ モードでのシミュレーション
- HDL エディタ
- ソース コードのデバッグ
- Verilog-2001 または VHDL-93 のサポート（ 単一言語の製品 ）
- MXE-III Starter バージョンは、MXE-II より 50 % 高速な HDL シミュレーションと 20 倍のデザイン容量を提供
- ModelSim PE や SE など、よりパワフルなシミュレータにアップグレード可能な XPower へのエクスポート機能
- テスト ベンチを作成するための統合型波形エディタ
- デザイン階層、波形、コンソールのビュー
- ソース レベルのデバッギング
- TCL インターフェイスを搭載したコマンド ライン コンソール
- FlexLM ライセンス不要
- 「 Generate Expected Results 」プロセスにより、入力スティミュラスに基づいて予想されるデザイン出力ビヘイビアを生成

ISE Software SIM 8.1i

ザイリンクスは、ISE Foundation のユーザー向けに、オプションのデザイン製品としてすべての機能を搭載した統合型の HDL シミュレータを提供しています（ 図 4 ）。また、ISE Foundation には ISE Simulator Lite という無償のシミュレータも用意されています。これは ISE Simulator の入門版であり、小型のデバイスに理想的です。

ISE SIM 8.1i には次のような特徴があります。

- Verilog 2001 および VHDL-93 の混在デザインをサポート
- シンプルなユーザー インターフェイス
- デバイスの消費電力を容易に見積もり可

階層デザインの容易な検証

ISE Foundation ツールキットの KEEP_HIERARCHY という機能を使うことにより、階層デザインをより簡単かつ高速にデバッグすることができます。このデザイン フローは、タイミング シミュレーションの実行時間を短縮するだけでなく、デバッグ中に問題を発見して解決するという大きな役割も果たします。

この機能の目的は、デザインが合成やインプリメンテーションのフロー時に選択したサブモジュールの階層を保ち、デザイン全体がアセンブルおよび検証される前にタイミング シミュレーションでこれらサブモジュールを検証することです。

各サブモジュールは、別個のネットリストとして書き出し、別個の関連する SDF ファイルを用いて RTL シミュレーションとタイミング シミュレーションの両方で検証することが可能です。タイミング ネットリスト用の各サブモジュールは、RTL 用と同じに見えるため（ 最上位のポート名は同じ ）、タイミング シミュレーションでは RTL シミュレーションで使用したテスト ベンチにほとんど、あるいはまったく手を加えずそのまま流用できます。

この方法を Virtex™-4 の代表的なデザインで解析したところ、シミュレーションの実行時間とシミュレーションに必要なメモリが両方とも大幅に減少しました。詳細は、「 ISE 8.1i Synthesis and Verification Design Guide 」の「 Design Hierarchy and Simulation 」のセクションをご覧ください。

より高速なシミュレータ
ザイリンクスのデバイス向けに
パフォーマンスを改善

ザイリンクスとパートナ各社にとって、シミュレーションの実行時間を短縮することは永遠の目標です。ハードコードしたザイリンクスのライブラリ プリミティブを備える Cadence 社の NC-Sim v5.5 と、「 vopt 」を搭載する Mentor社の ModelSim SE 6.1 は、シミュレーションの実行時間を改善しました。

高度な検証方法

アサーション ベース検証

アサーション ベース検証（ ABV ）は、アサーション、ファンクション カバレッジ、フォーマル モデル チェックの各テクノロジを融合した検証方法で、ASIC およびザイリンクスの FPGA デザインの両方に利用が可能です。アサーションとは、デザインの意図を明示的に表現したもので、回路構造が何をすべきで何をすべきではないかを規定します。デザインにアサーションを埋め込み、デザインの動作をモニタリングさせることで、そのデザインの観察性を改善できます。ザイリンクスの FPGA 向け ABV の詳細は、本号 .xx ページの「 ABV を活用する初期欠陥の発見 」をご覧ください。

等価チェック

等価チェックとは、フォーマル テクニックを使い、異なる開発段階にある 同一デザインの2つのバージョンが機能的に等価かどうかを判断するスタティックな検証テクノロジです。テスト ベクタは必要なく、検証時間を 10～100 倍高速化するうえ、100 % のファンクション カバレッジを提供します。また、ツールに起因するバグをチェックすることもできます。コードはタイミングの目標を満たすよう変更されているため、等価チェックがファンクション チェックを実施する際、シミュレーションの実行に長時間かかることはなくなります。

ザイリンクスのデバイスに等価チェックのフローを使うには、次のように行います。

1. http://www.xilinx.co.jp/ise/partner_libraries/ で等価チェック ライブラリをダウンロードする。
2. コンスタント レジスタの除去、レジスタの複製、レジスタの併合、ディスエーブル リタイミングなどの最適化をオフにすることで、「 フォーマル検証可能 」なネットリストを生成するよう、合成ツールとインプリメンテーション ツールをセットアップする。

   a. Synplify Pro では、等価チェックをイネーブルして自動化したセットアップ ファイルを書き出すため、次の変数を使う。

   ```
   <deign>.vif (text):
   set_option -verification_mode 1
   set_option -write_vif 1
   ```

   b. DC FPGA では、等価チェックをイネーブルして自動化したセットアップ ファイル「 .svf 」(暗号化) を書き出すため、set_fpga_default -formality を含める。

   c. ISE に対しては、<デザイン>.svf ファイルに ISE の最適化したコンスタント レジスタのリストを含めるため、「 netgen -ecb 」をイネーブルする。

3. CoreGen から等価チェック用の Verilog ネットリストを出力する。このネットリストは、インスタンシートされた各 CoreGen IP が合成後から PAR 後のチェックまで CoreGen ブロックを検証するためのファンクション モデルとして使われます。

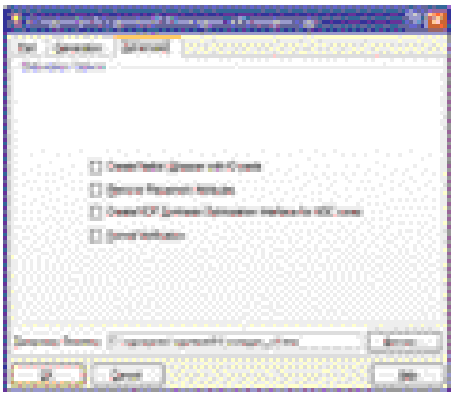
等価チェックを成功させるには次の点に注意してください。

- RTL で大規模なブロック RAM をインスタンシエートするには CoreGen を使う（ 図 5 ）
- 合成ツールのリタイミングをオフにする
- ワイド乗算器を別の階層に隔離し、それをブラックボックス化する
- RTL のターゲットを新しいアーキテクチャに置く際は、古いインスタンシエートされたコンポーネントを置き換える

Synplicity 社（ Synplify ）、Cadence Design Systems 社 （ Conformal ）、Synopsys 社（ DC FPGA と Formality ）は、設計者がこれを実現できるよう、ザイリンクスと協力してきました。次のものに関する詳細は各社にお問い合わせください。

- Synplicity 社の Synplify Pro V8.0+ と Conformal V5.0+ の合成
- Synplicity 社の Synplify Pro V8.0+ と eCheck V4.3+ の合成
- Synopsys 社の DC FPGA V2005.03+ と Formality V2005.03+

図 5　等価チェックのためのザイリンクス CoreGen ネットリスト



結論

検証は設計者にとって最も注意の必要な 3 大要素の 1 つになっています。ザイリンクスとアライアンス プログラム パートナー各社は、以下を通して設計者の検証作業を改善するため継続的な投資を行っています。

- タイミング精度の改善
- 統合性に優れ、使いやすいツール
- 階層デザインのサポート
- より高速なシミュレータ パフォーマンス
- 新しいメソッド

詳細は、http://support.xilinx.co.jp/ をご覧ください。

Early Defect Discovery with Assertion-Based Verification Accelerates Design Closure

ABV を活用する初期欠陥の発見

デザイン、合成、検証の統合によるアサーション ベースのバグ発見法

Ping Yeung
Principal Engineer
Mentor Graphics Corporation
ping_yeung@mentor.com

Darren Zacher
Technical Marketing Engineer
Mentor Graphics Corporation
darren_zacher@mentor.com

一般に、モバイルやワイヤレス、ネットワーキング、メディアといったアプリケーションに、プラットフォーム FPGA を用いてシステム オンチップ（ SoC ）をデザインする場合、設計者は短期間に多くの IP コンポーネントを統合することになります。先進のザイリンクス MicroBlaze™ とオンチップ Power PC™ プロセッサを搭載する Virtex™-II Pro や Virtex-4 ファミリなどのプラットフォーム FPGA を使えば、プロセッサ IP の統合にすぐに着手できます。

どの IP を選択しても、市場の要求に応えるにはさらに機能を増やす必要があり、複数のプロセッサ コアとマルチレイヤのバス アーキテクチャを採用するなど、プロセッサ ベースのプラットフォーム デザインはいっそう複雑さを増すばかりです。こうした中、プラットフォーム FPGA のデザインと検証に新たなアプローチが求められています。論理合成ロジックを押しボタンのように単純にマッピングし、後から思いついたように検証するようなやり方では、もはや通用しません。

プラットフォーム FPGA のデザイン フローが、すべてのステップで合成と検証を並行して進めていくという ASIC SoC のデザイン手法にかなり近づいてきたように、FPGA の開発者も、当初は複雑な ASIC SoC 向けだった合成/検証の方法を採用することに魅力を感じ始めています。

デザインと検証の統合という流れの中、最も注目されているのがアサーション ベースの検証（ ABV ）です。アサーションはマイクロプロセッサのデザインに 10 年以上にわたって利用されてきました。しかし、その潜在能力がフルに認識されるようになったのは、ごく最近のことです。特にサードパーティや自社の IP がますます増えつつあるプラットフォーム ベースのデザインなど、大規模かつ複雑なデザインに大きなメリットがあります。FPGA の複雑さとサイズが大規模 ASIC に近づきつつある現在、アサーションは FPGA の設計者にとって非常に有用になりました。

デザイン上の問題点の早期発見を可能にする統合的なフローの中で、ABV の価値を明らかにするため、デザイン クロージャの成功に向けてお勧めしたいいくつかの手法を紹介するとともに、ABV について詳しく見ていきます。

成功に不可欠な手法

欠陥は、エラーを修正する労力とコストが

少ないデザイン サイクルの初期段階に発見される必要があります。次に、デザイン クロージャを加速するためのベスト プラクティスをいくつか挙げます。

- **事前にできるだけ多くのタイミングとパフォーマンス解析を行う**
  選択したアービトレーション スキームが着信トラフィックについていけるか確認するために、デザインのファンクションが検証されて合成、配置・配線されるまで待っていられるでしょうか。スループットの問題を早期に発見するには、合成プロセスを通じてパフォーマンスとタイミングの問題をより深く解析する必要があります。配置・配線でタイミング制約を満たそうと何度も試行錯誤する前に、制約を完全にしておかなければならないのです。また、タイミングの問題を早期に発見するには、合成中に制約の影響範囲を解析する必要もあります。

- **予測性を高めるため、インタラクティブな合成テクニックを使う**
  FPGA デザイン ツールキットに欠かせない武器は、RTL から物理インプリメンテーションにいたるまでインタラクティブな合成・解析が行える環境です。インタラクティブな合成テクニックにより、デザイン サイクルの初期に「what-if（仮定）」シナリオを試すことができます。また、堅牢な合成環境は、高水準な演算子やアーキテクチャ固有のテクノロジ セルなど、いくつかのデザイン表現を提供します。インタラクティブな合成機能により、デザインの特性について初期段階から理解できるうえ、仕様を満たすかどうかもわかります。

- **HDL コードを初期のうちに頻繁にチェックする**
  今日のデザイン管理ツールでは、コードを設計チームやシリコン ベンダが合意したコーディング スタイルのルールと照合することができます。シミュレーション サイクルを繰り返す前に、コーディング スタイルのルール チェッカを使って実際の欠陥を見つけ、また潜在的な問題点を予測することで、デザイン サイクルの早い段階で欠陥を発見できるのです。

- **より効果的なファンクション検証戦略を導入する**
  プラットフォーム FPGA に対する合成ツールは、単にテクノロジをマッピングしたネットリストを生成するにとどまりません。一流の合成ツールには、サイクル内のあらゆる段階でデザインを詳しく検討できる重要な解析機能が用意されています。これらの機能を使えば、ファンクションの検証に細心の注意を払わなければならないクロック ドメインの交差点など、潜在的な問題エリアを識別できます。さらに、VHDL や Verilog テスト ベンチを使ってプラットフォーム FPGA に従来のファンクション検証アプローチを適用する場合、ランダムもしくは擬似ランダム スティミュラスをモデル化して、回路の応答が設計者の意図に即しているかをチェックするのは非常に面倒になります。こういう場合は ABV が大きな役割を果たします。

アサーション ベース検証

従来のシミュレーション ベースの検証をさらに一歩進めるには、デザインの検証プロセスと制御の観察性を改善する必要があります。そのためには、アサーション、ファンクション カバレッジ、フォーマル モデル チェックの各テクノロジを融合した ABV を使うのがベストです。アサーションとは、デザインの意図を明示的に表現することであり、回路構造が何をすべきで何をすべきでないかを捕捉します。アサーションをデザインに埋め込み、デザインの動作をモニタリングさせることで、観察性が改善されます。フォーマル モデル チェック（FMC）は RTL 構造を解析し、制御を強化するためデザインの内部特性を特徴付けます。

ABV はデザイン全体にわたり無数の観測地点（アサーション）を作成するため、エラーをその発生源近く、もしくは発生源で識別することで、デバッグを高速化します。また、アサーションはプライマリ出力に伝播されない問題も識別するので、ABV はアサーションがなければ検出されなかったかもしれないバグを顕在化させてデザインの品質を改善する働きもします。

アサーションは FMC の目標値として一式のプロパティを提供します。また、シミュレーション テスト ベクトルを使ってダイナミック フォーマル検証がモジュールのアサーションの潜在的な違反を発見することが可能です（詳しくは後述します）。

ABV の基本的なメソッドは、デザインのインプリメンテーションをその指定したアサーションと比較します。したがって、デザインのアサーションの品質と包括性は大変重要です。これらのアサーションは次のソースから派生します。

- さまざまな構造をインプリメントするとき、設計エンジニアは、独自のアサーション言語、もしくは SystemVerilog Assertions（SVA）や Property Specification Language（PSL）といった標準のアサーション言語を使ってアサーションを書くことができます。その際、不注意からくるミスでインプリメンテーションに失敗することもありますが、アサーションは意図するビヘイビアと実際のビヘイビアの間にクリティカルなクロス チェックを提供します。たとえば、ポインタの意図する最大値が 48 だと仮定します。コーディング ミスによってポインタが 48 を超えるような場合、アサーションはそのコーディング ミスを検出します。

- 検証エンジニアはデザインの仕様、設計書、もしくはデザインに関する個人的な知識をもとにしてアサーションを書くことができます。通常、こうしたアサーションは高レベルな要件に応え、デザインのクリティカル ビヘイビアをカバーします。たとえば、ある DMA コントローラが、10 回のメモリ転送を行うよう設定されたチャネルを持っているとします。そのチャネルを解除する前に、総転送回数が 10 ぴったりでないと、重大な問題が起こります。検証エンジニアはこうしたイベントをチェックするためのアサーションを追加できます。

- 設計エンジニアと検証エンジニアは、該当するインターフェイス プロトコルの違

図 1　アサーションを利用したシュミレーション ベースの検証

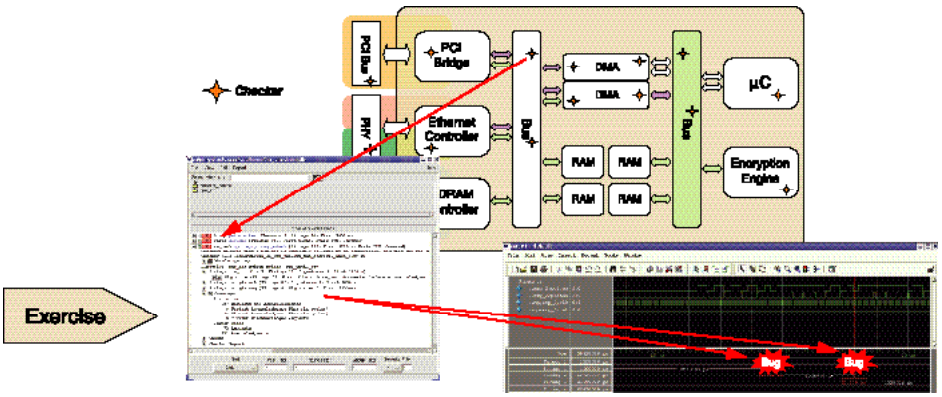
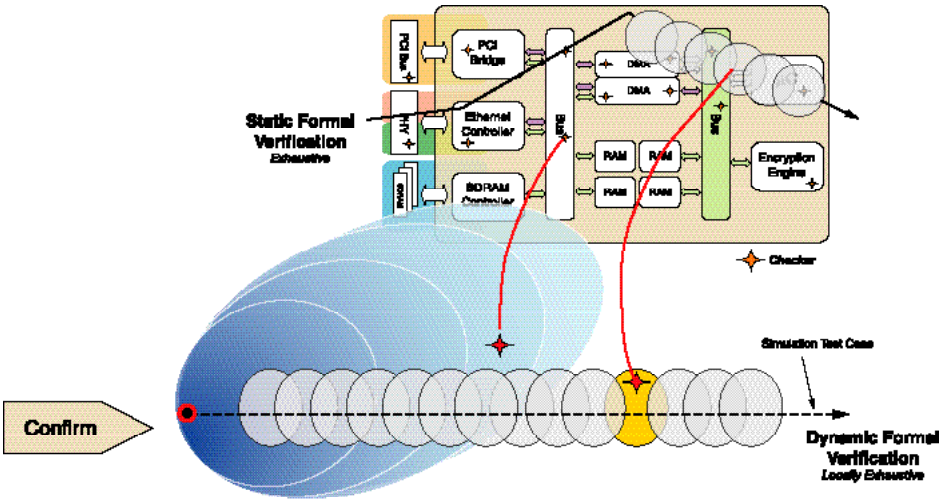


図 2　アサーションを利用したフォーマル モデルの確認



反をチェックするため、プロトコル アサーション モニタを挿入します。これらのモニタは、コンポーネント間のトランザクションが適切に生成され、正しく扱われることを保証します。EDA ベンダは、一般的な標準インターフェイス向けに市販のプロトコル アサーション モニタを開発しました。メンター グラフィック（ Mentor Graphics ）社の CheckerWare プロトコル モニタはその一例です。市販のモニタでは、プロトコルのルールはすべてカプセル化され、プロトコル規格の改定に応じて常にアップデートされています。シミュレーションと一緒に使うことで、プロトコル アサーション モニタは有益な統計情報を収集します。また、フォーマル検証のときには、制約として働きます。

- 一部の検証ツールは、RTL ソースコードの構造を解析することでデザインにアサーションを挿入します。多くの一般的なデザイン問題を扱うソリューションは、これを解析するためのベースを提供しており、計装プロセスの自動化が可能です。多くの問題は、ツールそのものによって（ ネットリストの解析を使用 ）、もしくはフォーマル解析を使うことで、スタティックに発見されます。それ以外の問題は、ファンクション シミュレーションやダイナミック フォーマル検証などのダイナミック検証メソッドによってのみ検出されます。アサーションの自動挿入により解析される問題としては、合成とシミュレーションのミスマッチの問題、コードまたは FSM のステートに到達できない問題、クロックドメイン交差（ CDC ）エラー、X セマンティックの問題、および CDC クロック ジッタがあります。

ABV の基礎知識

ABV は、バグの発生源近く、もしくは発生源で、そのバグを早期に発見して修正することで、デバッグを加速するとともにデザインの品質を改善します。

アサーションを用いたシミュレーション ベースの検証

ファンクションのシミュレーション中、デザインのアサーションはそのデザイン内部とインターフェイス上で動作をモニタリングします（ 図 1 ）。アサーション違反はただちに報告されるため、問題がデザインの出力ポートに伝播されてからテスト ベンチで検出する必要はありません。このような高い観察性により、バグを簡単に選別し、発見した問題の潜在的な原因について理解を深めることができるのです。

アサーションを用いたフォーマル モデル チェック

FMC はシミュレーションに対する補充的なテクノロジであり、ABV メソッドに不可欠な要素です。FMC は数学的手法を用いて、アサーション ライブラリから提供される一式の仮定を基に一部のアサーションを true、また反例を発見することで他のアサーションを false として証明します。証明とは、FMC がそのアサーションが取り得るあらゆるビヘイビアを調べた結果、違反はありえないと判断したということです。一方、反例はアサーションが満たされない状況を示します。FMC はデザインの考えられる限りのステートを調べて、どれか 1 つのステートでも、指定したプロパティ セットに違反しないかを判断します。

メンター グラフィックス社の 0-In ビジネス ユニットが開発した最新の FMC テクニックであるダイナミック フォーマル検証は、

与えられたシミュレーショントレースを増幅することで、スタティック フォーマル検証のメモリと計算の制約を克服しています。これは、スタティックおよびダイナミック フォーマル テクニックを使い、FMC をサブブロックとチップ レベルの両方で行えるようにすることで ABV を拡張します（図 2）。

アサーションやカバレッジ ポイントとして表される一式のターゲット ビヘイビアを指定すると、ダイナミック フォーマル検証は、シミュレーショントレースに沿って「気になるステート」のフォーマル解析を使い、カバレッジ ポイントのどれかに到達したり何らかのアサーションに違反したりする可能性がないかを判断します。ファンクション検証の場合、「気になるステート」とは、新しい、もしくは珍しいデザイン ビヘイビアが発生した部分のことです。

シミュレーションの何サイクルも深いところで起こるバグを考えてみましょう。このバグはデザイン内の特定の FIFO がフルで、特定の FSM が FOO ステートにある場合にのみ起こり得ます。シミュレーションの際、テストが FIFO を満たすようにして、同じテストでも別のテストでも FSM を FOO ステートにすることが可能です。ランダム スティミュラスがぴったりかみ合って両方のイベントが起こるという確率はかなり低いでしょう。

ダイナミック フォーマル検証を使用する場合、FIFO が満杯となっているシミュレーション ステートから、フォーマル解析がその時点に到達して超過するまでの考え得るすべてのステートを調べ、バグを招くステートに結び付く特定のシーケンスを識別して報告します。このようにダイナミック フォーマル検証と制約付きのランダム スティミュラスを併用することで、通常なら追加のテスト ベンチ コードを使わなければ検出できないバグを発見できるのです。

ダイナミック フォーマル検証を使用しない場合、潜在的なバグが隠れていることを念頭におき、テストを正しいステート セットに導くためにスティミュラスの制約を変更する必要があります。現実には、ダイナミック フォーマル検証はどれでも 1 つのシナリオを 10,000 以上の有効なテストに増幅させることができます。このことを考えれば、制約に前述の微調整を含める必要がないので、テスト ベンチ用の実際の制約セットははるかにシンプルになり、そうした綿密な制約環境を構想してコーディングするのにかかる膨大な時間が節約されます。

インターフェイス プロトコル モニタ

インターフェイス プロトコル モニタを使えば、プラットフォーム ベースのデザインにあるデバイスを切り離して検証できます。プロセッサ ベースのしっかりしたデザインでは、すべてのコンポーネントがインターフェイスと正しくやり取りできなければなりません。プロトコル モニタは、該当するインターフェイスの違反をブロック レベルまたはチップ レベルでチェックします。

プロトコル モニタには、プロトコルのルールを適用するためのアサーションが含まれています。このモニタは、コンポーネント間のトランザクションが適切に生成され、正しく扱われることを保証します。シミュレーションやフォーマル検証中、不正なプロトコル トランザクションがその発生源で識別されます。

ハードウェア関連の問題については、CheckerWare アサーション ライブラリを用いた ABV メソッドにより、プロセッサ ベースのプラットフォーム FPGA デザインに存在する問題をいち早く発見できます。シミュレーション中に違反をチェックする以外にも、CheckerWare プロトコル モニタはファンクション カバレッジと統計情報を収集し、フォーマル ファンクション検証に対する制約として働きます。モニタは実行されたトランザクションを確認し、リグレッション スイートの「ホール（穴）」をハイライト表示して、擬似ランダム スティミュラス生成環境の品質を監査します。

検証の難しい構造

プロセッサ ベースのデザインがますます複雑化する中、コンポーネントはデザインの奥深くに埋め込まれるため、その制御とテストを効果的に行うのは困難です。たとえば、マルチレイヤのバス アーキテクチャでは、複数のマスタが複数のスレーブに同時にアクセスできます。この機能は全体的なシステム帯域を高めるものの、検証の複雑さも増します。

このような検証の難しい構造に取り組むときは、アサーションとチェッカを用いてデザインの意図を把握し、RTL コードに埋め込めばいいのです。エンベデッド アサーションは、RTL コードそのもののエラーに起因する不正なビヘイビアを自動的にチェックし、設計者や検証エンジニアが手作業することなくデザインの問題をローカルに捉まえます。

エンベデッド アサーションにより、問題の発生源をより素早く突き止めることもできます。従来の検証フローでは、それぞれの障害にたくさんの原因が考えられたため、シミュレーション中に検出したエラーからその発生源までトレースするのにとても時間がかかります。たとえば、いくつかのインターフェイスからメモリにデータを転送するため複数の DMA チャネルをセットアップしてある場合、テストの最後にメモリ データにエラーを検出しても、そのデータ エラーを招いたトランザクションまで遡ってトレースするのは困難です。アサーションを使えば、データ損傷の正確な時間と、その問題に関わるハードウェア リソースを素早く突き止めることができるのです。

結論

最近のプラットフォーム FPGA はますます大容量化、複雑化しており、企業にとっては過去にないビジネス チャンスが開かれましたが、設計者にはいっそう厳しいチャレンジが課されることになりました。問題点を早期に発見するため、デザインから合成、検証にいたるまで統合的なフローを活用し、またそれに順応することは、繰り返しの減少とより高速なデザイン クロージャという点から FPGA エンジニアリング チームに必ずやメリットをもたらすでしょう。

ABV は、問題の発生源、もしくはその近くでバグを発見して修正することで、デバッグを加速するとともに、完成後まで検出されないかもしれないエラーも見つけることで、開発コストの削減とデザイン品質の改善を同時に達成します。

メンター グラフィックス社の高度な検証メソッドの詳細は、http://www.mentor.com/products/fv/ をご覧ください。

Simplifying FPGA Pin Assignment Closure

FPGA 設計における ピン配置の簡素化

PCB 上で FPGA を統合する際のピン配置変更の理由とツール紹介

Philippe Garrault
Technical Marketing Engineer
Xilinx, Inc.
philippe.garrault@xilinx.com

PCB と FPGA の両方の環境要件を満たすピン配置は、以前より困難になってきています。FPGA 周辺のインターフェイスでは、FPGA のさらなる高性能化、高密度化、そして I/O 数の増加により、FPGA との間を行き来する信号のレイアウトにいっそう厳しい制約を課しています。その結果、PCB 上でますます高速化する信号のタイミング、混雑、シグナル インテグリティが、FPGA のピン配置に制約を課しています。

こうした課題は、米 CMPMedia 社の「EE Times 」(http://www.eetimes.com/) が実施した最新の EDA 調査の結果にも表れています。回答者の 3 分の 2 が、最新のデザインに 2 つ以上のプログラマブル デバイスを使っていると答えています。また、彼らは FPGA デザイン プロジェクトの 2 番目に難しい課題として、「FPGA を PCB で正しく動作させるようにする 」ことを挙げています。

最近まで、FPGA のピン配置を支援するツールやプロセスはほとんどなかったのですが、そうした状況は変わりつつあります。本稿では、今日のデザイン環境でピン配置を変更する理由を見ていくとともに、変更にどういう意味があるのかを説明し、FPGA のピン配置を簡素化、自動化するための各種ツールを紹介します。

FPGA のピン アサインを変更する必要性

システムのデザイン プロセスを通じて、FPGA ピン アサインを変更する理由はたくさんあります。図 1 のフローチャートは、FPGA ピン アサイン クロージャと、これら変更の理由にスポットを当てた典型的なデザイン フローを表しています。変更を行うエリアはたくさんありますが、そのほとんどの要因は、次の 3 つです。

1. デザイン フローの制約によるピン アサインの変更

競争が激しく常に進化を続ける今日の電子機器市場では、マーケット ニーズの変化にいち早く対応するため、各企業ではデザイン サイクルを短縮することが急務となっています。このため、製品開発はできるだけ並行して行うというのが一般的です。これは FPGA と PCB のデザイン プロセスにも当てはまります。システム設計者はインターフェイス特性の初期リストを定義し、PCB と FPGA の設計者はそのリストを基にデザインに取り掛かります。この配置は、後で PCB と FPGA のチームが製品開発を進める中で変更されていきます。さらに、開発サイクルの途中で、市場動向に応じて新しいプロトコルに対するサポートを追加したり、完成直前に機能を追加したりするなど、ピン アサインへの変更が必要になることもあります。

2. PCB の制約によるピン アサインの変更

フォームファクタの規格やボードのコスト管理という理由から、ボード上の実装スペースが限られていることがあります。このような場合、FPGA ピン アサインのプログラマビリティと柔軟性を活かして、PCB の混雑や配線の問題を解決できます。FPGA の機能として、たとえば次のものを使用できます。

- ボード上のネットのもつれを解くためピンを交換します。これにより必要なビアの数が低減し、レイヤ (基板層) の数も減る可能性があります。レイヤの数が減ると信号の環境が変わり、シグナル インテグリティと電磁放射も改善されます。また、ボードのコストも低減できます。

- 信号の駆動強度やスルー レートを低減させることで、I/O の特性を調整し、ボードのシグナル インテグリティを強化することができます。

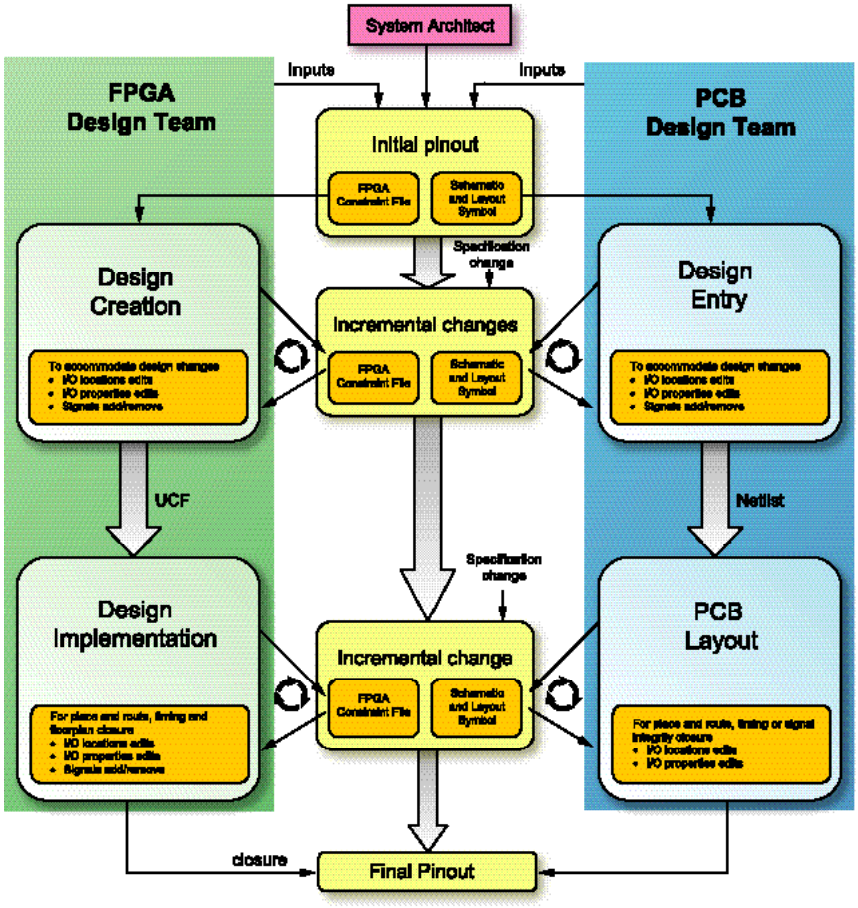
- ディスクリート コンポーネントのコスト節約や混雑エリアのボード スペース節約のため、プログラマブルな内部終端を使うことができます。

3. FPGA の制約によるピン アサイン変更

この場合はさらに FPGA が PCB のデザインに制約を課すため、インプリメンテーションの進行に伴いピン アサインの変更が必要になることがあります。このようなピン アサイン変更の理由は次のとおりです。

- タイミング :
  デバイスに出入りするいくつかの信号の

図 1　ピン配置決定の典型的なプロセス



マージンがタイトすぎて、一部のパッケージ ピンでしか動作しない場合があります。

- 専用/特定用途向けピン：
  グローバル クロックやリージョナル クロック、プログラミング ピンのような特定用途向けピンとして動作するようプログラミングできるのは、パッケージ ピンのサブセット（ 一部分 ）のみであるため、ボードには信号を対応する I/O に配線するという制約がかかります。

- 電圧/終端の互換性：
  FPGA の I/O はバンクにグループ化され、特定バンク内のすべてのピンが電源電圧と基準電圧を共有します。つまり、いったんこのバンクに特定の電圧を使うと、同じバンクには互換性のある電圧を持つ I/O しか割り当てることができません。その場合、PCB の配線という観点から見れば必ずしも適切ではない FPGA パッケージ上のピンを選択せざるを得ません。

- 同時スイッチング出力（ SSO ）：
  FPGA により駆動される信号が I/O 標準規格の電気特性を満たすよう、ザイリンクスはデバイスのエリアあたりの最大推奨 SSO 数を定めています。この推奨最大数を超える場合、ピンを複数のエリアに分散する必要のある場合があります。

- デバイス デカップリング回路：
  FPGA は、ボード上の他の IC と同様、たとえば回路に電力を供給するためローカル デカップリング ネットワーク（ ディスクリート コンポーネント ）を追加するなど、電力供給システムになんらかの特性を必要とします（ 通常、電圧レギュレータはデバイスの急激な需要の変化に対応できません ）。

ピン配置を変更する意味

ここまで述べてきた理由により、システムのデザイン中に FPGA ピン アサインが変更されるのはほぼ確実です。では、これらの変更は時間と労力の点で、FPGA と PCB の環境にどういう意味があるのでしょうか。両方の環境が同期化されていることと、両方の環境の制約が一致していることを検証するにはどうすればいいのでしょうか。

FPGA サイドから見た場合、ピン アサインを変更するたびにユーザー制約ファイル（ .UCF ）をアップデートし、内部のタイミング制約が満たされていることを検証する必要があります。また、ピン配置の I/O バンキングや SSO ガイドラインを検証するため PACE と SSO の計算を実行することもできます。

PCB サイドからは、ピン アサインの変更後、回路図シンボルとレイアウト シンボルをアップデートする必要があります。また、変更された信号の電気特性と物理特性が制約をパスすることを確認するため、デザイン ルール チェック（ DRC ）を実行します。さらに、タイミングと振幅のマージンがまだ十分なことを検証するため、シグナル インテグリティの解析を実施することもできます。

重要なのは、ピン配置の変更を他の環境に伝達することです。これまで、PCB と FPGA のデザイン環境は切り離されていて、お互いをつなぐ連絡路は限られていました。次のセクションで説明するとおり、現在では同期化とデータ転送にかかる手間と時間を大幅に抑えられるツールやオプションがあります。

利用可能なツールとプロセス

FPGA ピン アサインは変わるものだということ、また FPGA および PCB デザインにおける変更の頻度とその意味についても理解できたことと思います。では、こうした変更を取り込む負担を最小限に抑えるにはどういう選択肢があるでしょうか。両方の環境間で自動的にピン マッピングを伝達し、データが変換中に失われないようにするにはどうすればよいでしょうか。PCB と FPGA の両チームにとって満足のいくソリューションが見つかるまで何度も反復できるよう、ピン アサインの変更を素早く行うための方法はあるでしょうか。

FPGA 環境内

PACE（ ピン配置とエリア制約エディタ ）プログラムを使えば、パッケージ ピンにデザインの信号名をグラフィカルに割り当てることができます。PACE は、FPGA の内部ロジックをグラフィカルに表現したダイ ビューも表示します。したがって、PCB 設計者からのロケーションに関する要望を念頭に置きながら、ピンを駆動ロジックに近接して配置できます。I/O 規格、駆動、スルー レートに対するピン特性も入力できます。グラフィカル ビューは、グローバル クロックやリージョナル クロックなどの特別な用途向けのピンを簡単に識別します。また、PACE はピン配置が構造上正しいことを確かめるため、電圧準拠と SSO のチェックを実行します（ 図 2 ）。

このツールは、デザイン プロセスのさまざまな段階で役立ちます。ピン配置をゼロから作成したり、合成されたネットリストから信号名を読み込んだりできます。配置・配線後、インプリメンテーションや PCB ツールの制約を満たすため、ピン配置をインタラクティブに調整できます。PACE は Verilog モジュールや VHDL エンティティ、さらにピン アサイン ファイルを生成します。このファイルは、回路図シンボルとレイアウト シンボルを生成・アップデートするために、ほとんどの PCB ツールにロード可能です。また、PACE は PCB ツールからピン ファイルを読み込んで、ISE™ ソフトウェア用の UCF 制約ファイルを生成します。

ここでもう 1 つ紹介しておきたいツールとして、Product Acceleration 社（ http://www.prodacc.com/ ）の DesignF/X があります。このアプリケーションを使い、I/O 特性（ 標準や駆動強度 ）と、インターフェイスを配置するパッケージの一般的なエリアを定義します。その後、このツールは PCB の配線を容易にするためすべての I/O を配置するとともに、FPGA バンキング、クロッキング、および SSO のルールに準拠していることを確認します。

PCB 環境内

お使いの PCB デザイン ツールには、FPGA の回路図シンボルとレイアウト シンボルを作成・維持するうえで役立つ、ウィザードや文書化されたプロセスがあるはずです。ピン アサインの変更後に、これらシンボルを容易にアップデートできる方法も用意されていることでしょう。さらに、ピン数の多いデバイス向けに、ドキュメント化とコネクティビティ プロセスを簡素化するため、シンボルを複数のシートに分割するオプションもあります。

FPGA と PCB を橋渡しするツール

FPGA と PCB の両環境を橋渡しするツールは最近登場したばかりで、その機能リストはどんどん拡大しています。Mentor I/O Designer（ 図 3 ）のようなプログラムは、まだピン配置に関連する FPGA と PCB のすべての制約を理解しているわけでないので、万能薬ではありませんが、FPGA ピン アサイン クロージャの簡素化、加速化に向けて既にかなりの進歩を実現しています。どちらか一方の環境から、次のことが行えます。

- **ピン アサインの変更後、両環境の同期化を自動化する**
  プログラムはザイリンクスの UCF ファイルと回路図シンボル、レイアウト シンボルをアップデートします。これにより、エラーを招きやすい手作業でのデータ入力とその後の検証プロセスが大幅に減り、FPGA と PCB の両環境を同期化できます。

- **FPGA パッケージのグラフィカル ビューからピンを配置または交換する**
  PACE と同様、これはピン配置がデバイスのルールに違反しないことを確認するため DRC を実行します。

- **PCB のグラフィカル ビューからピンを配置または交換する**
  FPGA がインターフェイスしている他のコンポーネントの物理的ロケーションを視覚化することで、ボード上のワイヤ クロスオーバを減らし FPGA ブレークアウト（ 包囲環境 ）を簡素化するような形でピンを配置するための情報が得られます。この結果、デザインにかかる時間（ PCB ルータの反復使用による ）が短縮されるうえ、ボード スペースの有効利用とレイヤやビアの低減によりコストも節約されます。

図 2　PACE を使った PCB ツールへのピンアウトのエキスポート

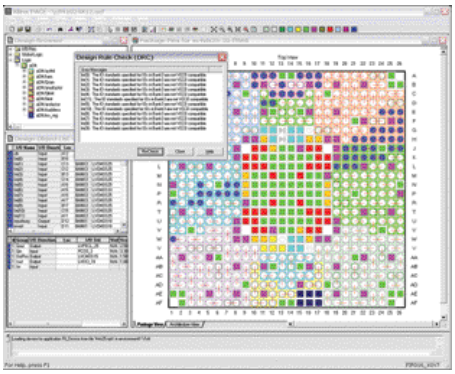
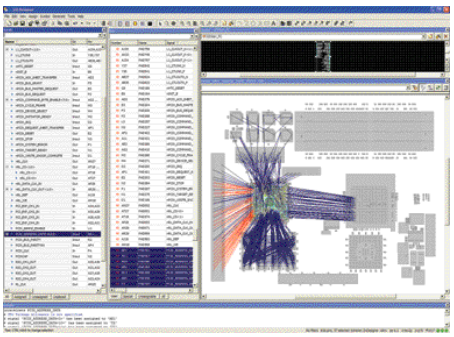


図 3　Mentor I/O Designer の PCB 配線図



結論

FPGA と PCB の両環境は、以前に増して厳しい制約、ときには矛盾する制約さえ課しており、従来のスプレッドシートによるアプローチではプログラマブル デバイスのピン アサインを達成するためにいっそう手間がかかるようになっています。両環境を同期化するため、時間をかけて手作業でデータ入力を行うのでは、設計者は最適とはほど遠い PCB しか作成できず、FPGA のパワーと柔軟性を十分に活かしきれません。しかし、幸いなことに、FPGA や EDA のベンダがピン アサイン データの作成、管理、同期化を支援する多数のツールとテクニックを発表しており、その種類はどんどん増えています。時間を有効に使うためにも、ぜひこれらのツールをご利用ください。

Power Considerations in 90 nm FPGA Designs

90 nm FPGA のデザインにおける消費電力の考察

Virtex-4 FPGA を使った消費電力を削減するデザインを学ぶ

Matt Klein
Sr. Staff Engineer, Applications EngineeringÅA
Advanced Products Division
Xilinx, Inc.
matt.klein@xilinx.com

システムの信頼性を維持するには、システムの電力バジェットをしっかり管理することが重要です。これを怠ると、コンポーネントが破壊され、信頼性が損なわれる場合があります。

半導体業界の 90 nm シリコン プロセスへの急速な移行は、パフォーマンスとコストの面では利点があるものの、電力バジェットには大きなプレッシャーをかけています。トランジスタのサイズが小型化するにつれ、リーク電流、すなわちスタティック消費電力は指数的に増加していきます。システムの高速化とデザインの高密度化に伴いダイナミック消費電力も増加しますが、その増加はより直線的です。

今日、多くのデザインにおけるスタティック消費電力とダイナミック消費電力の割合は 50/50 です。国際半導体技術ロードマップ（ ITRS ）は、スタティック消費電力はプロセス ノードごとに指数的に増えているため、革新的なプロセス技術が不可欠になると予測しています。

高性能化/高集積度化に加え、価格の低下に伴い、FPGA を採用する市場やシステムが年々増え続ける中、システム全体における FPGA の消費電力はますます重要になっています。大手 FPGA ベンダは、すでにスタティック消費電力とダイナミック消費電力の低減に向けて新しいテクニックを採用しています。

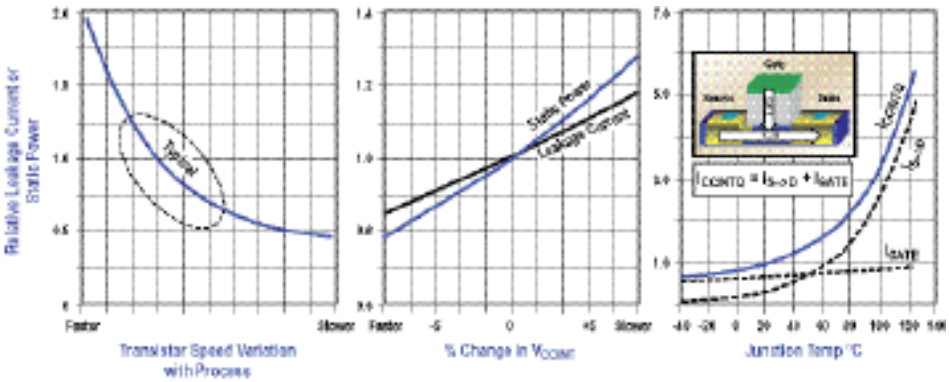
消費電力の削減方法

システムにおける消費電力は非常に重要であり、その大部分は、システム統合機能に使われることの多い FPGA で占められます。通常、システムや個別コンポーネントは、それぞれ決められた電力バジェットを持ち、このバジェットは大きく 2 つのカテゴリに分けられます。1 つ目のカテゴリは、デザインに使われている電源の電力容量を合わせたシンプルで実質的なバジェットで、これはシステムの所要電力を満たす必要があります。2 つ目のカテゴリは熱バジェットで、各種コンポーネントが温度仕様内でシステムを動かすために理解する必要があります。従って、使用する FPGA の電力が消費される場所と、それを最適化する方法について知ることが必要です。

電力バジェット内で動作

すでに述べたとおり、電力バジェットが問題となるのは、消費電力と熱についての懸念からです。ここで典型的な例を紹介しましょう。ボードの電力バジェットが 20W で、通常の動作環境は 10°C～40°C とします。ファンが故障した場合、特定のコンポーネント上の周辺温度が 70°C を超えることがあります。多くのコンポーネント メーカの動作条件は、ジャンクション温度で最高 85°C（ コマー

図 1　プロセス、電圧、温度によって変化するリーク電流とスタティック電力



シャル グレード)、またインダストリアル グレードの部品には 100°C と指定しています。ザイリンクスの ISE™ XPower や Web ベースの電力見積もりツールを使用すると、消費電力が落ち込む場所を調べるなど、FPGA の消費電力を最適化する必要がある場合に利用できます。また、FPGA で電力を消費する部分と、消費電力を減らすためのデザイン最適化の方法を習得することも重要です。

FPGA のどこで電力が消費されるか

FPGA には、電力を消費するエリアが主に 2 つあります。スタティック電力はトランジスタのリーク電流から、またダイナミック電力は電圧スイング、トグル レート、キャパシタンスから消費されます。両方とも、電力バジェットと電力最適化を満たすうえで重要な要素です。したがって、それぞれの要素の意味と、異なる動作条件でどう変動するかを知っておくことが大切です。

図 2　ダイナミック電力の変化はコア電圧の変化に依存

V_{CCINT} Variation from Nominal	Dynamic Power Change
-10.0%	-19.0%
-5.0%	-9.8%
0.0%	0.0%
5.0%	10.3%
10.0%	21.0%

スタティック電力

スタティック電力とは、リーク電流によりトランジスタが消費する電力です。90 nm のデバイスでは、このリーク電流は顕著です。トランジスタからより高いパフォーマンスを得るには、トランジスタの電圧スレッショルド（V_T）を低減する必要があり、しかし一方、これもリーク電流を増やす一因になります。

90 nm トランジスタのリーク電流はプロセスで大きく変わります。トランジスタの V_T はドーピング量により変化し、ゲート長はリソグラフィにより変化します。この結果、トランジスタのスピードとリーク電流は大きく変動することになります。V_T やゲート長を減らすとリーク電流とスピードが高まり、その逆もしかりです。リーク電流とスタティック電力の変動量は、ワースト ケースと一般的なプロセスで約 2:1 になります。

リーク電流とスタティック電力はコア電圧 VCCINT によっても大きく左右され、変動量はそれぞれ VCCINT の 2 乗および 3 乗になります。スタティック電力は、VCCINT が 5 % 増えるだけで約 15 % 増加します。リーク電流は、ジャンクション（チップ）温度 T_J によって非常に大きな影響を受けます。

プロセス、電圧、温度の各要素は、FPGA のリーク電流とスタティック電力に大きな影響を与えるため、それぞれの要素を理解すると共に、FPGA や ASIC の総合的な消費電力にどう影響するかを理解することが大切です。ゲートからサブストレートへのリーク電流も総リーク電流の一部ですが、温度に大きく左右されることはありません。

図 1 は、90 nm FPGA におけるトランジスタのリーク電流とスタティック電力が、プロセス、電圧、温度によってどう変化するかを示したものです。

高性能な 90 nm FPGA への移行に伴いトランジスタのリーク電流が増加することを受けて、ザイリンクスの IC 設計者は最新の Virtex™-4 FPGA のトランジスタに第3の厚さを持つゲート酸化膜を採用しました。従来の FPGA と ASIC には、コアトランジスタに薄い酸化膜、I/O トランジスタに厚い酸化膜という、2 種類の酸化膜（デュアル酸化膜）しかありませんでした。Virtex-4 FPGA のトランジスタの一部に、第 3 の中厚酸化膜（トリプル酸化膜）とより高い V_T を使用したことで、トリプル酸化膜を使わない競合他社の FPGA と比較して総合的なリーク電流とスタティック電力が大幅に減少しています。プロセス、電圧、温度に伴う変動は引き続き発生するものの、Virtex-4 FPGA における絶対リーク電流は、競合他社の高性能 90 nm FPGA の約 1/3 程度に抑えられているのです。

ダイナミック電力

ダイナミック電力とは、トグルするトランジスタとトレース（配線）によって消費される電力です。これは、内部電圧がロジック「0」からロジック「1」へ、あるいはロジック「1」からロジック「0」へ変化する際に、キャパシタンスがその電圧まで充放電されるために発生するものです。この頻度が多いほど消費電力は増えます。FPGA では、トランジスタはロジックと、メタル トレース間のプログラマブルなインターコネクトに使われます。ここで言うキャパシタンスとは、トランジスタの寄生キャパシタンスとメタル配線のキャパシタンスを指します。

ダイナミック電力の式は次のとおりです。

$$P_{DYNAMIC} = nCV2f$$

n=トグルするノード数、C=キャパシタンス、V=電圧スイング、f=周波数

FPGA 内のすべてのノードは、トランジスタの寄生キャパシタンスとメタル配線によるキャパシタンスの充放電の組み合わせ

図 3　ザイリンクスの Web ベースの電力見積もりツールの画面

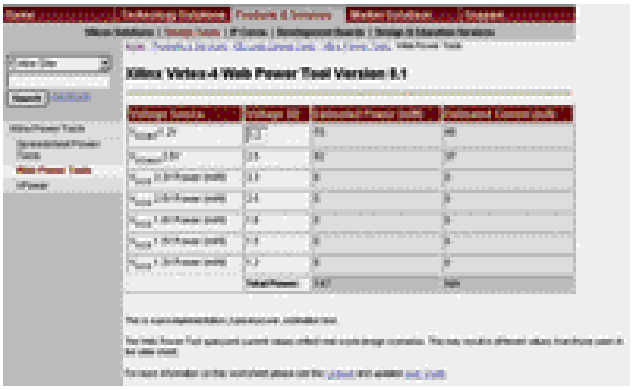
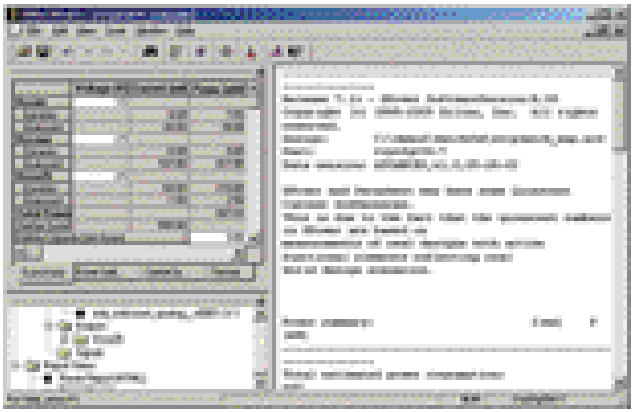


図 4　ISE XPower 電力見積もりツールの画面



で電力を消費します。後者は FPGA 内の配線の長さに左右され、ネット ノードのキャパシタンスはスイッチングするトランジスタの数で決まります。ロジックを高集積度にパッキングするほどスイッチングトランジスタの数が減り、配線の長さも短くてすむため、ダイナミック電力は減少します。

図 2 は、電圧スイングとコア電圧 VCCINT に伴いダイナミック電力がどう変化するかを示したものです。プロセスと温度がダイナミック電力に及ぼす影響はほとんどありません。この 2 つを併せても、影響は 5～10 % 未満です。

FPGA の環境を変えることでデザインの消費電力を低減

デザインの消費電力を最適化するには、FPGA 内のデザインとは独立して行えることがいくつかあります。そのためには、まず環境を知ることが重要です。

温度

温度制御はスタティック電力を削減するのに役立ちます。図 2 のとおり、ジャンクション温度を 100°C から 85°C に下げるだけで、スタティック電力は約 20 % 低減されます。Virtex-4 FPGA のスタティック電力は他の 90 nm FPGA に比べてもともと低いのですが、デザインによっては FPGA のスタティック電力が総電力バジェットのかなりの割合（ 30 ～40 % ）を占めるため、これを 20 % 削減することは大きなメリットです。ジャンクション温度は、エアフローを増やし、FPGA から熱を放出する大型のヒート シンクを使うことで低下することができます。ジャンクション温度の低減は信頼性のさらなる向上にもつながります。

電圧

コア電圧を名目値以下に維持することで、スタティック電力とダイナミック電力を低減できます。コア電圧 VCCINT で消費されるスタティック電力とダイナミック電力は、FPGA で最大の消費電力であることが多々あります。通常、FPGA は名目値 +/- 5 % の電源電圧で動作させることでパフォーマンスを満たすように設計されています。図 2 のとおり、VCCINT が +/- 5 % 変わると、スタティック電力とダイナミック電力はそれぞれ ± 15 % と ± 10 % 変動します。VCCINT の電源電圧をより厳密に指定できるのであれば、名目値より 5 % 高いワースト ケースではなく、名目値と同じかそれを少し下回る値に設定することができます。

デザインの変更により消費電力を低減

FPGA のデザインに関連する消費電力のトレードオフ（ 利害得失 ）を決めるには、どこから手を付けるべきかを知る必要があります。ザイリンクスは、デザインの消費電力を初期の段階で、あるいは詳細に見積もるための、いくつかのデザイン ツールを用意しています。

図 3 に示すザイリンクスの Web Power ツールを使用することにより、デザインのサイズ（ ロジックとフリップフロップ 数 ）、動作周波数、トグル レート、エンベデッド ブロックのユーティリゼーション、さらに温度など環境条件の見積もりに基づいて、デザインの消費電力を早い段階で見積もることができます。これは、正確な配置・配線やユーティリゼーションといったデザインについての詳細情報に依存しません。

ISE ソフトウェアのすべてのコンフィギュレーションに搭載されている XPower ツールは、デザインに対するスティミュラス ベクタを入力することで消費電力を詳しく解析できるプログラムです。このツールは、実際に配置・配線したデザインの情報を使い、図 4 のようにはるかに正確に消費電力を算出します。

消費電力を低減するための FPGA デザイン テクニック

いくつかのデザイン固有のテクニックを使って消費電力を低減することもできます。たとえば、可能な限りロジックを小さなエリアに制約する、エリアを最小限に抑えるよう合成フラグを設定する、ロジックのレイヤを極力少なくする、といったテクニックがあります。また、パイプライン化を行うことでより高いタイミング制約が設定可能で、キャパシタンスとダイナミック電力が減少します。

配置とタイミングの制約を設定する

デザインのフロアプランを正しく作成することによりダイナミック電力は低減します。ISE Floorplanner では配置制約を作成できます。また、PlanAhead™ デザイン

図 5　ISE 8.1i ソフトウェアが備える消費電力ベースの回路最適化機能を使ったインターコネクト キャパシタンスの削減

FPGA Family	Average Reduction Capadtance	Maximum % Capadtance Reduction
Spartan-3	11%	21%
Virtex-4	6%	9.5%
Virtex-II Pro	13%	18%

ツールというザイリンクスのより高度なフロアプランニング ツールを利用することにより、デザイン内の階層を観察し、関連するロジックのセットを小さなエリアにグループ化することが可能です。配置制約とグループ化制約を使うことで物理エリアは減り、より高いパフォーマンスを達成できると共に、配線キャパシタンスとダイナミック消費電力の低減につながります。

他に、デザイン内のタイミングを制約するテクニックもあります。合成ツールと ISE の配置・配線ツールにはタイミング制約を入力する機能が用意されています。ターゲットのタイミング制約、特にクロック ターゲットのタイミング制約を上げると、ルータはいっそう配置・配線に努力し、なんとかその制約を満たそうとします。これにより配線が最小限になり、配線による消費電力が低減します。

その他の部分的シャットダウンやリプログラミング方法

ダイナミック消費電力を低減するには、他にもよく使われるデザイン テクニックがあります。その 1 つがクロック マルチプレクシングを使う方法です。この方法では、FPGA の一部をオフにします。Virtex-4 FPGA とそれ以前の FPGA には、クロック ゲート化ブロックというハードウェアの機能があり、これを使ってグローバル クロック ネットをスムーズにオン/オフに切り換えることができます。この方法はフリップフロップのクロック イネーブルより優れており、トグルする大規模なクロック ネット全体のゲートをオフにできるため、ネットとフリップフロップの消費電力の削減につながります。

バッテリ型のアプリケーションに使われているデザインなどでは、使用するときのみ電力を消費します。このようなアプリケーションでは、クロックをオフにし、FPGA データが消えないぎりぎりのレベルまでコア電圧を低減できます。Virtex-4 デバイスの場合、最小レベルは 0.9 V です。このレベルでは、名目値の 1.2 V と比較して、スタティック消費電力は 60 % 以上も低減されます。

一定のタスクに必要な FPGA のサイズを縮小するための方法として、ダイナミック リコンフィギュレーション ポート（DRP）があります。DRP は Virtex-4 を始めとするザイリンクスの各種 FPGA で利用できます。共存する必要のないファンクションがある場合、このポートを使って FPGA の一部だけをリロードすることが可能です。こうすることにより、より小さな FPGA を選択でき、結果的にスタティック消費電力の低減になります。

エンベデッド ブロックの使用

消費電力を減らすためのもう 1 つの方法に、エンベデッド ブロックの使用があります。場合により、特別なブロックをインスタンシエートするほうがよいこともありますが、Virtex-4 FPGA は基本的に ASIC ゲートである多数のハード IP を備えています。実際、これらのファンクションの中には最新の合成ツールにより自動的に合成されるものもあります。これらの新しいブロックは、プログラマブル ロジックとプログラマブル インターコネクトのインプリメンテーションより、1/5 ～1/20 の電力しか消費しません。エンベデッド ブロックがスタティック電力を減らすのは、プログラマブル ロジックと異なり追加のトランジスタを持たず、プログラマブル インターコネクト用のトランジスタをまったく使わないためです。また、メタル インターコネクトとプログラマブル インターコネクトを両方使うのでなくメタル インターコネクトだけを使うこと、トレース（配線）長が短縮されること、パストランジスタがないため余分なノード キャパシタンスが少なくてすむこと、ロジックのレイヤ数が最小限に抑えられることなどで、ダイナミック電力も低減することになります。

このようなブロックとしては次のものがあります。

- PowerPC™ - 高性能なエンベデッド プロセッサ
- DSP - 高性能で先進の多機能型算術および論理ブロックである XtremeDSP™ スライス
- SSIO - すべての I/O ピンで利用できる新しい ChipSync™ ブロックで、ソース同期 I/O デザインに対するロジック セルの数を削減
- エンベデッド Ethernet MAC - MAC ファンクションにロジックとインターコネクトの使用が不要
- FIFO - ビルトインの FIFO コントローラを含む SmartRAM メモリ
- SRL16 - プログラマブル インターコネクトなしで複数のカスケード接続されたフリップフロップを使用可能

消費電力を考慮した配線の最適化

ISE 8.1i ソフトウェアでは、消費電力を考慮したデザインの最適化を図る優れたツールを提供します。このツールは新しくリリースするもので、より高速なタイミング制約を入力しなくても、自動的にキャパシタンスを最小化できます。

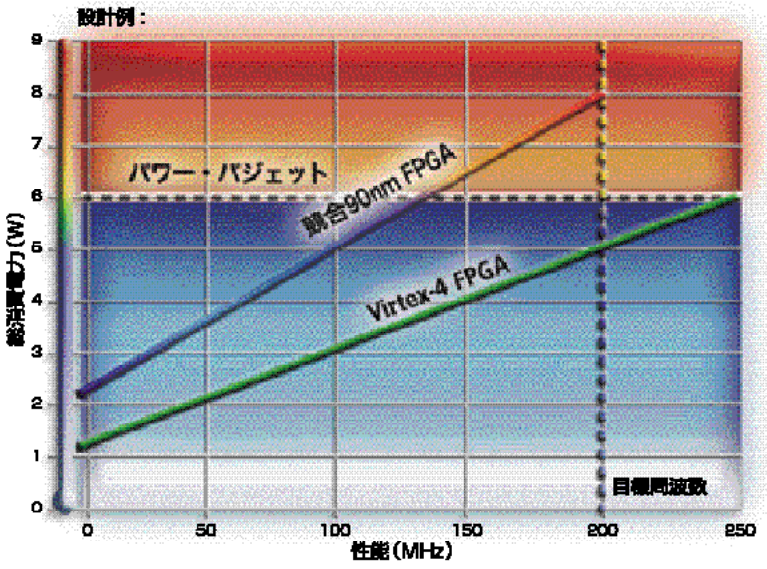
図 5 は、このインターコネクト キャパシタンスを低減する機能のテスト結果です。インターコネクト キャパシタンスはダイナミック消費電力の約 2/3 と見積もられており、これだけ改善されればダイナミック消費電力の削減に貴重なツールと言えるでしょう。

ISE ソフトウェアの将来のリリースでは、消費電力を最適化する合成や配置など、消費電力の最適化機能をさらに拡充していく予定です。

結論

システムの電力バジェットと動作環境を知ることは非常に重要です。さまざまな形の消費電力がどこから発生するのかを理解することにより、FPGA 環境とデザイン特性を調整し、消費電力を最小限に抑えられるうえ、目標とする電力バジェットをうまく満たすことができます。

低消費電力のXILINX FPGAで、
あなたのシステムをいつもクールに



設計例：LX60 vs. 2560、目標周波数＝200 MHz、ワーストケース・プロセス
20K LUT、20Kフリップフロップ、1Mbitオンチップ RAM、64 DSPブロック、125　2.5V I/O
ザイリンクス ツール v4.0.8および競合ツール v8.1にて
高集積度デバイスでは最大で3ワットの消費電力削減が可能

FPGA1個あたり1～5ワットも消費電力が削減できるのはVirtex-4だけ

現実的な動作温度（Tj＝85℃）での仕様をご自身でお確かめください。
トータルな消費電力削減に関して、Virtex-4に優るFPGAはありません。

- スタティック消費電力を73%削減
- ダイナミック消費電力を最大86%削減

トリプル酸化膜テクノロジとエンベデッドIP

90nmテクノロジ・ノードにおけるシステム・レベル設計者が直面する次の問題は消費電力です。デバイスによっては、漏れ電流、スタティック消費電力の急増、さらには熱暴走を引き起こしてしまうのもあり、このため、Virtex-4 FPGAはトリプル酸化膜テクノロジ、エンベデッドIP、省電力設定回路を使用して設計されました。これにより予定消費電力内で目標とする性能が達成できます。

www.xilinx.co.jp/virtex4/lowpower

最少のシステムコストで、最高のシステム・パフォーマンスを。

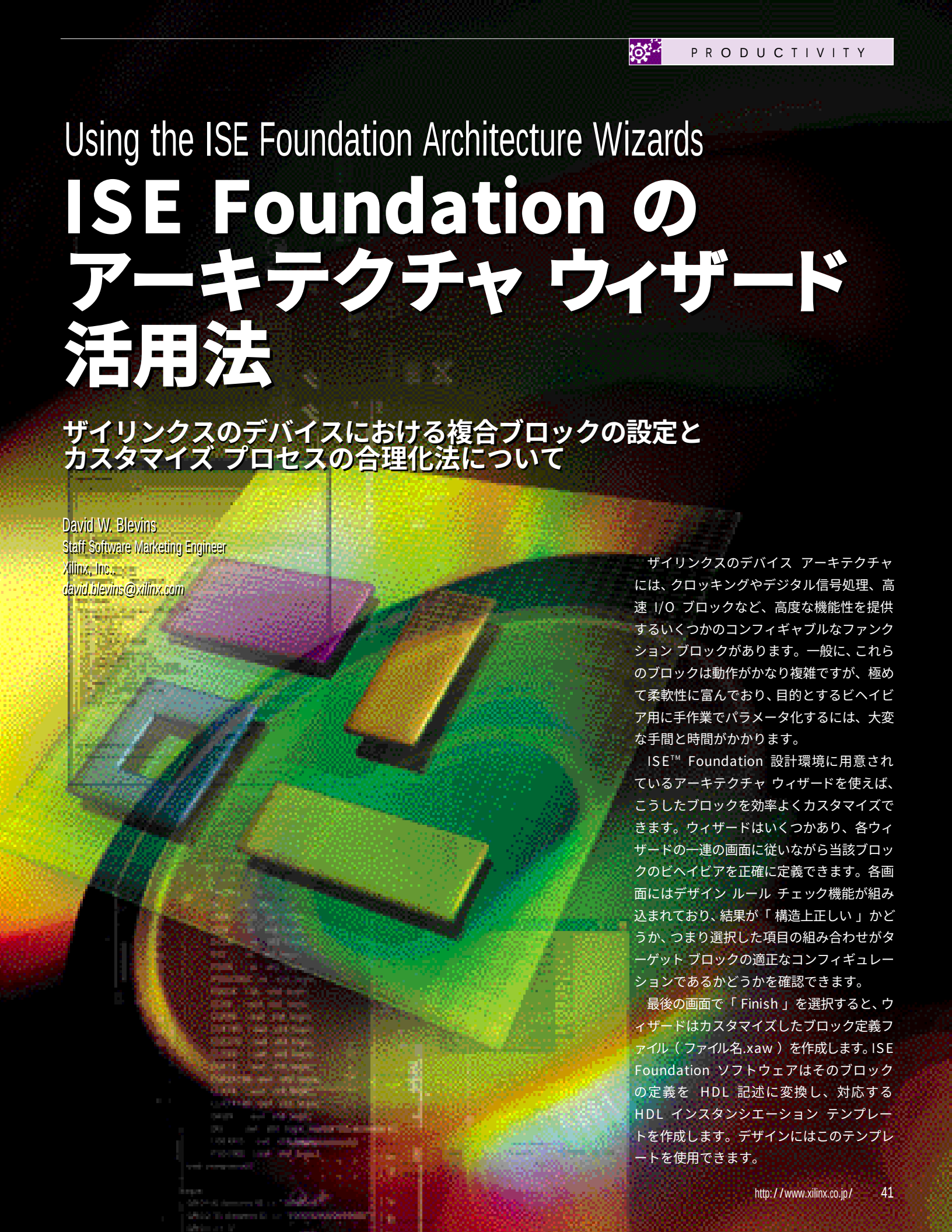
XILINX®
The Programmable Logic Company™

Using the ISE Foundation Architecture Wizards

ISE Foundation のアーキテクチャ ウィザード活用法

ザイリンクスのデバイスにおける複合ブロックの設定とカスタマイズ プロセスの合理化法について

David W. Blevins
Staff Software Marketing Engineer
Xilinx, Inc.
david.blevins@xilinx.com



ザイリンクスのデバイス アーキテクチャには、クロッキングやデジタル信号処理、高速 I/O ブロックなど、高度な機能性を提供するいくつかのコンフィギャブルなファンクション ブロックがあります。一般に、これらのブロックは動作がかなり複雑ですが、極めて柔軟性に富んでおり、目的とするビヘイビア用に手作業でパラメータ化するには、大変な手間と時間がかかります。

ISE™ Foundation 設計環境に用意されているアーキテクチャ ウィザードを使えば、こうしたブロックを効率よくカスタマイズできます。ウィザードはいくつかあり、各ウィザードの一連の画面に従いながら当該ブロックのビヘイビアを正確に定義できます。各画面にはデザイン ルール チェック機能が組み込まれており、結果が「構造上正しい」かどうか、つまり選択した項目の組み合わせがターゲット ブロックの適正なコンフィギュレーションであるかどうかを確認できます。

最後の画面で「Finish」を選択すると、ウィザードはカスタマイズしたブロック定義ファイル（ファイル名.xaw）を作成します。ISE Foundation ソフトウェアはそのブロックの定義を HDL 記述に変換し、対応する HDL インスタンシエーション テンプレートを作成します。デザインにはこのテンプレートを使用できます。

ウィザードの終了後、ISE Foundation ソフトウェアが生成した HDL コードとインスタンシエーション テンプレートを見れば、この種のタスクにアーキテクチャ ウィザードを使うことのメリットがはっきりとわかるでしょう。

クロッキング ウィザード

例として、デバイス外部のクロックにより駆動される Virtex™-4 FPGA 用のデジタル クロッキング マネージャ モジュールを作成するため、クロッキング ウィザードを使ってみましょう。このモジュールは、主クロック信号と、イネーブル バッファを駆動する周波数逓倍（ 2 倍 ）クロックを生成します。LOCKED 信号は、FPGA が電源投入、もしくはリセットされた後、DCM クロック信号が安定したことを示します。

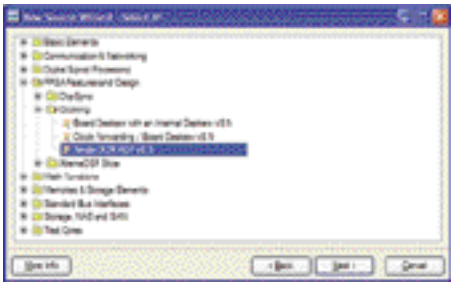
ウィザードの終了後、ISE Foundation ソフトウェアが生成した HDL コードとインスタンシエーション テンプレートを見れば、この種のタスクにアーキテクチャ ウィザードを使うことのメリットがはっきりとわかるでしょう。

クロッキング ウィザードの起動

まず、ISE の Project Navigator 内から「Add New Source 」を選択して「 IP (CoreGen and Architecture Wizard)」ソース タイプを選択することで、クロッキング ウィザードを起動します。するとダイアログ ボックスが表示されるので、そこで「 Single DCM ADV v8.1i 」(図 1) を選択します。

以降のダイアログ ボックスで「 Next 」と「 Finish 」をクリックすれば、クロッキング ウィザードが起動されます。

図 1　IP ダイヤログ ボックスの選択



一般設定画面

クロッキング ウィザードの最初の画面で、必要とするブロックの入力と出力、クロック ソース、フェーズ シフトの種類、周波数、フィードバック コンフィギュレーションを選択します（ 図 2 ）。

図 2　一般的なセットアップ画面

図 3　クロック バッファ画面

クロック バッファ画面

「 Next 」をクリックすると、クロック バッファ画面に進みます。デフォルトの場合、すべてのクロック出力はグローバル バッファを駆動しますが、このデザイン例では、周波数逓倍クロックの出力の後にイネーブル バ

図 4　バッファのビュー/編集画面

図 5　サマリ画面

ッファを配置する必要があります。そこで、CLK2X のグローバル バッファをイネーブル バッファに変更するため、「Customize Buffers」を選択し、CLK2X の横にある「Global Buffer」ボタンをクリックします（図 3）。

次の画面で「Add Buffer」ボタンをクリックし、イネーブル バッファをブロックのクロック入力に配置します（図 4）。

サマリ画面

「OK」ボタンを押してバッファを作成したら、クロック バッファ画面の「Next」をクリックしてサマリ ページに進みます。サマリ ページには生成されるブロックについての詳細なレポートが表示されます（図 5）。

「Finish」ボタンをクリックすると、ISE Foundation プロジェクト ディレクトリに「ファイル名.xaw」というバイナリのソースファイルが作成されます。「.xaw」は、ISE Foundation ソフトウェアで、デザインをインプリメントする際、合成に必要となる HDL 記述に自動的に変換されますが、「.xaw」ファイルの作成後は、その HDL をいつでも見ることができます。サポートされる HDL は、VHDL と Verilog のいずれでも使用できます。この例では VHDL を使用しています。

次のコードを手作業で書くことを考えれば、アーキテクチャ ウィザードの真価がおわかりいただけるでしょう。

```
-- Module dcm1
-- Generated by Xilinx Architecture Wizard
-- Written for synthesis tool: XST

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity dcm1 is
  port ( CLKIN_ENABLE_IN : in   std_logic;
         CLKIN_IN        : in   std_logic;
         RST_IN          : in   std_logic;
         CLKIN_IBUFG_OUT : out  std_logic;
         CLKIN_OUT       : out  std_logic;
         CLK0_OUT        : out  std_logic;
         LOCKED_OUT      : out  std_logic);
end dcm1;

architecture BEHAVIORAL of dcm1 is
  signal CLKFB_IN       : std_logic;
  signal CLKIN_IBUFG    : std_logic;
  signal CLK0_BUF       : std_logic;
  signal GND1           : std_logic_vector (6 downto 0);
  signal GND2           : std_logic_vector (15 downto 0);
  signal GND3           : std_logic;
  component BUFGCE
    port ( I  : in   std_logic;
           CE : in   std_logic;
           O  : out  std_logic);
  end component;

  component IBUFG
    port ( I : in   std_logic;
           O : out  std_logic);
  end component;

  component BUFG
    port ( I : in   std_logic;
           O : out  std_logic);
  end component;

  component DCM_ADV
    generic( CLK_FEEDBACK : string :=  "1X";
             CLKDV_DIVIDE : real :=  2.000000;
             CLKFX_DIVIDE : integer :=  1;
             CLKFX_MULTIPLY : integer :=  4;
             CLKIN_DIVIDE_BY_2 : boolean :=  FALSE;
             CLKIN_PERIOD : real :=  10.000000;
             CLKOUT_PHASE_SHIFT : string :=  "NONE";
             DCM_AUTOCALIBRATION : boolean :=  TRUE;
             DCM_PERFORMANCE_MODE : string :=
"MAX_SPEED";
             DESKEW_ADJUST : string :=
             "SYSTEM_SYNCHRONOUS";
             DFS_FREQUENCY_MODE : string :=  "LOW";
             DLL_FREQUENCY_MODE : string :=  "LOW";
             DUTY_CYCLE_CORRECTION : boolean :=  TRUE;
             FACTORY_JF : bit_vector :=  x"F0F0";
             PHASE_SHIFT : integer :=  0;
             STARTUP_WAIT : boolean :=  FALSE);
      port ( CLKIN    : in   std_logic;
             CLKFB    : in   std_logic;
             DADDR    : in   std_logic_vector (6 downto 0);
             DI       : in   std_logic_vector (15 downto 0);
             DWE      : in   std_logic;
             DEN      : in   std_logic;
             DCLK     : in   std_logic;
             RST      : in   std_logic;
             PSEN     : in   std_logic;
             PSINCDEC : in   std_logic;
             PSCLK    : in   std_logic;
             CLK0     : out  std_logic;
             CLK90    : out  std_logic;
             CLK180   : out  std_logic;
             CLK270   : out  std_logic;
             CLKDV    : out  std_logic;
             CLK2X    : out  std_logic;
             CLK2X180 : out  std_logic;
             CLKFX    : out  std_logic;
             CLKFX180 : out  std_logic;
             DRDY     : out  std_logic;
             DO       : out  std_logic_vector (15 downto 0);
             LOCKED   : out  std_logic;
             PSDONE   : out  std_logic);
    end component;

begin
  GND1(6 downto 0) <= "0000000";
  GND2(15 downto 0) <= "0000000000000000";
  GND3 <= '0';
  CLK0_OUT <= CLKFB_IN;
  CLKIN_BUFGCE_INST : BUFGCE
    port map (CE=>CLKIN_ENABLE_IN,
              I=>CLKIN_IBUFG,
              O=>CLKIN_OUT);

  CLKIN_IBUFG_INST : IBUFG
    port map (I=>CLKIN_IN,
              O=>CLKIN_IBUFG);

  CLK0_BUFG_INST : BUFG
    port map (I=>CLK0_BUF,
              O=>CLKFB_IN);

  DCM_ADV_INST : DCM_ADV
  generic map( CLK_FEEDBACK => "1X",
               CLKDV_DIVIDE => 2.000000,
               CLKFX_DIVIDE => 1,
               CLKFX_MULTIPLY => 4,
               CLKIN_DIVIDE_BY_2 => FALSE,
               CLKIN_PERIOD => 10.000000,
               CLKOUT_PHASE_SHIFT => "NONE",
               DCM_AUTOCALIBRATION => TRUE,
               DCM_PERFORMANCE_MODE => "MAX_SPEED",
               DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
               DFS_FREQUENCY_MODE => "LOW",
               DLL_FREQUENCY_MODE => "LOW",
               DUTY_CYCLE_CORRECTION => TRUE,
               FACTORY_JF => x"F0F0",
               PHASE_SHIFT => 0,
               STARTUP_WAIT => FALSE)
    port map (CLKFB=>CLKFB_IN,
              CLKIN=>CLKIN_IBUFG,
              DADDR(6 downto 0)=>GND1(6 downto 0),
              DCLK=>GND3,
              DEN=>GND3,
              DI(15 downto 0)=>GND2(15 downto 0),
              DWE=>GND3,
              PSCLK=>GND3,
              PSEN=>GND3,
              PSINCDEC=>GND3,
              RST=>RST_IN,
              CLKDV=>open,
              CLKFX=>open,
              CLKFX180=>open,
              CLK0=>CLK0_BUF,
              CLK2X=>open,
              CLK2X180=>open,
              CLK90=>open,
              CLK180=>open,
              CLK270=>open,
              DO=>open,
              DRDY=>open,
              LOCKED=>LOCKED_OUT,
              PSDONE=>open);

end BEHAVIORAL;
```

この HDL ソース コードは、この例で使っている合成ツールへの DCM モジュールを記述したものですが、デザインにこのモジュールのインスタンスを作成する必要があります。インスタンスの作成は、Project Navigator 内の「 View HDL Instantiation Template 」プロセスを使って作成する、「インスタンシエーション テンプレート」で行います。こうして生成したコードをデザインに挿入することで、DCM モジュールのインスタンスを作成できるのです。

```
COMPONENT dcm1
        PORT(
                CLKIN_ENABLE_IN : IN std_logic;
                CLKIN_IN : IN std_logic;
                RST_IN : IN std_logic;
                CLKIN_IBUFG_OUT : OUT std_logic;
                CLKIN_OUT : OUT std_logic;
                CLK0_OUT : OUT std_logic;
                LOCKED_OUT : OUT std_logic
                );
        END COMPONENT;

        Inst_dcm1: dcm1 PORT MAP(
                CLKIN_ENABLE_IN => ,
                CLKIN_IN => ,
                RST_IN => ,
                CLKIN_IBUFG_OUT => ,
                CLKIN_OUT => ,
                CLK0_OUT => ,
                LOCKED_OUT =>
        );
```

サポートされている ブロック コンフィギュレーション

上の例は、アーキテクチャ ウィザードがサポートする 1 種類のブロックの1つのコンフィギュレーションを表しているにすぎません。他にも、次のブロックとコンフィギュレーションがサポートされています。

クロッキング ウィザード

デジタル クロック管理モジュールを使えば、デザイン内のグローバル クロッキング コンフィギュレーションを幅広く制御できるうえ、外部もしくは内部クロック ソースの選択、クロック デスキュー、フェーズ シフト制御、周波数合成を利用できます。

クロッキング ウィザードは、1 つ以上の DCM モジュールを使って複数の DCM コンフィギュレーションを作成できます。

- シングル DCM
- シングル DCM_ADV
- クロック転送/ボード デスキュー
- 内部デスキューを用いたボード デスキュー
- 2 個の DCM を用いたクロック スイッチング
- 2 個の DCM を用いた直列カスケード接続
- PMCD （ Virtex-4 FPGA ）

Rocket IO ウィザード

ザイリンクスの Rocket IO™ ウィザードは、Virtex-II Pro と Virtex-II Pro X デバイスでマルチギガビットのトランシーバ ブロックをコンフィギュレーションします（ Virtex-4 の Rocket IO は、CORE Generator™ ソフトウェアでサポートしています ）。プロトコルによっては、複数チャネルのトランシーバをコンフィギュレーションすることも可能です。

Virtex-II Pro デバイスについては、ウィザードを通して次の Rocket IO プロトコルを利用できます。

- ファイバ チャネル
- ギガビット イーサネット
- XAUI
- Infiniband
- Aurora
- PCI Express
- SONET OC-48 （ Virtex-II Pro X デバイス ）
- SONET OC-192 （ Virtex-II Pro X デバイス ）
- ユーザー定義のカスタム コンフィギュレーション

XtremeDSP ウィザード （ Virtex-4 デバイス ）

このウィザードは、Virtex-4 XtremeDSP™ スライスを使って、いくつかの異なる種類の DSP ブロックを作成できます。

- 乗算器
- アキュムレータ
- 乗算アキュムレータ （ MAC ）
- 加算器/減算器

ChipSync ウィザード （ Virtex-4 デバイス）

2 種類の基本的な ChipSync コンフィギュレーションを作成できます。

- メモリ アプリケーション モードは、メモリ アプリケーションに使う 1 ブロックの I/O をコンフィギュレーションします。このウィザードでは、遅延情報の指定を含めて、データ バスとクロック/ストローブをセットアップできます。また、アドレス バス、リファレンス クロック、制御信号をコンフィギュレーションすることも可能です。

- 非メモリ アプリケーション モードは、ネットワーキングのケースなど、非メモリ アプリケーションに使う 1 ブロックの I/O をコンフィギュレーションします。このウィザードでは、遅延情報の指定を含めて、データ バスとクロックをセットアップできます。また、リファレンス クロックと制御信号をコンフィギュレーションすることも可能です。

結論

本稿で述べてきたように、アーキテクチャ ウィザードはザイリンクスの FPGA にビルトインされている各種の複雑なブロックのカスタム インスタンスを容易に作成できます。その結果、効率のよい設計を推進できるでしょう。

ザイリンクスは Platform Studio、CORE Generator ソフトウェア、System Generator for DSP など、他にも製品の Time-to-Market を最小限にするための IP ブロック作成ツールを用意しており、アーキテクチャ ウィザードはこうしたツールをさらに補強する役目を果たします。

PlanAhead Software as a Platform for Partial Reconfiguration

パーシャル リコンフィギュレーションのプラットフォームとして活用する設計・解析ツール PlanAhead ソフトウェア

合理的なスペース、重量、電力、コスト削減環境を提供する PlanAhead

Nij Dorairaj
Senior Staff Engineer
Xilinx, Inc..
nij.dorairaj@xilinx.com

Eric Shiflet
Technical Marketing Engineer
Xilinx, Inc..
eric.shiflet@xilinx.com

Mark Goosman
Product Marketing Manager
Xilinx, Inc..
mark.goosman@xilinx.com

ザイリンクスの Virtex™ プラットフォーム ファミリの FPGA アーキテクチャでは、パーシャル リコンフィギュレーション（PR）というメソッドを使い、デザイン モジュールを動作中に交換することが可能です。このパワフルな機能により、基本デザインの動作中でも割り込みをかけることなく、複数のデザイン モジュールが 1 つのデバイス上のリソースをタイムシェアリングすることが可能です。

パーシャル リコンフィギュレーションとは、デバイスの他の部分が動作している間、FPGA のあらかじめ定義した部分だけを限定的に再構成できるデバイス コンフィギュレーション プロセスです。この機能は、特にデバイスが中断の許されないミッション クリティカルな環境で動作中に、一部のサブシステムだけを再定義する場合に有用です。

パーシャル リコンフィギュレーションを使うことにより、1 個の FPGA の機能性を大幅に高めることが可能で、デバイスの数とサイズの削減につながります。このテクノロジの重要なアプリケーションとしては、リコンフィギャブルな通信システムや暗号化システムが挙げられます。

Virtex のコンフィギュレーションの背景

パーシャル リコンフィギュレーションは、Virtex および Spartan™ ファミリの両方でサポートされています。本稿では、Virtex-II と Virtex-II Pro FPGA におけるパーシャル リコンフィギュレーションのインプリメントについてご紹介します。

Virtex FPGA 内のユーザー プログラマブルな機能は、すべて電源投入時にコンフィギュレーションする必要のある揮発性メモリセルによって制御されます。これらのメモリセルはコンフィギュレーション メモリと呼ばれ、ルックアップ テーブル（LUT）の等式やシグナル ルーティング（信号経路）、入出力ブロック（IOB）の電圧規格など、デザインのあらゆる面を定義付けします。

コンフィギュレーション メモリをプログラミングするため、コンフィギュレーション制御ロジックに対する命令とコンフィギュレーション メモリ用のデータがビットストリームの形で提供され、このビットストリームは JTAG、SelectMAP、シリアル、もしくは ICAP の各コンフィギュレーション インターフェイスを介してデバイスに送られます。

プログラミングした Virtex FPGA は、

図 1　マイクロプロセッサのコンテキスト スイッチングと FPGA のパーシャル リコンフィギュレーション間の解析

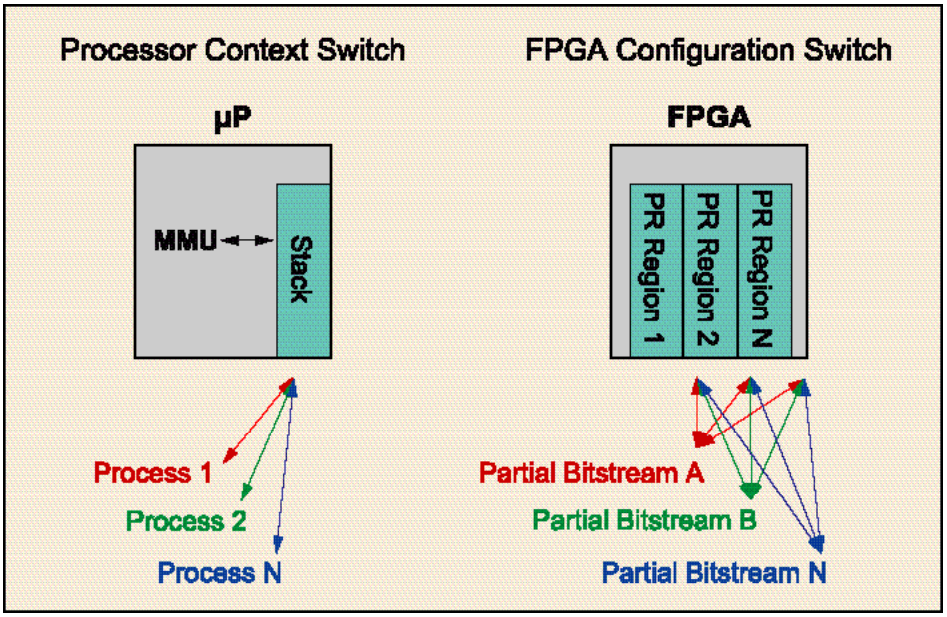
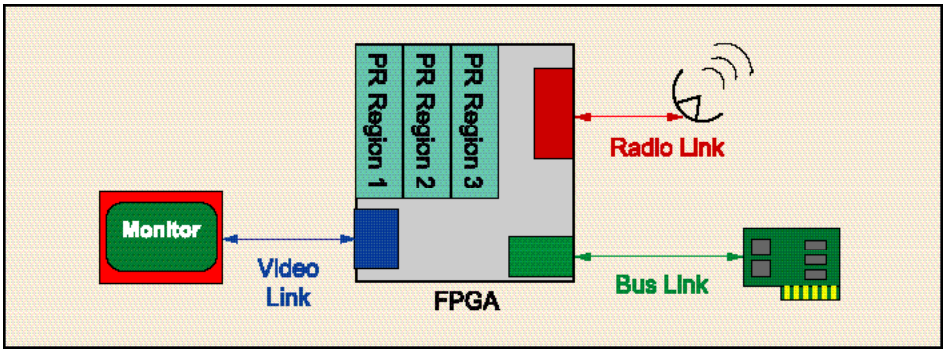


図 2　パーシャルリコンフィギュレーション中のリアルタイムリンクのメンテナンス



パーシャル ビットストリームを使って部分的なリコンフィギュレーションが可能です。パーシャル リコンフィギュレーションを使うと、FPGA デザインの残りの部分が動作中でも、そのデザインの一部の構造だけを変更できます。

使用例

パーシャル リコンフィギュレーションは、同じ FPGA デバイスのリソースをタイムシェアリングできる複数のファンクションを備えたシステムに便利です。FPGA の一部を動作させながら他の部分をディスエーブルにして部分的にリコンフィギュレーションを行い、新しい機能を実装することが可能です。これは、マイクロプロセッサがソフトウェア プロセス間のコンテキスト スイッチングを管理する図 1 の状況に似ています。FPGA のパーシャル リコンフィギュレーションを使う際に異なるのは、ソフトウェアでなくハードウェアがスイッチングされる点です。

フル リコンフィギュレーションでは不可能な継続的動作を必要とするアプリケーションでは、複数のフル ビットストリームを使うよりパーシャル リコンフィギュレーションを行うほうが便利です。図 2 の例は、水平同期と垂直同期を利用するグラフィックス ディスプレイを表しています。このアプリケーションが動作する環境上、ラジオとビデオのリンクから出力される信号を保持する必要がありますが、そのフォーマットとデータ処理フォーマットは動作中にアップデートして変更する必要があります。パーシャル リコンフィギュレーションを使うことにより、システムはこれらのリアルタイム リンクを維持しながら、FPGA 内の他のモジュールをいつでもすぐに変更することが可能になります。

方法

パーシャル リコンフィギュレーションのデザインを効果的にインプリメントするには、デザイン手法に厳密に従う必要があります。ガイドラインは次のとおりです。

- 取り外すモジュール（ パーシャル リコンフィギュレーション モジュール、略称 PRM ）とデザインの残りの部分（ スタティック ロジック）の間にバス マクロを挿入する。
- パーシャル リコンフィギュレーションが可能なネットリストを生成するための合成ガイドラインに従う。
- PRM のフロアプランを作成し、すべてのスタティック モジュールをクラスタ化する。
- バス マクロを配置する。
- パーシャル リコンフィギュレーション特有のデザイン ルールに従う。
- パーシャル リコンフィギュレーションのインプリメンテーション フローを実行する。

PlanAhead™ ソフトウェアは、上記ガイドラインを管理するための 1 つの環境（ プラットフォーム ）を提供しています。

PlanAhead のデザイン ツールを使用してパーシャル リコンフィギュレーションのデザインをインプリメントするステップは、次のとおりです。

- Netlist をインポートする。
- パーシャル リコンフィギュレーション用にフロアプランを作成する。
- デザイン ルールをチェックする。
- ネットリストをエクスポートする。
- インプリメンテーション フローを管理する。
- ビットストリームのサイズを見積もる。

ネットリストをインポートする

PlanAhead ソフトウェアには、XST や Synplify など、合成した任意のネットリストを利用できます。どのような階層ネットリ

ストでもインポートが可能です（ 1 個または複数の edf/ngc ファイル ）。通常のガイドラインに従って PlanAhead デザイン ツールにデザインをインポートし、そのデザインのフロアプランを作成してください。

パーシャル リコンフィギュレーション用にフロアプランを作成する

これは、パーシャル リコンフィギュレーション フローの中で重要なステップです。PlanAhead ソフトウェアでは、物理ブロック（ Pblock ）というデザイン パーティションに基づいてフロアプランを作成します。Pblock は、ロジックを制約するため、FPGA デバイスに定義されている長方形などのエリアを持つことができます。長方形以外で Pblock を定義すると、ISE™ ソフトウェアはそのロジックを配置中にグループ化しようとします。Pblock 内に配置したネットリスト ロジックは、AREA_GROUP の制約を受けることになります。

パーシャル リコンフィギュレーションのフロアプランを作成するための主なタスクは次のとおりです。

1. PRM をセットアップする。

- ファブリック内に定義されているエリアを使って Pblock を作成することで、PRM に対するエリアを割り当てる。
- Pblock に対して RANGES を割り当てる。
- PRM に MODE=RECONFIG 属性を定義する。
  （Pblock property->attribute section）

2. スタティック ロジックをセットアップする。

- PRM を除くすべてのトップレベル モジュールは、1 個の PBlock にグループ化する必要があります。これをスタティック ロジック ブロックといいます。このブロックには RANGE を定義してはいけません。そうすることによりスタティック ロジックは 1 個の Pblock にクラスタ化されます。PRM を除くトップレベル モジュールをすべて選択し、Pblock に割り当てます。図 3 は、スタティック ロジックを AG_base という Pblock にグループ化したところです。
- PlanAhead デザイン ツールでフロアプランの作成が完了したら、物理階層は図 4 のようになります。

3. バス マクロを配置する。

- バス マクロとは、スタティック ロジックに PRM を接続する物理ポートです。RPM からスタティック ロジックへの接続は、バス マクロを経由する必要があります。バス マクロは、RTL でブラック ボックスとしてインスタンシエートされ、「 .nmc 」ファイルとしてあらかじめ定義された配線マクロで満たされます。バス マクロは PRM の境界に配置されます。PRM に接続されたスタティック ロジックは、配置中にバス マクロのほうににマイグレート（ 移動 ）されます。

デザイン ルールをチェックする

フローの複雑さにより、ベースとなる RTL とフロアプランの作成中に間違いを犯すことがよくあります。そこで、PlanAhead の

図 3　すべてのスタティック ロジックを AG_base という 1 つの Pblock にグループ化

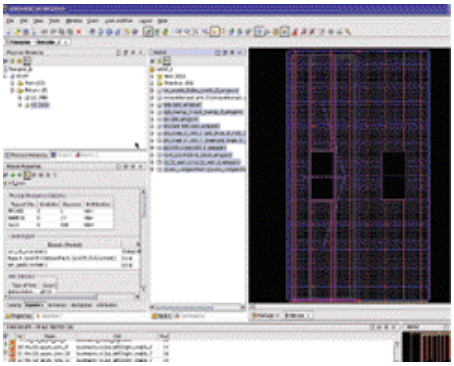


図 4　パーシャルリコンフィギュレーションのデザインの物理階層

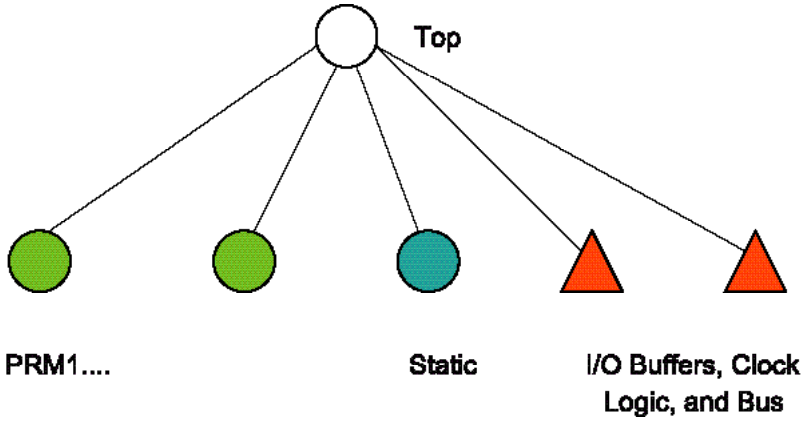
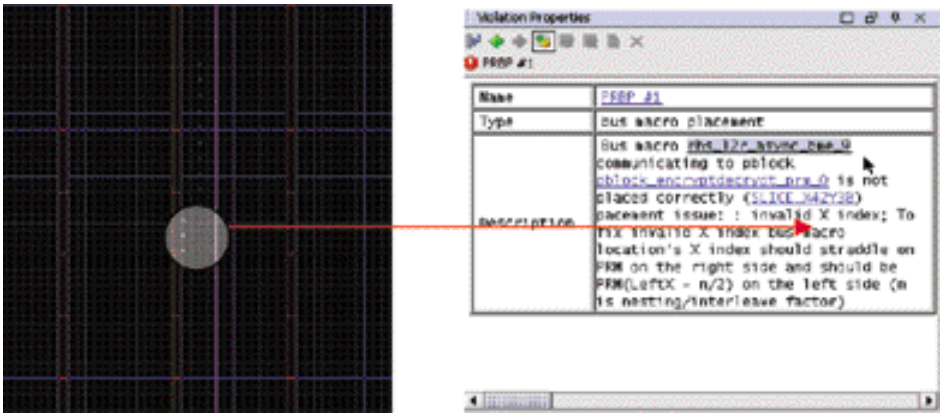


図 5　PRBP DRC でバス マクロの配置で従うべきすべてのルールを検証



PR デザイン ツールは、デザインの違反をチェックします。また、この機能には、デザインを改善する方法についてフィードバックを与える PRAdvisor も組み込まれています。

パーシャル リコンフィギュレーションのデザイン ルール チェックは次のとおりです。

1. バス マクロ DRC：
バス マクロの接続と配置に関するすべてのデザイン ルールを検証します。バス マクロ DRC の一例として PRBP チェックがあります。この DRC は、バス マクロの配置について従うべきすべてのルールをチェックします。図 5 は、PRBP DRC をパスしなかったデザインの例です。このケースでは、SLICE_X41Y* にインターリーブ/ネストされたマクロを配置する必要があります。

2. フロアプランニング DRC：
フロアプランニングのルールをチェックします。クロック オブジェクト（グローバル クロック バッファ、DCM ）と I/O を配置し、スタティック ロジックをクラスタ化する必要があります。

3. グリッチ ロジック DRC：
PRM リージョン上下のグリッチ ロジック エレメント（ SRL と分散 RAM ）検証します。

4. タイミング アドバイザ/DRC：
タイミング関連の問題をチェックします。タイミング DRC の一例として PRTP チェックがあります。インプリメンテーション中、PRM の前にスタティック モジュールがインプリメントされます。通常のタイミング制約は、スタティック モジュールと PRM が交差するパスをカバーしません。バス マクロが同期であれば問題ありませんが、非同期バス マクロの場合、スタティック モジュールは非同期パスの伝播を認識しないことになります（ 図 6 ）。これらのパスがタイミング クリティカルな場合、これは重大な問題です。この情報をスタティック モジュールへ渡すための 1 つの方法として、バス マクロ出力ネットに TPSYNC 制約を指定します。Plan Ahead ソフトウェアでは、「 .UCF 」ファイルに追加できる TPSYNC 制約の使用を推奨しています。

図 6　非同期パスを含むタイミング クリティカルなパスは特別な考察が必要

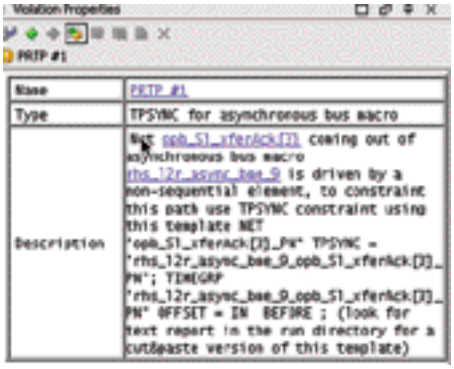


図 7　デザインのフロアプランを作成して DRC チェッカーにパスしたモジュールは、エクスポート フロアプランのダイヤログ ボックス中に黒文字で表示

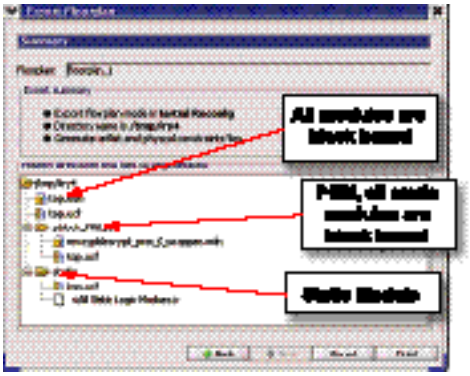
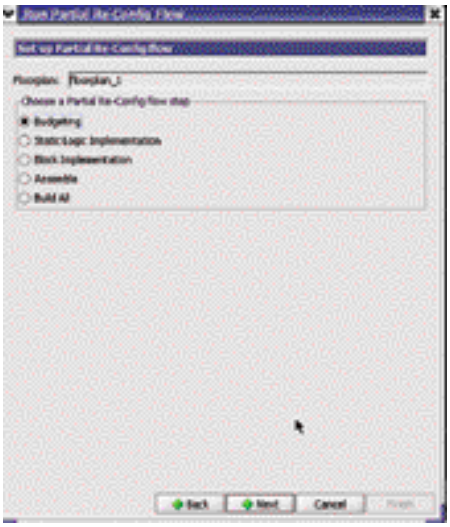


図 8　PlanAhead のパーシャル リコンフィギュレーション フロー ウィザードは扱いやすいフローを提供



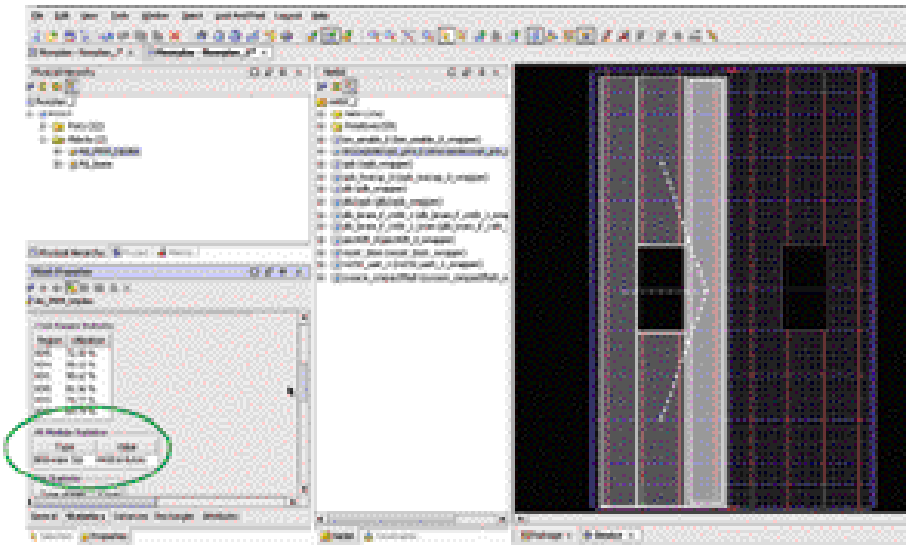
ネットリストをエクスポートする

デザインのフロアプランを作成して DRC チェッカーにパスしたら、いつでもエクスポートが可能です。PlanAhead デザイン ツールは、元の階層ネットリストを、特定のフォーマット（ スタティック モジュールと PRM は別々のディレクトリ ）を持つパーシャル リコンフィギュレーション型のネットリストにエクスポートします。図 7 のようなエクスポート ディレクトリが表示されます。

インプリメンテーション フローを管理する

パーシャル リコンフィギュレーション フロー ウィザード（ 図 8 ）がエクスポートしたデザインに、パーシャル リコンフィギュ

図 9　Pblock の統計レポートには、PRM ビットストリーム サイズを報告するセクションがある



レーションのインプリメンテーションを実行します。このウィザードは、デザイン全体に対してフル ビットストリームを、また、各 PRM に対してパーシャル ビットストリームを生成します。このインプリメンテーション ステップは次のとおりです。

- 最初のバジェット作成
- スタティック モジュールのインプリメンテーション
- PRM インプリメンテーション（全 PRM のそれぞれのバージョンごとに 1 回のインプリメンテーション）
- アセンブリとビットストリームの生成（結果はマージ ディレクトリに格納）

ツールを走らせるには、[Tools] → [Run Partial Reconfig] を選択し、フロー ウィザードを起動します。このウィザードに従い、各インプリメンテーション ステップを進めていきます。ここでインプリメンテーションを実行することも可能ですが、スクリプトだけ生成しておいて、後で PlanAhead ユーザー インターフェイス以外のコマンド ラインから使用することも可能です。

ビットストリームのサイズを見積もる

Pblock の統計レポートには、PRM ビットストリーム サイズを報告するセクションがあります（ 図 9 ）。この情報を使って、外付けのフラッシュ メモリや DDR など、コンフィギュレーション メモリのストレージ サイズを見積もることができます。また、この情報は、ビットストリーム メモリ インターフェイスに基づいたモジュールの交換にかかる時間を計算するためにも役立ちます。

結論

PlanAhead ソフトウェアは、パーシャル リコンフィギュレーションのための初のグラフィカル環境です。パーシャル リコンフィギュレーション アプリケーションのプラットフォームとして PlanAhead デザイン ツールを使うことにより、こうした最先端アプリケーションのダイナミックな動作環境に手を加える煩わしさは大幅に緩和され、以前なら複数の FPGA を必要としていたアプリケーションに 1 個のデバイスを利用するだけですむようになります。

PlanAhead ソフトウェアのメソッドは、パーシャル リコンフィギュレーションを使う設計者に生産性の大幅な向上とソリューションまでの時間短縮（ Time-to-Solution ）をもたらします。

デザインへのパーシャル リコンフィギュレーションは、あらゆるアプリケーションに役立ちます。Virtex-II と Virtex-II Pro FPGA をターゲットとしたデザインにパーシャル リコンフィギュレーションの利点を活用するプラットフォームを提供することで、PlanAhead デザイン ツールはデザインの機能性を劇的に拡充します。

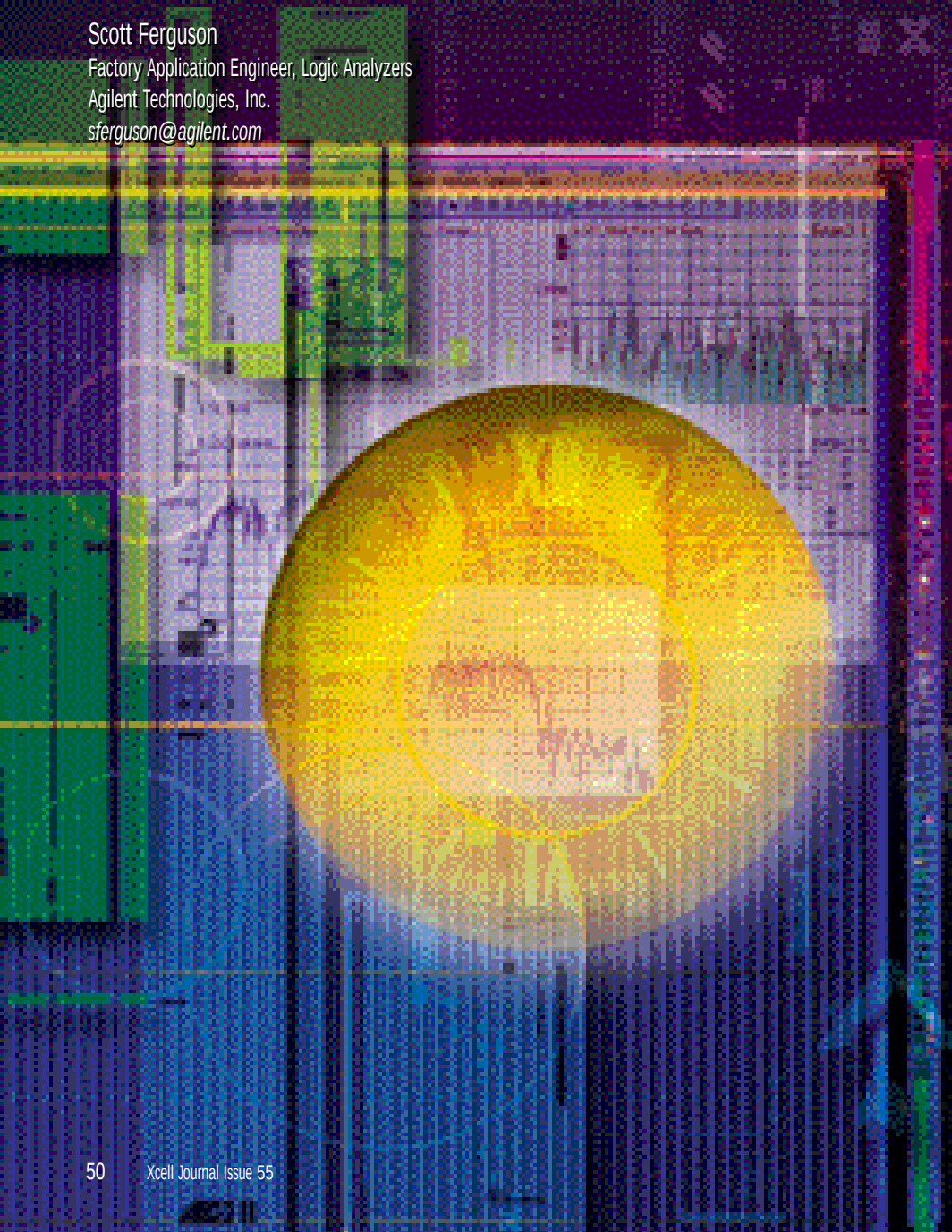
近い将来には、Virtex-4 マルチプラットフォーム FPGA をターゲットとしたデザインのサポートも予定しており、パーシャル リコンフィギュレーションの利用範囲は事実上無限に広がります。

Real-Time Analysis of DSP Designs

DSP デザインにおけるリアルタイム解析法

FPGA Dynamic Prob とデジタル VSAを結合するAgilent 社のツール

Scott Ferguson
Factory Application Engineer, Logic Analyzers
Agilent Technologies, Inc.
sferguson@agilent.com



FPGA が、携帯電話の基地局や衛星通信、レーダーといったデジタル通信機器など高性能な信号処理を実現するための現実的な選択肢になってきている状況下において、回路にできるだけ短時間で最適なパフォーマンスを達成するには、解析・デバッグ ツールに新たな手法を用意する必要があります。

シミュレーションと RF アナログ信号に接続する信号解析ツールはあるものの、FPGA のサブ回路における信号品質（ 周波数スペクトル、I-Q コンステレーション、エラー ベクタ マグニチュード（ EVM ））を測定できることが重要です。そこで、Agilent 社は、当社の 89601A Vector Signal Anal ysis（ VSA ）ソフトウェアとロジック アナライザ製品ライン（ 1680、1690、16900 ファミリ ）をリンクし、デジタル VSA ツールを作成しました。この VSA ツールをザイリンクスの ChipScope™ Pro Agilent Trace Core と併用することで、FPGA デザイン内のどこでも信号解析を素早く簡単に実行できます。

本稿では、このツールセットのしくみと、ザイリンクスをベースとする DSP 回路を最大限有効に活用するうえでどのように役立つかについて解説します。

デジタル VSA

VSA は、高速フーリエ変換（ FFT ）ベースのデータ処理を使って、時間および周波数ドメインのディスプレイと測定値を提供します。図 1 は、代表的な VSA ディスプレイです。このディスプレイは非常にフレキシブルでコンフィギャブルですが、メイン コンポーネントは、I-Q コンステレーション プロット（左上 ）、マグニチュード スペクトル（ 左下 ）、エラー ベクタ（ 右上 ）、および測定値（ 右下 ）です。 EVM は測定値のセクションに表示されます。この 1 個の値が、変調信号の品質の主要指標となります。

EVM は、キャプチャしたデータから I-Q シンボルを抽出することで計算されます。シンボルとは、QPSK や QAM などの変調方式により定義されるコンステレーション内のグリッド ポイントです。測定した信号から抽出されると、「 リファレンス 」信号という理想的（ 理論上は完璧 ）な信号を作成するために、このシンボル シーケンスが使われます。測定した信号はそれぞれリファレンス信号と比較されますが、その比較結果の差をエラー ベクタといいます（ エラーには I と Q、つまりマグニチュードとフェーズの両コンポーネントを含んでいます ）。1回のキャプチャにおける個々のエラー ベクタを組み合わせると 1 個の EVM 測定値となります。

本来、この解析ソフトウェアはアナログ RF 信号を解析する目的で作られましたが、ハードウェアに依存しない PC ベースのソフトウェア パッケージとして開発されました。Agilent 社のロジック アナライザも PC ベースですので、 VSA ソフトウェアを拡張してロジック アナライザにリンクさせるのは容易でした。

アナログ信号の代替として、デジタル ベースバンドと IF の各信号があります。これらの信号は最初からデジタルであるため、FFT 解析を行うために RF 信号アナライザなどの信号をデジタル化する装置を使う必要がありません。こうしたアナログ信号のデジタル バージョンを、オシロスコープのディスプレイに似たチャート形の波形としてロジック アナライザに表示することができます（ 図 2 ）。

この図からわかるとおり、バスが同期的にサンプリングされ、サンプル レートが Nyquist（ ナイキスト ）の定理を満たしていれば、ロジック アナライザは「 かつての 」アナログ信号や「 もうすぐになるであろう 」アナログ信号の十分に正確なデータをキャプチャします。

FPGA Dynamic Probe

FPGA Dynamic Probe を ChipScope Pro アナライザと連携させることで、再コ

図 3　FPGA Dynamic Probe の概要

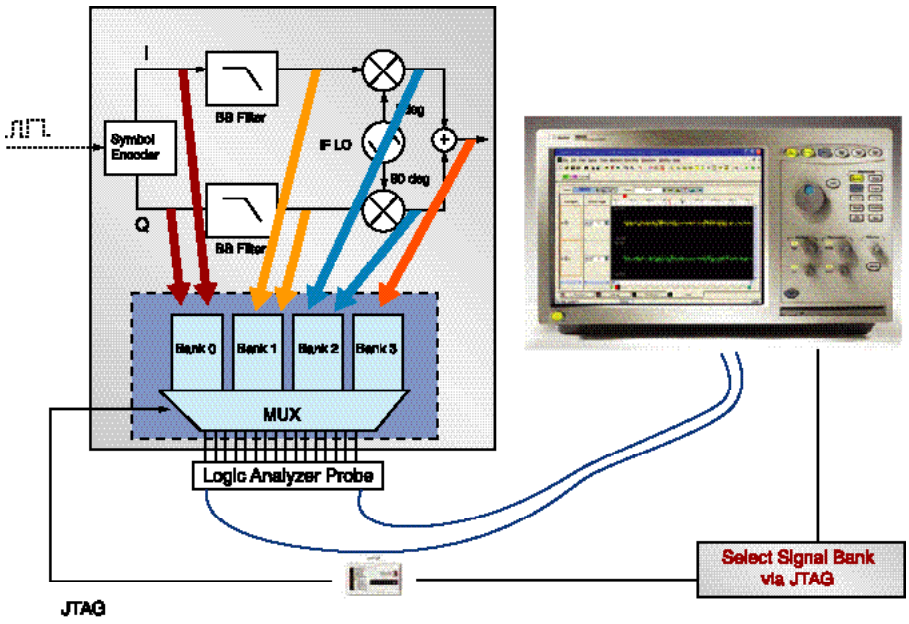


図 1　VSA の画面

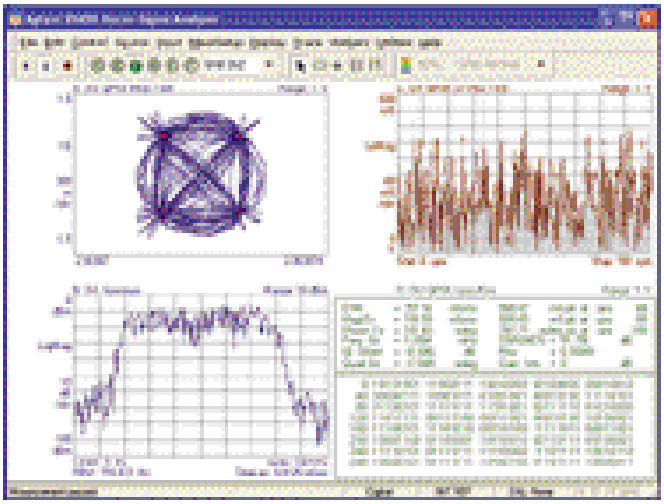
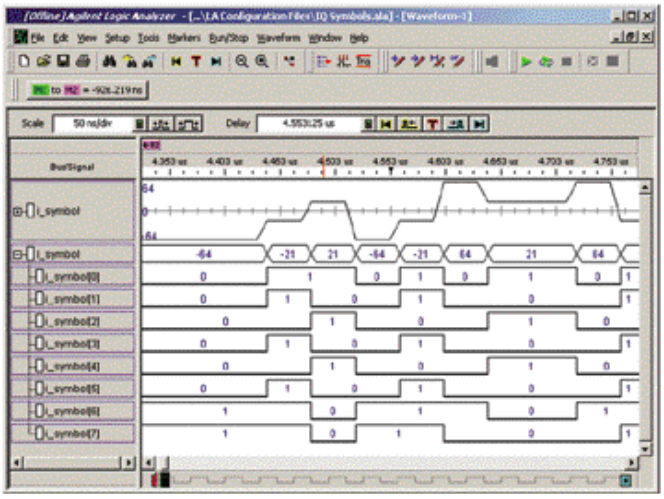


図 2　デジタル バスのチャート画面



ンパイルすることなく DSP デザインのどこにでもアクセスできます。図 3 は、デジタル ラジオトランスミッタの単純化したデザインを Agilent Trace Core 2（ATC2）に接続しているところです。このコアは、ChipScope Pro Core Inserter を使って、一般には合成後にデザインに取り込んだスイッチング マルチプレクサ（MUX）です。コアの挿入中、トレース コアに接続する内部ネットと、MUX 出力を接続する物理パッドを選択します。その後、これらのパッドを回路基板上でロジック アナライザ プローブに配線します。

ロジック アナライザは JTAG を通して FPGA を制御します（ビット ファイルをダウンロードしてバンクを選択）。新しいバンクを選択すると、ロジック アナライザはそれ自体を自動的に再構成し、プローブに接続されているネットの名前と一致させます。

デザイン サンプル - QAM 16 モジュレータ

Agilent 社は、ザイリンクスの FAE の協力を得て、Xilinx System Generator for DSP を使用して小型の Virtex™-II 部品（XC2V250-FG256）に収まるデモを作成しました。このツールを使うと、DSP デザインを素早く簡単に作成できます。このデザイン（図 3 のブロック図）には、25 MHz のシンボル エンコーダ、24 タップと 4X インターポレーションを備えるルートコサイン フィルタ（出力 100 MHz）、および 25 MHz ローカル オシレータを備える IF 変調ステージがあります。

System Generator デザインに ATC2 コアを統合

Agilent 社は、このデザインを VHDL にコンパイル後、ATC2 コアを挿入しました。ロジック アナライザのディスプレイに表示される信号名をわかりやすくするため、VHDL を少し手作業で編集しました（System Generator 上でユニークなネット名を使用すればこのステップを回避できるはずです）。次に、ネット名をロジック アナライザの画面に収まる長さにするため、関係のあるネットのほとんどをトップレベル オブジェクトからの出力ポートとして接続しました。

FPGA Dynamic Probe で使うためだけにネットを出力ポートに接続する場合、VHDL で「keep」属性を使うと便利です。合成が終わるまではデザインに ATC2 コアの追加は行われないため、多くのネットは何の接続もされず、最適化は行われません。VHDL で keep 属性を使うためのシンタックスは次のとおりです。

```
attribute keep : string;
attribute keep of i_symbol:
signal is “true”;
attribute keep of q_symbol:
signal is “true”;
```

この例では、48 個の信号を持つ 4 つのバンクを ATC2 コアに入れました。ATC2 コアの 2X TDM オプション（各パッド上で一度に2 つの信号をタイムスライス）を使うと、コア用に必要な FPGA のパッケージ パッドは 25 個のみです（クロックに 1 個、データに 24 個）。これで 192 個の信号にアクセスできることになります。実際には約 92 個の信号を見るだけで十分です。

- I-Q シンボル、各 8 ビット（16）
- I-Q フィルタ出力、各24 ビット（48）
- IF ローカル オシレータのサインとコサイン、各 2 ビット（4）
- 組み合わせた IF 信号（24）

24 ビットの I 信号と Q 信号を持つ RRC フィルタの出力は、ピンの所要数を定義する最大の要件でした。24 個のピンを利用できない場合、最下位ビットを捨てることでダイナミック レンジを失うことがありますが、それでも信号を見ることはできます。

時間ドメイン、ロジック、VSA 測定値

ロジック アナライザは、ATC2 コアの出力をキャプチャするため、同期サンプリング（「ステート モード」）を使います。つまり、データは ATC2 の出力クロックの各エッジでサンプリングされるわけです。このデザイン例では、RRC フィルタ前のシンボル

図 4　フィルタ内のグループ遅延の測定

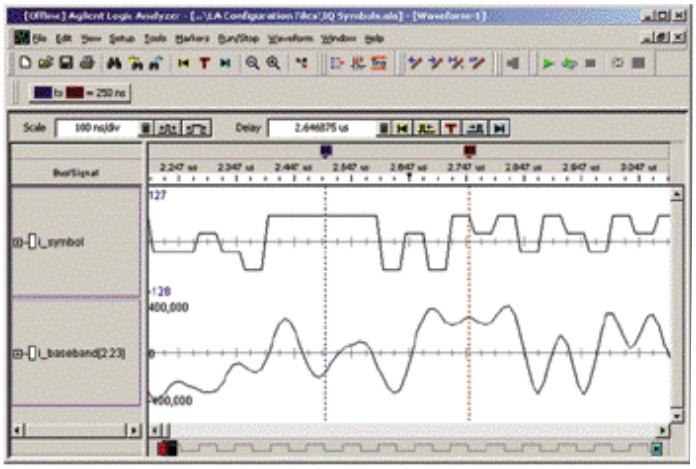
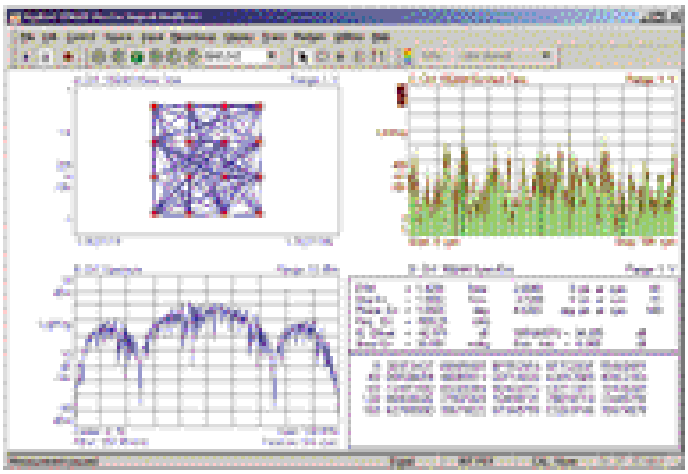


図 5　フィルタリング前の QAM16 I-Q シンボル



データに 25 MHz、フィルタ後のすべての部分に 100 MHz と、回路内に 2 つのクロック レートを使用しています。ATC2 コアはコアあたり 1 個のクロックしかサポートしないため、デバッグには 2 つのオプションがあります。

- 各クロック レートに 1 つずつ、2 つのコアを使用
- より高速なクロック レートの 1 コアを使い、25 MHz バスをオーバーサンプリング

2 つのクロックには相関関係があり、一方はもう一方の整数倍ですので、遅い方のバスをオーバーサンプリングするだけで十分です。オーバーサンプリングが望ましくない場合、ロジック アナライザはサンプルを 4 つおきに格納するセットアップを使うことで、25 MHz クロックあたり 1 つのサンプルで 25 MHz バスを正確にキャプチャすることが可能です。

この例では、MUX で利用できる追加の信号を使い、対象となる信号のいくつかをダブルプローブすることができました。たとえば、バンク 0 にはフィルタの前の I および Q のシンボル、また、RRC フィルタの後に I コンポーネントがあります。つまり、図 4 に示すように、フィルタ内のグループ遅延を測定するため、ロジック アナライザで時間ドメインの解析を行えるということです。2 つのマーカーはワイドで上部が平坦であるという信号の共通の特徴を表し、マーカー測定ディスプレイは 250 ns の間隔を示しています。

回路の測定部分にプローブした後、信号にベクタ信号解析を行い、RRC フィルタと IF 変調の各ステージの品質を測定しました。

図 5 に示すように、フィルタリング前の QAM16 I-Q シンボルを見ると、16 ポイントの QAM コンステレーションがあることがわかります（左上のグラフ）。

1 シンボルあたり 1 つのポイントを示すので、コンステレーション ポイント間のラインは直線になります。周波数スペクトラム（左下のグラフ）は 0 Hz にセンタリングされており、パスバンドは 25 MHz で、隣接するチャネルの信号が干渉しています。もちろん、RF 信号では隣接チャネル間で信号が干渉されることは望ましくなく、それがベースバンド フィルタを使う理由です。

ATC2 コアに別のバンクを選択することで（これはロジック アナライザで制御されます）、図 6 に示すように、ベースバンド フィルタの後の IQ 信号の解析を行うことができます。これでスペクトルのサイドバンドは除去され、測定値のディスプレイ（右下の四分円）は 0.5 % の EVM を示しています。今度 RF チームがベースバンド デザインにおけるエラーについて不満を言ってきたら、彼らが熟知しているこの測定値を指摘して、フィルタの責任ではないことを証明すればいいでしょう。

多くのデジタル ラジオ デザインでは、この IQ 信号はアナログに変換されます。とはいえ、ここでは同じ FPGA 内で IF 変調をデジタルで行いました。図 7 に示すように、FPGA Dynamic Probe でバンクをスイッチングすることでデジタル IF にアクセスできます（この場合も追加の合成および配置・配線ステップは不要）。スペクトルと I-Q コンステレーションは、現在 25 MHz にセンタリングされている以外、ほぼ同じです。EVM は若干高いので、これを低くするためにより高品質なローカル オシレータか別のフィルタ ステージを使うこともできるでしょう。

結論

ザイリンクスの System Generator と ChipScope Pro アナライザを Agilent 社のロジック アナライザおよび Agilent VSA ソフトウェアと併用することで、ザイリンクスの FPGA 内でデジタル ベースバンドと IF 信号をリアルタイムに詳しく解析できます。これは時間の節約になるだけでなく、シミュレーションと実際のハードウェア間の違いに関する疑念を払拭してくれます。また、RF 設計チームの同僚と意志の疎通を図り、相互理解を深められるうえ、彼らの言葉で話すことができ、信号のフォーマット（アナログ、デジタル、ベースバンド、RF）にかかわらず同じ解析ソフトウェアを使用できます。

これらアプリケーションの詳細は、http://www.agilent.com/find/logic-sw-apps/ をご覧ください。

図 6　ベースバンド フィルタの後の IQ 信号の解析

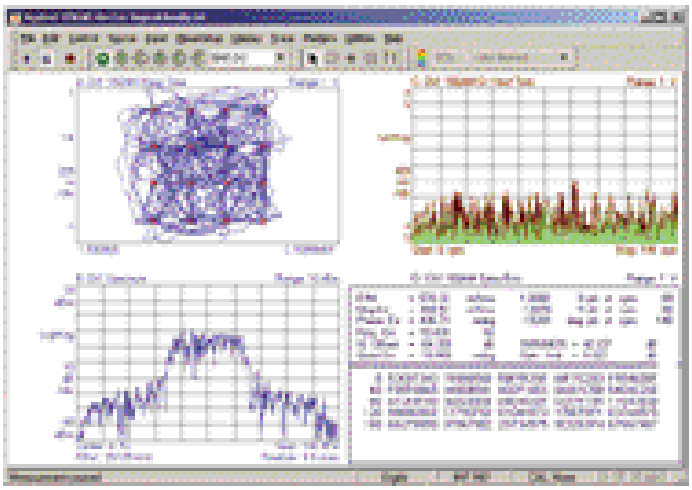
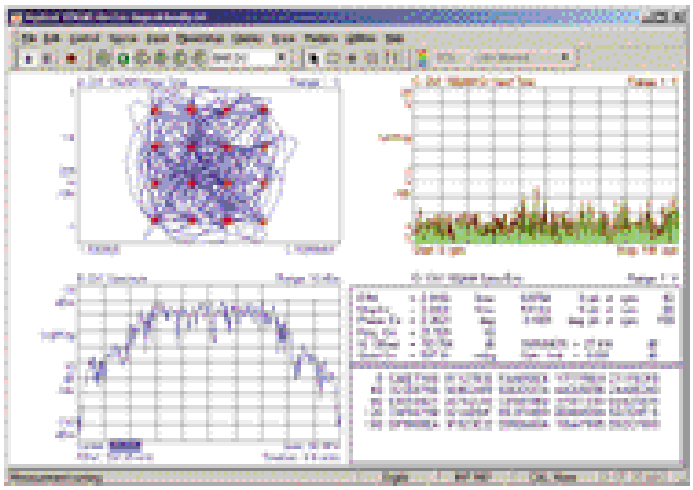


図 7　デジタル IF 信号



Program SPI Serial Flash from Xilinx FPGAs and CPLDs

ザイリンクスの FPGA と CPLD による SPI シリアル フラッシュのプログラミング

簡単にできる現在利用可能なツールの使用法

Rick Folea
CTO
Ricreations, Inc.
rfolea@UniversalScan.com

ザイリンクスの Spartan™-3E デバイスでは、同社の FPGA をコンフィギュレーションに業界標準の SPI シリアル フラッシュを使えるようになりました。さらに、ザイリンクスが今後発表するデバイスもすべて SPI シリアル フラッシュに対応する予定で、すなわち、すぐに使える安価なメモリを複数のベンダから選択できるようになりました。

ただし、ザイリンクスは同社のプログラミング ツールでこれらのデバイスをサポートする予定はなく、設計者は自分でプログラミングする必要があります。本稿では、SPI フラッシュをプログラミングする際の設計者の選択肢を紹介し、迅速に作業を進めるための方法について解説します。

SPI フラッシュの特長

これまで、ザイリンクスのデバイスをコンフィギュレーションする際に使うメモリは、ザイリンクスがサポートする特定のメモリのみでした。これは、1 バイトあたりのストレージが割高なうえ、デバイスの入手先と在庫に苦労することもあります。

SPI フラッシュに移行することで、メモリ ベンダの選択肢が増え、メモリの密度と種類を幅広く選択できます。中でも最も重要なメリットは、コストの削減とアベイラビリティ（ 入手のしやすさ ）でしょう。表 1 は、SPI フラッシュのベンダと、ザイリンクスの FPGA がサポートする SPI フラッシュの互換デバイスの一覧です。

SPI フラッシュのバリエーション

SPI フラッシュは 3 種類の一般的なカテゴリに分類できます。コード ストレージ用の SPI シリアル フラッシュ、データ ストレージ用の SPI シリアル フラッシュ、および Atmel 社の DataFlash です。

DataFlash デバイスは、Atmel 社の実績あるシリアル フラッシュ AT45DB というパーツです。これらのデバイスは、技術的に成熟しており数も豊富で、最高 128 MB の集積度を持ちます。ただ、プログラミングと消去アルゴリズムが SPI フラッシュ アルゴリズムに準拠せず若干使いにくいうえ、ベンダの選択肢もあまり多くはありません。したがって、どうしても 128 MB のストレージが必要でない限り、比較的最近の SPI フラッシュ メモリに目が向くでしょう。

図 1 に、コードとデータ用の 2 種類の SPI メモリを示します。いずれも基本的には同じですが、データ用メモリは、先にデータを消去することなくメモリに書き込めるインストラクションなど、いくつか追加のインストラクションを備えています。このインストラクションは、アプリケーションから素早くデータを読み出し/書き込みするときに便利です。

しかしながら、コード用メモリはベンダ、デバイスの種類とも選択肢がはるかに広いため、データ用メモリの特殊な機能を本当に必要とする場合を除き、当面はコード用メモリのプログラミング機能が使用されるでしょう。データ用メモリはコード用メモリのスーパーセット（ 上位互換デバイス ）ですので、データ用メモリを使っても問題はありませんが、選択肢をオープンにしておく場合は、データ用メモリの特殊なコマンド セットに

図 1　各ベンダ提供の各種 SPI Flash メモリ。青＝利用可能、黄色＝計画中、水色＝利用不可、「 Series 」項目中の太字＝ザイリンクスのラボでテスト済み

Vendor	Series	Type	1M	2M	4M	8M	16M	32M	64M	128M
ST	**M25P**	Code								
ST	M2/45PE	Data								
ATMEL	**AT45DB**	Data								
ATMEL	AT25F	Code								
NexFlash	**NX25P**	Code								
SST	**SST25VF**	Code								
SST	**SST25LF**	Code								
AMD/Fujitsu	S25FL	Code								
PMC	**Pm25LV**	Code								
SAIFUN	SA25F	Code								
SAIFUN	SA25C	Code								
Sanyo	LE25FW	Code								
MXIC	MX25L	Code								

依存するのは避けたほうが賢明でしょう。

SPI フラッシュ メモリの注意点

SPI フラッシュ メモリはどのベンダでもほぼ同じですが、いくつか注意点があります。最近のデバイスの多くは、ポーリングすることでデバイスを識別できる「 ReadID 」命令をサポートしています。データシートに記載の詳細な注意事項を必ず参照してください。たとえば、M25P シリーズの一部デバイスの ST データシートには、ReadID 命令が記載されていますが、注意書きには、この機能をサポートするのはパーツ ナンバーの末尾が「 X 」で終わるデバイスのみと記載されています。ところが、いろいろ探してもそうしたデバイスは見つかりませんでした。

また、どのベンダのデバイスも読み出し/消去/プログラムの各インストラクションをサポートしますが、それぞれ使用するコマンド セットは異なります。どのコマンドもやることは同じですが、コマンドごとに使うビットが異なります（ 例 : ST デバイスのページ プログラムは 0xD8 ですが、ATMEL デバイスのページ プログラムは 0x52 ）。詳しい注意点は、次の各インストラクションを参照してください。

SPI フラッシュ メモリのプログラミング

基本的なインストラクションを使い、前述の注意点にさえ留意すれば、小さな専用の SPI フラッシュ プログラマを自分で作成するのはとても簡単です。すべてのデータとコマンドは、最初に最上位ビット（ MSB ）からシフトインされます。

デバイスの消去は次の方法で簡単に行えます。

- チップ イネーブルを Low にする
- 8 ビットの消去コマンドをシリアルにシフトインする
- チップ イネーブルを High にする

後は、ステータス レジスタをチェックして消去サイクルが終了したことを確認するか、予想される最大消去時間（ ベンダとデバイスの密度に応じて 1 ～数百秒 ）待機します。

メモリ アレイからデータを読み出すには、次のように行います。

- チップ イネーブルを Low にする
- 8 ビットの ReadData コマンドをシリアルにシフトインする
- アドレスをシリアルにシフトインする（ ほとんどのデバイスで 24 ビット、最初に MSB ）
- クロックの発行を続け、メモリのシリアル出力からデータを取り込む
- 必要なバイト数をシフトアウトし終えたら、チップ イネーブルを High にする

ほとんどのデバイスでは、クロックを入力し続けるとデバイスの終わりまで続行し、終わりまでくると最初のアドレスに戻り、続けて使用できます。

メモリ アレイにデータを書き込むには、次のように行います。

- チップ イネーブルを Low にする
- WriteEnable コマンドをシリアルにシフトインする
- チップ イネーブルを High にする
- チップ イネーブルを Low にする
- 8 ビットの PageProgram コマンドをシリアルにシフトインする
- アドレスをシリアルにシフトインする（ ほとんどのデバイスで 24 ビット、最初に MSB ）
- 最初に MSB から 8 ビット データをシフトインする（ ほとんどのデバイスでは一度に 256 バイトまでクロック可能 ）
- チップ イネーブルを High にする
- 書き込みサイクルの完了を確認するには、ステータス レジスタを監視するか、最大時間待機する（ 通常は数ミリ秒 ）
- 次のページに同じことを繰り返す（ 通常は 256 バイト ）

ページの境界を超えると、データがラップしてページの下部に上書きされ、次のページに進むことはありません。

プログラミングを始めるにあたって知っておくべきことは以上です。専用アプリケーション向けに小さなプログラマを作成し、基本的なコマンド セットの使用さえ守れば、SPI フラッシュ メモリをプログラミングするのは

とても簡単です。

今日利用できるツール

独自のプログラマの作成は、SPI ポートと、メモリに追加ファンクション用のスペースがあるマイクロプロセッサで問題なく行えます。では、小型の CPLD や FPGA に接続されているメモリをプログラミングする場合はどうでしょう。安価でシンプルな汎用プログラマがいかに少ないか、意外に思うかもしれません。今日、マイクロプロセッサ以外のアプリケーションに使える選択肢は、次のカテゴリに分類されます。

専用の FPGA コンフィギュレーション

コマンド セットが小さくシンプルなことから、一部の設計者は SPI フラッシュのプログラミングに使える専用の FPGA コンフィギュレーションを作成するという方法に頼ってきました。

Avnet社（旧 Memec 社）は、この方法をさらに進化させています。同社のコンフィギュレーションは、MicroBlaze™ ソフト コア プロセッサとオンチップ ペリフェラル バス（OPB）SPI コアを使います。同社は、単純に FPGA 内のプロセッサを使って、SPI メモリを読み書きするアプリケーションを作成しました。2005 年の春、同社は SPI フラッシュを次の目的で使うための方法を実証しました。

- FPGA をコンフィギュレーションする（Spartan-3E デバイス）
- MicroBlaze プロセッサをブートする
- 不揮発性データ ストレージに使う

デモや詳しい情報については、Avnet 社の FAE までお問い合わせください。

これらのメソッドを使うと SPI フラッシュに非常に高速にアクセスできますが、デバイスにコンフィギュレーションを収容するための十分なスペースが必要です。CPLD などのコンフィギュレーションができないデバイスでは使用できません。

X-SPI ユーティリティ

ザイリンクスの Web サイトには「X_SPI」というユーティリティがあります。ボードに追加のヘッダを置き、SPI バスに接続されているすべてのデバイスを 3 ステートにできれば、このコマンド ライン ユーティリティを使い、ザイリンクスの Parallel-III または Parallel-IV ケーブルで SPI フラッシュを直接プログラミングできます。プログラミングの方法と、CoolRunner™ CPLD を使った SPI の信号をザイリンクスの FPGA（Spartan-3E 以前のデバイス）が必要とする信号に変換する方法については、XAPP 800「Configuring Xilinx FPGAs with SPI フラッシュ Memories Using Cool Runner-II CPLDs」に解説されています。この方法では、ボードに別途のヘッダを追加しなければならないこと、SPI バス上のすべてのデバイスを 3 ステートにできること、使いたい SPI フラッシュ デバイスをこのユーティティがサポートしてなければならないという、3 つの条件が揃っている必要があります。

よりよい方法 - JTAG

XAPP800 は JTAG についても触れていますが、当時は JTAG を通して SPI フラッシュをプログラミングするためのシンプルで安価な汎用ツールはありませんでした。しかし、現在は Universal Scan の最新リリースが SPI フラッシュのプログラミングをサポートしています。Ricreations 社は無償の試用版を用意しており、また、価格を据え置きながら、このハードウェア デバッグ/メモリ プログラミング ツールに SPI プログラミング機能を追加しました。

ザイリンクスのデバイスで Boundary Scan チェーンを使って SPI フラッシュをプログラミングするのは簡単で、SPI デバイスに接続されているピンを指定し、データ ファイルを選んで「program」をクリックするだけです。Boundary Scan でのテストとプログラミングに伴うテスト ベクタやテスト エグゼクティブ、CAD データなどはいっさい不要です。ザイリンクスのデバイス上のピンは EXTEST コマンドの状態になります。ベクタは SPI フラッシュ メモリ ピンを駆動するため自動的にフォーマット/シフトインされ、気付いたらメモリがプログラミングされているという簡単さです。これには通常の消去、プログラム、検証の各機能に加え、メモリ ビューア、各種ユーティリティ、セクタ消去機能、そして自動デバイス ID インテロゲータも含まれています。

このメソッドの優れたところは、FPGA、CPLD、マイクロプロセッサ、Ethernet スイッチ、DSP など、JTAG ポートさえ備えていればどのようなデバイスでも使えることです。デバイスをコンフィギュレーションしたりプログラミングしたりする必要はなく、完全に正しく動作する必要さえありません。さらに、FPGA や CPLD に追加の貴重なデバイス リソースを必要とせず、特別なコードやコンフィギュレーションが不要なうえ、回路基板に特別なヘッダやその他のサポートデバイスも必要ありません。

1 つ難点は、クロック エッジ、データ セットアップ、チップ イネーブルのたびに JTAG チェーンにフルスキャンのベクタをシフトしなければならないため、他のメソッドよりかなり遅くなることです。たとえば、2,000 ビットのスキャン チェーンを持つ FPGA があるとします。シリアル データを SPI フラッシュにセットアップするのに 1 スキャン、クロックを上げるのに 1 スキャン、クロックを下げるのにさらにもう 1 スキャンかかります。SPI フラッシュ メモリに 1 バイトをシフト

図 2　素早く安く簡単に一般向けの SPI Flash プログラミングを行うための各種オプション

Method	Use on FPGAS	Use on CPLDs	Use on JTAG Device	Use on Purpose	Use on HW Reqd.	Use on Vendors	Use on Time
Specia Configuration	Some	No	No	No	No	Yes	Fast
X-SPI	N.A.	N.A.	N.A.	Yes	No	Some	Medium
JTAG	Yes	Yes	Yes	Yes	No	Yes	Slow or Medium
Miscel	No	No	N.A.	Yes	Yes	No	Medium

するのに 8 セット必要ですので、フラッシュ デバイスに 8ビット入れるには 48,000 クロックとなります。これにプログラミングするバイト数を掛けるわけですから、時間がかかるのは当然です。基本的に数百キロヘルツの TCK レートに限られるパラレル ポートを使う場合はなおさらです。

プログラミングの時間を高速化する場合は、これらの製品の新しい USB 2.0 バージョンを選択してください。新しいバージョンでは TCK レートが数桁増加しています。また、スピードを改善するために SPI デバイスが短いスキャン チェーンのデバイスに接続されおり、他のデバイスはすべてバイパス モードに入れてあることも必要です。

従来の JTAG ツールの中にも SPI フラッシュをプログラミングするものはありますので、入手可能であればぜひ活用してください。そうしたツールは非常にパワフルで、メモリを素早くプログラミングできます。唯一の欠点は、本稿で解説したメソッドより多くのセットアップを必要とし、高価になりがちな点ですが、スピードが必要な場合は、従来の Boundary Scan ツールはよい選択肢となります（ JTAG ツールのベンダについては別表を参照 ）。

その他のメソッド

ほとんどの SPI フラッシュ ベンダは、自社のデバイスをプログラミングできる一種のユーティリティを提供していますが、通常は自社デバイス専用であり、そのデバイスに直接接続する必要があったり、別のユーティリティ セットに追加するためコストの負担が増えたりします。

結論

図 2 に、これまで取り上げてきた選択肢をまとめます。スピード重視の場合は自分でデザインしてみてください。ボードの変更やヘッダの追加が苦にならないようであれば、無償の X-SPI ユーティリティを使ってみてください。あるいは、SPI フラッシュをプログラミングが可能で、使いやすく安価な新しい JTAG ツールを探すのも方法です。

SPI フラッシュ デバイスの普及やメモリの高密度化、使いやすさなどから、SPI フラッシュは今後のデザインに理想的なコンフィギュレーション ソリューション、もしくはデータ ストレージ ソリューションになり得ます。

Universal Scan についての詳細は、http://www.universalscan.com/ をご覧ください。

JTAG ツール ベンダ
ベンダ名（ URL ）

JTAG Technologies	(http://www.JTAG.com./)
Acculogic	(http://www.acculogic.com/)
Corelis	(http://www.corelis.com/)
Goepel	(http://www.goepel.com/)
Assett-Intertech	(http://www.assett-intertech.com/)
Intellitech	(http://www.intellitech.com/)
Flynn	(http://www.flynn.com/)
Universal Scan	(http://www.universalscan.com/)

ザイリンクス イベント カレンダー

1～3月

ザイリンクスは、年間を通じて多数のトレードショーやイベントに参加しています。これらのイベントは、ザイリンクスのシリコンやソフトウェアの専門家がお客様からの質問にお答えしたり、最新製品やザイリンクスのカスタマのサクセスストーリをご紹介する機会です。

ザイリンクス主催イベント

PCI Express セミナ

2 月 9 日(木)　東京開催

会　場：東京コンファレンスセンター 品川

2 月 14 日(火)　大阪開催

会　場：ニッセイ新大阪ビル

http://www.xilinx.co.jp/pcie_seminar/

展示会

1 月 18 日(水)～ 20 日(金)

リード エグジビション ジャパン主催

プリント配線版EXPO

アンソフト社ブースにて共同出展

◇会　場：東京ビッグサイト
◇U R L：http://www.pwb.jp/

1 月 26 日(木)～ 27 日(金)

JEITA 主催

Electronic Design and Solution Fair

菱洋エレクトロ株式会社と共同出展
日本セロックシカ社ブースにてプレゼンテーション

◇会　場：パシフィコ横浜
◇U R L：http://www.edsfair.com/

ザイリンクス ウェブ セミナ

◇好評配信中

第1回　Virtex™- 4 シグナル インテグリティ

高速インターフェイスをご使用になるエンジニアおよびシグナル インテグリティでお困りの方々を対象に、その理論と Virtex-4 のシグナル インテグリティ特性を競合デバイスとの実測比較データを使いながらご紹介いたします。
所要時間：約 25分

第2回　Virtex™- 4 ローパワー・アドバンテージ

システム設計時における消費電力の低減の重要性と、Virtex-4 マルチプラットフォーム FPGA の低消費電力の優位性とテクノロジについてご紹介いたします。また、競合デバイスとの比較により具体的に何が優れているかを実測データを使用して解説いたします。
所要時間：約 25分

第3回　Virtex™- 4 メモリインターフェイス・アドバンテージ

パワフルな高速メモリインターフェイスの構築を容易にする Virtex-4 は、すべてのプラットフォームにコンフィギャブルなハイパフォーマンス SelectIO™ テクノロジが搭載され、多様な I/O 規格に対応しています。本セミナでは、DDR2 を中心に Virtex-4 を用いたメモリインターフェイス設計の利点や関連する開発ボード、リファレンス デザインをご紹介いたします。
所要時間：約 25 分

第4回　FPGA を用いたプロセッサ設計の特徴と手法

Virtex-4 はハードコアである PPC を用いた高速処理に加え、ソフトコアである MicroBlaze を用いて、システムに最適なパフォーマンスと低価格を提供します。本セミナではこの特徴と設計手法に関して COMPUTEX 社の協力をいただき最適なデバック手法までを説明します。
所要時間：約 35 分

1 月 24 日(火) 配信開始

第5回　ローコスト Spartan™-3E FPGA コンフィギュレーション オプション

本セミナでは、ザイリンクス Spartan-3E FPGA のご紹介とコストなコンフィギュレーションオプション、利用できるオプションの種類、そしてユーザーシステムに最適なソリューション選択方法について説明します。これにより、システム開発コスト、PCB コスト、検証コスト、および最終的にシステムを構築するまでに必要なコストなどを最適化できます。
所要時間：約 35 分

http://www.xilinx.co.jp/webseminar/

XILINX®

PCI Express セミナ開催のご案内

参加無料
ランチ付

ザイリンクスの大きな市場である組み込み市場の中でも、急速な成長が見込まれる PCI Express の各種ソリューションをご紹介するセミナを開催いたします。ザイリンクスのパートナとともに、シリコンデバイス、IP コア、開発ボード、計測・デバッグのノウハウ、基板設計のヒントなどをご紹介いたします。

開催要項

東　京

期　日：2006 年 2 月 9 日（木）
会　場：東京コンファレンスセンター 品川
〒 108-0075
東京都港区港南 1-9-36 アレア品川 5 F
JR「品川」駅港南口より徒歩 2 分
定　員：200名様

大　阪

期　日：2006 年 2 月 14 日（火）
会　場：ニッセイ新大阪ビル　13 階
〒 532-0003
大阪市淀川区宮原 3-4-30
JR「新大阪」駅徒歩 7 分
定　員：50 名様

参加無料／ランチ付／事前登録制
主催：ザイリンクス株式会社

展　示

- アジレント・テクノロジー株式会社
- アンソフト・ジャパン株式会社
- 株式会社沖情報システムズ
- Genesys logic Inc.／株式会社スピナカー・システムズ
- 東京エレクトロンデバイス株式会社
- 株式会社東陽テクニカ
- PLD Applications／菱洋エレクトロ株式会社
- 株式会社ミッシュインターナショナル

セミナ プログラム（両会場共通）

受付は9:30より開始いたします。

10:00	**ザイリンクス PCI Express ソリューションの概要とロードマップのご紹介** Xilinx Inc. Connectivity Solutions Sr. Marketing Manager, Abhijit Athvale
11:20	ご休憩（展示をご覧ください）
11:30	**GENESYS Logic 社 PCI-Express Application とシステムデザインについて** Genesys Logic Inc. Architect and Director of Marketing, Jerry Chen
12:00	**PCI ExpressPHYトランシーバー PX1011A のご紹介** Philips Semiconductors Interface Products事業部 Marketing Manager, Ho Wai Wong-Lam
12:30	ご昼食（展示をご覧ください）
13:30	**PCI Expressの評価とコンプライアンス・テスト** アジレント・テクノロジー株式会社 電子計測本部 アプリケーションエンジニアリング部 荒井 信隆／重見 尚志
14:50	ご休憩（展示をご覧ください）
15:20	**ギガビット伝送を実現するインターコネクト設計技術** アンソフト・ジャパン株式会社 SIプロダクトグループ マーケティングマネージャー　渥辺 亨
15:40	**東京エレクトロンデバイス オリジナル PCI-Express 基板作成時の注意点** 東京エレクトロンデバイス株式会社 設計開発センター 主任　小田島 大介
16:30	ご休憩（展示をご覧ください）
17:40	**PCI Express Solution（高速転送能力の活かし方）** 株式会社沖情報システムズ 東京オフィス支店長　門田 晴信
17:00	**PLDA社 PCI Express Solution のご紹介** PLD Applications Senior Designer Philippe Legros
17:30	**質疑応答**

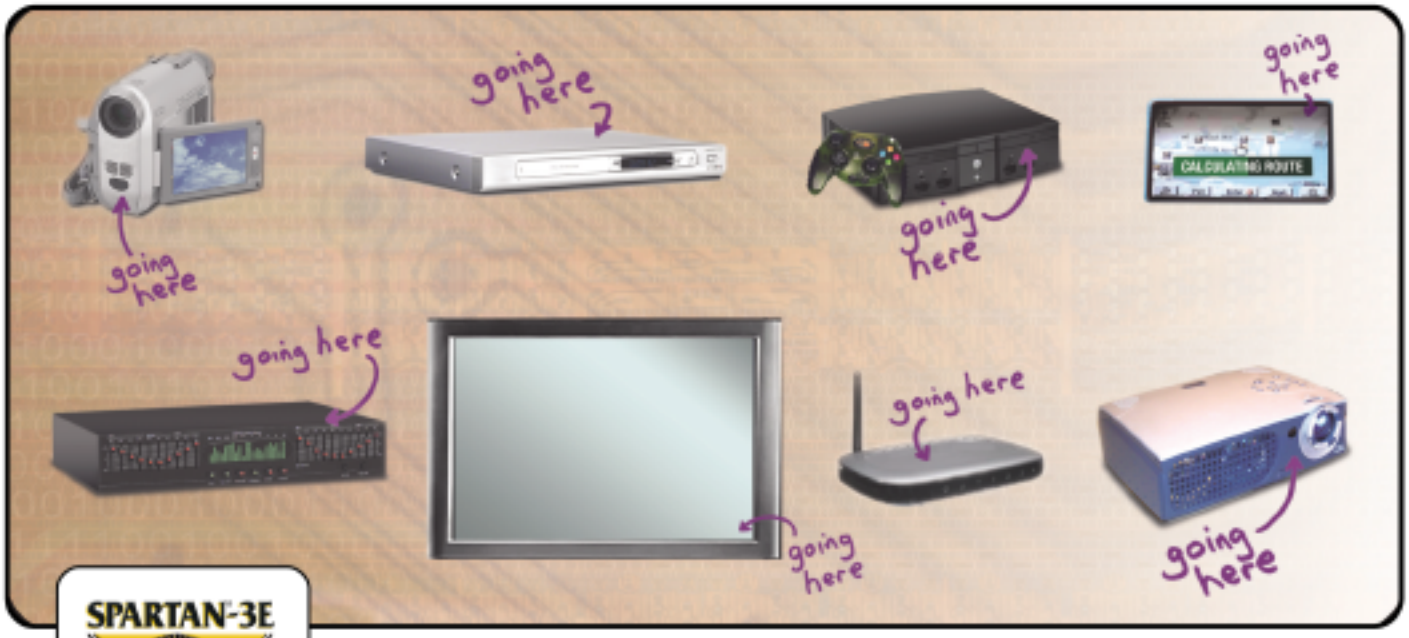
プログラム内容は、セミナ当日に若干変更する場合があります。また、大阪会場では講演者が一部変更になる場合があります。あらかじめご了承をお願いします。

お問い合わせ

ザイリンクス PCI Express セミナ事務局
TEL：03（5333）3342　FAX：03（5333）3466
E-mail：xilinx-event@visions-et.jp

誕生! Spartan-3E ファミリ ―――― いまだかつてない低コストを実現!!

Spartan-3Eは、
これらの量産向けアプリケーションに
採用が決定されています。



デジタルコンシューマ アプリケーションに最適!

高性能90nmプロセステクノロジを採用した最新のSpartan-3Eデバイスは、ロジック重視のデザインに最適化された製品で、特にデジタルコンシューマ アプリケーションに最適な機能を備えています。
従来比30%以上安価になったSpartan-3E FPGAは、ASICを置き換えるだけでなく、柔軟でかつタイム トゥ マーケットを実現できます。

業界初の10万システムゲートFPGAがわずか2ドル*で手に入ります

Spartan-3E FPGAは、10万システムゲートの柔軟なアーキテクチャに加え、低コスト、デジタル信号処理を実現可能な高性能エンベデッド乗算器、72Kビットのブロック RAM、柔軟なクロックシステムを構築することのできるDCM、デジタルクロックマネージャ機能に加え、mini-LVDS、RSDSインターフェースを含むさまざまなI/O標準をサポートしています。また、Spartan-3Eファミリでは、最大160万システムゲートまでの高集積度をサポート。これらは、既に実証済みの90nmプロセスを使用しており、これら業界最先端クラスの製造技術とコスト効率に優れたアーキテクチャの融合で今までにないプライスポイントと最高の価値をお届けします。

すぐに使いこなせる
ソフトウェア

MAKE IT YOUR ASIC

* XC3S100Eの2006年7月以降、50万個での価格です。
また、出荷時期および納期は変更される場合があります。

XILINX®
The Programmable Logic Company™
www.xilinx.co.jp/spartan3e