ASIC から FPGA への移行に関するデザイン手法およびガイドライン

2003 年 7 月 ver. 1.0 Application Note 311

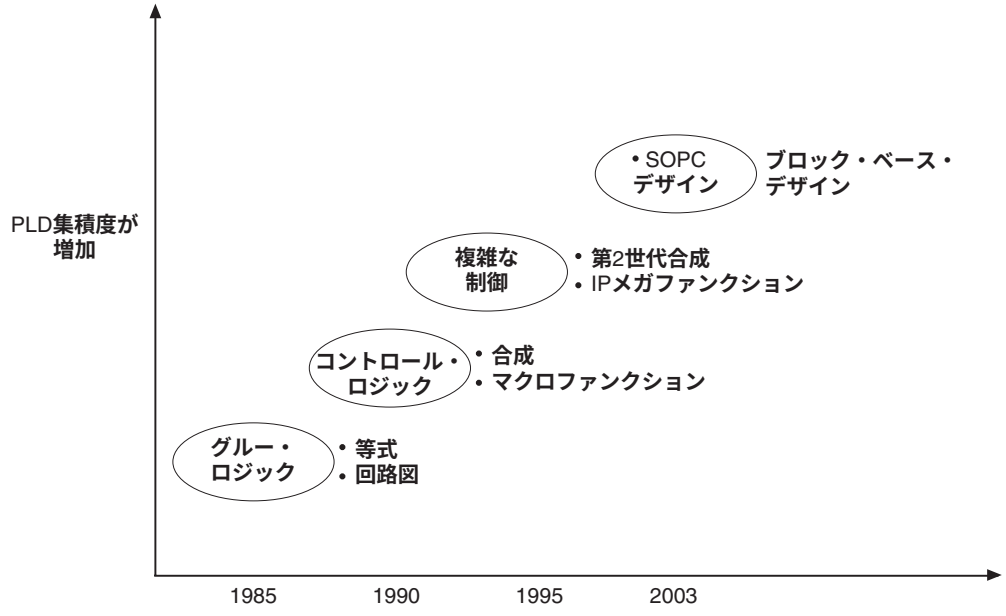
はじめに

ASICデザインのコストは年々増加しています。NRE(Non-Recurring Engineering)コストおよびマスクのコストに加えて、ASIC デザインの複雑さのために開発コストが増え続けています。電力、シグナル・インテグリティ、クロック・ツリー合成などの問題や製造上の欠陥によってリスクが非常に大きくなり、「Time-to-Market」が長くなる可能性があります。リスピンのリスクや NRE の高コストに対応し、「Time-to-Market」の遅れを短縮するために、FPGA は ASIC 開発に対して有効で競争力の高い手段になります。

プログラマブル・ロジックはグルー・ロジックとして使用されていた状態から今日の FPGA に進化し、1 つのデバイスに完全なシステム・デザインを実装できるようになりました。ゲートおよび機能数が劇的に増加し、従来は ASIC デバイスでのみ提供されていた機能と競合できるようになったのです。図 1 に、高集積度デバイス、IP（Intellectual Property）統合、および高速 I/O インタコネクト・テクノロジへと至る FPGA アプリケーションの進化を示します。これらすべての要素によって、FPGA はデジタル・システム実装において中心的な役割を果たします。数百万ゲートの FPGA アーキテクチャの登場と、さまざまなサードパーティ EDA ツールのサポートにより、設計者は ASIC デバイスと同様のデザイン・フローを使用して、FPGA で SOPC（system-on-a-programmable-chip）デザインを作成できます。

ここでは、FPGA デバイスの性能と面積効率を向上させるための、FPGA デザインのガイドラインと手法について説明します。また、アルテラの Quartus® II ソフトウェア・ツールで使用できる SOPC Builder や MegaWizard® Plug-In Manager などの新機能についても説明します。これらの機能を使用すれば、真の SOPC ソリューションを迅速かつ簡単に実現できます。最後に、標準的な FPGA および ASIC のデザイン・プロセスの類似点、および長所と短所を示します。

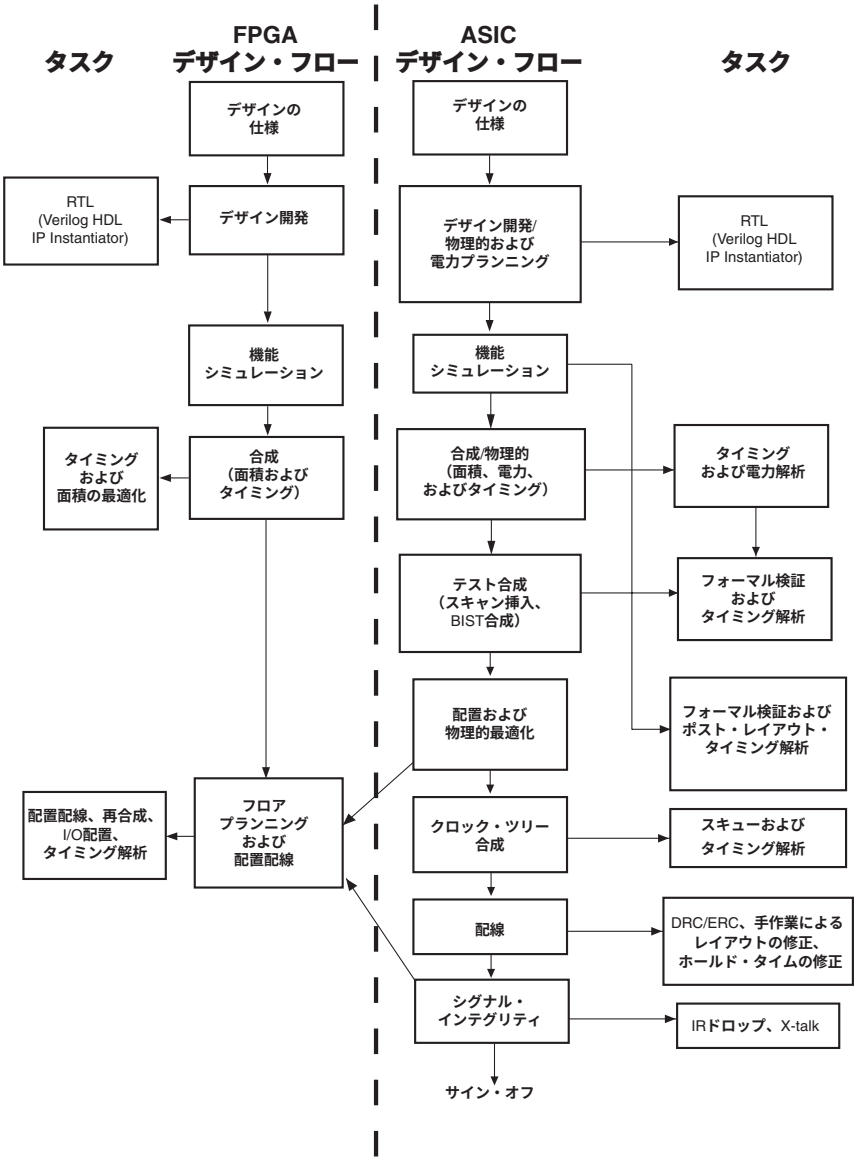
図 1. 1985 年から現在に至る FPGA デバイス・アプリケーションの変遷



ASIC および FPGA のデザイン・フロー

標準的な ASIC および FPGA のデザイン・フローを図 2 に示します。ASIC デバイスのバックエンド・デザインには、配置および物理的最適化、クロック・ツリー合成、シグナル・インテグリティ、異なる EDA ソフトウェア・ツールを使用する配線など、さまざまな複雑なタスクが関係しています。ASIC デバイスと比較すると、FPGA デバイスの物理的（すなわち、バックエンド）デザインは非常に単純であり、Quartus II ソフトウェアという 1 つのソフトウェア・ツールによって行われます。これにより、デザイン・プロセスの複雑さが軽減されるだけでなく、コストも大幅に低減されます。ここでは、FPGA デバイスと ASIC デバイスの両方のデザイン・フローに関係する各タスクについて説明し、それらを比較します。

図 2.　ASIC と FPGA のデザイン・フローの比較



デザイン仕様

デザイン仕様は以下を含む仕様の定義から成ります。

- 標準 I/O 規格、I/O ピン数、各種規格での I/O ピンの配置
- グローバル・クロックおよび周波数要件
- メモリ要件
- テスト方法
- FPGA ファミリおよびデバイスの選択 (スピード・グレード含む)

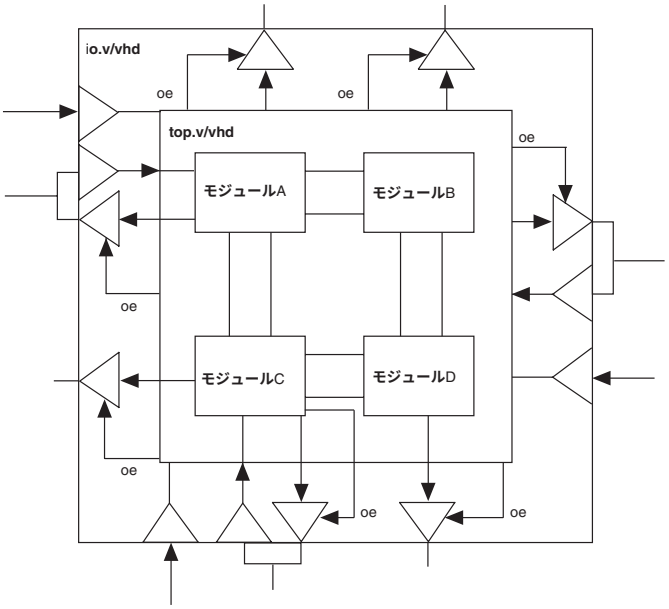
FPGA ファミリはそれぞれ異なる標準 I/O 規格をサポートしており、またファミリ内のデバイスによって I/O ピン数が異なるので、上記の仕様をそれぞれに定義する必要があります。

標準 I/O 規格

FPGA デバイスの選択は、スピード・グレード、信号方式テクノロジ、ターミネーション・テクノロジ、使用可能なロジック・エレメント (LE) 数、外部リセット、グローバル・クロック・ネットワーク数などの条件によって決まりますが、最も重要なのは I/O ピン数とサポートされる標準 I/O 規格です。I/O リソースはデバイスやファミリによって異なります。今日のデザインの複雑さに対応し、複雑な標準 I/O 規格をサポートするために、アルテラ・デバイスはさまざまな標準 I/O 規格をサポートしています。

ASIC デザイン手法は、Verilog HDL または VHDL ファイルでテクノロジの I/O バッファを指定し、シミュレーションと合成を実行することによって、デザインの I/O パッドをインスタンス化します。図 3 に示すとおり、I/O ファイルはトップレベル・ファイルであり、トップレベル・デザインをインスタンス化します。

図 3. I/O 規格 (トップレベル・デザイン)

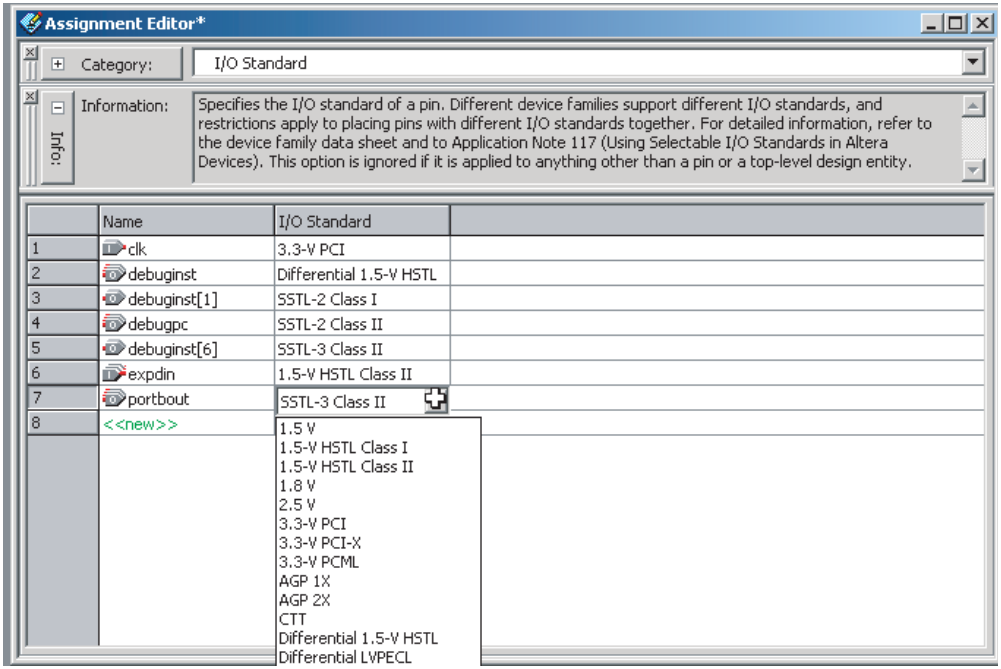


デザインがファンダリに渡されると、**io.v** および **io.vhd** ファイルが取り除かれて、テクノロジ I/O パッドに置き換えられます。

FPGA デザイン・フローでは、サードパーティ合成ツールによってデザインに I/O パッドを合成するか否かを選択できます。I/O パッドを合成する場合は、I/O バッファ経由の遅延を適切に見積もることができます。図 4 に示すように、実際の I/O アサインメントは Quartus II ソフトウェア内で、**Assignment Editor**(Assignment メニュー) を使用して行われます。設計者は、合成されたデザインについて、I/O パッドを含めて Quartus II ソフトウェア内で特定の標準 I/O 規格で置き換えるか、または特定の標準 I/O 規格をトップ・レベル・デザインに追加できるようにするかを選択できます。これらの選択は配置配線前に行われます。

詳しくは、「*Using the Assignment Editor in the Quartus II Software*」ホワイトペーパーを参照してください。

図 4. Quartus II I/O アサインメント・エディタ



アルテラの FPGA デバイスはさまざまな標準 I/O 規格をサポートします。特定のアルテラ・デバイスでサポートされる標準 I/O 規格については、アルテラの各デバイスのデータ・シートを参照してください。

I/O ピン数

デザインで使用される正確な I/O ピン数を決定する必要があります。最終的にこの数値を使用して、パッケージのタイプとパッケージに必要な総ピン数が決定されます。

I/O ピンの位置

ボード・レイアウトに関する潜在的な問題を極力抑えるため、I/O ピンの位置には十分な配慮が必要です。ボード・デザインに高速信号を取り入れる場合、ビア・インタコネクトやクロスオーバー信号を少なくして、配線プロセスを単純にする必要があります。FPGA フローでは、I/O バンクが特定の標準 I/O 規格をサポートするので、設計者は特定の標準 I/O 規格を必要とするデザインのモジュールを物理的に対応する I/O バンクの近くに配置できるよう、そのデバイスがサポート可能な各種標準 I/O 規格を把握していなければなりません。

I/O バンクの位置と I/O バンク内のサポートされる標準規格については、アルテラの各デバイスのデータ・シートを参照してください。

I/O タイミング

I/O タイミングは、標準 I/O 規格とドライブ強度の影響を受けます。I/O 要件に対応するために、アルテラ FPGA はレジスタ付き I/O ピンを備えています。デバイスは、クロック・セットアップ・タイム（t_{SU}）と「Clock-to-Output」遅延（t_{CO}）要求値に応じてこれらのレジスタを使用します。これらのレジスタは、高速入力レジスタや高速出力レジスタなど、Quartus II ソフトウェア内でデザイン階層のトップ・レベルでの制約を設定することによってイネーブルされ、I/O タイミングに対するレジスタの配置を決定します。

グローバル・クロック数

使用可能な FPGA のタイプを判断するために、デザインで必要なクロック数を確認することが必要です。デザイン内に存在する異なるクロックの数には通常、上限があります。ASIC フローではクロック・ツリー挿入は手動で処理しますが、FPGA フローでは自動的に処理されます。

クロック・ツリー合成とクロック要求値について詳しくは、41 ページの「CTS（クロック・ツリー合成）」を参照してください。

メモリ要件

エンベデッド・ダイナミック・ランダム・アクセス・メモリ (DRAM) および SRAM 容量と速度要件の確認は、テクノロジ選択プロセスにおける重要な部分であり、プロセス初期段階で実行する必要があります。Stratix デバイスは、さまざまな種類のアプリケーションで使用可能な各種サイズのメモリ・ブロック内に最大 4M ビットのエンベデッド・メモリを内蔵しています。ASIC テクノロジではメモリ容量が大きくなりますが、ASIC デバイスでこのメモリの統合やテストを行うには多大な労力を要します。FPGA フローでは、Quartis II ソフトウェア・ツールを使用して、ユーザがシステム要件に適合するメモリ・ブロックを自動的に配列することができ、ユーザによるインスタンス化は必要ありません。ASIC フローにおいて、エンベデッド・メモリの統合とテストには、通常 RAM ブロック周囲にテスト用のカスタム回路を作成するだけでなく、レジスタ転送レベル (RTL) デザインで特殊なモジュールをインスタンス化する作業が必要になります。

FPGA のメモリ要件と使用可能なコンフィギュレーションについて詳しくは、25 ページの「内部および外部メモリの仕様」を参照してください。

PLL(Phase-Locked-Loop) 要件

デザインで必要な PLL 数の確認は、FPGA デザイン・フローで選択可能なデバイスのタイプおよびファミリに影響します。ASIC デバイスへの PLL 回路の挿入は通常、デザインで特殊な PLL ブロックをインスタンス化することによって手動で行われます。PLL は、Quartus II ソフトウェア内で利用できる MegaWizard Plug-In Manager ツールを使用して、コンフィギュレーション可能な PLL を作成した後、FPGA デザインでインスタンス化されます。このプロセスは通常 Quartus II ソフトウェアでは自動的に実行されます。FPGA テクノロジでは使用可能な PLL の数が通常は制限されますが、ASIC テクノロジでは PLL の数は事実上無制限です。Stratix デバイスは最大 12 個の PLL をサポートします。

デザインでサポートされる PLL の数とタイプについては、アルテラの各デバイスのデータ・シートを参照してください。

テスト方法

ASIC のテストと故障検出は、ASIC 開発プロセスの重要な部分です。スキャン挿入、BIST(ビルトイン・セルフ・テスト)、シグネチャ解析、I_{DDQ}、および ATPG (自動テスト・パターン生成) はすべて、ASIC デザイン・フロー内で検討する必要があります。ASIC テストでは、通常 ATPG ツールを使用して「単一スタック」モデルの下でデバイスの製造欠陥をテストするテスト・ベクタの生成を行います。ATPG は通常スキャン挿入後に実行されます。スキャン挿入は、シーケンシャル・テスト問題を組み合わせテスト問題に絞り込むことによって、既存の故障検出を改善するために実行されます。FPGA デバイスは工場で事前に検証されているので、FPGA テクノロジを使用すれば、デバイス・テストに悩む必要はなくなります。これらのデバイスでは、大部分のロジック・ファンクションはさまざまなイメージをデバイスにダウンロードすることによって、徹底的にテストされています。したがって、ASIC デバイスと比較して、FPGA デバイスは出荷後の欠陥発生率が非常に低くなります。

合成ツールによってトップレベルの I/O パッドを合成でき、テスト用にバウンダリ・スキャンがすでに内蔵されている FPGA とは異なり、ASIC デザイン・フローでは一般的に、サードパーティの EDA ツールを使用して、デバイス・ロジック上にバウンダリ・スキャンを手動で挿入し、シミュレーションするという多大な労力が必要です。また、FPGA とは異なり、デバイス製造後に故障時の単一スタックの発生を特定するために、ASIC のテスト方法ではデザインのすべてのフリップ・フロップを通過するテスト・ベクタのシーケンシャル伝播が必要です。フリップ・フロップの数が多いため、1 台のテスタで並列にテスト可能な数個のチェインなど、スキャン・チェインは管理可能な多数のチェインに分割されます。スキャン・チェインの数が非常に多く、パッケージ上で使用可能な I/O ピンの数が制限されているため、同じピンを通常動作モードではファンクション・ピンとして、またテスト・モードではスキャン・チェイン出力ピンとして使用できるように、ASIC の I/O ピンは手動でスキャン・チェイン出力と多重化されます。

クロック周波数

デザインに適したアルテラFPGAを選択するには、回路の速度も決定要因になります。

同時スイッチング出力（SSO）数

SSO の数と配置は、ASIC に必要な電源ピンとグランド・ピンの数に直接影響します。FPGA では電源ピンとグランド・ピンの配置と数は固定されていて、Quartus II ソフトウェアで割り当てるので、通常 FPGA にはこのような問題はありません。

FPGA デバイスのサイズ

電力要求値と放熱要求値を推測できるように、必要なロジックの潜在的サイズを見積もる必要があります。ロジックのサイズを見積もり、動作周波数を確認した後、デバイスのおおよそのサイズとスピード・グレードを選択できます。

アルテラ FPGA は、同一パッケージ内でのバーティカル・マイグレーションもサポートしています。バーティカル・マイグレーションでは、専用ピン、コンフィギュレーション・ピン、電源ピンのボード上のレイアウトを変更することなくデバイスの集積度を変更できます。例えば、672 ピン・パッケージの EP1S10B672C6 デバイスから EP1S20B672C6 デバイスにデザインを移行することができます。

電力要件

見積もったロジック・サイズと速度に基づいて、暫定的な電力解析を実行し、デバイスの電力要件を決定します。通常、これによってデバイスの冷却要件とパッケージ要件が定義されます。

デザイン開発

デザイン開発は、FPGA および ASIC 共に以下のプロセスで構成されます。

- 手法の指定 (トップダウンまたはボトムアップ)
- 性能向上のための RTL コーディング・ガイドライン
- 階層デザインの分割
- 外部メモリと内部メモリの規定
- IP の使用とフロー

手法の指定

FPGA デザイン手法は、トップダウンとボトムアップの両方のデザイン手法をサポートします。FPGA デザイン・フローは、ASIC デバイスに使用されるプロセスと同様、ボトムアップ手法のためのモジュラー・デザイン・アプローチ、およびトップダウン・デザイン手法のための階層デザイン分割をサポートしています。

RTL コーディング・ガイドライン

以下に示す簡単なデザイン・ガイドラインによって、デバイスの性能とデザインのデバッグ能力を向上させることができます。ここでは、FPGA アーキテクチャを活用してデバイスの性能を向上させる一般的なデザイン・ガイドラインについて説明します。

- 同期デザインと非同期デザイン
- 同期リセットと非同期リセット
- ゲート付きクロックとクロック・イネーブル
- クロック・ディバイダの使用
- クロック・イネーブルの効率的な使用
- アルテラ PLL の使用によるクロック・スキューの低減
- データ・パイプラインの使用
- エンコーディング方式の使用
- Look-Ahead 手法の使用
- ロジック複製の使用
- 内部バスの使用

同期デザインと非同期デザイン

ここでは、2 つの RTL コード例によって同期カウンタと非同期カウンタの相違を示します。

同期デザイン

同期デザインでは、すべてのレジスタがデザインへのプライマリ入力である 1 つのグローバル・クロックによってクロックされます。同期デザインには組み合わせフィードバック・ループやディレイ・ラインはありません。以下の RTL コードは同期カウンタの一例です。

```
module sync_counter (clk,count_en,reset,count_out);

    input clk,count_en,reset;
    output [3:0] count_out;
    reg [3:0] count_out;

always @(posedge clk) begin
        if (~reset)
            count_out = 4'b0000;
    else if (count_en == 1'b1) begin
        if (count_out == 4'b1111)
    count_out = 4'b0000;
        else
    count_out = count_out + 1'b1;
    end
end

endmodule
```

非同期デザイン

クロック周波数の相違、ゲート付きクロック、または信号が 2 つの異なるクロック・ドメイン間を通過するときに、非同期動作が発生します。非同期信号に起因する最悪の問題は、ディスティネーション・レジスタでの潜在的メタスタビリティです。これは、信号状態がディスティネーション・レジスタのアクティブ・クロック・エッジと同時に変化する可能性があるために発生します。ディスティネーション・レジスタの前に、2 つのフリップフロップ・ステージを追加することによって、このメタスタビリティの危険性を回避することができます。信号によってメタスタビリティが発生しない場合は、目的のレジスタに到達する前に、追加した 2 つのフリップフロップを通過させて安定させます。非同期信号によって発生するその他の問題としては、開始クロックと比較して受信側のサンプリング・クロックが遅すぎて、信号が完全に失われるという問題が考えられます。この場合、デバイスの非同期ブロック間に適切なハンドシェイク・プロトコルを実装して、確実に正しくデバイスを動作させる必要があります。図 5 に、互いに非同期となる複数のクロックを使用し、一方のクロック・ドメインから別のクロック・ドメインに通過する信号を持つ非同期デザインの例を示します。

図 5.　2 つのクロックを使用する非同期デザイン　　注 (1)

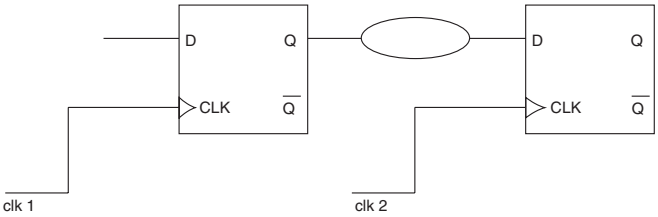


図 5 の注：

(1)　Clk1 と Clk2 のクロック・ソースは異なります。

非同期回路を構成する一般的なデザイン形式には以下の要素が含まれます。

- ゲート付きクロック
- ラッチ推定
- 複数クロック
- 派生クロック

非同期ロジックの短所の 1 つは、このタイプのロジック特有の問題であるレーシング状態が存在することです。以下の RTL コードは非同期カウンタの一例を示しています。

```
module async_counter(clk,count_en,reset,count_out);

input clk,count_en,reset;
output [3:0] count_out;
reg [3:0] count_out;
wire gated_clk = clk & count_en;

always @(posedge gated_clk or negedge reset) begin

    if (~reset)
        count_out <= 4'b0000;
    else begin
        if (count_out == 4'b1111)
        count_out <= 4'b0000;
    else
        count_out <= count_out + 1'b1;
    end
end
endmodule
```

同期リセットと非同期リセット

非同期リセットは、関連付けられているクロックに関係なく、レジスタの内容をクリアする方法として定義されます。ASIC ライブラリは、ビルトイン・リセット / クリア・ピン付きレジスタとピンのないレジスタで構成されています。ビルトイン・リセット / クリア・ピン付きレジスタは、ピンのないレジスタより大きくなります。

ASIC 設計者は、多くの場合ビルトイン・リセット・ピンのないレジスタを使用し、リセット用レジスタのデータ・パス上の外部ゲートを使用することにより、デザインにおいて速度と面積効率の向上を図ります。リセットがデータ・ピンを通して配線されている場合、リセットのアサート時にはクロックが動作していなければなりません。これに対し、同期リセット信号は他のデータ信号として扱われるので、配線およびタイミング最適化フェーズで余分な処理が必要になることはありません。

ASIC デザインでは、ステート・マシンからの内部生成された非同期リセットによって、スキャン・テストで問題が発生する可能性があります。通常、ステート・マシンのフリップ・フロップを通してテスト・ベクタをシフトするとこのような問題が発生し、誤ったリセットを発生させます。FPGA はスキャン・テストが行われないので、FPGA では問題ありません。

リセット信号をゲーティングする代わりに、ASIC 設計者がビルトイン・リセット / クリア・ピン付きレジスタを使用する場合、非同期リセットはデバイスのプライマリ入力によって制御され、クロック・ツリーと同様にバッファされて、すべてのデバイス・レジスタのすべての非同期リセット・ピンに到達します。ASIC では、クロック・ツリーと同様にリセット・ツリーを生成する必要があります。異なるステート・マシンがリセット解放直後に同期が外れるのを防止するには、タイミング解析または個別のスタート信号を使用します。

FPGA ではリセット・ツリーが既に配置されています。すべてのレジスタにはビルトイン非同期リセットがあるため、このリセット機能を使用しないことが面積の節約にはなりありません。リセット信号の解放後直ちにすべてが同期化されるようなデザイン上の安全策を講じる必要があります。

ゲート付きクロックとクロック・イネーブル

ASIC デザインでは、電力最適化のためにクロック・ゲーティングが使用されます。例えば、ライト・イネーブル信号が High のときにのみメモリにデータを書き込む必要がある場合は、ライト・イネーブルが High のときにのみレジスタのスイッチングが行われるよう、クロックをライト・イネーブルでゲートすることができます。ゲート付きクロックでは、組み合わせパスに関連する遅延によってグリッチが発生し、誤ったデータがクロックされる可能性があります。以下の RTL コードはクロック・ゲーティングの例です。図 6 に対応する回路図を示します。

```
module gated_clk(in,out,en,clk);
input in,clk,en;
output out;
reg out;
wire gate_clk;
assign gate_clk = clk & en;
always @(posedge gate_clk) begin
    out <= in;
    end
endmodule
```

図 6. クロック・ゲーティング

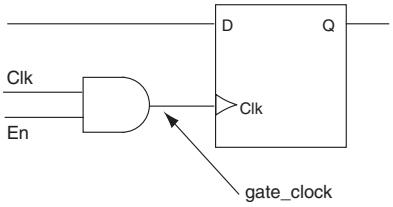
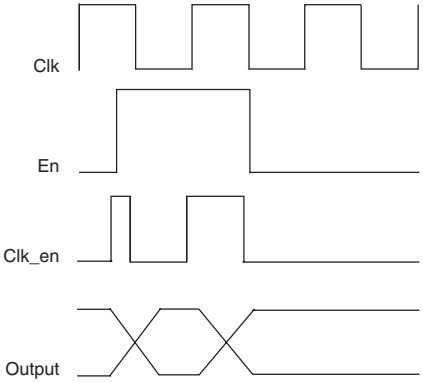


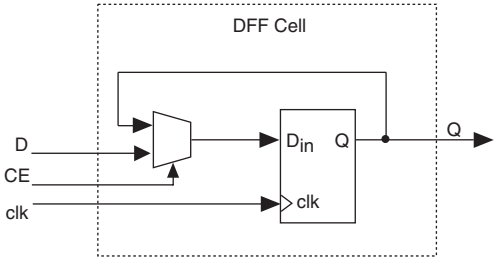
図 7 のタイミング図に、クロックのゲーティングに関連するグリッチを示します。

図 7. クロック・ゲーティングに関連する問題



FPGA には、ゲート・クロッキングを回避するのに使用できるクロック・イネーブル・ピンがあります。クロック・イネーブル・ピンを使用してフリップ・フロップのクロッキングをディセーブルしても、フリップ・フロップがクロックされないということではありません。単に、フリップ・フロップの現在の状態が継続的にクロックされるだけです。フリップ・フロップのこのような実装の例を図 8 に示します。

図 8. クロック・イネーブル・ピンによるフリップ・フロップのクロッキングのディセーブル

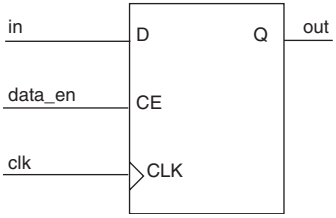


以下の RGL コードは、クロック・イネーブル・ピンを使用して図 8 の例を修正したものです。

```
module clock_en(in,out,clk,data_en);
input in,clk,data_en;
output out;
reg out;

always @(posedge clk) begin
    if (data_en)
        out <= in;
    else
        out <= out;
end
endmodule
```

図 9 に、このクロック・イネーブル・コードの回路図を示します。

図 9. FPGA デバイスのビルトイン・クロック・イネーブル・リソース



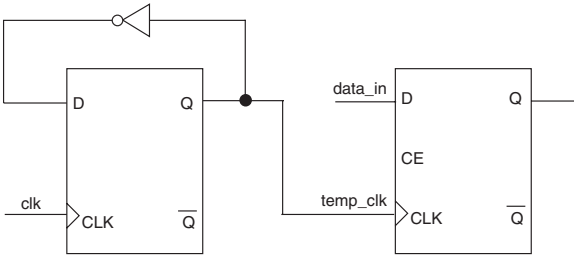
クロック・ディバイダの使用

データ入力に Q バー出力を供給するのは、クロック分周で広く使用されている方法であり、リップル・クロック分周と呼ばれています。以下の RTL コードはこのようなディバイダの例です。

```
module ripple_clk_divider(clk,data_in,data_out);
input clk,data_in;
output data_out;
reg data_out;
reg temp_clk;

always @(posedge clk)
temp_clk <= ~temp_clk;

always @(posedge temp_clk)
data_out <= data_in;

endmodule
```

図 10 に、このコード例の回路図を示します。

図 10. リップル・クロック分周

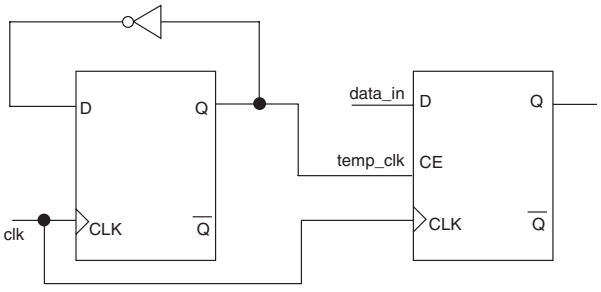


以下の RTL コード例に示すように、クロック・イネーブル・リソースを使用してクロックを分周し、同期クロック分周によるリップルの影響を軽減します。

```
module clock_divider_ce(clk,data_in,data_out);
    input clk,data_in;
    output data_out;
    reg data_out;
    reg temp_clk;

always @(posedge clk)
    temp_clk <= ~temp_clk;

always @(posedge clk) begin
    if (temp_clk)
        data_out <= data_in;
    end

endmodule
```

図 11 に、このコードの回路図を示します。

図 11. 同期クロック分周



クロック・イネーブルの効率的な使用

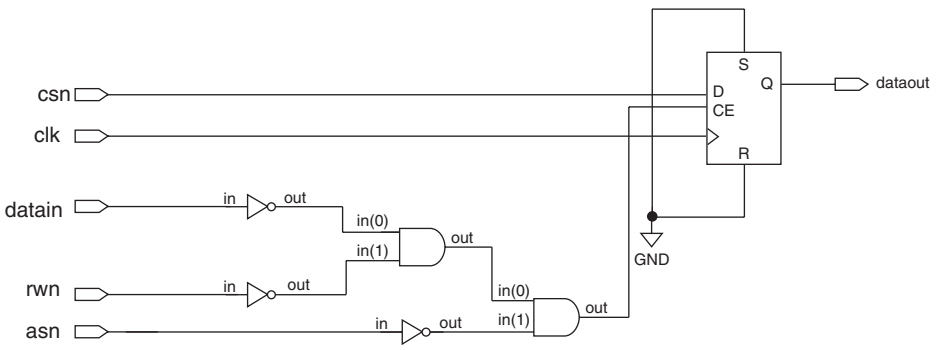
レジスタのクロック・イネーブル・ピンをドライブする組み合わせロジックがクリティカル・パス内に存在する場合があります。クロック・イネーブルには、同じ LE 内のルックアップ・テーブル (LUT) を介してアクセスすることはできません。このため、ロジック・ファンクションがフリップ・フロップのクロック・イネーブルをドライブする場合は、関連するロジックおよび配線遅延を持つ外部LUTにクロック・イネーブルが実装されます。ロジックのレイアウト数を少なくするために、マルチプレクサを使用してデータ・パス内に同じクロック・イネーブルを実装できる場合があります。

次の例は、ロジック・レベル数を減らす RTL コードを示します。

```
module inefficient_ce(clk,datain,rwn,csn,asn,dataout);
input       clk,datain,rwn,csn,asn;
output      dataout;
reg         dataout;

always @(posedge clk)
    begin
        if ((rwn == 1'b0) && (csn == 1'b0) && (asn == 1'b0))
            dataout <= datain;
    end

endmodule
```

図 12 に、上のコードの回路図を示します。

図 12. ロジック・レベル数を低減するための RTL の変更

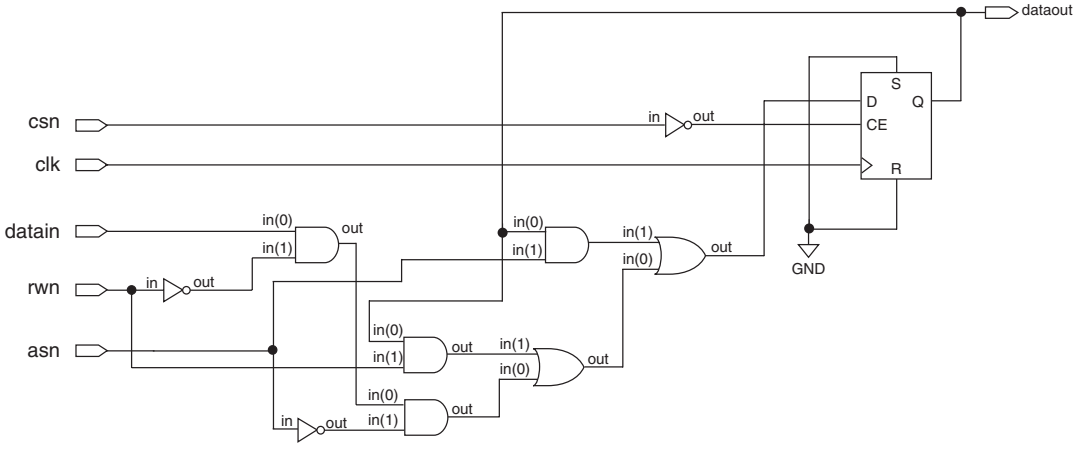


以下の RTL コードを使用すると、レジスタのクロック・イネーブルをドライブするロジック・レベル数を 1 LUT から 0 LUT に低減できます。

```
module efficient_ce(clk,datain,rwn,csn,asn,dataout);

inputclk,datain,rwn,csn,asn;
outputdataout;
reg     dataout;

always @(posedge clk)
    begin
        if (csn == 1'b0)
            dataout <= (datain && (rwn == 1'b0) && (asn ==
1'b0))
                                || (dataout && rwn == 1'b1)
                                || (dataout && asn == 1'b1);
    end

endmodule
```

図 13 に、この RTL コードの回路図を示します。

図 13. 1LUT から 0LUT へのロジック・レベルの低減



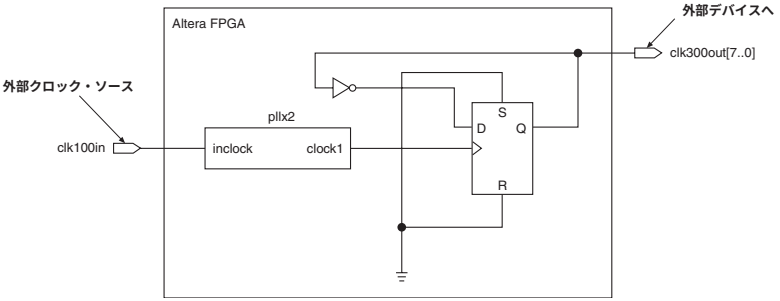
ゲート・クロッキングが唯一の代替方法でない限り、クロック・イネーブル・リソースを利用したデザインを実装してください。

PLL を使用したクロック・スキューの低減

アルテラ PLL は、スキューの発生を最小限に抑え、オフチップで使用される複数のクロック・ソースを提供できます。以下の RTL 実装と図 14 の回路図に、入力クロックの 3 倍となる 8 つのピンをドライブする方法を示します。PLL を使用して入力クロック周波数を 6 倍にし、トグル・フリップフロップを使用して入力周波数の 3 倍の周波数を取得します。図 14 に示すように、Quartus II ソフトウェア MegaWizard Plug-in Manager がインスタンス化されたモジュール pllx2 を生成します。

```
module off_chip(clk100in,clk300out);

inputclk100in;
wireclk600;

output[7:0]clk300out;
reg     [7:0]clk300out;

reg     clk300int /* synthesis syn_preserve = 1 */;

pllx2 U1(
    .inclock(clk100in),
    .clock1(clk600));

always @(posedge clk600)
    begin
        clk300int <= ~clk300int;
    end

always @(posedge clk600)
    begin
        clk300out[0]<=~clk300int;
        clk300out[1]<=~clk300int;
        clk300out[2]<=~clk300int;
        clk300out[3]<=~clk300int;
        clk300out[4]<=~clk300int;
        clk300out[5]<=~clk300int;
        clk300out[6]<=~clk300int;
        clk300out[7]<=~clk300int;
    end

endmodule
```

図 14. 複数クロックへのソースとして使用されるアルテラ PLL

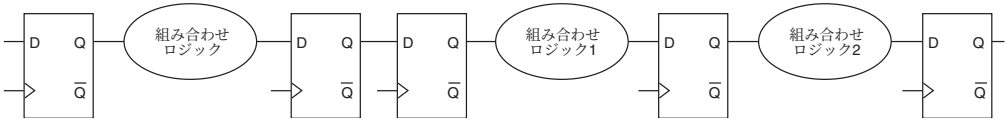


ゲート・クロッキングが唯一の代替方法である場合を除いて、クロック・イネーブル・リソースを利用したデザインを実現してください。

データ・パイプラインの使用

FPGA には各ロジック・エレメントで使用可能なフリップ・フロップがあるため多数のレジスタがあり、これらのレジスタを使用して、面積を増やすことなくデバイスの性能を向上させることができます。データ・パイプライン化は、レジスタを挿入して長い組み合わせパスを分断することによってデバイスの性能を向上させことができる重要な概念です。図15 は、パイプライン化の概念を示しています。

図 15. データ・パイプライン化の概念



データ・パイプライン化によってデザインの性能が向上する場合でも、データとコントロール・パスの待ち時間を等しくし、挿入されたレジスタからの信号キャプチャに対応するためにテスト・ベンチを変更してください。

エンコーディング方式の使用

ステート・マシンは今日のほとんどすべての SOPC デザインで広く使用されています。ステート・マシンは、一般に使用されているワンホット・バイナリ・エンコーディング方式など、さまざまなエンコーディング方式によって実装できます。ワンホット・エンコーディングでは各ステートにストレージ・エレメントが必要ですが、バイナリ・エンコーディングでは「n」個のストレージ・エレメントを使用して 2^n 個のステートを表すことができます。高度にエンコードされたバイナリ・ステート・マシンをデコードすると、単純なワンホット・エンコーディング・ステート・マシンと比較して通常、ステート間のロジック・レベルが増加します。ワンホット・エンコーディングの利点は、ステートが既にデコードされたフォーマットになっており、単純なデコーディング・ロジックしか必要ないので、ステート間のロジック・レベル数が低減されることです。

FPGA には多数のレジスタが実装されており、また大部分のデザインについてはワンホット・エンコーディングの方がバイナリ・エンコーディングよりも優れた性能結果が得られるため、ワンホット・エンコーディングによる有限ステート・マシン（FSM）を実装してください。

Look-Ahead 手法の使用

Look-Ahead 手法は、大きい組み合わせロジック・ファンクションの一部を前のクロック・サイクルに強制的に送り込み、残りのロジック・ファンクションを後のクロック・サイクルで強制的に実行します。この手法は、レジスタ・バランシングとも呼ばれ、レジスタ間のロジック・レベルのバランスを取ります。その結果、実行速度が大幅に向上します。このため、組み合わせロジックを 2 つのクロック・サイクルに分割することにより、以下の RTL コード例に示すとおり、両方のサイクルの動作速度が向上し、デザイン全体でレイテンシが生じません。

```
module without_lookahead (clk,a,b,q,d);
input clk,d;
input [15:0] a,b;
output q;
reg q;

always @(posedge clk) begin
if (^a && (a == b)) // a and b are 16 bit registered input
busses
    q <= d;
end
endmodule
```

上記の例を変更して組み合わせロジックを 2 つのレジスタ・ステージに分割する場合、コードは以下のとおりです。

```
module with_lookahead (clk,a,b,q,d);
input clk,d;
input [15:0] a,b;
output q;
reg q;
reg ce0, ce1;

always @(posedge clk) begin
    ce0 <= (a[7:0]==b[7:0]);
    ce1 <= (a[15:8]==b[15:8]);
end

always @(posedge clk) begin
    if (^a && (ce0 == 1'b1) && (ce1==1'b1))
        q<=d;
end
endmodule
```

ロジック複製の使用

配線遅延を最小化することによって、FPGA の性能を向上させることができます。高ファン・アウトは、ロジック複製によって最小化が可能な配線遅延ソースの 1 つです。Quartus II ソフトウェアは、ロジック複製をネットリスト最適化の一部としてサポートしています。以下の RTL コード例は、高ファン・アウトを低減する手法を示します。

詳しくは、44 ページの「ネットリスト最適化」を参照してください。

```
module large_load(in,en,clk,out);
input [31:0] in;
input clk,en;
output [31:0] out;
reg [31:0] out;

always @(posedge clk) begin
    if (en)
        out <= in;
    else
        out <=32'bz;
end

endmodule
```

信号 en は、上記のとおり、32 個のトライ・ステート・バッファをドライブし、配線遅延の一因となります。以下の修正された RTL コードに示すとおり、信号 en を複製して、32 個ではなく 16 個から成るトライ・ステート・バッファのセットをドライブすることができます。しかし、合成ツールはこの繰り返しを冗長とみなし、複製されたロジックを最適化する傾向にあります。LeonardoSpectrum™ と Synplify に共通に使用される以下のコードに示すように、対象となる複製ロジックを維持するために合成アトリビュートが必要です。

```
module logic_duplicate(in,en,clk,out);
input [31:0] in;
input clk,en;
output [31:0] out;
reg [31:0] out;
reg en1 /* synthesis syn_preserve=1 */; // Synplify
reg en2 /* synthesis syn_preserve=1 */; // Synplify
//exemplar attribute en2 PRESERVE_DRIVER -value TRUE // LS
//exemplar attribute en1 PRESERVE_DRIVER -value TRUE // LS
always @(posedge clk) begin
    en1 <= en;
    en2 <= en;
end

always @(posedge clk) begin
    if (en1)
        out[31:16] <= in[31:16];
    else
        out[31:16] <=16'bz;
end
always @(posedge clk) begin
    if (en2)
        out[15:0] <= in[15:0];
    else
        out[15:0] <=16'bz;
end

endmodule
```

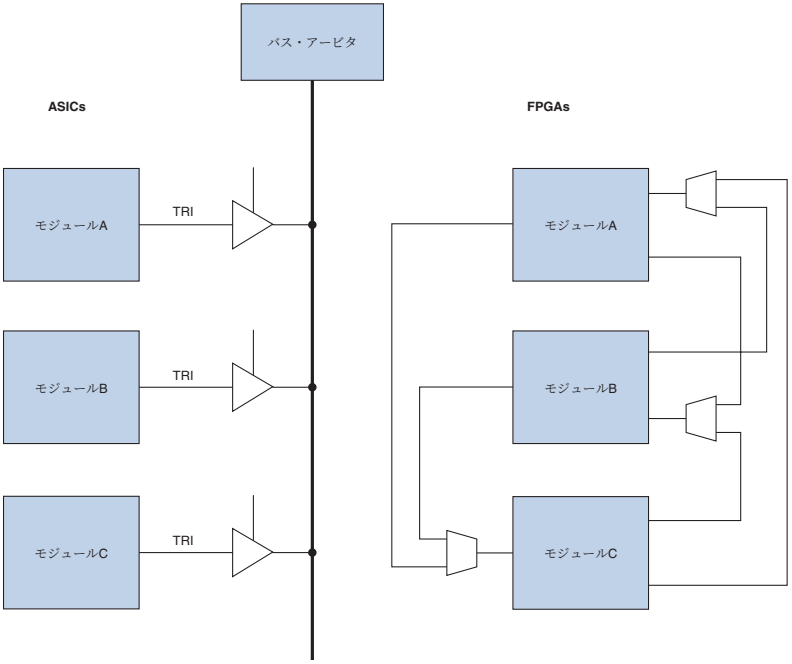
内部バスの使用

ASIC デバイスの内部バスによって、各種内部モジュールと各種外部デバイスとの通信が可能になります。ASIC デバイス内には多数の内部トライ・ステート・バスが存在する可能性があります。

FPGA のデザイン作成時、これらの内部トライ・ステート・バスはディープ・ワイド・マルチプレクサに変換されてしまいます。ただし、FPGA は I/O インタフェースを介してトライ・ステート・バスをサポートし、各種オンボード・デバイス間の通信を仲介します。

図 16 は、FPGA でのディープ・ワイド・マルチプレクサ実装と ASIC での内部バス実装との比較を示します。

図 16. ASIC デバイスと FPGA デバイスの内部バス



以下の RTL コードは内部バスをマルチプレクサに変換し、トライ・ステート・ゲートは OR ゲートに変換されます。

```
module bustest (clk, ina, inb, inc, sel, outputvalue);

input ina, inb, inc,clk;
input [2:0] sel;
inout outputvalue;

//Internal Signals
reg reg_ina, reg_inb, reg_inc;
reg bus;
reg reg_bus;
always @(posedge clk)
begin
    reg_ina <= ina;
    reg_inb <= inb;
    reg_inc <= inc;
    reg_bus <= bus;
end
```

```
always @(sel or reg_ina or reg_inb or reg_inc)
begin
    case(sel)
            3'b100 : begin
                        bus <= reg_ina;
                     end
        3'b010 : begin
                     bus <= reg_inb;
                 end
        3'b001 : begin
                     bus <= reg_inc;
                 end
        default begin
                     bus <= 1'bZ;
            end
    endcase
end

assign outputvalue = reg_bus;
endmodule
```

内部および外部メモリの仕様

メモリ・ブロックのサイズとコンフィギュレーションは SOPC デザインの主要検討事項であり、デバイス・メモリをオンチップまたはオフチップのどちらにするかは性能目標によって決まります。大容量メモリ要件に対する需要を満たすために、アルテラの FPGA は多様なメモリ・ファンクション・セットを備えており、これらは Quartus II ソフトウェアを使用してさまざまな要件に対応してコンフィギュレーション可能です。StratixおよびStratix GXデバイス・ファミリはTriMatrix™メモリを内蔵しており、このメモリは表 1 に示すとおり、さまざまなメモリ・サイズにコンフィギュレーション可能です。

Stratix デバイスおよび Stratix GX デバイスの使用可能なメモリ・コンフィギュレーションと機能について詳しくは、*Stratix デバイス・ハンドブック*の「*Stratix GX FPGA ファミリ・データ・シート*」および「*Stratix デバイス・ファミリ・データ・シート*」を参照してください。

表 1. TriMatrix メモリの特長の要約 (1 / 2)

特長	M512 ブロック	M4K ブロック	M-RAM ブロック
性能	312 MHz	312 MHz	300 MHz
トータル RAM ビット数 (パリティ・ビットを含む)	576	4,608	589,824

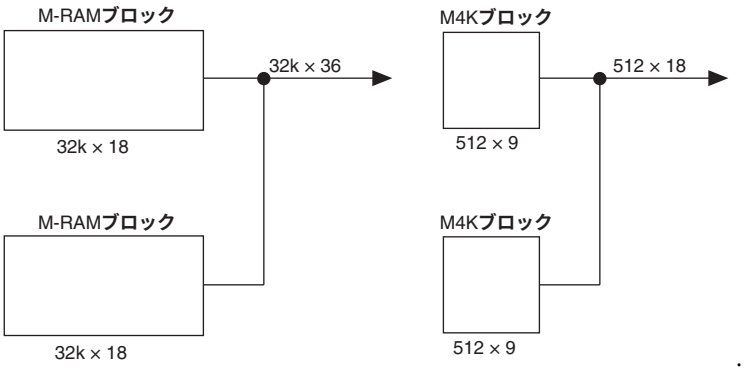
表 1.　TriMatrix メモリの特長の要約 (2 / 2)

特長	M512 ブロック	M4K ブロック	M-RAM ブロック
コンフィギュレーション	512 × 1 256 × 2 128 × 4 64 × 8 64 × 9 32 × 16 32 × 18	4K × 1 2K × 2 1K × 4 512 × 8 512 × 9 256 × 16 256 × 18 128 × 32 128 × 36	64K × 8 64K × 9 32K × 16 32K × 18 16K × 32 16K × 36 8K × 64 8K × 72 4K × 128 4K × 144
パリティ・ビット	✓	✓	✓
バイト・イネーブル		✓	✓
シングル・ポート・メモリ	✓	✓	✓
シンプル・デュアル・ポート・メモリ	✓	✓	✓
トゥルー・デュアル・ポート・メモリ		✓	✓
エンベデッド・シフト・レジスタ	✓	✓	
ROM	✓	✓	
FIFO バッファ	✓	✓	✓
シンプル・デュアル・ポート・データ幅混在サポート	✓	✓	✓
トゥルー・デュアル・ポート・データ幅混在サポート		✓	✓
メモリ初期設定 (.mif)	✓	✓	
異周波数でのアクセス	✓	✓	✓
パワー・アップ条件	出力がクリア	出力がクリア	出力が未知

メモリの実装 — 柔軟性および効率

メモリ・コンフィギュレーションが 1 つのメモリ・ブロックに収まらない場合、Quartus II ソフトウェアは 2 つ以上のメモリ・ブロックを結合することによって必要なコンフィギュレーションを実現します。例えば、デザインが 512 × 18 ビットのメモリ・ブロックを必要とする場合、Quartus II ソフトウェアは 512 × 9 ビットのブロックを 2 つ結合します。同様に、Quartus II ソフトウェアは、図 17 に示すとおり、2 つの 32K × 18 ビットの M-RAM ブロックを結合して、32K × 36 ビットの RAM ファンクションを実現することができます。

図 17. メモリ・ブロックのコンフィギュレーション

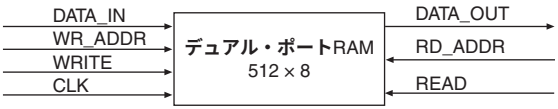


メモリ・コンフィギュレーションおよび Quartus II ソフトウェアの設定について詳しくは、アルテラ Web サイトおよび「*AN 207: TriMatrix Memory Selection Using the Quartus II Software*」を参照してください。

ASIC RAM に代わるアルテラ RAM のインスタンス化

ASIC のデザイン・フローでのメモリのインスタンス化は、ファンダリが提供したメモリ・モデルをインスタンス化します。例えば、図 18 はシンプル・デュアル・ポート・メモリ・インタフェースを示します。

図 18. デュアル・ポート RAM



以下の RTL コードは、図 18 に示すデュアル・ポート・メモリ・インタフェースのインスタンス化を示します。

```
module my_design (port declarations)

input and output declarations

module instantiations

dual_port_ram dual_port_ram_inst

.clk (clk),
.data_in (data_in),
.rd_address (rd_address),
.read (read),
.data_out (data_out),
.wr_address (wr_address),
.write (write));
```

ASIC ファンダリが提供するメモリ・モデルは、合成用タイミング情報とシミュレーション用ビヘイビア・モデルを搭載しています。デザインをアルテラの FPGA に移植しなければならない場合、アルテラの MegaWizard Plug-In Manager 内からメモリ要件をコンフィギュレーションする必要があります。次に、このウィザードは以下のファイル・セットを生成します。

- **memory.v/memory.vhd**
  このファイルは、アルテラの LPM(library-of-parameterized-modules) メモリ・モジュールをインスタンス化するラッパ・ファイルで、シミュレーションに使用されます。
- **memory_bb.v/memory_bb.vhd**
  このファイルは合成に使用されるモジュール宣言です。
- **memory_inst.v/memory_inst.vhd**
  このファイルはメモリ・モジュールのインスタンス化の例です。

メモリ・モデルのタイミング情報は「クリア・ボックス」とも呼ばれ、Quartus II ソフトウェア・バージョン 3.0 で使用できます。さらにこのタイミング情報は、合成のために Synplify Pro 7.3 と併用することができます。Synplify Pro は、これらのモデルを使用して面積とタイミングを正確に見積もることができます。

シミュレーション

アルテラは、すべてのデバイス・ファミリに MegaWizard Plug-In Manager を使用して、生成されたメモリ・モジュールのシミュレーション・モデルを提供します。シミュレータ・ツールを選択する場合は、シミュレーション・フローについて説明したアプリケーション・ノートを参照してください。

合成

合成ツールを選択する場合は、合成フローについて説明したアプリケーション・ノートを参照してください。

Quartus II ソフトウェアのヘルプ・システムは、アルテラのメガファンクションおよびLPMファンクションに関するガイドラインについての追加情報を提供しています。

配置配線

ASIC のデザイン・フローと同様に、Quartus II ソフトウェアの配置配線ツールは適切なメモリ・モジュールをインスタンス化し、デザインの配置配線を実行します。

Quartus II ソフトウェアのヘルプ・システムは、LPM メガファンクションのインスタンス化についての追加情報を提供します。

アルテラの FPGA で設計するときは、使用可能なメモリ・リソースがファミリおよびデバイスによって異なるため、デザイン計画の過程でデバイスを指定する必要があります。

アルテラの Stratix および Stratix GX デバイス・ファミリは同期メモリ・インタフェースのみをサポートします。デザインに既にメモリ・ブロックへの非同期リファレンスが含まれている場合は、「*AN 210*: *Converting Memory from Asynchronous to Synchronous for Stratix & Stratix GX Designs*」を参照してください。

デザイン開発ツール

アルテラは以下の開発ツールを提供します。これらの開発ツールはすべて Quartus II ソフトウェア内で使用できます。

- SOPC Builder
- DSP Builder
- MegaWizard Plug-In Manager

これらのツールを使用して SOPC デザインの開発を容易にすることができ、結果的に「Time-to-Market」が短縮されます。以下のセクションでは、これらのツールの概要と利点を示します。

SOPC Builder

SOPC Builder は、CPU、メモリ・インタフェース、標準ペリフェラル、ユーザ定義ペリフェラルなどのコンポーネントで構成される、SOPC デザインの作成のための統一されたグラフィカルな環境を提供します。

SOPC Builder によって、エンベデッド・プロセッサ、標準ペリフェラル、IP コア、オンチップ・メモリ、オフチップ・メモリへのインタフェース、ユーザ定義ロジックなどのコンポーネントを組み合わせて 1 つのカスタム・システム・モジュールを作成することができます。このツールは、これらのコンポーネントをインスタンス化する 1 つのシステム・モジュールを生成し、これらのコンポーネントを接続するのに必要なバス・ロジックを自動的に生成します。SOPC Builder は、シミュレーション・モデルとして使用可能な、システムを記述した Verilog デザイン・ファイル **(.v)** および / または VHDL デザイン・ファイル **(.vhd)** も自動的に生成します。

図 19 は、SOPC Builder と併用してシステム・レベルのモジュールを数分間で生成できる標準的なシステム・コンポーネント・セットの例を示します。SOPC Builder はテスト・ベンチも生成します。

図 19. 標準的な SOPC デザインの要素

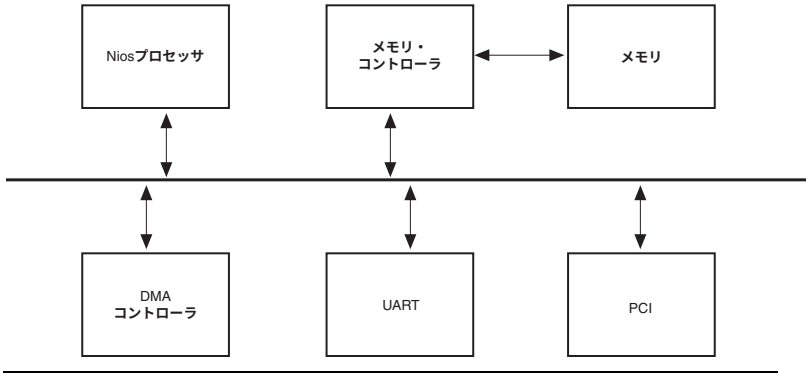
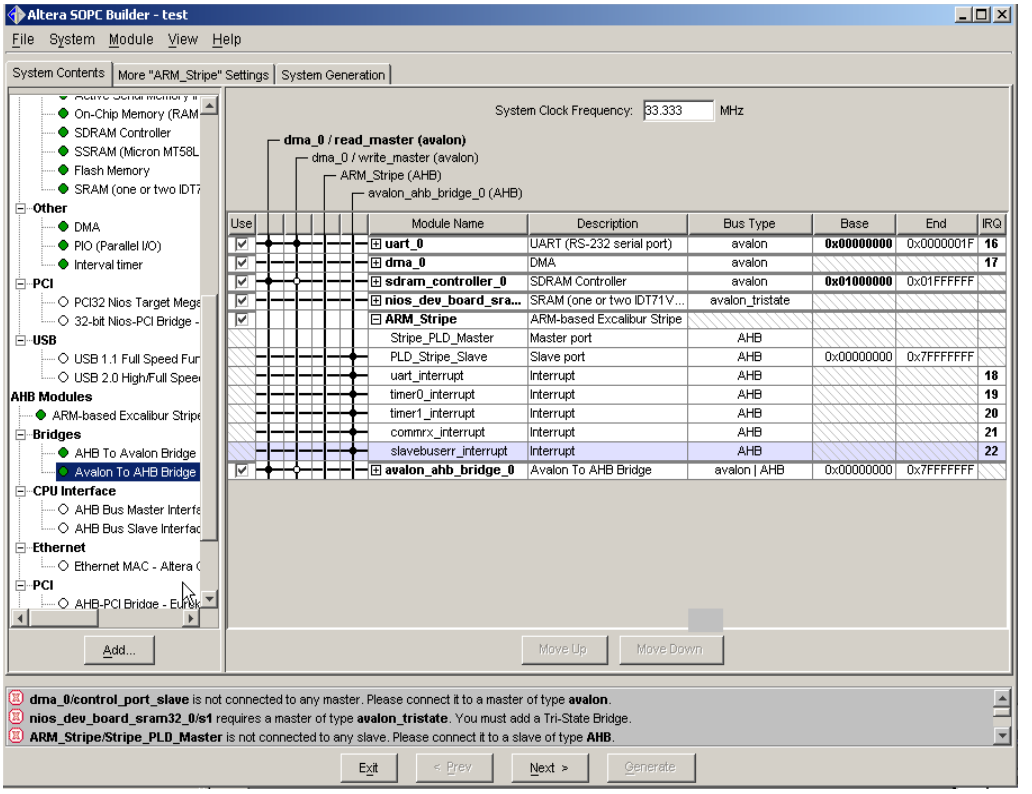


図20は、ペリフェラル・セットが Nios® プロセッサに接続された後の SOPC Builder ユーザ・インタフェースを示します。

SOPC Builder について詳しくは、**www.altera.co.jp/literature** の資料を参照してください。

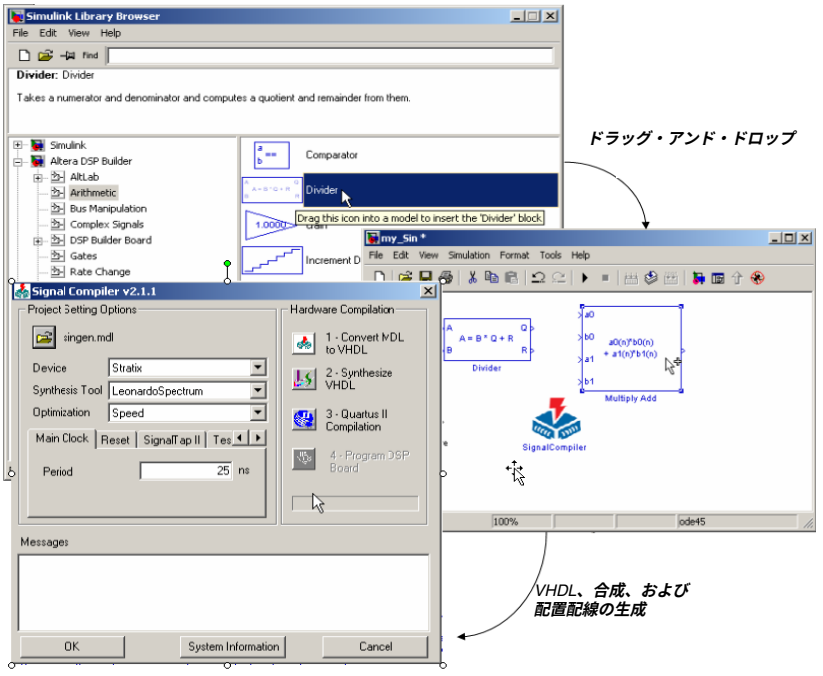
図 20. アルテラの SOPC Builder



DSP Builder

アルテラの DSP Builder は、Quartus II ソフトウェアと、MATLAB および Simulink ソフトウェアなどの高度なアルゴリズム開発ツールとを統合します。DSP Builder ソフトウェアは、設計者がアルゴリズムと親和性のある開発環境で DSP デザインのハードウェア表現を作成するのを支援することによって、DSP の設計サイクルを短縮するビルディング・ブロックのセットです。図 21 はコンセプトから実装まで、DSP Builder を使用して構築されたシンプルなデザインを示します。MATLAB および Simulink の両方のツール内では、すぐに使えるいくつかの数学関数をシミュレーション・モデルと共に利用できます。これらを使用して回路図を作成できます。DSP Builder は、テストベンチと共にデザインの VHDL 記述を生成することができます。DSP Builder には、Quartus II ソフトウェアと併用して合成、シミュレーション、および配置配線を実行可能なサード・パーティ EDA ツールへのリンクもあります。

図 21. DSP Builder のデザイン・フロー

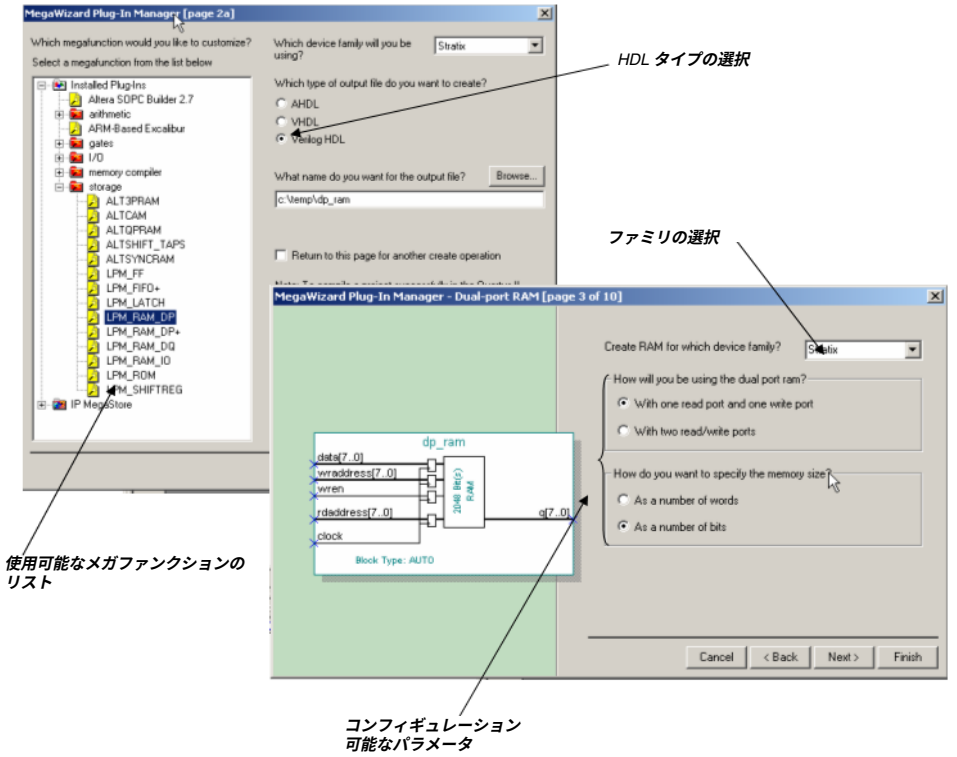


MegaWizard Plug-In Manager

Quartus II ソフトウェアで使用可能な MegaWizard Plug-In Manager は、カスタム・メガファンクション・バリエーションを収めたデザイン・ファイルの作成または変更に役立ち、さらにデザイン・ファイルでこれらのカスタム・メガファンクションをインスタンス化することができます。これらのカスタム・メガファンクションの種類は、LPM ファンクションを含むアルテラのメガファンクションが基本となっており、単純なブール・ゲートから複雑なメモリ構造に至るまで多種多様です。

図 22 に、アルテラの LPM エレメントにアクセスしてこれらを編集するために MegaWizard Plug-In Manager で要求される選択を示します。

図 22. MegaWizard Plug-in Manager LPM コンポーネントのコンフィギュレーション



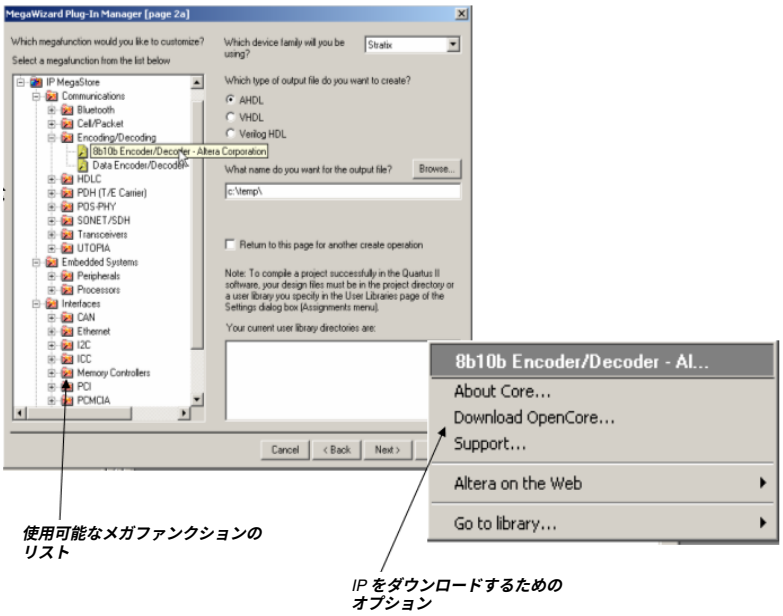
IP の供給状況とフロー

事前に検証された IP ブロックは、設計時間を短縮し、「Time-to-Market」に関する多数の問題を解決し、さらに検証を簡略化します。

MegaWizard Plug-In Manager への MegaWizard Portal Extension により、設計者は Quartus II 環境を離れずに IP MegaStore™ サイトで入手可能な MegaCore® ファンクションを直接インストールして使用できます。設計者は、リスト内の使用可能な MegaCore ファンクションの選択、選択した MegaCore ファンクションに関する情報の入手、Altera IP MegaStore サイトへの登録、MegaCore ファンクションのダウンロード、対応する MegaWizard Plug-In のインストールと起動のすべてをQuartus IIソフトウェア内から実行できます。この機能によって、IP MegaStore にある MegaCore ファンクションの最新バージョンにアクセスできます。

図 23 は、MegaWizard Plug-In Manager の初期画面と使用可能ないくつかの IP コアのサンプル・リストを示します。

図 23. IP の作成

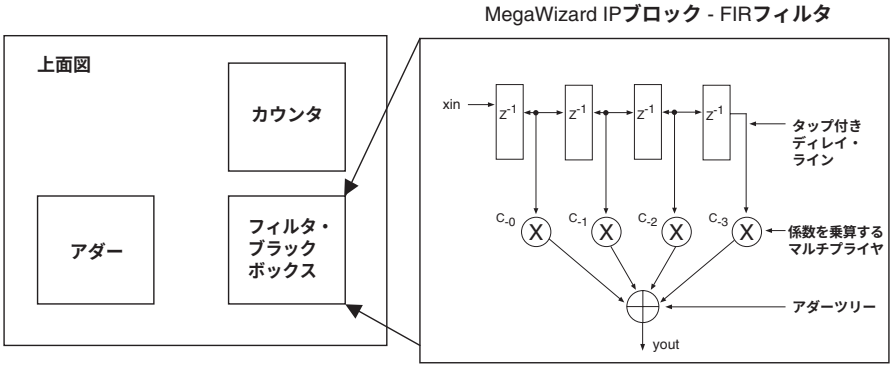


IP のインスタンス化

Altera MegaWizard Plug-In Manager で使用できる IP ブロックが暗号化されたフォーマットになっていて、サードパーティ合成ツールでは読み出せない場合、IP のインスタンス化には、合成ツール内で IP をブラック・ボックス化してから Quartus II ソフトウェアに暗号化された IP を読み込んで、配置配線を実行する必要があります。ブラック・ボックスのコンセプトを図 24 に示します。

IP デザイン・フローについて詳しくは、**www.altera.com/literature/lit-ip.html** の資料を参照してください。

図 24. IP のインスタンス化



LPM は合成ツールでは合成されないため、「ブラック・ボックス化」が必要です。

機能シミュレーション

機能シミュレーションは RTL デザインの機能性を検証します。以下は、サポートされるサード・パーティ EDA ツールのリストです。

- NC-SIM および Verilog-XL（Cadence 社から提供）
- VCS および VSS（Synopsys 社から提供）
- ModelSim®（Mentor Graphics® 社から提供）

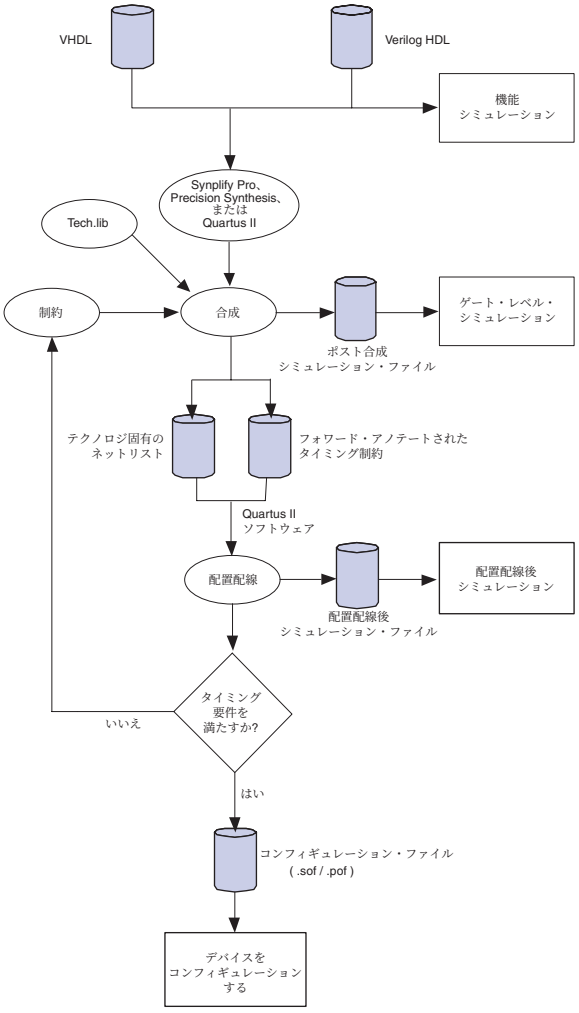
Quartus II ソフトウェアのネイティブ・シミュレータを使用してシミュレーションを実行することもできます。

合成

合成とは、RTL コードからゲート・レベルへのデザイン表現の変換プロセスのことです。図 25 は、標準的な合成フローを示します。ASIC ツールと比較して、複雑さとスクリプティングの面でFPGA合成ツールのほうがはるかに使いやすくなっています。業界標準の FPGA 合成ツールは、対応する ASIC ツールと比較してサポートする制約は限定されていますが、FPGA ツールは以下をはじめとして、一般的な ASIC 合成手法をすべてサポートしています。

- トップダウンまたはボトムアップ・アプローチ
- モジュール・ベース・デザイン・フロー
- スクリプティング

図 25. 標準的な論理合成デザイン・フロー



スクリプティング

Synplicity 社の Synplify や Synplify Pro、Mentor Graphics 社の Precision Synthesis などの FPGA 合成ツールは、ツール・コマンド言語 (Tcl) と SDC（Synopsys DC）フォーマットで書かれたスクリプトをサポートします。以下のサンプル・スクリプトは基本合成ステップを実行するのに標準的に使用されますが、これらは Synplify / Synplify Pro に適用されます。

```
project -new C:/test/fpga_risc8/risc.prj // 新しいプロジェクト
risc.prj をオープンする
add_file C:/test/fpga_risc8/src/risc8.v // プロジェクトにソース・
ファイルを追加する
set_option -technology APEX20KE // ターゲット・テクノロジを設定す
る
set_option -frequency 50.000000 // スピードを設定する
set_option -num_critical_paths 1 // レポートするクリティカル・パス
の数を設定する
set_option -num_startend_points 1 // タイミング・レポートの開始ポ
イント数を設定する
set_option -pipe 1 // パイプライン化オプションを TRUE に設定する
set_option -retiming 1 // リタイミングのオプションを TRUE に設定する
add_file -constraint test.sdc // test.sdc から Synopsys DC ス
タイル・コマンドを読み込む
project -run synthesis // 合成を実行する
project -save C:/test/fpga_risc8/risc.prj // プロジェクトを保存
する
```

SDC スクリプト

以下は制約を設定するための SDC スクリプトのサンプルです。

```
define_clock -name {clk} -freq 70.000 -clockgroup
default_clkgroup // クロックの作成
define_output_delay {dds_out[7:0]} 3.00 -ref clk:r // clk の立
ち上がりに対する出力遅延を設定する //
define_input_delay {expaddr[6:0]} 2.00 -ref clk:f //clk の立ち
下がりに対する出力遅延を設定する //
define_multicycle_path -from {i:ctl[0]} -to {i:accum[9:0]} 2
// マルチサイクル・パスの設定
define_false_path -from {i:ddsstep[7:0]} -to {i:sinout[7:0]}
// 偽のパスの設定
define_attribute {expread} syn_maxfan {4}
// 最大ファンナウト属性の設定
```

コマンド・ラインの実行

以下はコマンド・ラインを実行するための SDC スクリプトのサンプルです。

```
synplify_pro -batch <file_name>.tcl
```

スクリプティングおよびサポートされるSDCコマンドについて詳しくは、Synplify または LeonardoSpectrum のユーザおよびリファレンス・マニュアルを参照してください。

合成用サードパーティ EDA ツールのサポート

以下のサードパーティ・ツールを使用して合成を実行できます。

- Synplify Pro および Synplify（Synplicity 社から提供）
- LeonardoSpectrum および Precision Synthesis（Mentor Graphics 社から提供）
- FPGA Compiler II（Synopsys 社から提供）

上記のツールの他に Quartus II ソフトウェアを使用して合成を実行することもできます。

テスト合成

ベンダが FPGA デバイスの製造欠陥をテストすれば、メモリ BIST、SCAN 挿入、またはその他の製造欠陥の検出に標準的に使用されるテストが不要になります。これによって、複雑なテスト合成の作業が一切不要になります。

ゲート・レベル・シミュレーションおよびタイミング解析

業界標準のシミュレーション用 EDA ツールによって、ゲート・レベル・シミュレーションを実行することができます。また、Quartus II ソフトウェアに付属して、アルテラのすべての FPGA デバイス・ファミリ用のシミュレーション・ライブラリが出荷されます。

シミュレーションの実行について詳しくは、**www.altera.co.jp** に掲載されている以下のアプリケーション・ノートを参照してください。

- 「*AN 204: Using ModelSim-Altera in a Quartus II Design Flow*」

アルテラの FPGA デザイン・フローは、Quartus II ソフトウェアでサポートされるタイミング解析に加えて、Synopsys 社から提供されるスタティック・タイミング解析実行用の PrimeTime もサポートしています。大部分の FPGA ファミリに対応する PrimeTime ライブラリも Quartus II ソフトウェアに含まれています。

配置配線

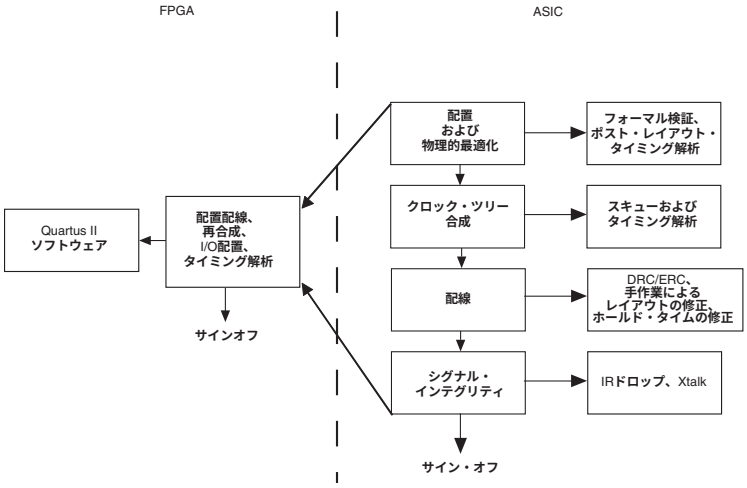
FPGA デザインの配置配線はアルテラからの Quartus II ソフトウェアを使用して実行されます。Quartus II ソフトウェアは、標準 EDIF、VHDL、および Verilog HDL ネットリスト・ファイルを読み込み、VHDL および Verilog HDL ネットリスト・ファイルを生成して、他の業界標準 EDA ツールへの簡便なインタフェースを実現します。

Quartus II ソフトウェアは、さまざまなタイプのネットリスト最適化を実行して、性能要求と面積要求を満たすことができます。Quartus II ソフトウェアは、ネットリスト最適化の他に、LogicLock™ ブロック・ベース・デザイン手法もサポートしており、t_{SU} および t_{CO} 要件を満足する高速入力 / 高速出力レジスタを可能にする最新機能を備えています。

Quartus II ソフトウェア内のタイミング・クロージャ・フロアプランによって、設計者はデバイスまたはロジック・セル（エンベデッド・セルおよび I/O セルを含む）内のロウまたはカラムにロジックを割り当てることができます。このフロアプランを使用すれば、**フィッタ**やユーザ・アサインメントによるロジック配置の表示、LogicLock 領域アサインメントの作成と表示、クリティカル・パス情報、フィジカル・タイミング見積もり、および配線の輻輳の表示を行うことができます。

図 26 は、FPGA と ASIC の配置配線プロセスを比較し、FPGA のバックエンド・タスクの実行 ASIC と比較して多くの利点を持つことを示します。これらの利点には、プロセス全体がシンプル、EDA ツールのコストが最低、「Time-to-Market」の短縮などが挙げられます。

図 26. FPGA と ASIC の配置配線プロセスの比較



配置配線後の検証

配置配線ネットリストの検証は、タイミング要件、論理等価性を検証し、電力消費を見積もるために実行されます。Quartus II ソフトウェアは、各種サードパーティEDA ツールをサポートするネットリストを生成し、さまざまな検証オペレーションを実行します。

ゲート・レベル・シミュレーション

シミュレーション・ツールを選択した後、Quartus II ソフトウェアはタイミング情報を含むネットリストを出力し、このネットリストと選択されたツールとの間に互換性があることを自動的に保証します。Quartus II ソフトウェアは、以下のいずれかを生成するオプションを備えています。

- 機能シミュレーションを実行する Verilog HDL または VHDL ネットリスト
- タイミング・シミュレーションを実行する Verilog HDL または VHDL ネットリストとタイミング情報（SDF ファイル）

スタティック・タイミング解析

タイミング解析に PrimeTime EDA ツールを選択すれば、Quartus II ソフトウェアは Verilog HDL または VHDL ゲート・レベル・ネットリスト、SDF ファイル、および Tcl スクリプトを出力して、PrimeTime 内でスタティック・タイミング解析を実行します。また、PrimeTime ライブラリが Quartus II ソフトウェアに付属して出荷されます。

フォーマル検証

Quartus II ソフトウェアは、徹底的な数学的手法を使用してデザインの機能性を検証するフォーマル検証をサポートします。アルテラの FPGA デザイン・フロー内では、合成されたネットリストと Quartus II のフィッティング後のネットリストとの間でフォーマル検証を実行できます。現在、フォーマル検証は Synplify Pro ツールで合成されたデザインに対してのみサポートされています。

消費電力見積もり

アルテラの FPGA デザイン・フローは 2 つの消費電力の見積もり方法をサポートします。1 つの方法は、オンライン・パワー・カリキュレータと呼ばれるデザイン・ユーティリティを使用します。各種デバイスの予測消費電力を標準条件に基づいて算出することができます。このオンライン・カリキュレータは、各デバイス・ファミリのデザイン・ユーティリティ・セクションにあります。

このカリキュレータについて詳しくは、**www.altera.co.jp/products/devices/dev-index.jsp** をご覧ください。

Quartus IIソフトウェアは、Model Technology™ ModelSim®ソフトウェアもサポートしており、消費電力見積もりデータを含むシミュレーションを実行します。設計者は、Quartus II ソフトウェアが消費電力見積もりデータを、APEX™ 20KE、APEX 20KC、および Mercury™ デバイス用に生成された Verilog Output File (**.vo**) または VHDL Output File (**.vho**) に含めるよう指定できます。ModelSim ソフトウェアは、ソフトウェア内でデザインをシミュレートした後、消費電力見積もりデータを使用して、デザインの消費電力を見積もるために Quartus II ソフトウェア内で使用できる電力入力ファイル（**.pwf**）を生成します。

消費電力見積もりに関する追加情報については、Quartus II ソフトウェアのヘルプをご覧ください。

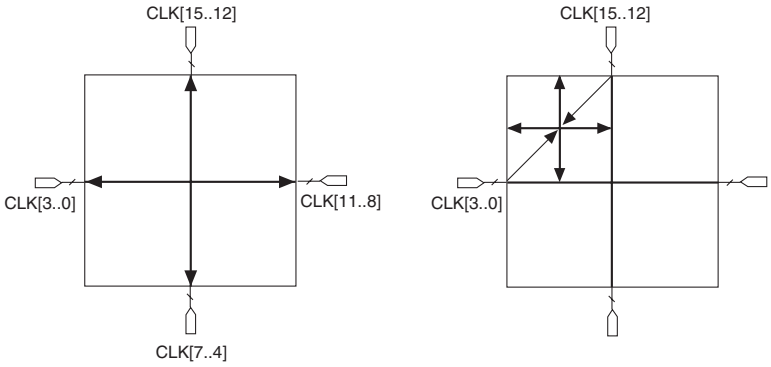
CTS（クロック・ツリー合成）

クロック・ツリー合成（CTS）は ASIC デザイン手法の重要なステップで、配置後に実行されます。CTS はクロック・ネットワークを構築してデザインでのレジスタ間のクロック・スキューを低減します。CTS は、フィジカル・シンセシスを実行する機能を備えたサードパーティEDA ツールまたはファンダリ提供ツールのいずれかによって実行されます。CTS には多大な時間を要し、また要求されるクロック・スキューが得られるまでに数回の繰り返しが必要な場合もあります。

アルテラの FPGA は、PLL リソースの他にグローバルまたはリージョナルの専用クロック・ネットワークを備えています。これらのクロック・ネットワークは、ドライブ強度が高くてスキューが小さく、CTS のない FPGA デザイン・フローを実現できるため、デザインのサイクル・タイムが短縮されます。使用可能なグローバル・クロック・リソース数はファミリおよびデバイスによって異なるため、必要なグローバル・クロック・リソース数を把握し、それに応じてデバイスを選択しなければなりません。図 27 は Stratix デバイスのグローバル・クロック・リソースとリージョナル・クロック・リソースを示します。

PLL およびグローバル・クロック・ネットワークについて詳しくは、該当するデバイスのデータ・シートを参照してください。

図 27. Stratix デバイスのクロック・リソース



グローバル・クロック・ピン数と配置の選択肢がほぼ無制限の ASIC デバイスとは異なり、FPGA デバイスのグローバル・クロック・ピン数と配置の選択肢の数は一定です。

デザインの実装に必要なクロック・リソースの数が、確実に FPGA デバイス内で使用可能なグローバル・クロック・リソースおよび高速クロック・リソース数以下になるようにしてください。クロック・ピンの配置も固定されており、クロック・ピンの物理的位置を活かすようなデザインの配置を行う必要があります。

Quartus II ソフトウェアの機能

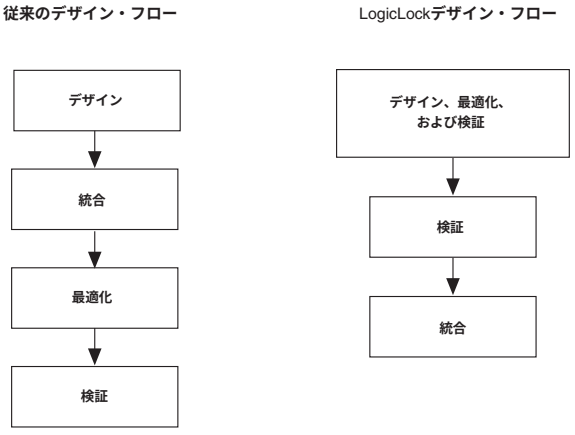
Quartus II ソフトウェアは、アルテラのプログラマブル・ロジック・デバイス（PLD）でロジックを設計するための、完全に統合された、アーキテクチャに依存しないパッケージ・ソフトウェアです。Quartus II ソフトウェアは、以下を含む広い範囲のロジック・デザイン機能を提供します：

- LogicLock インクリメンタル・デザイン
- Tcl スクリプト
- SignalTap® II ロジック・アナライザ
- SignalProbe™ デバッグ・テクノロジ
- 回路図、ブロック図、AHDL、VHDL、および Verilog HDL を使用したデザイン・エントリ
- フロアプランの編集
- リタイミングを含む強力なロジック合成
- 機能シミュレーションおよびタイミング・シミュレーション
- タイミング解析
- プログラミング・ファイルを生成するための、ソフトウェア・ソース・ファイルのインポート、作成、およびリンク
- デバイスのプログラミングと検証

LogicLock

Quartus II ソフトウェアは、LogicLock 機能を使用したブロック・ベース・アプローチを可能にします。この手法を使用して、各ロジック・モジュールの作成と実装を個別に行い、すべての最適化されたモジュールをトップレベル・デザインに統合することができます。図 28 に、従来のデザイン・フローと LogicLock デザイン・フローとの比較を示します。

図 28. 従来のデザイン・フローと LogicLock デザイン・フローとの比較

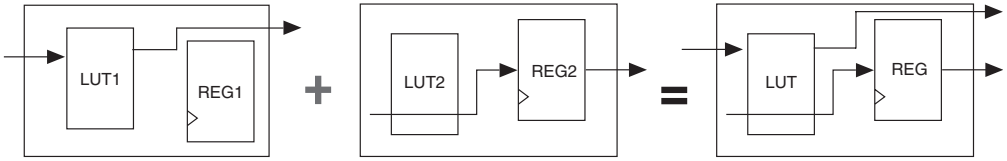


レジスタ・パッキング

レジスタ・パッキングは、図 29 に示すとおり、レジスタのみを使用するロジック・セルと LUT のみを使用するロジック・セルを結合して 1 つのロジック・セルに統合して面積を縮小します。レジスタ・パッキングには、以下の 4 つのオプションが用意されています。

- **オフ** : レジスタをパッキングしません。
- **ノーマル**: デフォルト設定でタイミングが損なわれないと予測されるとき、デフォルト設定でレジスタをパッキングします。
- **面積の最小化** : Quartus II ソフトウェアが積極的にレジスタをパッキングして、面積を縮小します。このソフトウェアはキャリー・アンド・カスケード・チェインの一部であるレジスタはパッキングしません。
- **チェインによる面積最小化** : Quartus II ソフトウェアがキャリー・アンド・カスケード・チェインのコンポーネントであるレジスタを含むレジスタを積極的にパッキングします。このオプションは、Stratix デバイス、Stratix GX デバイス、および Cyclone デバイスでのみ使用できます。

図 29. レジスタ・パッキング



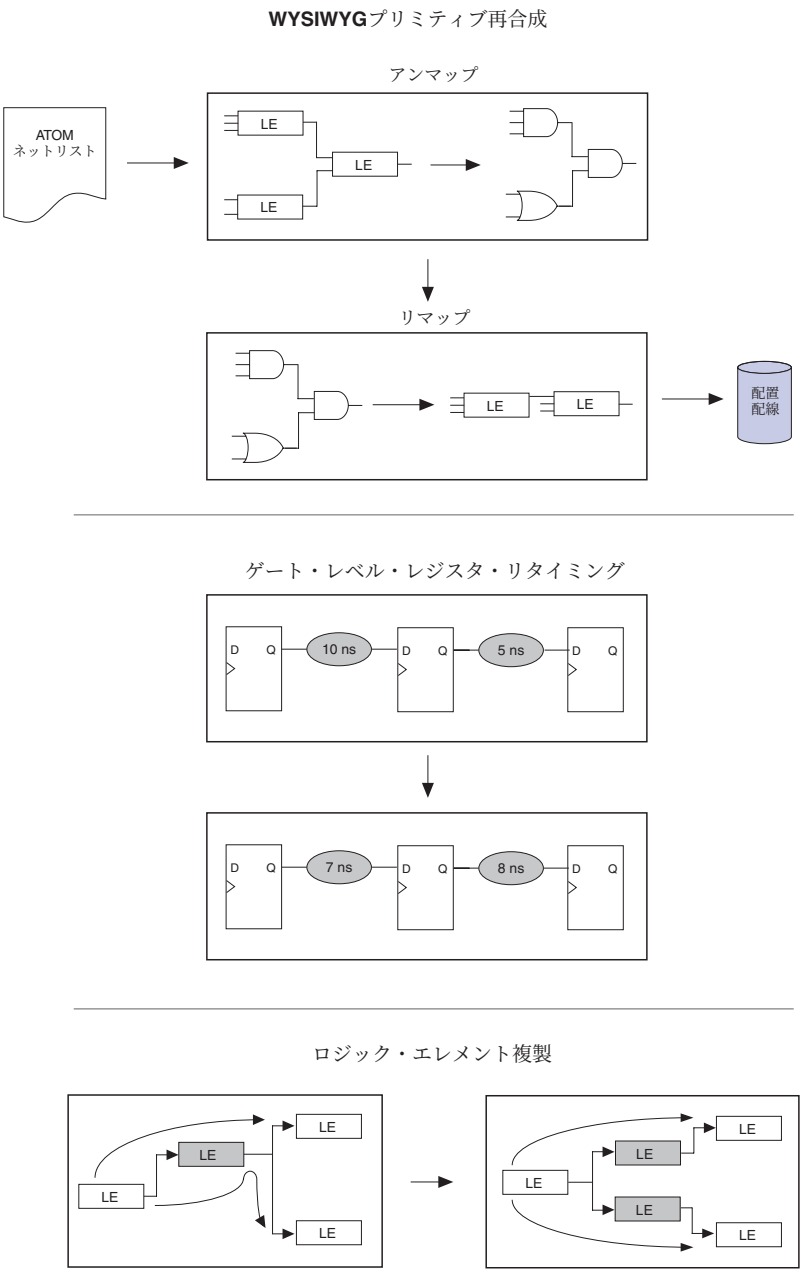
ネットリスト最適化

QuartusII ソフトウェアは、合成後および配置配線前に、デザインをさらに最適化するためのネットリスト最適化オプションを備えています。このオプションは使用する合成ツールに関係なく適用できます。デザインによっては、他のオプションよりも効果的なオプションがあります。以下の 3 つのオプションが用意されています。

- WYSIWYG プリミティブ再合成
- ゲート・レベル・レジスタ・リタイミング
- ロジック・エレメント複製

図 30 に、上記の各オプションを示します。

図 30. ネットリスト最適化の方法



Tcl スクリプティング

QuartusII ソフトウェアは、Tcl スクリプティングを Tcl コマンドとして使用可能なアプリケーション・プログラム・インタフェース（API）と併せてサポートします。QuartusII ソフトウェアによって作成された既存のプロジェクトから Tcl スクリプト・ファイル（**.tcl**）を生成することができます。あるいは、Tcl 言語テンプレートと Quartus II の Tcl テンプレートを使用して、Tcl スクリプトを作成することも可能です。

QuartusII ソフトウェア内、および他の EDA ソフトウェア・ツール（Synplicity 社の Synplify、Mentor Graphics 社の LeonardoSpectrum など）内から、Tcl スクリプトまたは個々の Tcl コマンドを実行できます。

以下の Tcl コードは、基本的な配置配線を実行するための Tcl スクリプトのサンプルです。

```
# Quartus II Tcl プロジェクト・パッケージのロード
package require ::quartus::project

}

# プロジェクトの作成とオープン
if {[project_exists test]} {
    project_open -cmp clock_sync test
} else {
    project_new test
}

# プロジェクトのアサインメント
set_global_assignment -name "VQM_FILE"
"..\\synplify\\clock_sync.vqm"
set_global_assignment -name "COMPILER_SETTINGS" "clock_sync"

# 合成、シミュレーション、およびタイミング検証ツールのアサインメント
set_global_assignment -name "EDA_SIMULATION_TOOL" -section_id
"test" "ModelSim (Verilog HDL output from Quartus II)"
set_global_assignment -name "EDA_TIMING_ANALYSIS_TOOL" -
section_id "test" "PrimeTime (Verilog HDL output from Quartus
II)"


# 入出力フォーマットと読み込み

set_global_assignment -name "EDA_INPUT_DATA_FORMAT" -
section_id "eda_design_synthesis" "EDIF"
set_global_assignment -name "EDA_OUTPUT_DATA_FORMAT" -
section_id "eda_design_synthesis" "EDIF"
```

```
# デバイスおよびファミリの選択

set_global_assignment -name "FAMILY" "Stratix"
set_global_assignment -name "DEVICE" "EP1S40F1508C5"

# タイミング仕様

set_global_assignment -name "FMAX_REQUIREMENT" "40.0 MHz"
set_global_assignment -name "TCO_REQUIREMENT" "25ns"
set_global_assignment -name "TPD_REQUIREMENT" "25ns"
set_global_assignment -name "TSU_REQUIREMENT" "25ns"

# プロジェクトをコンパイルし、
# エラーがある場合は、"qexit" を使用して終了する
if [catch {qexec "quartus_fit $project_name"} result] {
    qexit -error
}
if [catch {qexec "quartus_tan $project_name"} result] {
    qexit -error
}

#------ レポートから Fmax を報告 ------#
set actual_fmax [get_fmax_from_report]
puts ""
puts "-------------------------------------------------"
puts "Required Fmax: $required_fmax Actual Fmax:
$actual_fmax"
puts "-------------------------------------------------"

#------ プロジェクトをクローズする ------#
project_close
```

Tclスクリプティングについて詳しくは、「*AN 195: Scripting with Tcl in the Quartus II Software*」と Quartus II ソフトウェアのヘルプを参照してください。

モジュラー QuartusII ソフトウェア

QuartusII ソフトウェア・バージョン 3.0 により、アルテラはモジュラー実行コマンドの柔軟性を提供します。モジュラー実行コマンドは、合成からシミュレーションおよびプログラミングまでの FPGA デザイン・フローの各ステップに対して個別のプログラムを提供します。モジュラー実行コマンドの利点には、メモリ要件の軽減と性能の向上、デザイン・フローの各ステップに対するコマンド・ライン制御、Makefiles を含むスクリプト化されたデザイン・フローとの容易な統合、使いやすい Quartus II グラフィカル・ユーザ・インタフェース（GUI）に基づくデザイン・フローとの完全な互換性などがあります。

イン・システム検証

ASIC デバイスでは、テスト・ポイントとテスト・ピンを使用して各種ノードを検査し、問題の要因の識別を支援します。これらのテスト・ポイント / ピンとロジック・アナライザにより、大部分の問題の原因を効果的に特定することができます。

このテスト方法は FPGA でも使用されています。ただし、リコンフィギュレーション可能な FPGA では、さらに多くの利点が得られます。アルテラの FPGA は、テスト・ポイントとテスト・ピンによる方法に似た自動化プロセスを使用して内部ノードを未使用 I/O ピンに配線する SignalProbe™ ユーティリティをサポートしています。ASIC の手法と同様に、ロジック・アナライザにこれらの SignalProbe ピンを使用することもできます。ただし、FPGA はリコンフィギュレーション可能なので、SignalTap II ロジック・アナライザを FPGA 自体に埋め込むことができます。

SignalTap II エンベデッド・ロジック・アナライザ

SignalTap II エンベデッド・ロジック・アナライザは、内部ノードのデータをキャプチャし、ダウンロード・ケーブルを通して Quartus II ソフトウェアにリアルタイムでデータを転送し、波形表示させます。このエンベデッド・ロジック・アナライザは、標準の外部ロジック・アナライザと同じ方法で、トリガ位置およびトリガ・イベントをサポートします。

SignalTap II ロジック・アナライザでは、実行中に測定ノードを変更することができます。このツールは、複数の SignalTap II インスタンスを 1 つのデバイスにインスタンス化することによって、デザインでの多数のクロック・ドメインもサポートします。

SignalTap II ロジック・アナライザは、プログラミング・インタフェースを使用して、コンピュータに返信するため、テスト・ピンを追加する必要はありません。キャプチャされたデータは、デバイスの内部 RAM に保存された後、JTAG 通信ポートを介してデバイスからストリーミングされます。これは、BGA パッケージを使用していて使用可能なピンへのアクセスが困難あるいは不可能なときに便利です。

SignalProbe 機能

SignalProbe 機能はインクリメンタル配線をサポートしているため、設計者は信号を素早く I/O ピンに配線して、デザインに影響を与えずに効率的に信号を検証することができます。デバイスのメモリが制限されていて JTAG 通信ポートへのアクセスがない場合は、SignalProbe 機能で信号のデバッグとハードウェア検証を実行することができます。

Quartus II ソフトウェアでは、事前に指定された SignalProbe ピンに配線するノードの選択を変更することができます。次に、これらの配線された SignalProbe ピンにより、ロジック・アナライザを使用して選択した内部ノードを解析することができます。

SignalProbe 機能によって、完全なリコンパイルに必要な時間の 5% の時間で信号を再配線することができます。SignalProbe 機能によって、アルテラの設計者は内部デザイン信号に素早くアクセスしてシステム・レベルのデバッグを行うことができます。

SignalProbe 機能について詳しくは、Quartus II ヘルプを参照してください。

BSDL（Boundary Scan Description Language）テスト

ASIC デバイスと FPGA デバイスのどちらも BSDL テストを使用してピンの接続性と機能性を保証します。表面実装パッケージおよび PC ボードの製造の進歩によってボードの小型化が進み、外部テスト・プローブなどの従来式のテスト方法の実施が困難になっています。BSDL テストでは、物理的なテスト・プローブを使用しないでピンの接続をテストでき、またデバイスの通常動作中に機能データをキャプチャすることができます。

アルテラのデバイスはすべて BSDL テストをサポートします。アルテラの FPGA デバイスには、JTAG（TAP）コントローラが既にデバイスに埋め込まれており、ASIC デバイスと異なりエンジニアリング作業は必要ありません。アルテラはまた各ファミリの各デバイス用の BSDL ファイルも提供しています。これらの BSDL ファイルはアルテラの Web サイトで閲覧できます。

BSDL テストについて詳しくは、「*AN 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*」を参照してください。

デバイスのプログラミング（プロトタイプ）

ASIC デバイスと FPGA デバイスの違いの 1 つは、FPGA デバイスとは異なり ASIC の機能は変更できないことです。ASIC デバイスがファンダリから戻ると、デバイス機能が固定されるだけで、多くのことを行うことはできません。他方、FPGA デバイスは何回もリコンフィギュレーションでき、はるかに高い柔軟性を提供します。

プロトタイプ作成段階では、デバイスの機能を変更し、拡張機能とバグ修正を追加する能力が求められます。FPGA デバイスはさまざまな方法でコンフィギュレーションが可能です。

アルテラの SRAM ベース FPGA デバイス（Stratix、Stratix GX、Cyclone デバイスなど）は、表 2 に示すとおり、多くの方法でコンフィギュレーションできます。

表 2. アルテラの SRAM ベース・コンフィギュレーション手法

コンフィギュレーション手法	デバイス・ファミリ	標準的な使用法
コンフィギュレーション・デバイス	APEX II、APEX 20K、Mercury、ACEX 1K、FLEX 10K、	EPC16、EPC8、EPC2、EPC1、または EPC1441 のコンフィギュレーション・デバイスによるコンフィギュレーション
	FLEX 6000	EPC1 または EPC1441 のコンフィギュレーション・デバイスによるコンフィギュレーション
パッシブ・シリアル（PS）	APEX II、APEX 20K、Mercury、ACEX 1K、FLEX 10K、FLEX 6000	シリアル同期マイクロプロセッサ・インタフェースと、MasterBlaster™ 通信ケーブルまたは ByteBlasterMV™ パラレル・ポート・ダウンロード・ケーブルによるコンフィギュレーション *(1)*
パッシブ・パラレル同期（PPS）	APEX II、APEX 20K、Mercury、ACEX 1K、FLEX 10K	パラレル同期マイクロプロセッサ・インタフェースによるコンフィギュレーション
ファースト・パッシブ・パラレル（FPP）	APEX II	各クロック・サイクルで 8 ビットのコンフィギュレーション・データがロードされるパラレル同期コンフィギュレーション・デバイスまたはマイクロプロセッサ・インタフェースによるコンフィギュレーション PPS より 8 倍高速
パッシブ・パラレル非同期（PPA）	APEX II、APEX 20K、Mercury、ACEX 1K、FLEX 10K	パラレル非同期マイクロプロセッサ・インタフェースによるコンフィギュレーションこの方式では、マイクロプロセッサはターゲット・デバイスをメモリとして扱います。
パッシブ・シリアル非同期（PSA）	FLEX 6000	シリアル非同期マイクロプロセッサ・インタフェースによるコンフィギュレーション
JTAG（Joint Test Action Group）	APEX II、APEX 20K、Mercury、ACEX 1K、FLEX 10K	IEEE Std. 1149.1（JTAG）ピンを使用したコンフィギュレーション *(2)*

表 1 の注：

(1) MasterBlaster 通信ケーブルは、標準的な PC シリアル・ハードウェア・インタフェースまたは標準的なユニバーサル・シリアル・バス（USB）ハードウェア・インタフェースを使用して、コンフィギュレーション・データを APEX II、APEX 20K、Mercury、ACEX 1K、FLEX 10K、および FLEX 6000 の各デバイスにダウンロードします。この通信ケーブルは、5.0 V、3.3 V、2.5 V、または 1.8 V の V_{CC} による動作をサポートし、Quartus II ソフトウェアによってサポートされます。MasterBlaster ケーブルについて詳しくは、「*MasterBlaster Serial/USB Communications Cable Data Sheet*」を参照してください。

(2) JTAG ピンを使用して FLEX 6000 デバイスをコンフィギュレーションすることはできませんが、JTAG バウンダリ・スキャン・テストを実行することはできます。

一般に、FPGA デバイスをコンフィギュレーションする方法は 2 つあります。

- ダウンロード・ケーブルを使用
- コンフィギュレーション・デバイスを使用

ダウンロード・ケーブルの使用

SRAM ベース FPGA デバイスを使用する場合、プロトタイプ作成段階ではダウンロード・ケーブルの方が多く使用されます。プロトタイプ作成段階では、必然的にデザインの変更と修正が必要になると予想されます。ASIC デザインでは、デバイスを取り外して新しい ASIC デバイスに交換する必要があります。あるいはボード全体が廃棄されます。FPGA デバイスの場合、ボードに実装されたままデバイスをプログラムすることができます。

Quartus II ソフトウェアでは、ダウンロード・ケーブルを通してデバイスをリコンフィギュレーションすることができます。コンピュータのシリアル・ポート、USB ポート、またはパラレル・ポートに接続するのに使用できる各種ダウンロード・ケーブルがあります。JTAG コンフィギュレーションまたはパッシブ・シリアル・コンフィギュレーションでは、ダウンロード・ケーブルを接続できます。

コンフィギュレーション・デバイスの使用

SRAM ベースの FPGA デバイスは、ボードに電源を投入するたびにコンフィギュレーションする必要があります。アルテラはこの目的に使用できる各種サイズのコンフィギュレーション・デバイスを提供しています。

これらの EEPROM ベース・コンフィギュレーション・デバイスは、コンフィギュレーション・データを格納し、対象となるデバイスをコンフィギュレーションします。標準的なシナリオの 1 つは、FPGA デバイスをコンフィギュレーション・デバイスでパッシブにシリアルにコンフィギュレーションすることです。ボードに電源が投入されると、コンフィギュレーション・デバイスは FPGA デバイスに自動的にシリアル・データ・ストリームを送信し、ボードをコンフィギュレーションします。

あるいは、オンボード・メモリにコンフィギュレーション・データを格納して、コントローラとして機能する別の PLD（通常は EPROM ベース・デバイスまたはマイクロプロセッサ）を使用することもできます。ボードに電源が投入されると、コントローラ・デバイスは、メモリからターゲットの FPGA デバイスにコンフィギュレーション・データを送信します。

これらのコンフィギュレーション方法は、最終製品のボードに最適です。

HardCopy デバイス

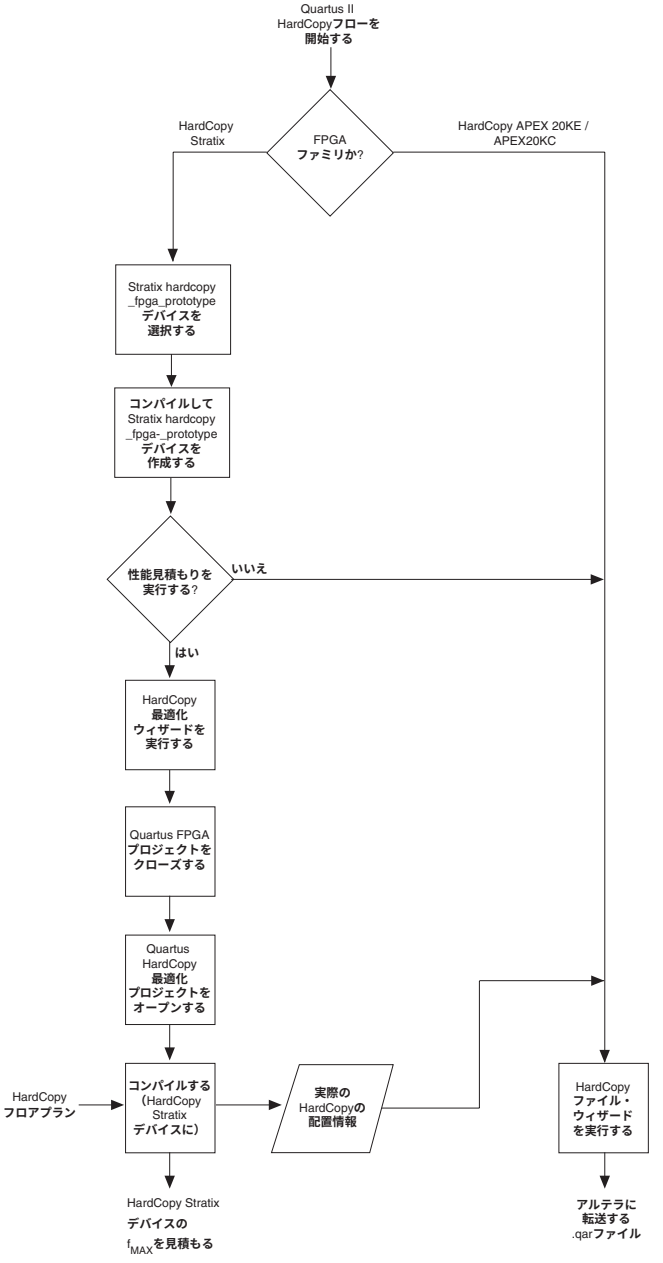
アルテラは、コスト効果が高いストラクチャードASICデバイスである HardCopy™ デバイス・ファミリを提供しています。HardCopy デバイスは、高集積 FPGA の柔軟性を拡張し、コスト効果の高い量産ソリューションを実現します。アルテラの FPGA から HardCopy デバイスへの変換プロセスは、高集積 system-on-a-programmable-chip から低コストの代替デバイスへのシームレスな移行を最小のリスクで実現します。HardCopy デバイスを利用すれば、プロトタイプから生産に至るまでアルテラの SOPC ソリューションを活用することができ、しかもコストの削減と「Time-to-Market」の短縮が同時に達成されます。

HardCopy デバイスの最大の利点は、設計者がデバイスの変換プロセスに関わらずに済むことです。ASIC デバイスの開発とは異なり、HardCopy デザインはテスト・ベンチ、テスト・ベクタ、タイミングおよび機能シミュレーションを一切必要としません。HardCopy の変換プロセスで必要とされるのは、Quartus II ソフトウェアで生成した出力ファイルだけです。アルテラは、変換処理を含めわずか 8 週間でファンクション・プロトタイプをお届けします。

HardCopy デバイスについて詳しくは、「*HardCopy Device Handbook, Volume 1*」を参照してください。

図 31 は、Quartus II ソフトウェアによる **.sof** ファイル生成後の HardCopy デザイン・フローを示します。

図 31. HardCopy デザイン・フロー



FPGA の経済的メリット

アルテラは、FPGA の経済的メリットを ASIC と比較して計算する手段も提供しています。このツールは、入力パラメータを取り込んで予想総コストと予想総収益の見積もりを算出します。このツールはアルテラの Web サイトで入手できます。

まとめ

ASIC デザイン・フローと FPGA デザイン・フローと比較すると（3 ページの図 2 参照）わかるように、ASIC デザイン・フローは FPGA に必要なステップより多くのステップで構成されており、関連作業を実行するのに FPGA より多くの EDA ツールを使用する必要があります。

ASIC デザイン・フローの欠点

ここでは、FPGA デザイン・フローと比較した場合の ASIC デザイン・フローの欠点をいくつか示します。

- **初期計画およびフィジカル・デザインに関する専門知識**
  ASIC デザイン手法は、以下の初期のフィジカル・プランニングから構成されています。

  - I/O 配置
  - メモリ配置
  - 電力およびフィジカル・プランニング

  この初期のフィジカル・プランニングにより、デザイン段階の早期にデバイスの面積とスピードを把握し、管理することができます。これらの作業を実行するには、デザイン・チーム内にフィジカル・デザインに関する専門知識を持つ設計者もいなければなりません。

- **テスト合成**
  スキャン・セルとメモリ BIST ロジックによって、面積および製造欠陥を検出するためのテスト・オーバヘッドが増大します。

- **ファンダリの介在**
  バックエンド・タスクには、デザイン・チームとファンダリ間でのフィジカル・デザイン・プランニングが伴います。ファンダリの要員がデザインの配置配線を実行し、デザイン・チームが検証とタイミング解析を担当します。最終サイン・オフまでに繰り返し作業が発生し、その結果「Time-to-Market」に遅延が生じます。

- **EDA ツール**
  FPGA デザイン・フローと比較して、EDA ツールの数が多くなるためコストが増加し作業が複雑になります。

- **NRE コスト**
  デザインのリスピンによって、NRE コストが増大します。また、マスクがきわめて高価で、0.13µm テクノロジを使用して製造される 1 枚のマスクにかかるコストは約 850,000 ドルです。

- **ファームウェア開発**
  デザイン製造後にファームウェアのテストを行う必要があり、ファームウェアに変更があると「Time-to-Market」に遅延が生じます。

- **「Time-to-Market」と高いリスク**
  ASIC デザイン・サイクルの期間は、FPGA デザイン・フローと比較してかなり長く、デザインの変更によって高い NRE コストと「Time-to-Market」の遅延が生じます。

FPGA デザイン・フローの利点

ここでは、FPGA デザイン手法の利点を示します。

- **design-for-test ツールが不要**
  FPGA デバイスはスキャンのないデザインを行うことができ、オリジナル・デザインとスキャン・デザイン間のフォーマル検証が不要です。

- **短期間でのプロトタイプ作成と量産**
  FPGA デザインは、ASIC と比較して 10 ～ 12 週間という短いリード・タイムを実現できます。デバイスを選択した後、デザインが完成するまでに余裕を持ってパーツを注文し、設計者の準備が整い次第パーツを入手することも可能です。

- **HardCopy**
  アルテラが提供するストラクチャード ASIC デバイスの HardCopy ファミリは、ASIC よりもコスト効果の高い製品です。

- **ファームウェア開発**
  デザインのプロトタイプ作成後、すぐにファームウェアの開発とテストを行うことが可能で、ファームウェアやデザインの変更は短いターンアラウンド時間で実行できます。

- **NRE コスト と機能の柔軟性**
  デバイスはリコンフィギュレーション可能なため、デザインのリスピンに関連する NRE コストが不要で、またデザイン変更にも容易に対応でき、ASIC の数ヶ月に対してわずか数分しかかかりません。

- **配置配線**
  アルテラの Quartus II ソフトウェアは、配置配線を実行し、FPGA の構築に必要なすべてのファイルを生成します。このため、外部ファンダリとの連携が不要になります。Quartus II ソフトウェアは、業界標準のシミュレーション・ツールやタイミング検証 EDA ツールで使用するのに必要なネットリストも生成します。

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com