パーシャル リコンフィギュレーション ユーザー ガイド

UG702 (v13.1) 2011 年 3 月 1 日

XILINX®

本資料は英語版 (v 13.1) を翻訳したもので、内容に相違が生じる場合には原文を優先します。
資料によっては英語版の更新に対応していないものがあります。
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

改定履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改定内容
2010 年 5 月 3 日	12.1	ISE 12.1 初期リリース
2010 年 7 月 23 日	12.2	ISE 12.2 リリース用にアップデート
2010 年 10月 5日	12.3	ISE 12.3 リリース用にアップデート
2011 年 3月 1 日	13.1	ISE 13.1 リリース用にアップデート

目次

XILINX®

第 1 章

概要

パーシャル リコンフィギュレーションとは、パーシャル コンフィギュレーション ファイルを読み込んで動作中の FPGA デザインを修正することです。本書では、「パーティション」というモジュール デザインの手法を使用して、部分的にリコンフィギュレーション可能な FPGA デザインを作成およびインプリメントする方法について説明します。デザインのモジュール インスタンスは、新しいハードウェア ファンクションを定義するパーシャル ビット ファイルに変換されます。『Differencing Method for Partial Reconfiguration』(XAPP290)に記述される差分方法などのその他の手法については、本書では説明しません。その他のマニュアルについては、付録 C「その他のリソース」を参照してください。

本書の対象は、次のとおりです。

- 部分的にリコンフィギュレーション可能な FPGA デザインを作成する設計者用です。
- ザイリンクス® ISE® Design Suite および PlanAhead™ ソフトウェアなどの FPGA デザイン ソフトウェアに既に精通している方を対象とします。
- 特に ISE Design Suite 13.1 用に記述されています。このリリースでサポートされるのは、Virtex®-4、Virtex-5、Virtex-6 デバイスのパーシャル リコンフィギュレーションのみです。

パーシャル リコンフィギュレーションの概要

FPGA では、デザインを変更しても製造し直す必要なく、オンサイト プログラムまたはリプログラムが柔軟に実行できます。パーシャル リコンフィギュレーション (PR) は、この柔軟性をさらに推進したもので、パーシャル コンフィギュレーション ファイル (通常はパーシャル ビット ファイル) を読み込むことで、動作中の FPGA デザインが修正できます。フル ビット ファイルで FPGA をコンフィギュレーションした後に、パーシャル ビット ファイルをダウンロードすると、デバイスの該当パーツに対してアプリケーションをまとめて実行しなくても、FPGA のリコンフィギャブル領域を変更できます。

図 1-1 は、パーシャル リコンフィギュレーションの概念を示しています。

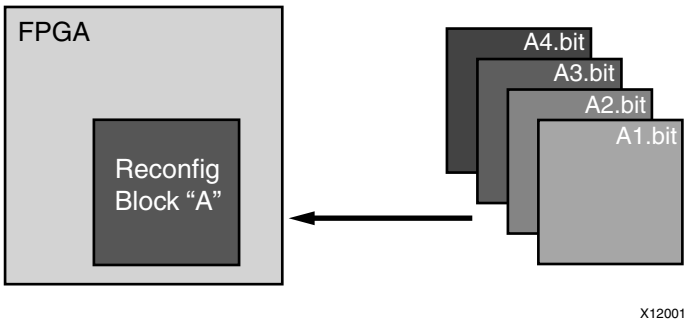

図 1-1 : パーシャル リコンフィギュレーションの基本的概念

図に示すように、Reconfig Block A にインプリメントされるファンクションは、A1.bit、A2.bit、A3.bit、または A4.bit などの複数のパーシャル ビット ファイルのいずれか 1 つをダウンロードすると修正できます。FPGA デザイン ロジックは、リコンフィギャブル ロジックとスタティック ロジックの 2 種類に分類されます。図の FPGA ブロックのグレーの部分は、スタティック ロジックを表し、Reconfig Block “A” と記述された黒い部分がリコンフィギャブル ロジックを表しています。パーシャル ビット ファイルが読み込まれても、スタティック ロジックは動作したままで、まったく影響を受けません。リコンフィギャブル ロジックはパーシャル ビット ファイルの内容で置き換えられます。

1 つの FPGA デバイスにハードウェアをダイナミックに時分割多重化できると効果的なのには、多くの理由があります。

この理由は、次のとおりです。

- 指定ファンクションをインプリメントするために必要な FPGA デバイスのサイズを削減するので、コストおよび消費電力も削減できます。
- アプリケーションに対して使用可能なアルゴリズムまたはプロトコルが柔軟に選択できます。
- デザインを保護したまま新技術を使用可能になります。
- FPGA のフォールト トレランスを改善します。
- コンフィギャブルな計算を促進します。

サイズ、重さ、電力、コストを削減するだけでなく、パーシャル リコンフィギュレーションなしにインプリメントできない新しいタイプの FPGA デザインに対応します。

用語

次は、このマニュアルで使用されるパーシャル リコンフィギュレーションに特有の用語です。

ボトムアップ合成

ボトムアップ合成は、1 つのプロジェクトまたは複数プロジェクトで、モジュールごとにデザインを合成する方法で、パーティションごとに別々のネットリストが必要です。これらのバウンダリを越えた最適化は実行されないので、デザインの各部分が別々に合成されます。最上位レベル ロジックは、パーティション用にブラックボックスを使用して合成される必要があります。

コンフィギュレーション

コンフィギュレーションは、リコンフィギャブル パーティションごとに 1 つのリコンフィギャブル モジュールを含む完全なデザインです。パーシャル リコンフィギュレーション FPGA プロジェクトには、多くのコンフィギュレーションが含まれることがあります。コンフィギュレーションはそれぞれ 1 つのフル ビット ファイルを生成するほか、リコンフィギャブル モジュールごとに 1 つのパーシャル ビット ファイルを生成します。

コンフィギュレーション フレーム

コンフィギュレーション フレームは、FPGA コンフィギュレーション メモリ空間の中で最小のアドレス指定可能なセグメントです。リコンフィギャブル フレームは、これらの最下位レベルのエレメントの一部から構築されます。

フレーム

フレームは、FPGA デバイス内の最小のリコンフィギャブル領域を表します (このマニュアルではコンフィギュレーション フレームを除く)。リコンフィギャブル フレームのビットストリーム サイズは、フレーム内に含まれるロジック タイプによって異なります。

内部コンフィギュレーション アクセス ポート (ICAP)

内部コンフィギュレーション アクセス ポート (ICAP) は、元々 SelectMAP インターフェイスの内部バージョンです。詳細は、該当デバイスのコンフィギュレーション ユーザー ガイドを参照してください。

パーシャル リコンフィギュレーション (PR)

パーシャル リコンフィギュレーション (PR) は、パーシャル コンフィギュレーション ファイルをダウンロードして、動作中の FPGA デザインのロジックを修正することを示します。

パーティション

パーティションは、階層バウンダリでユーザーにより定義され、デザイン再利用の際に考慮されるデザインの論理セクションです。パーティションは、新規としてインプリメントされるか、前のインプリメンテーションから保持されることもあります。保持されたパーティションには、同一の機能だけでなく、同一のインプリメンテーション結果も保持されます。

パーティション ピン

パーティション ピンは、スタティック ロジックとリコンフィギャブル ロジック間の論理および物理接続です。パーティション ピンは、すべてのリコンフィギャブル パーティション ポートに対して自動的に作成されます。

プロキシ ロジック

プロキシ ロジックとは、専用配線以外で、パーティション ピンごとにソフトウェアで自動的に挿入される 1 つの LUT1 エレメントです。プロキシ ロジックは固定される必要があり、スタティック ロジックとリコンフィギャブル ロジック間のインターフェイスとしての既知のポイントです。

リコンフィギャブル ロジック

リコンフィギャブル ロジックとは、リコンフィギャブル モジュールの一部である論理エレメントのことです。これらの論理エレメントは、パーシャル ビット ファイルが読み込まれると変更されます。LUT、フリップフロップ、BRAM、DSP ブロックおよび I/O など、ほとんどのタイプの論理コンポーネントがリコンフィギュレーションできます。

リコンフィギャブル モジュール (RM)

リコンフィギャブル モジュール (RM) とは、リコンフィギャブル パーティションであるインスタンスによってインスタンシエートされるとインプリメントされるネットリストまたは HDL 記述のことです。1 つのリコンフィギャブル パーティションに対して複数のリコンフィギャブル モジュールがあることもあります。

リコンフィギャブル パーティション (RP)

リコンフィギャブル パーティション (RP) とは、インスタンスをリコンフィギャブルとして定義するインスタンシエーションの属性セットです。インスタンスのリコンフィギャブル パーティション属性は、NGDBuild、MAP、および PAR などのソフトウェア ツールで検出され、正しく処理されます。

インスタンスがリコンフィギャブル パーティションの場合、リコンフィギャブル パーティションは「インスタンス」とよく同じ意味で使用されます。

スタティック ロジック

スタティック ロジックとは、リコンフィギャブル モジュールには含まれない論理エレメントです。この論理エレメントが部分的にリコンフィギュレーションされることはなく、リコンフィギャブル パーティションがリコンフィギュレーションされる場合、常にアクティブになります。スタティック ロジックとは、最上位レベル ロジックのことです。

ISE 13.1 のパーシャル リコンフィギュレーション デザインの基準

パーシャル リコンフィギュレーション (PR) は、ISE® Design Suite のアドバンス フローです。このソフトウェアにはさまざまな機能が含まれていますが、PR プロジェクトを始める前に、まず次の要件を理解してく必要があります。

各項目の詳細は、このユーザー ガイドで説明されます。

デザイン要件とガイドライン

- パーシャル リコンフィギュレーションには、ISE 12.1 以降のバージョンを使用する必要があります。
- デバイス サポート :Virtex®-4、Virtex-5、Virtex-6
  - これらのファミリのすべての派生デバイスがサポートされます。
  - Xilinx® 7 シリーズ デバイス (Virtex-7、Kintex-7、Artix-7) はまだサポートされていません。
- PR は、PlanAhead™ またはコマンド ラインでのみサポートされます。Project Navigator ではサポートされません。

- エレメント タイプごとにリコンフィギャブル領域を定義するには、フロアプランが必要になります。
  - 可能であれば、フレーム/クロック領域のバウンダリに対してアライメントしておくと、効率的です。
- ボトムアップ合成 (複数ネットファイルの作成) とリコンフィギャブル モジュールのネットリスト ファイルの管理は、ユーザーの責任で行ってください。
  - 合成は PlanAhead の外部で実行します。どの合成ツールでも使用できます。
- パーシャル　リコンフィギュレーション中にリコンフィギャブル領域とデザインのスタティック部分の接続を解除するには、ロジックをデカップリングしてください。
  - リコンフィギャブル エレメントが FPGA 外部にある場合、デカップリングはオフチップで実行される必要があります。
- 標準的なタイミング制約がサポートされるほか、必要であればその他のタイミング バジェット機能も使用できます。
- デザインを問題なく完成させるために、独自のデザイン ルール チェック (DRC) が含まれています。
- PR デザインでは、パーシャル リコンフィギュレーションの開始と、パーシャル ビット ファイルを FPGA 内部で出力するか、システム デザインの一部として出力するかを考慮する必要があります。
- すべてのインプリメンテーション オプションが PR フローで使用できるわけではありません。MAP コマンドの -global_opt オプションとその子オプションおよび SmartGuide™ は、デザイン全体の最適化を実行するので、パーティションまたは PR フローでは使用できません。
- -power オプションは、MAP と PAR のどちらにも使用できますが、すべてのオプションが使用できるわけではありません。MAP で high および xe 値を指定すると、デザインをフラットにするのに必要な Intelligent Clock Gating 機能が起動されますので、パーシャル リコンフィギュレーションでは使用できません。
- リコンフィギャブル パーティションには、パーティション用にインプリメントされるさまざまなリコンフィギャブル モジュールで使用されるように、すべてのピンのスーパーセットが含まれる必要があります。このため、一部のモジュール パターンの入力または出力が未使用になります。未使用の入力は、モジュール内でどこにも接続されないままになり、インプリメンテーション ツールではそれを無視できることを示すメッセージが表示されます。未使用の出力は 1 に接続されます。リコンフィギャブル パーティションには、1 つのモジュールに使用されても別のモジュールでは使用されないようなピンが含まれることがあるので、PR フローの場合、PXML ファイルで BoundaryOpt 制約をパーティションに適用することはできません。

デザイン パフォーマンス

- パフォーマンスの測定基準はデザインによって異なりますが、階層デザイン手法を使用すると、悪影響を最低限に抑えることができます。階層デザインの詳細は、『階層デザイン手法ガイド』(UG748)およびホワイトペーパー「Repeatable Results with Design Preservation」(WP632) を参照してください。ただし、ほとんどのデザインがシリコン分離に必要なその他の制限の影響を受ける可能性があります。

  通常、次のようになります。

  - クロック周波数が 10% 低下
  - 記録密度が 80% スライス未満

- これらの追加要件を考慮すると、ほとんどの場合デザイン実行時間は増加します。MAP が最も影響を受けますが、NGDBuild および PAR も PR デザインの処理の影響を受けることがあります。
- リコンフィギャブル領域が小さすぎたり、長方形以外の形で形成される場合は、配線が困難になることもあります。

設計に関する考慮事項

- すべてではありませんが、ほとんどのコンポーネント タイプはリコンフィギュレーションできます。
  - グローバル クロックとクロック修正ロジックは、スタティック領域に含まれる必要があります。
    - BUFG、MMCM、PLL、DCM および同様のものが含まれます。
    - BSCAN や STARTUP などのアーキテクチャ特有のコンポーネントは、それぞれデザインのスタティック領域に含まれている必要があります。
- IP をインプリメントするのに使用されるコンポーネントのために、IP 制限が発生することがあります。具体的な例は、次のとおりです。
  - ChipScope ICON (BUFG)
  - グローバル バッファの付いた EDK ブロック
  - MIG コントローラ (MMCM)
- スタティック領域とリコンフィギャブル領域の間には双方向インターフェイスは使用できません。
- 専用の暗号化は、Virtex-6 でサポートされるほか、Virtex-5 の場合は IP コアを介してサポートされます。
  - ユーザーは独自のソフトウェア暗号化エンジンを構築してパーシャル ビット ファイルを修正でき、FPGA 内にハードウェア暗号化エンジンを構築して暗号化のニーズを処理することもできます。
- Virtex デバイスの場合、パーシャル リコンフィギュレーションの終わりに専用の CRC 機能はありませんが、パーシャル ビット ファイルの統合が有効かどうかを、ビット ファイル出力機能の一部として挿入される IP コアを介してチェックできます。

特定の IP ソリューションもありますが (ザイリンクス アプリケーション ノート 『PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration』 (XAPP887) 参照)、前述のように、独自のソリューションを開発して、デザイン内で CRC チェックを実行することもできます。

パーシャル リコンフィギュレーションは、ザイリンクス FPGA に含まれる優れた機能で、シリコンの機能とソフトウェアの機能を理解することが重要なポイントとなります。開発プロセスにおけるトレードオフを理解して考慮する必要はありますが、全体的に見ると、FPGA デザインのインプリメンテーションはより柔軟になります。

パーシャル リコンフィギュレーションは、ザイリンクス サポート、デザイン サービスおよび Titanium テクニカル サービスで完全にサポートされています。これらのサポート サービスからは、ユーザーのデザインに合ったソリューションを提供しています。

第 2 章

よく使用されるアプリケーション

パーシャル リコンフィギュレーションでは、FPGA ハードウェア リソースをマイクロプロセッサの機能と同様に、時分割してタスクを切り替えることができます。FPGA デバイスの場合、ハードウェアでタスクを切り替えるので、ソフトウェア インプリメンテーションの柔軟性とハードウェア インプリメンテーションのパフォーマンスの両方の利点を生かすことができます。ここでは、さまざまなシナリオを使用して、このテクノロジの詳細を説明します。

ネットワーク マルチポート インターフェイス

パーシャル リコンフィギュレーションでは、サイズ、重量、電力、コストなどを削減することで FPGA アプリケーションを最適化します。時間非依存のファンクションはリコンフィギャブル モジュールとして識別し、分離し、インプリメントでき、必要に応じて デバイスに入れたり出したりできます。よくある例は、ネットワーク スイッチです。このスイッチのポートでは複数のインターフェイス プロトコルがサポートされることもありますが、FPGA デバイスのコンフィギュレーション前にどのプロトコルが使用されるかは予測できません。すべてのポートをディスエーブルにして FPGA デバイスをリコンフィギュレーションしないですむようにするには、図 2-1 のように、可能性のあるすべてのインターフェイス プロトコルを全ポートにインプリメントします。

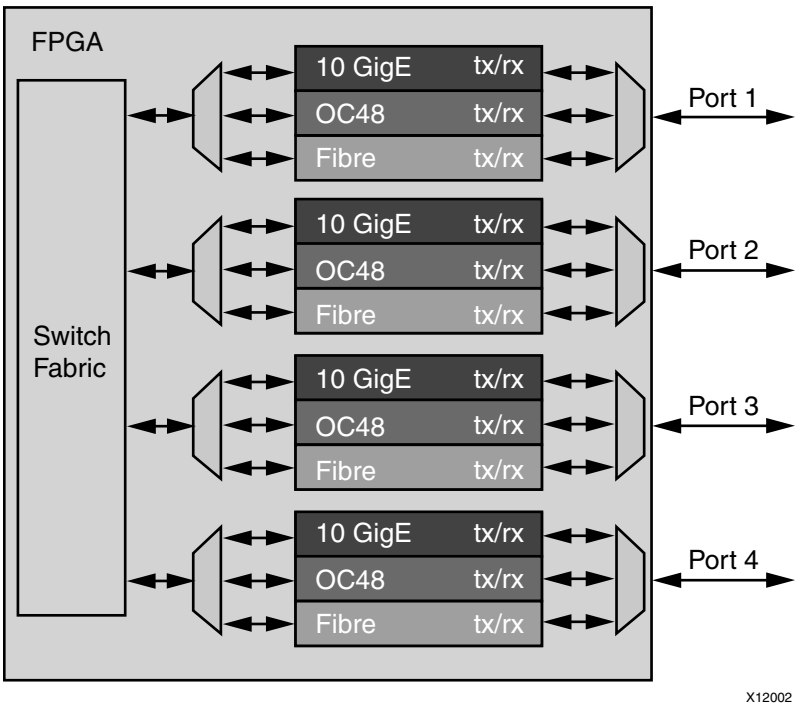

図 2-1 : パーシャル リコンフィギュレーションを使用しない場合のネットワーク スイッチ

この場合、各ポートに 1 つの規格しか使用されないので、効率の悪いデザインになります。14 ページの図 2-2 に示すように、パーシャル リコンフィギュレーションを使用すると、各ポート インターフェイスをリコンフィギャブル モジュールにすることで、効率の良いデザインを作成できます。これにより、複数のプロトコル エンジンを 1 つのポートに接続するために必要であった MUX エレメントは必要なくなります。

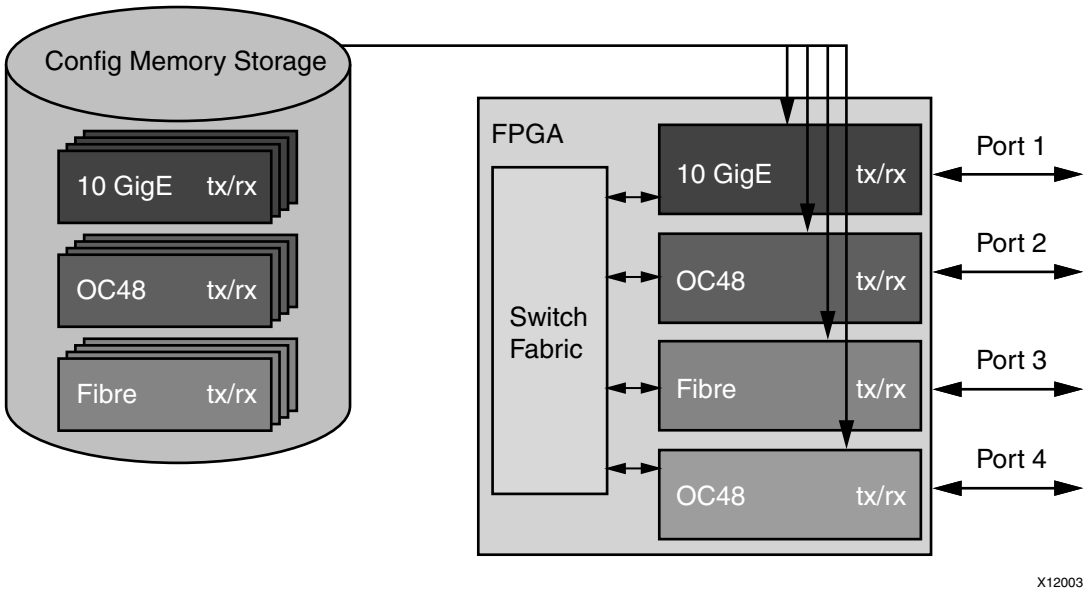

図 2-2 : パーシャル リコンフィギュレーションを使用した場合のネットワーク スイッチ

この基本的な方法は、さまざまなデザインで利用できます。たとえば、相互に排他的な機能を持つ多くのアプリケーションの 1 つである Software Defined Radio (SDR) でこの機能が分割されると、柔軟性およびリソース使用率がかなり改善されます。

パーシャル リコンフィギュレーションを使用したデザインの利点は、効率だけではありません。たとえば、新規プロトコルは 図 2-2 のように、スタティック ロジック (この例ではスイッチ ファブリック) に影響なく、どの時点でもサポートされます。新しい標準がポートに読み込まれても、それ以外のポートには何の影響もありません。追加される標準は、デザイン全体をやり直さなくても、作成して、コンフィギュレーション メモリに追加できるので、スイッチ ファブリックおよびそのポートでの柔軟性および信頼度が増し、ダウンタイムも削減できます。デバッグ モジュールを作成することで、ポートでエラーが発生する場合、未使用のポートに解析/修正ロジックが読み込まれるようにすると、リアルタイムで問題を処理できます。

図 2-2 の例では、各プロトコルでターゲットとなる可能性のある独自の物理ロケーションに対して、独自のパーシャル ビット ファイルが生成される必要があります。パーシャル ビット ファイルは、デバイスの明示的な領域に関連付けられます。この例では、16 個のパーシャル ビット ファイルが 4 つのロケーションの 4 つのプロトコルに対応しています。今後のパーシャル リコンフィギュレーションの開発により、ビット ファイルを別々の物理ロケーションに配置し直せるようになる可能性もあります。

PCIe インターフェイスを使用したコンフィギュレーション

パーシャル リコンフィギュレーションでは、システム アーキテクチャとより互換性のあるインターフェイス標準を使用して、新しいコンフィギュレーション ポートを作成できます。たとえば、FPGA デバイスは PCIe バスのペリフェラルにできるので、システム ホストはこの FPGA を PCIe 接続を介してコンフィギュレーションできます。電源投入リセット後は、FPGA デバイスをフル ビット ファイルでコンフィギュレーションする必要がありますが、フル ビット ファイルには PCIe インターフェイスと Internal Configuration Access Port (ICAP) への接続しか含まれていない可能性があります。

ビットストリーム圧縮を使用すると、サイズを削減できるので、この内部デバイス ロードのコンフィギュレーション時間も削減でき、FPGA コンフィギュレーションが PCIe の列挙仕様を満たしやすくなります。

これで、図 2-3 のように、システム ホストが PCIe ポートを介してパーシャル ビット ファイルをダウンロードし、ほとんどの FPGA の機能をコンフィギュレーションできるようになります。

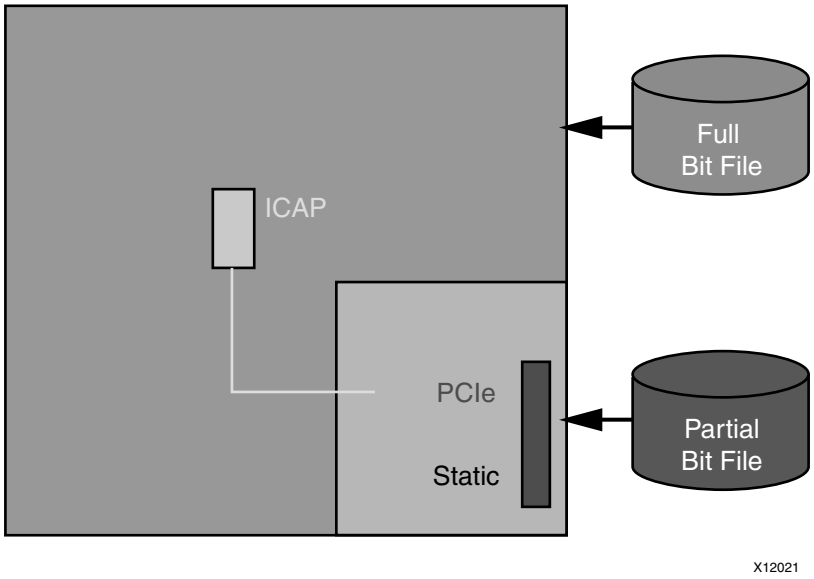

図 2-3 : PCIe インターフェイスを使用したコンフィギュレーション

PCIe 標準には、リクエストを処理できなくても、リクエストを認識するためのペリフェラル (この場合は FPGA) が必要です。FPGA デバイス全体をコンフィギュレーションし直すと、この要件に違反してしまう可能性があります。PCIe はスタティック ロジックの一部なので、パーシャル リコンフィギュレーションのプロセス中も常にアクティブです。このため、FPGA デバイスはリコンフィギュレーション中でも PCIe コマンドに反応できるようになっています。この使用例は、ザイリンクス アプリケーション ノート『Fast Configuration of PCI Express Technology through Partial Reconfiguration』(XAPP883) に記述されています。ML605 評価ボードをターゲットにするリファレンス デザインは、このアプリケーション ノートに含まれています。

ダイナミック リコンフィギャブル パケット プロセッサ

パケット プロセッサにパーシャル リコンフィギュレーションを使用すると、受信するパケットの種類に基づいて、その処理ファンクションを素早く変更できます。図 2-4 では、パケットにパーシャル ビット ファイルを含むヘッダがあるか、特定パケットにパーシャル ビット ファイルが含まれています。パーシャル ビット ファイルは処理後に、FPGA デバイスのコプロセッサをリコンフィ

ギュレーションするために使用されます。これは、パーシャル ビット ファイルの定義済みライブラリに依存するのではなく、受信したデータ パケットに基づいて FPGA デバイスをリコンフィギュレーションする例です。

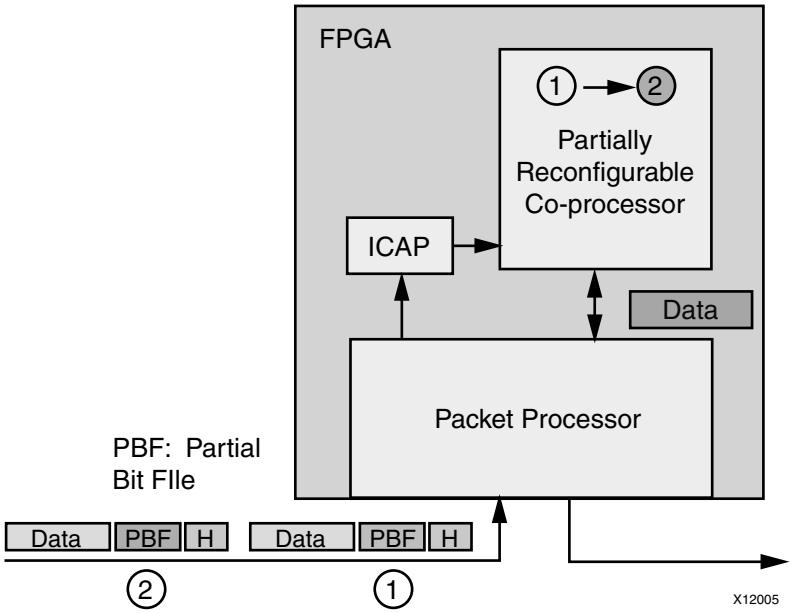

図 2-4 : ダイナミック リコンフィギャブル パケット プロセッサ

非対称鍵暗号化方式

パーシャル リコンフィギュレーションなしではデザインできない新しいアプリケーションもあります。FPGA コンフィギュレーション ファイルを保護するには、パーシャル リコンフィギュレーションと非対称の暗号文を組み合わせます(非対称鍵暗号の詳細は、「Public-key cryptography」を参照してください)。

図 2-5 の場合、青いボックスのすべてのファンクションが FPGA の物理パッケージ内でインプリメントできます。cleartext 情報と Private Key は、保護されたコンテナ外部には出せません。

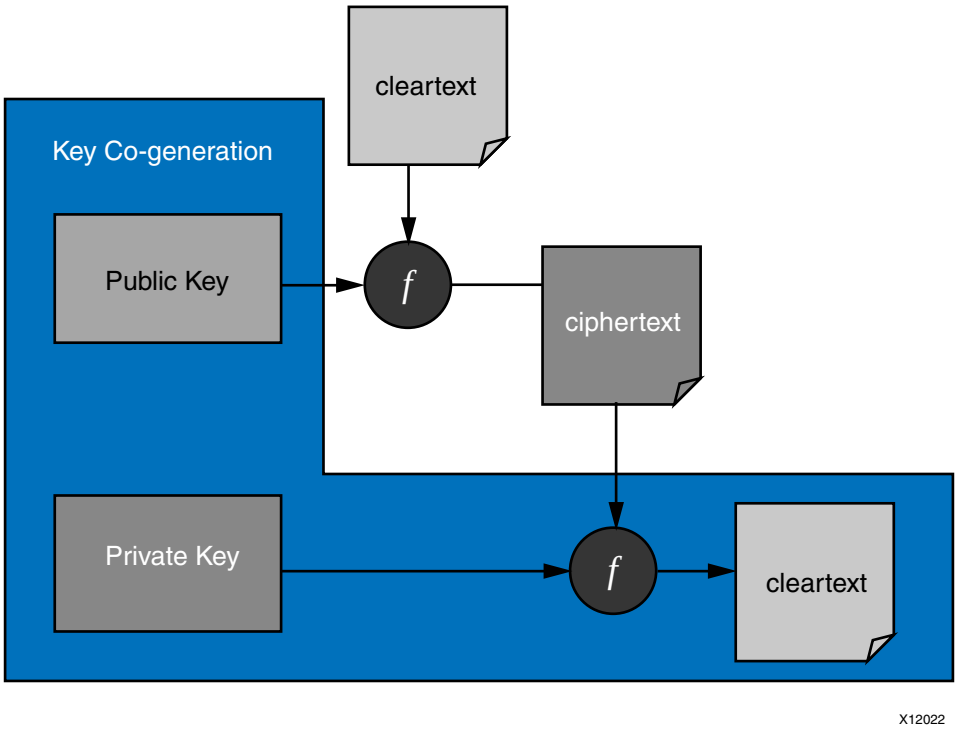

図 2-5 : 非対称鍵暗号化方式

このデザインを実際にインプリメントする場合、最初のビット ファイルは所有権情報を含まない暗号化されていないデザインです。最初のデザインには、公開鍵と秘密鍵のペアを生成するアルゴリズムと、ホスト、FPGA、ICAP を接続するインターフェイスのみが含まれています。

最初のビット ファイルが読み込まれると、FPGA デバイスで公開鍵と秘密鍵のペアが生成されます。公開鍵はホストに送信され、ホストでパーシャル ビット ファイルを暗号化するのにその鍵が使用されます。図 2-6 に示すように、暗号化されたパーシャル ビット ファイルが FPGA デバイスにダウンロードされ、解読され、ICAP に送信されると、FPGA が部分的にコンフィギュレーションし直されます。

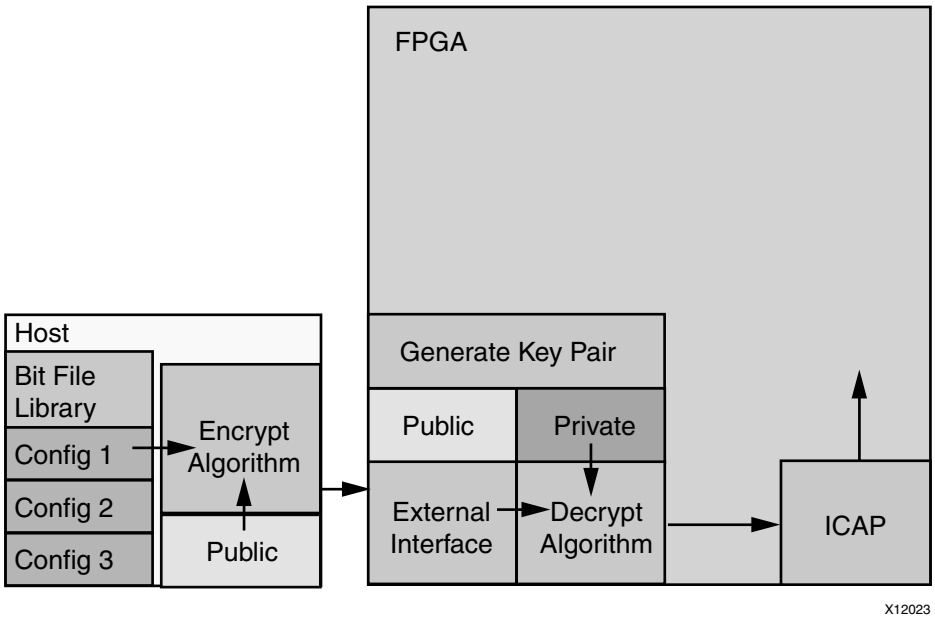

図 2-6 : 暗号化されたパーシャル ビット ファイルの読み込み

パーシャル ビット ファイルは、スタティック デザインのロジックも含め、FPGA デザインのかなりの部分を占めます。また、FPGA リソース全体のわずかな割合しか消費しません。

この方法には、次のような利点があります。

- 公開鍵と秘密鍵のペアはどの時点でも再生成できます。新しいコンフィギュレーションがホストからダウンロードされると、異なる公開鍵で暗号化できます。FPGA デバイスが電源投入リセット後などの同じパーシャル ビット ファイルを使用してコンフィギュレーションされる場合、同じビット ファイルであっても別の公開鍵ペアが使用されます。
- 秘密鍵は SRAM に格納されます。FPGA デバイスが電力を失うと、秘密鍵は存在しなくなります。
- FPGA デバイスに電源が入った状態でシステムが盗難にあったとしても、秘密鍵は汎用の FPGA ファブリックに格納されているため、非常に見つけにくくなっています。秘密鍵は特定のレジスタに格納されているわけではありません。設計者は、物理的なリモート領域および無関連の領域に秘密鍵を格納するレジスタ ビットをそれぞれ手動で探すことができます。暗号化機能の例については、ザイリンクス アプリケーション ノート 『PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration』 (XAPP887) を参照してください。Virtex-5 および Virtex-6 のサンプル デザインは、このアプリケーション ノートに含まれます。

まとめ

パーシャル リコンフィギュレーションは、サイズ、重さ、電力、コストを削減するだけでなく、パーシャル リコンフィギュレーションなしにインプリメントできない新しいタイプの FPGA デザインにも対応しています。

XILINX®

第 3 章

ソフトウェア ツール フロー

この章では、基本的なソフトウェア ツール フロー、部分的にリコンフィギャブルな FPGA とデザインの構造をサポートするシステム、および制約の適用などについて説明します。

部分的にリコンフィギャブルな FPGA デザインをインプリメントする方法は、共通ロジックを共有するパーシャル リコンフィギュレーション以外の複数デザインをインプリメントする方法と類似しています。複数デザイン間で同じ共通ロジックが使用されるようにするには、パーティションが使用されます。図 3-1 は、この概念を示しています。

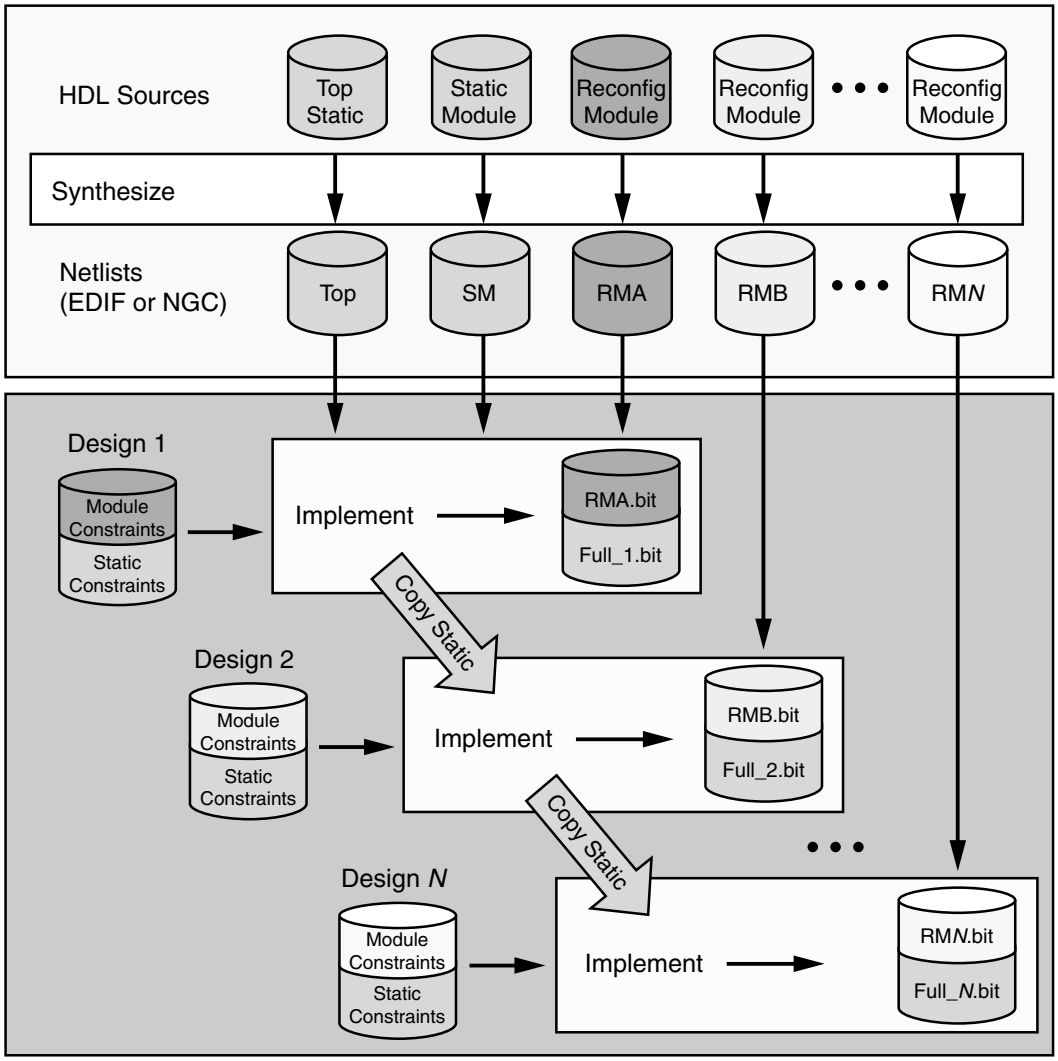

図 3-1 : パーシャル リコンフィギュレーション ソフトウェア フローの概要

一番上の薄いグレーのボックス部分は、各モジュールの HDL ソースからネットリストへの合成を示しています。最適なネットリストが各領域にインプリメントされ、そのコンフィギュレーション用のフル ビット ファイルとパーシャル ビット ファイルが生成されます。最初のインプリメンテーションからのスタティック ロジックは、その後に続くデザイン インプリメンテーションすべてで共有されます。

デザイン構造例

このマニュアルでは、デザイン フローと手法を説明するのに Color2 サンプル デザインを使用します。このデザインは主に赤、青、緑 (この色は主には使用されません) の DVI サポートのモニタ カラー バーに表示されるほか、主な色を混ぜた別の形でも表示されます。パーシャル リコンフィギャブル モジュールは赤、青、緑のモジュールで表示されます。各モジュールには、赤、青、緑ごとに高速および低速の種類があります。この色の速度は、LED がデモ ボードで点滅する速さを示しています。このデザインでは、Virtex®-6 ML-605 評価版プラットフォームをターゲットにしています。

参照デザインのデザイン ファイルは、次からダウンロードできます。

http://japan.xilinx.com/tools/partial-reconfiguration

図 3-2 は、階層ネットリストの図を示しています。Top、IIC_init、DVI_IF、VGA はデザインのスタティック領域にあるモジュールで、ほかのモジュールがリコンフィギュレーション可能な場合も、これらのロジックは通常の動作状態を保ちます。赤、青、緑は、それぞれ Red、Blue、Green ファンクションのリコンフィギャブル モジュールのインスタンシエーションを示しています。各色モジュールには、それぞれ高速および低速パターンがあります。

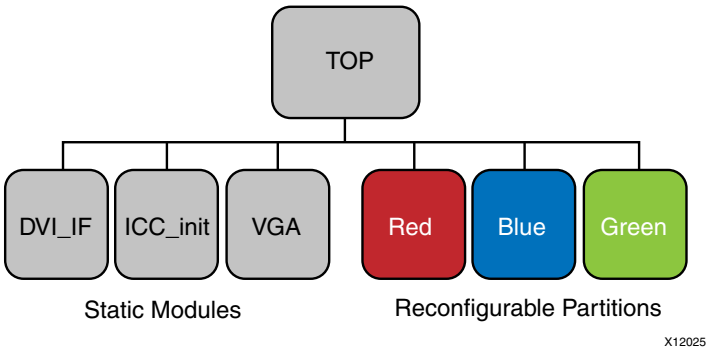

図 3-2 : Color2 デザイン階層

次は、デザイン ソースの階層と Color2 というパーシャル リコンフィギュレーション プロジェクト全体のリコンフィギャブル モジュールのコード例です。

```
Design source hierarchy and Reconfigurable Module variants for overall
PR project named Color2:

  Top.v ......... top module which is static
   red.......... instantiation of a Reconfigurable Module
     red_fast.v......Reconfigurable Module
     red_slow.v......""
   blue ......... instantiation of a Reconfigurable Module
     blue_fast.v .....Reconfigurable Module
     blue_slow.v .....""
   green......... instantiation of a Reconfigurable Module
     green_fast.v.....Reconfigurable Module
     green_slow.v.....""
   DVI_IF.v ....... static module
   IIC_init.v ......""
   VGA.v........."
```

Red、Green、Blue は部分的にリコンフィギャブルなインスタンスです。デザインのその他すべてのロジックはスタティックです。

Red、Green、Blue インスタンスにはほかのロジックが含まれません。これらのインスタンスは単にインスタンシエーション文で、red_fast および blue_slow などのモジュール定義には、インプリメントするロジックが含まれます。

プロジェクト ファイルの構造例

部分的にリコンフィギャブルな FPGA デザイン プロジェクトは、平均的な FPGA デザイン プロジェクトよりも複雑ですが、明確に定義されたファイルとディレクトリ構造があると、プロジェクトが管理しやすくなります。

プロジェクト全体で、各リコンフィギャブル パーティションには複数のリコンフィギャブル モジュールがあります。モジュールはボトムアップ方式で合成されるので、各リコンフィギャブル

パーティションに関連するネットリストが複数生成されます。インプリメンテーションはトップダウン方式で実行され、「コンフィギュレーション」という特定のネットリスト セットが定義されます。

ソース、制約、合成結果、インプリメンテーション結果の混同を避けるため、デザイン インプリメンテーションの各段階ごとに別々のディレクトリを使用することをお勧めします。パーシャル リコンフィギュレーション デザインに最もよく使用されるディレクトリ構造は、次のとおりです。

```
project_name ..... プロジェクト全体の名前
  Docs......... ユーザーまたはデザイン ドキュメント
  Implementation....ザイリンクス ソフトウェア インプリメンテーション結果
   modules....... スタティックまたはリコンフィギャブル モジュールのネットリスト
   configurations ...コンフィギュレーションのインプリメンテーション結果
  Source........ ソース ファイル
   modules.......スタティックおよびリコンフィギャブル モジュールの HDL ソース
ファイル
   UCF......... 制約ファイル
  Synth ........ 合成結果
   modules....... 各スタティックおよびリコンフィギャブル モジュールのネットリスト
  Tools ........Tcl スクリプトまたはその他のユーザー スクリプト
```

ファイルに Color2 デザインが記述されているとすると、フロー ベースのディレクトリ構造は次のようになります。

```
Color2.......... プロジェクト全体の名前
  Docs
   readme.txt
  Source ........HDL ソース ファイル
   Static....... スタティック ロジックのすべての HDL のコレクション
    Top ....... 最上位レベルのスタティック モジュール
    DVI_IF...... 下位レベルのスタティック モジュール
    IIC_init....."
    VGA ......."
   red_fast......Red のリコンフィギャブル モジュール
   red_slow....."
   blue_fast .....Blue のリコンフィギャブル モジュール
   blue_slow ....."
   green_fast.....Green のリコンフィギャブル モジュール
   green_slow....."
   UCF ........ 制約ファイル
  Synth......... 合成済みネットリスト
   static....... top、DVI_IF、IIC_init および VGA
   red_fast
   red_slow
   blue_fast
   blue_slow
   green_fast
   green_slow
  Implementation .... スクリプトから実行したインプリメンテーション結果
   FastConfig..... インプリメンテーション結果とビット ファイル
   SlowConfig....."
   FSFConfig ....."
   BlankConfig .... これらの色のブラック ボックス
  PlanAhead....... PlanAhead から実行したインプリメンテーション結果
   FFF ........ インプリメンテーション結果とビット ファイル
   SSS ......."
   FSF ......."
```

```
BB......... これらの色のブラック ボックス
Tools.........Tcl スクリプトまたはその他のユーザー スクリプト
```

合成

リコンフィギャブル モジュールはそれぞれボトムアップ形式でほかのモジュールとは別々に合成されます。これは、グラフィカル インターフェイスかコマンド ラインのいずれかから個別のプロジェクトを使用すると実行できます。これらのモジュールのポートはパッケージ ピンに接続されず、その上のスタティック ロジックに接続されるので、モジュールごとに、I/O 挿入をディスエーブルにしてください。I/O ポートは、リコンフィギュレーションに含めることができます。詳細は、第 7 章の「リコンフィギャブル モジュールの I/O」を参照してください。

スタティック モジュール同士は一緒に合成して 1 つのネットリストを生成できるほか、複数のスタティック ネットリストを個別に生成したりできます。NGDBuild ではスタティック モジュールとリコンフィギャブル モジュールがまとめられ、リコンフィギャブル パーティションの定義でスタティックとリコンフィギャブル ロジック間のインターフェイスが表されます。スタティックまたはリコンフィギャブル モジュール合成には、別のオプションも使用でいます。

次は、Color2 デザイン例で生成された一番短いコード例です。

```
Netlists generated for the PR project named Color2:

 Netlist for Top which contains DVI_IF, IIC_init and VGA modules

 Netlists for the reconfigurable instance Red:
 --------------
 Netlist for red_fast
 Netlist for red_slow

 Netlists for the reconfigurable instance Blue:
 --------------
 Netlist for blue_fast
 Netlist for blue_slow

 Netlists for the reconfigurable instance Green:
 --------------
 Netlist for green_fast
 Netlist for green_slow
```

注意：ネットリスト名は HDL ファイル名ではなく、モジュール名に関連しています。Red のモジュール/ネットリスト名は、それぞれ同じにしておかないと、スタティック ロジック モジュールのインスタンシエーションからリコンフィギャブル　モジュールを呼び出すことができなくなります。また、各リコンフィギャブル モジュールのポートも同じにしないと、デザインのアセンブリがうまくいきません。

リコンフィギャブル モジュールの各インスタンシエーションには、独自のモジュール名を付ける必要があります。このサンプル デザインでは、Red は 1 度だけインスタンシエートできます。これにより、インプリメンテーション ツールでどのリコンフィギャブル モジュールがどのリコンフィギャブル パーティションに関連しているかが決定されるようになります。

実際には、各リコンフィギャブル モジュールのネットリスト名は同一なので、各ネットリストはそれぞれのディレクトリにある必要があります。

```
Netlist directory for the PR project named Color2:

Static/Top.ngc (contains logic for all static logic including
          DVI_IF, IIC_init and VGA)

 Netlists for the reconfigurable instance Red:
 --------------
 red_fast/red.ngc
 red_slow/red.ngc

 Netlists for the reconfigurable instance Blue:
 --------------
 blue_fast/blue.ngc
 blue_slow/blue.ngc

 Netlists for the reconfigurable instance Green:
 --------------
 green_fast/green.ngc
 green_slow/green.ngc
```

コンフィギュレーション

パーシャル リコンフィギャブル ソフトウェアでは、スタティック ロジックと各リコンフィギャブル パーティションごとに 1 つのリコンフィギャブル モジュールを含む完全なデザインがインプリメントされます。インプリメンテーションはそれぞれ本文中で実行されます。これにより、リソース使用率、グローバル信号、デザイン制約およびその他の要件などの情報すべてがツールに渡されます。すべてのリコンフィギャブル モジュールをインプリメントするには、すべての可能性のあるリコンフィギャブル モジュールの組み合わせのサブセットを選択し、それらを独自のデザインとしてインプリメントする必要があります。独自のインプリメンテーションはそれぞれ「コンフィギュレーション」と呼ばれます。

各リコンフィギャブル パーティションはオプションでブラック ボックスとして設定できるので、空のビットストリームをリコンフィギャブル モジュールにできます。このため、Color2 デザインでは、リコンフィギャブル モジュールのフルセットと、それに伴ってパーシャル ビット ファイル (BIT) も次のようにインプリメントできます。

```
Red { red_fast, red_slow, black box }
Blue { blue_fast, blue_slow, black box }
Green { green_fast, green_slow, black box }
```

各リコンフィギャブル パーティションに対する 3 つの選択肢と、このデザインの 3 つのリコンフィギャブル パーティションを使用した場合、コンフィギュレーションとして定義できる独自の組み合わせは 27 個ありますが、各組み合わせごとにコンフィギュレーションを作成する必要はありません。モジュールのパーシャル ビット ファイルはそれ以外のリコンフィギャブル モジュールからは独立しているので、各モジュールを含むコンフィギュレーションのみを 1 回インプリメントするだけで十分です。

Color2 デザインの場合に最低限必要なコンフィギュレーション数は次のコード例のようになります。

```
Minimum number of FPGA designs (Configurations) required to implement
the PR project Color2:

First Configuration      Second Configuration      Third Configuration
-------------------      --------------------      -------------------
 Top                       Top                       Top
    Red                       Red                       Red
       red_fast                  red_slow                  black box
    Blue                      Blue                      Blue
       blue_fast                 blue_slow                 black box
    Green                     Green                     Green
       green_fast                green_slow                black box
    DVI_IF                    DVI_IF                    DVI_IF
    IIC_init                  IIC_init                  IIC_init
    VGA                       VGA                       VGA
```

Red、Green、Blue にはそれぞれ 3 つの異なるモジュールがあります。このため、すべてのリコンフィギャブル モジュールをインプリメントするのに必要なコンフィギュレーション数は最低 3 つになります。必要であれば、もっとコンフィギュレーションを作成して、独自のフル ビット ファイル *(BIT) を作成することもできます。

たとえば、red_fast、blue_slow、green_fast モジュールを含む 4 つ目のコンフィギュレーションを作成することができます。この場合、3 つのリコンフィギャブル モジュールすべてがこのコンフィギュレーションで再使用されます。これらのモジュールのインプリメンテーション結果とパーシャル ビット ファイルは、複数のコンフィギュレーションで同じになります。

パーシャル ビットストリームを作成すると、パーシャル リコンフィギュレーション プロジェクトで作成されたフル ビットストリームまたはパーシャル ビットストリームのどの組み合わせでも FPGA デバイスに読み込めるようになりますが、その特定の組み合わせが予測どおりに動作するかどうかを確認するには、その組み合わせのモジュールのコンフィギュレーションを作成する必要のあることがあります。パーシャル リコンフィギュレーション デザインの全デザイン レベルのシミュレーションと検証フローは、標準のデザインと同じです。

制約

スタティック ロジックの制約は通常 UCF ファイルに格納され、すべてのコンフィギュレーション間で共有されます。ngdbuild -uc オプションを使用すると、1 つの共通の UCF ファイルをすべてのコンフィギュレーション間で共有することで、すべてのスタティック制約を同じにできます。

スタティック ロジック制約に含めることのできないモジュール特有の制約があることもあります。たとえば、タイミング制約が red_fast にのみ存在するパスに設定される場合、この制約は上記の最初のコンフィギュレーションにだけ適用されます。これは、PlanAhead™ を使用して制約ファイルを管理するか、特定モジュールのネットリスト内に制約を埋め込むと実行できます。ngdbuild -uc オプションは、コマンド ラインごとに何度でも使用できるので、各 run で複数の UCF ファイルを指定することもできます。

エリア グループ制約

AREA_GROUP は、論理デザイン エレメントを特定のラベルまたはグループに関連付けるグループ制約です。AREA_GROUP 制約とパーティション定義は、スタティック ロジックをリコンフィギャブル ロジックと区別するために必要で、スタティック領域内のロジックがリコンフィギャブル モジュールに統合されないようにしたり、リコンフィギャブル ロジックがスタティック領域のロ

ジックに統合されないようにします。AREA_GROUP 制約は、リコンフィギャブル パーティションごとに定義する必要があります。次の例では、reconfig_red というリコンフィギャブル パーティションに対して pblock_reconfig_red という AREA_GROUP 制約を設定しています。

```
INST “reconfig_red” AREA_GROUP = “pblock_reconfig_red”;
```

少なくとも 1 つ、できれば複数の AREA_GROUP RANGE 制約を各リコンフィギャブル領域に定義して、パーシャル リコンフィギュレーション領域の形や配置を設定する必要があります。主な範囲 (RANCE) 制約は、通常スライス範囲になります。スライスはパーシャル リコンフィギュレーション領域内にあります。スライスには、基本的な LUT および FF などの論理エレメントが含まれます。リコンフィギャブル モジュールにブロック RAM、I/O、その他の種類の論理エレメントも含まれる場合は、別の範囲制約を作成する必要があります。

AREA_GROUP RANGE 制約を設定するための要件は、PlanAhead で管理できます。

- AREA_GROUP RANGE 制約はリコンフィギャブル パーティション領域のサイズと形を定義するので、リコンフィギャブル パーティションごとに必要です。
- リコンフィギャブル パーティション内に配置されるリコンフィギャブル モジュールに含まれるすべてのデバイス リソース (スライス、I/O、ブロック RAM、DSP ブロック、マルチギガビット トランシーバ (MGT) など) に、それぞれ対応する AREA_GROUP RANGE 制約を指定する必要があります。シングル サイトのリソースでも関連する RANGE 制約は必要です。
- リコンフィギュレーションしない、またはリコンフィギュレーションできないエレメントには、AREA_GROUP RANGE 制約を作成しないでください。たとえば、DCM、PLL、または BUFG には AREA_GROUP RANGE 制約を作成しないでください。
- 1 つのリコンフィギャブル パーティションが複数の AREA_GROUP RANGE 制約で定義される場合、これらは隣接している必要があります。
- リコンフィギャブル パーティションの AREA_GROUP RANGE 制約は、別のリコンフィギャブル パーティションの AREA_GROUP RANGE 制約とは重複しないようにします。
- パーシャル リコンフィギュレーションのスライス領域は、左下端 (minX, minY) から右上端 (maxX, maxY) に向かって定義する必要があります。次に例を示します。

  ```
  INST “reconfig_red” AREA_GROUP = “pblock_reconfig_red”;
  AREA_GROUP “pblock_reconfig_red” RANGE = SLICE_X20Y76:SLICE_X25Y79;

  INST “reconfig_blue” AREA_GROUP = “pblock_reconfig_blue”;
  AREA_GROUP “pblock_reconfig_blue” RANGE = SLICE_X28Y64:SLICE_X33Y67;

  INST “reconfig_green” AREA_GROUP = “pblock_reconfig_green”;
  AREA_GROUP “pblock_reconfig_green” RANGE = SLICE_X20Y50:SLICE_X25Y53;
  ```

- ほとんどのロジック タイプは、スライス、ブロック RAM、DSP48、IOB、MGT などのリコンフィギャブル パーティション内に設定できます。DCM、PLL、または PMCD のようなクロック修正ロジックなどを含むグローバル クロック ロジックは、スタティック モジュール内に含まれている必要があります。リコンフィギャブル パーティションの規則に関する詳細は、第 7 章「デザインの注意事項」を参照してください。
- スライス範囲は、CLB バウンダリに設定される必要があります (CLB は分割しません)。この規則に従うと、AREA_GROUP RANGE 制約が CLB すべてに適用されます (Virtex®-5 デバイス)。
  - AREA_GROUP SLICE 範囲の横軸 (minX) は常に偶数
  - AREA_GROUP SLICE 範囲の横軸 (maxX) は常に奇数

  この規則に従うと、リコンフィギャブル パーティションの RANGE が Virtex-5 デバイスの CLB バウンダリになります。この場合、リコンフィギャブル フレーム規則には従ったことにな

りません。第 7 章「デザインの注意事項」で説明されるリコンフィギャブル フレーム規則に従うようにしてください。

- ブロック RAM の AREA_GROUP RANGE には、奇数または偶数のいずれかになる座標 (minX, minY) および (maxX, maxY) が含まれます。AREA_GROUP のブロック RAM 範囲は、PlanAhead または FPGA Editor で指定できます。

  27 ページの図 3-3 は、AREA_GROUP RANGE の例を示しています。

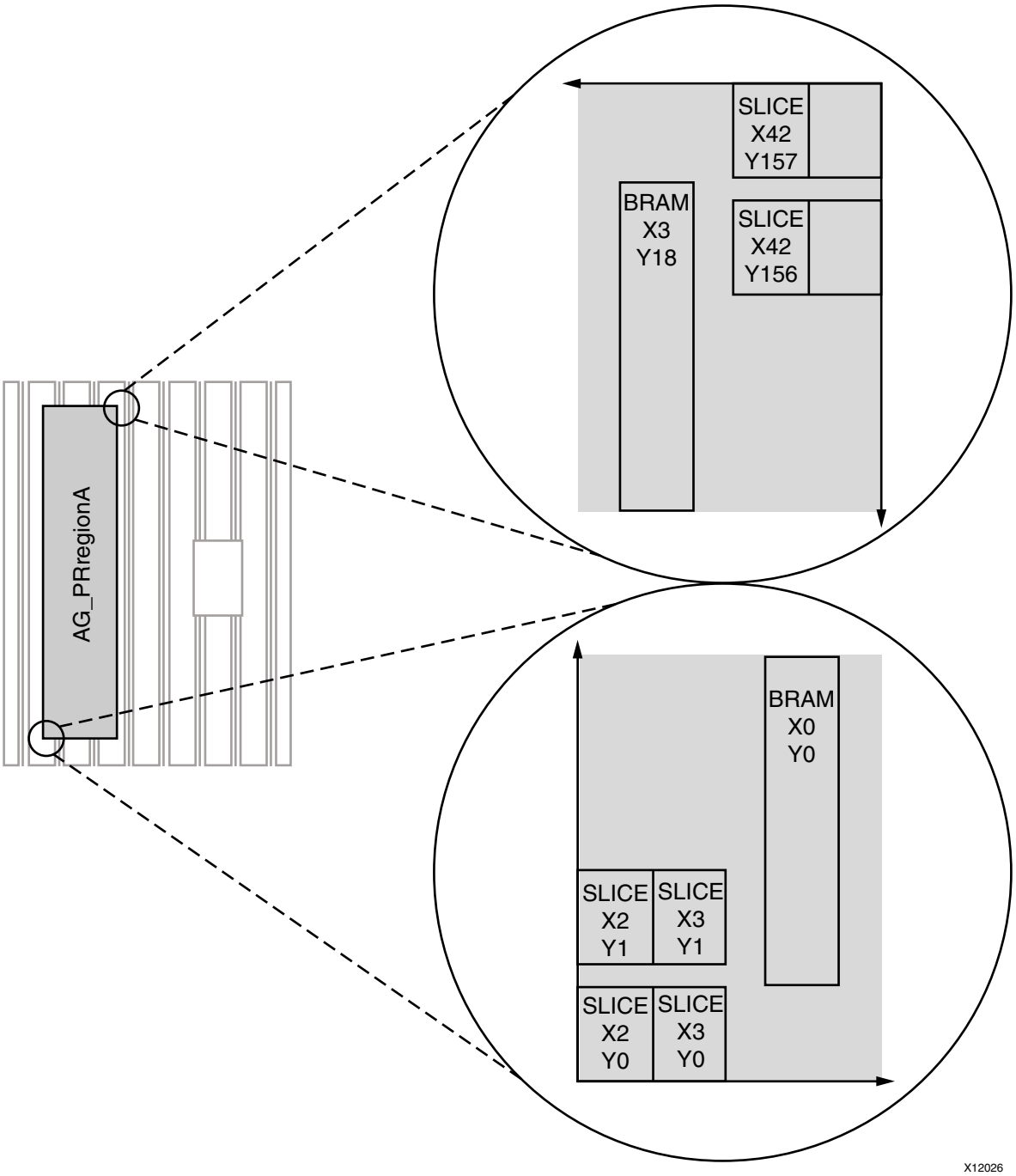

図 3-3 : パーシャル リコンフィギュレーション領域のスライス範囲と BRAM 範囲

次のコード例は、スライスおよび BRAM の AREA_GROUP RANGE 制約を示しています。

```
AREA_GROUP “AG_PRregionA” RANGE = SLICE_X2Y0:SLICE_X43Y157;
AREA_GROUP “AG_PRregionA” RANGE = RAMB16_X0Y0:RAMB16_X3Y18;
```

PlanAhead ソフトウェアでは、各リコンフィギャブル モジュールのサイズが概算され、使用されるリソースが表示されるので、ブロック RAM または I/O に必要な AREA_GROUP RANGE を決定する際に便利です。

ただし、リコンフィギャブル パーティションの形や配置に関する推奨案は表示されません。AREA_GROUP RANGE は、各リソース タイプの最大のリコンフィギャブル モジュールを適用するのに十分な大きさにする必要があります。つまり、ほとんどのスライスを使用するリコンフィギャブル モジュールは、ほとんどの BRAM を使用するリコンフィギャブル モジュールとは異なる可能性があります。また、AREA_GROUP RANGE は、デザインがタイミングを満たすことができるような形および配置にする必要もあります。

パーティション ピン

パーシャル リコンフィギュレーションには、スタティック ロジックとリコンフィギャブル ロジック間のポート バウンダリに「パーティション ピン」というコンポーネントが含まれます。パーティション ピンは、スタティック ロジックと各リコンフィギャブル パーティション間の異なるリコンフィギャブル モジュールの回路接続を同じにするために必要です。また、パーティション ピンはリコンフィギャブル パーティション バウンダリを始点、終点、通過点とするネットにタイミング制約を作成する際に便利なコンポーネントです。

パーティション ピンはインプリメンテーション ソフトウェアで自動的に挿入されます。第 7 章「デザインの注意事項」で説明される制御配線を除き、特別なインスタンシエーションや考慮は必要ありません。

メモ：パーティション ピンは、リコンフィギャブル領域への入力または出力接続にすることができます。パーティション ピンは、双方向にはできません。

パーティション ピンのタイミング制約のフォーマットは、29 ページの図 3-4 に示すパス構造のいずれかになります。黄色のリコンフィギャブル モジュールのボックスは、物理範囲またはフロアプランの必要がない論理バウンダリを表しています。

パス A) パーティション ピンへのスタティック ネット入力
パス B) パーティション ピンのリコンフィギャブル ネット出力
パス C) パーティション ピンへのリコンフィギャブル ネット入力
パス D) パーティション ピンのスタティック ネット出力
パス X、Y、Z) パスにパーティション ピンを含む register-to-register パス

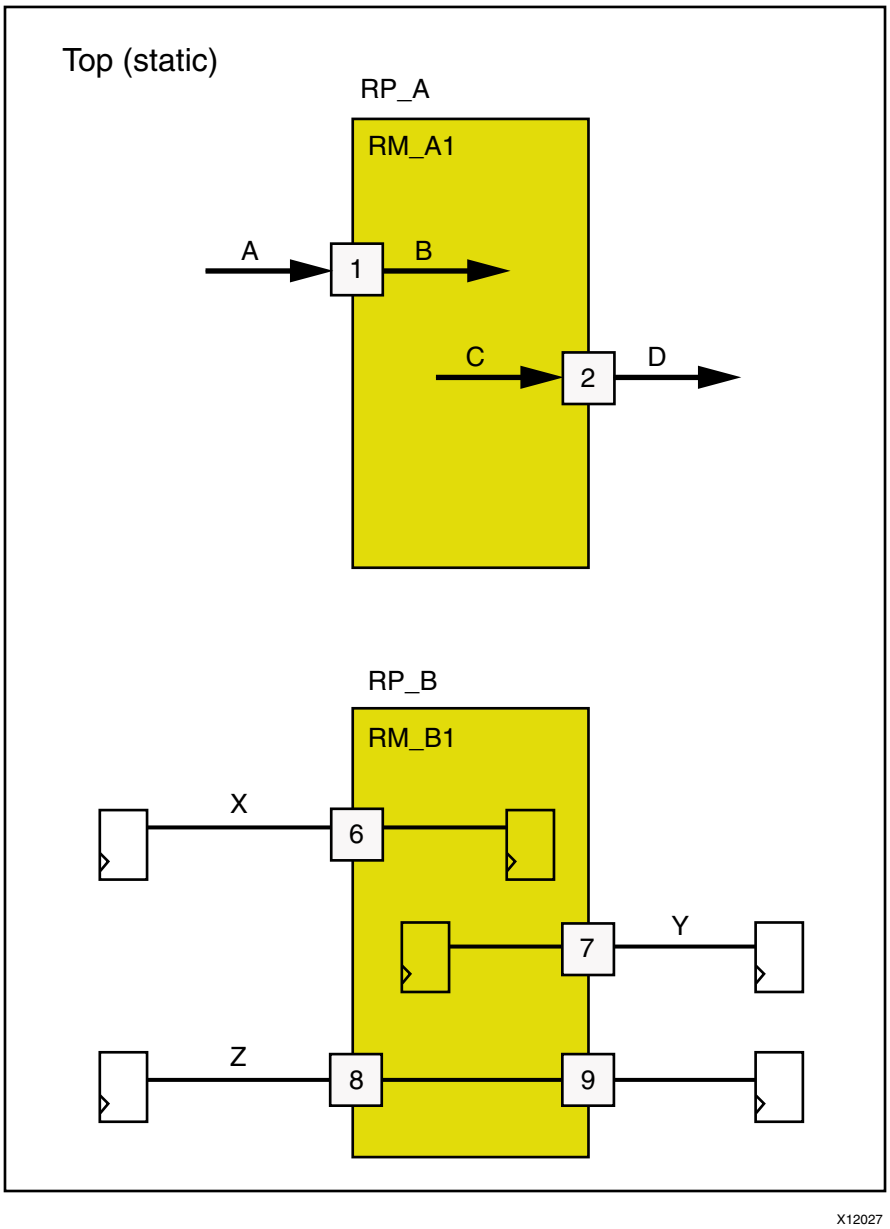

図 3-4 : リコンフィギャブル パーティションを通るタイミング パス

タイミング制約を作成する前に、PIN-TPSYNC 制約を使用してネットをリコンフィギャブル モジュールへの入力またはリコンフィギャブル モジュールからの出力によってグループ分けする必要があります。

ピン名の構文は、<*Partition_name*>.<*port_name*> です。次のコードはその具体例です。

```
PIN “RP_A.1” TPSYNC = group_RP_A_input;
PIN “RP_A.2” TPSYNC = group_RP_A_output;
```

パーティション ピンに TPSYNC 制約を使用した場合、PERIOD 制約のみを使用するよりも広範囲にこれらのパスがカバーされます。TPSYNC を使用すると、スタティック領域からパーティション ピンまでの遅延が最小になるように初期配置を実行できます。これにより、リコンフィギャブル モ

ジュールへのタイミング バジェットが多くなるので、インプリメンテーション ツールでリコンフィギャブル モジュールのタイミング要件が満たしやすくなります。

PIN-TPSYNC グループ制約では、標準の UCF のワイルドカード表記規則がサポートされます。たとえば、RP_A へのデータ バス入力があるとすると、次の方法で、この制約を使用した前述の例の入力グループに追加できます。

```
PIN “RP_A.data*” TPSYNC = group_RP_A_input;
```

パーティション ピン RP_A.1 へのスタティック ネットすべて、およびパーティション ピン RP_A.1 からのリコンフィギャブル ネットすべて (上記のパス A & B) に対してタイミング制約を作成するには、次を使用します。

```
TIMESPEC TS_from_static_to_PP_input = TO “group_RP_A_input” 4.5 ns;
TIMESPEC TS_from_PP_input_to_RM = FROM “group_RP_A_input” 4.5 ns;
```

パーティション ピン RP_A.2 へのリコンフィギャブル ネットすべて、およびパーティション ピン RP_A.2 からのスタティックル ネットすべて (上記のパス C & D) に対してタイミング制約を作成するには、次を使用します。

```
TIMESPEC TS_from_RM_to_PP_output = TO “group_RP_A_output” 4.5 ns;
TIMESPEC TS_from_PP_output_to_static = FROM “group_RP_A_output” 4.5 ns;
```

これらの制約が非同期パスに適用されることもあるので、リコンフィギャブル パスへのパスとリコンフィギャブル パスからのパスすべてを同期しておくことをお勧めします。®

最初のインプリメンテーション中には、リコンフィギャブル モジュールの 1 つのみでタイミング目的が考慮されます。ツールで生成されるタイミング バジェットでは、その他すべてのリコンフィギャブル モジュールのタイミング マージンが十分に提供されないので、後でインプリメントされたときにタイミングが満たせないことがあります。TPSYNC オプションを使用すると、デザインのスタティック部分を各リコンフィギャブル モジュールとは別に含めることができるので、スタティック領域と各リコンフィギャブル モジュールにタイミング バジェットが適切に分配されるようになります。

TPSYNC の詳細は、付録 A「既知の問題と制限」を参照してください。

標準の PERIOD タイミング制約は、パーティション ピンを含む register-to-register パスに使用されます。前述のネット X、Y、Z には、次のように制約が付けられます。

```
NET clk TNM_NET = clk_group;
TIMESPEC TS_clk_period = PERIOD clk_group 10 ns;
```

この制約を使用すると、register-to-register パスにパーティション ピンの遅延が含まれ、タイミング制約が満たされます。ネット遅延のどの部分がネットのスタティック部分およびリコンフィギャブル部分に分配されるかは指定されません。このため、PERIOD 制約を FROM、TO、および FROM:TO 制約と組み合わせて使用し、パス全体のバジェットを正確に分配する必要があります。

パーティションに直接入力パッドを接続したり、パーティションから出力パッドに出力を直接接続すると、最適なタイミング パフォーマンスにはならないことがあります。パーティション ピンは、組み合わせロジックとパス遅延で構成されています。パーティション ピンを使用すると、IOB パッキングが実行されないので、パッキングが必要な場合に入力および出力でタイミング エラーになることがあります。

リコンフィギャブル パーティション バウンダリを通るグローバル クロックを除くすべての信号がレジスタ入力されるようにしてください。これにより、タイミング制約が簡素化されて、タイミング制約が満たされる可能性が上がります。®ただし、パッドがリコンフィギャブル パーティションの同期コンポーネントに直接接続されない場合は、OFFSET 制約を使用してパスに正しく制約を付けます。

入力パッドによりパーティション内の同期コンポーネントが駆動される場合は、OFFSET IN 制約を入力に適用できます。これにより、パーティション ピンの遅延が考慮されるようになります。グローバル OFFSET IN は次のように適用できます。

```
OFFSET = IN 3 ns VALID 8 ns BEFORE “clk”;
```

同期コンポーネントがパーティション出力を駆動し、パーティション出力が出力パッドを駆動する場合は、OFFSET OUT 制約を出力に適用できます。これにより、パーティション ピンの遅延が考慮されるようになります。グローバル OFFSET OUT は次のように適用できます。

```
OFFSET = OUT 5 ns AFTER “clk”;
```

オプションで、パーティション ピンをリコンフィギャブル パーティションの area_group 範囲内のサイトに物理的にロックすることもできます。これらはパーティション リコンフィギュレーション ソフトウェアで自動的に配置されるので、必ずしもロックする必要はありませんが、ロックしておくと、インプリメンテーション結果をさらに詳細に制御することができます。この方法は最後の手段として使用するべきで、自動配置の後にのみ、タイミング制約を使用して実行できます。次の UCF コマンドでは、パーティション ピンをサイトに物理的にロックしています。

```
PIN “RP_A.1” LOC = SLICE_X4Y4;
```

ICAP 用のタイミング制約

FPGA のパーシャル リコンフィギュレーションのコンフィギュレーション ポートとして Internal Configuration Access Port (ICAP) が使用される場合、タイミング制約を使用すると、このインターフェイスの潜在的なパフォーマンスが理解しやすくなります。

Virtex-6 の ICAP のタイミング制約

Virtex-6 FPGA では、ICAP が TRACE で同期コンポーネントとして記述されています。これは、PERIOD、FROM:TO、およびすべてのグループ ベースの制約が ICAP サイトへのパスと ICAP サイトからのパスに正しく適用されることを意味します。ICAP コンポーネントが適用可能なタイミング グループに追加されている限り、その他の制約が必要になることはありません。

Virtex-5 および Virtex-4 の ICAP のタイミング制約

Virtex-5 および Virtex-4 FPGA の場合、PERIOD 制約が ICAP へのパスと ICAP からのパスに適用されません。ICAP 入力および出力は TRACE で同期とは認識されません。BUSY、CE、WRITE 信号も同様です。このため、ICAP への入力と ICAP からの出力には、次の例外制約を付ける必要があります。NET MAXDELAY

NET MAXDELAY 制約を使用した場合、構文は次のようになります。

```
NET “to_icap<*>” MAXDELAY = 15 ns;
NET “from_icap<*>” MAXDELAY = 15 ns;
NET “busy_from_icap” MAXDELAY = 15 ns;
NET “write_to_icap” MAXDELAY = 15 ns;
NET “ce_to_icap” MAXDELAY = 15 ns;
```

この例の場合、to_icap および from_icap ネットワークはすべての幅のバスです。アスタリスクは、バス全体 (0、1、2...) を表しています。NET MAXDELAY 制約は、ネット遅延のみに制約を付けます。セットアップ タイムや clock-to-out タイムは考慮されません。

ICAP コンポーネントは、同期エレメントで考慮されないのでタイム グループに追加できません。このため、ICAP は TPSYNC 制約を使用して同期コンポーネントにはできません。ICAP は特殊なコンポーネントなので、デザインで使用する際にはタイミングを特に考慮する必要があります。

パーティション ピン情報の抽出

パーティション ピンはインプリメンテーション ツールで追加されるので、論理ソース デザインには存在しません。パーティション ピンは予測可能な形式で命名されますが、正しい名前が確実に使われているかどうかを確認するには、デザインをインプリメンテーションし直す必要があります。この後、pr2ucf ユーティリティを使用すると、インプリメント済みデザインからパーティション ピンの配置を抽出できます。このユーティリティは、Configuration ディレクトリ内の配置配線済み NCD ファイルに対して実行します。

```
pr2ucf design_routed.ncd –o partition_pins.ucf
```

PIN 位置制約は partition_pins.ucf ファイルから design.ucf ファイルへコピーすると、デザインの UCF ファイルにバックアノテートできます。ただし、これは 1 つのコンフィギュレーションから次のコンフィギュレーションへの配置を維持するためには必要ありません。

パーティション ピンは、リコンフィギャブル領域内に物理的に配置されていても、スタティック ロジックの論理的な部分なので、パーティション ピンに設定する制約は最上位レベルの UCF に含める必要があります。パーティション ピンの配置は FPGA Editor 内で表示すると、ほかのロジックとの関連を確認できます。

Constraints Editor

パーティション ピン グループを作成するには、Constraints Editor を使用できます。最初のインプリメンテーション後のタイミング制約は、少なくとも 1 つのコンフィギュレーションに対して実行されます。

デザイン ファイルを要求するダイアログ ボックスが表示されたら、最新のコンフィギュレーションの NGD ファイルを選択します。ただし、UCF は新しいファイル (Constraints Editor を開く前に作成されたファイル) にする必要があります。既に PlanAhead にインポート済みの UCF ファイル名や現在コンフィギュレーションに含まれているファイル名にはできません。

Constraints Editor 内には、[Constraint Type] ウィンドウに `Group Constraints` カテゴリがあります。[By Combinatorial Pins] を選択し、パーティション ピンに基づいて 制約を作成します。開いたダイアログ ボックスの [Design element type] フィールドを [Partition Pins] に設定すると、デザイン内のパーティション ピン インスタンスを簡単に見つけることができます。ここで作成したグループを使用して、タイミング仕様を定義してください。図 3-5 は、[Group Constraints by Combinatorial Pins] ダイアログ ボックスを示しています。

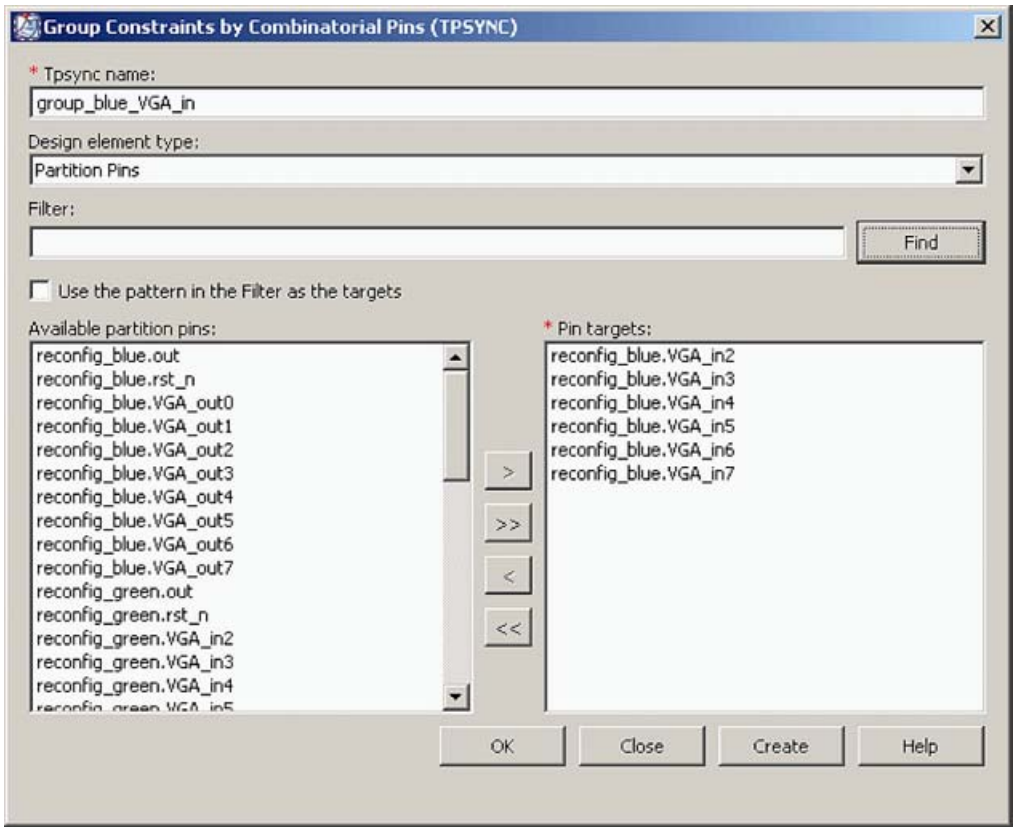

図 3-5 : Constraints Editor でのパーティション ピン グループの作成

Constraints Editor で生成した新しい制約は、PlanAhead ソフトウェアにインポートして、デザインに反映させる必要があります。[File] → [Import Constraint] をクリックし、Constraints Editor でアップデートした UCF から制約をインポートします。

PlanAhead のリコンフィギャブル モジュール制約

PlanAhead には、パーシャル リコンフィギュレーション デザインを管理する効率的な機能が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 デザインに制約を付けるのは複雑なので、PlanAhead を使用してこれらの制約を管理する場合は、まず計画する必要があります。

リコンフィギャブル モジュール制約を PlanAhead のパーシャル リコンフィギュレーション デザインに含めるには、主に 3 つの方法があります。

- **最上位レベルの UCF** – この方法では、制約が PlanAhead プロジェクトよりも前に 1 つまたは複数の最上位レベルの UCF ファイルに存在します。これらの制約は、リコンフィギャブル モジュール ロジックへの完全な階層パスを含み、指定したインスタンスを含むすべてのリコンフィギャブル モジュールに適用されます。リコンフィギャブル モジュール ロジックに関連する制約は、最上位レベルの UCF から抽出され、PlanAhead で生成されたパーティション UCF に追加されます。この方法は、リコンフィギャブル モジュール ロジックに制約を付ける際には推奨されません。
- **リコンフィギャブル モジュールの UCF** – この方法では、制約が PlanAhead プロジェクトよりも前にリコンフィギャブル モジュール レベルの UCF ファイルに存在します。これらの制約の階層はリコンフィギャブル モジュール階層に特有のものです (最上位からの完全な階層パスではない)。

複数のリコンフィギャブル モジュールに同じ制約が必要な場合は、その制約をリコンフィギャブル モジュール UCF で重複させる必要があります。この方法は、リコンフィギャブル モジュール特有の制約を追加する際に推奨される方法です。

- **GUI 手法** – この方法では、制約が PlanAhead の GUI または Tcl コマンドを使用して PlanAhead プロジェクトが作成された後に生成されます。リコンフィギャブル モジュール特有の制約は、制約の作成時にアクティブなリコンフィギャブル モジュールにのみ適用され、PlanAhead で生成されたリコンフィギャブル モジュール UCF に追加されます (これらは最上位レベルのターゲット UCF には表示されません)。これらの制約はユーザー定義のリコンフィギャブル モジュール UCF それぞれに手動で追加し、[Update Reconfigurable Module] コマンドを使用してリコンフィギャブル モジュールをアップデートするようにしてください。

PlanAhead の UCF に関する規則

PlanAhead フローを使用する場合は、UCF 制約に関する規則に従う必要があります。これらの規則は、制約管理システムが今後の PlanAhead リリースで修正されると変更される可能性もありますが、ISE® 12.3 ソフトウェアの場合は、次の規則に従ってください。

- PlanAhead プロジェクト用に UCF を指定する際は、[Copy into Project] オプションを使用します。このオプションを使用すると、PlanAhead に読み込まれるリコンフィギャブル モジュール制約が自動的に処理されます。これにより、PlanAhead での変更がローカル コピーの UCF にのみ反映されるようになります。
- すべてのリコンフィギャブル モジュール制約をリコンフィギャブル モジュール用の UCF ファイルに含めます。リコンフィギャブル モジュール制約を最上位レベルの UCF に含めたり、GUI を使用してリコンフィギャブル モジュールの UCF を作成すると、ビヘイビアで問題になることがあります。

PlanAhead UCF の既知の問題

- 最上位レベルの UCF にリコンフィギャブル モジュール用の制約が含まれる場合、これらの制約はそのリコンフィギャブル モジュールが該当するリコンフィギャブル パーティションに対して定義されるまで読み込まれません。この場合、[Netlist Design] ビューを一旦閉じ、リコンフィギャブル モジュールのネットリストを追加してから、開きなおしてください。これは既知の問題で、今後のリリースで修正される予定ですが、上記の方法で回避できます。
- [Netlist Design] ビューは、run を起動する前に開く必要があります。こうしないと、run ファイルが記述される前に、リコンフィギャブル モジュール ロジックに制約が正しく適用されないことがあります。
- PlanAhead の GUI で制約を変更した場合は、プロジェクトを保存してから、[Netlist Design] ビューを一旦閉じ、run を開始する前に開きなおしてください。

パーティションとインポート

パーティションを使用すると、スタティック ロジックなどの共通モジュールをすべてのコンフィギュレーションで同じにできます。パーティションは、インスタンス (または最上位レベル モジュール) に設定される属性で、ザイリンクス ソフトウェアではこれらの属性に従ってロジックが特定の方法でインプリメントされます。パーティションには、パーティション ロジックのインプリメント方法をさらに詳細に指定する RECONFIGURABLE および STATE などの属性があります。

RECONFIGURABLE 属性では、インスタンスまたはモジュールが最終的にパーシャル ビットファイルにインプリメントされる方法を指定します。リコンフィギャブル モジュールにはリコンフィギャブル以外のモジュールには必要のない物理要件が多く含まれているので、RECONFIGURABLE 属性はインプリメンテーション ツール フローを実行する前に設定する必要

があります。この属性により、モジュールの最終的なインプリメンテーションでかなり影響が出ます。

STATE 属性では、モジュールをインプリメントするか、前のインプリメンテーション済み (保存した) デザインからインポートするかを指定できます。

パーティションをインポートする場合、配置配線を含むインプリメンテーションはインポート先のデザインと同じになります。たとえば、最初のコンフィギュレーションでスタティック ロジックをインプリメントし、ユーザーがこの結果をエクスポート (プロモート) したとします。この後のインプリメンテーションには、すべてこのプロモートされたコンフィギュレーションからのスタティックなパーティションがインポートされます。スタティック ロジックを変更して再びエクスポートした場合、続くコンフィギュレーションにはこの新しいスタティック ロジックをインポートし、コンフィギュレーションをインプリメントし直して、アップデートする必要があります。

PXML ファイルの役割

パーティション情報は、インプリメンテーション ディレクトリの xpartition.pxml ファイルに格納されます。コンフィギュレーションは、それぞれデザイン ディレクトリに PXML ファイルを含んでいます。

xpartition.pxml ファイルには、次の特徴があります。

- XML 形式のテキスト ファイルです。
- PlanAhead ソフトウェアまたは提供されている gen_xp.tcl スクリプトで自動的に生成されます。gen_xp.tcl の詳細は、第 5 章「コマンド ライン スクリプト」を参照してください。
- ユーザーが作成または変更できます。
- MAP や PAR などのインプリメンテーション ツールの入力になります。
- リビジョン コントロールの必要に応じてソースを考慮できます。

NGDBuild、MAP、および PAR などのザイリンクス ソフトウェアでは、自動的にインプリメンテーション ディレクトリから xpartition.pxml が検索され使用されます。パーティション情報を含む XML ファイルには、xpartition.pxml という名前を付け、インプリメンテーション ディレクトリに含めないと、リコンフィギャブル パーティションが認識されません。

xpartition.pxml ファイルを変更した場合は、フローの一部を実行し直す必要があります。STATE 属性を変更した場合は、MAP または PAR のいずれかを実行し直します。MAP と PAR の両方を実行し直すと、配置配線で xpartition.pxml ファイルからの STATE 属性が使用されます。PAR のみを実行し直した場合、配置では前の run からの STATE 属性が維持され、配線でのみ現在の xpartition.pxml ファイルの STATE 属性が使用されます。ImportLocation または Reconfigurable 属性を変更した場合は、NGDBuild、MAP、および PAR のすべてを実行し直す必要があります。

メモ：PXML ファイルのパーティションに適用される BoundaryOpt 属性は、パーシャル リコンフィギュレーション フローでは使用できません。

次のセクションでは、1 つ目から 3 つ目までのコンフィギュレーション PXML ファイルを示しています。

1 つ目のコンフィギュレーション PXML ファイル

最初のコンフィギュレーション PXML ファイルは、次のようになります。

```
First Configuration's xpartition.pxml file:

 <Project FileVersion=”1.2” Name=”FFF” ProjectVersion=”2.0”>
```

```
 <Partition Name=”/top” State=”implement”
 ImportLocation=”NONE” >

  <Partition Name=”/top/red” State=”implement”
  ImportLocation=”NONE” Reconfigurable=”true”
  ReconfigModuleName=”red_fast”>

  <Partition Name=”/top/blue” State=”implement”
  ImportLocation=”NONE" Reconfigurable="true"
  ReconfigModuleName="blue_fast">

  <Partition Name="/top/green" State="implement"
  ImportLocation="NONE" Reconfigurable="true"
  ReconfigModuleName="green_fast">

  </Partition>
 </Partition>
</Project>
```

2 つ目のコンフィギュレーション PXML ファイル

スタティック ロジックにインポートされる 2 つ目のコンフィギュレーションは、次のようになります。

```
Second Configuration's xpartition.pxml file:

 <Project FileVersion=”1.2” Name=”SSS” ProjectVersion=”2.0”>

  <Partition Name=”/top” State=”import”
  ImportLocation=”../XFFF" >

   <Partition Name="/top/red"
   State="implement" ImportLocation="NONE" Reconfigurable="true"
   ReconfigModuleName="red_slow" >

   <Partition Name="/top/blue"
   State="implement" ImportLocation="NONE" Reconfigurable="true"
   ReconfigModuleName="blue_slow" >

   <Partition Name="/top/green"
   State="implement" ImportLocation="NONE" Reconfigurable="true"
   ReconfigModuleName="green_slow" >

   </Partition>
  </Partition>
 </Project>
```

3 つ目のコンフィギュレーション PXML ファイル

スタティック ロジックと 3 つすべてのリコンフィギャブル モジュールにインポートされる 3 つ目のコンフィギュレーションは、次のようになります。

```
Third Configuration's xpartition.pxml file:

 <Project FileVersion="1.2“ Name=”FSF" ProjectVersion="2.0">

  <Partition Name="/top" State="import"
```

```
ImportLocation="../XFFF" >

 <Partition Name="/top/red" State="import"
 ImportLocation="../XFFF" Reconfigurable="true"
 ReconfigModuleName="red_fast" >

 <Partition Name="/top/blue" State="import"
 ImportLocation="../XSSS" Reconfigurable="true"
 ReconfigModuleName="blue_slow" >

 <Partition Name="/top/green" State="import"
 ImportLocation="../XFFF" Reconfigurable="true"
 ReconfigModuleName="green_fast" >

 </Partition>
</Partition>
</Project>
```

スタティック ロジックと Red、Green モジュールは 1 つ目のコンフィギュレーションからインポートされます。Blue モジュールは、2 つ目のコンフィギュレーションからインポートされます。

インプリメンテーション

FPGA デザインをインプリメントするには、NGDBuild、MAP、および PAR をパーシャル リコンフィギュレーション デザイン以外のデザインと同じように実行する必要があります。ほとんどのパーシャル リコンフィギュレーション特有の情報は xpartition.pxml ファイルと UCF ファイルに含まれます。パーシャル リコンフィギュレーション特有のコマンド ライン オプションはありません。次の例は、パーシャル リコンフィギュレーション デザインをインプリメントするコマンドを示しています。

```
ngdbuild -sd ../red_fast -sd ../blue_fast –sd ../green_fast -uc
../UCF/design.ucf ../Static/top.edf FFF.ngd
map -w -o FFF_map.ncd FFF.ngd FFF.pcf
par -w FFF_map.ncd FFF.ncd FFF.pcf
```

すべてのインプリメンテーション オプションがパーシャル リコンフィギュレーションに使用できるわけではありません。使用できないオプションは次のとおりです。

- MAP コマンドの global_opt オプションとそれに関する子コマンド
- MAP コマンドの -power オプションの high および xe 値
- BoundaryOpt 制約 (PXML ファイルでパーティションに適用される)
- SmartGuide™

配置配線問題のデバッグ

パーシャル リコンフィギュレーション デザインが配置されれると、次のようになります (図 3-6 参照)。

- スタティック配線は、リコンフィギャブル パーティションを通して配線できます。
- リコンフィギャブル モジュール内の配線は、そのリコンフィギャブル パーティションに関連するエリア グループの外部には配線できません。
- インポートされた配線がインプリメント済みの配線より優先されます。

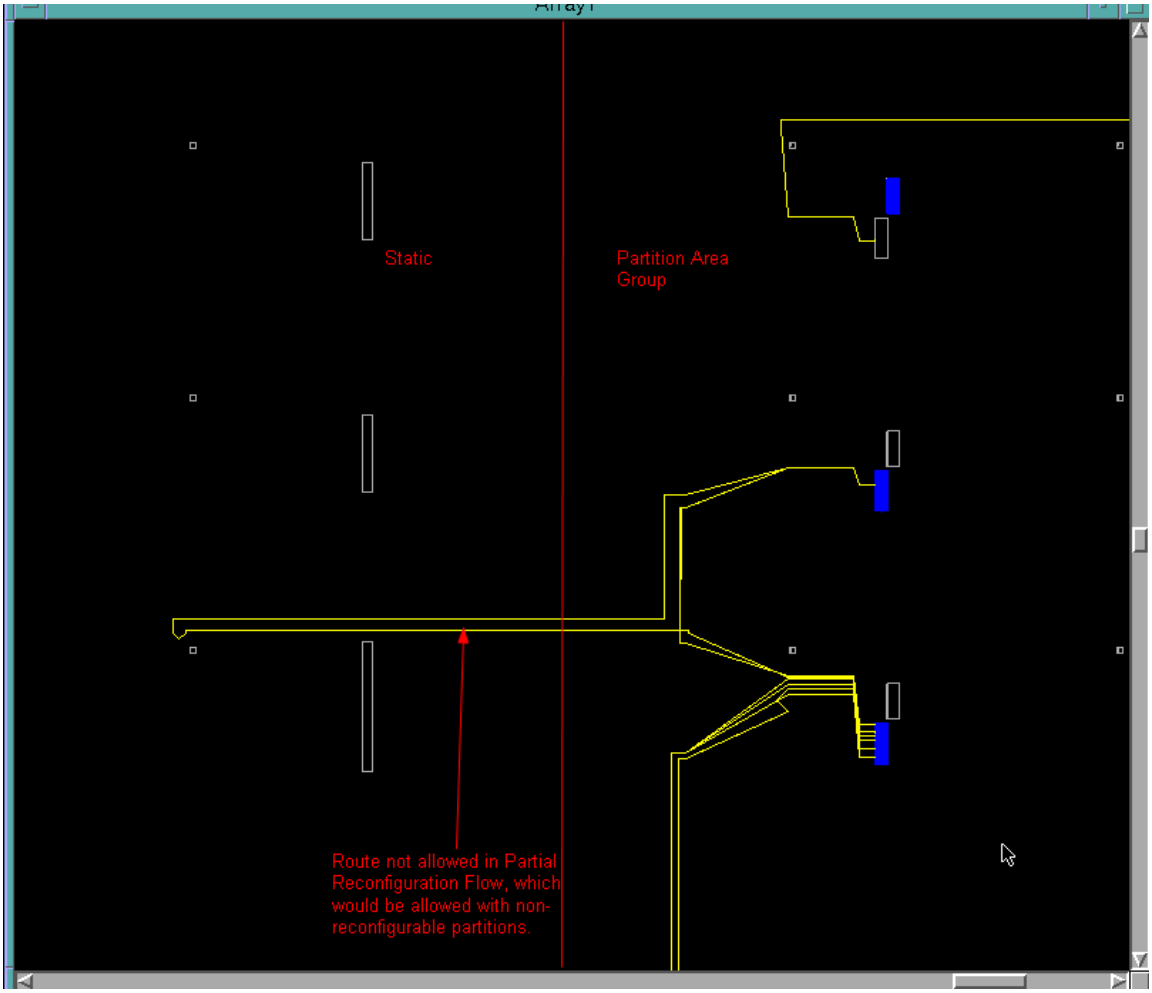

図 3-6 : パーシャル リコンフィギュレーションの配線制限

ここでは、配置配線問題のデバッグについて記述します。

- リコンフィギャブル パーティションのエリア グループは、平坦なデザインの同じエリア グループよりも大きくする必要があります。
- 配置ツールではこれらの配線制限が考慮されるので、使用できない配線リソースがあると配置エラーが発生することがあります。

配置でエラーが発生した場合は、xpartition.pxml ファイルで reconfigurable=”true” を削除して、リコンフィギャブル パーティション以外をテストしてみます。この修正を加える前、ファイルは次のようになっています。

```
<Project FileVersion="1.2" Name="FastConfig" ProjectVersion="2.0">
  <Partition Name="/top" State="implement">
    <Partition Name="/top/reconfig_red" Reconfigurable="true" State="implement" ReconfigModuleName="Red_Fast">
    </Partition>
    <Partition Name="/top/reconfig_green" Reconfigurable="true" State="implement" ReconfigModuleName="Green_Fast">
    </Partition>
    <Partition Name="/top/reconfig_blue" Reconfigurable="true" State="implement" ReconfigModuleName="Blue_Fast">
    </Partition>
  </Partition>
</Project>
```

修正を加えると、ファイルは次のようになります。

```
<Project FileVersion="1.2" Name="FastConfig" ProjectVersion="2.0">
  <Partition Name="/top" State="implement">
    <Partition Name="/top/reconfig_red" State="implement">
    </Partition>
    <Partition Name="/top/reconfig_green" State="implement">
    </Partition>
    <Partition Name="/top/reconfig_blue" State="implement">
    </Partition>
  </Partition>
</Project>
```

リコンフィギャブル パーティション以外の部分には同じ配線制限がないので、リコンフィギャブル パーティションがこの変更で問題なく配置配線される場合、エリア グループをそのリコンフィギャブル パーティションを配置配線するのに十分な大きさにする必要があります。

この変更後は、NGDBuild、MAP、および PAR を実行し直してください。

ビット ファイルの生成

NCD ファイルで bitgen コマンドを実行すると、フル ビット ファイルとパーシャル ビット ファイルの両方が生成されます。パーシャル ビット ファイルを生成するのに特別なオプションは必要ありませんが、パーシャル リコンフィギュレーション機能専用のオプションについては、このセクションの後半にリストしています。

```
bitgen -w FFF.ncd
```

デザインにリコンフィギャブル パーティションが含まれる場合、それぞれに対してパーシャル ビット ファイルが自動的に生成されます。フル ビット ファイルには、コンフィギュレーションで使用されたパーシャル モジュールが含まれます。

たとえば、デザイン例の 1 つ目のコンフィギュレーションでは次のファイルが生成されます。

```
fff.bit
```

(スタティック ロジックおよび red_fast、blue_fast、green_fast モジュール)

```
fff_reconfig_red_red_fast_partial.bit
```

(リコンフィギャブル パーティションの `red` に定義した範囲にあるロジックのみ)

```
fff_reconfig_blue_blue_fast_partial.bit
```

(リコンフィギャブル パーティションの `blue` に定義した範囲にあるロジックのみ)

```
fff_reconfig_green_green_fast_partial.bit
```

(リコンフィギャブル パーティションの `green` に定義した範囲にあるロジックのみ)

次の BitGen オプションをパーシャル リコンフィギュレーション デザインの該当箇所に設定する必要があります。

- **`-g ActiveReconfig:Yes`**

  ActiveReconfig オプションは、通常パーシャル リコンフィギュレーションで使用され、FPGA がシャットダウンされないようにします (GHIGH と GSR がアサートされないようにします)。

- **`-g Binary:Yes`**

  これにより、コンフィギュレーション データのみを含むバイナリ コンフィギュレーション (BIT ファイルからヘッダ情報を抜いたものと同じ) が生成されます。BIT ファイルには、さまざまな長さのヘッダ情報が含まれるので (常にワード バウンダリにあるわけではない)、カスタム コンフィギュレーション インターフェイスには BIN ファイルがよく使用されます。

- **`-g ConfigFallback:Disable`**

  このオプションを使用すると、パーシャル ビットストリームでコンフィギュレーション エラー (CRC エラー) 後にフル デバイス コンフィギュレーションがトリガされなくなります。このオプションは、Virtex-5 およびそれ以降のアーキテクチャで使用してください。

- **`-g CRC:enable`**

  これはデフォルトです。CRC をディスエーブルにするのはお勧めしません。

- **`-g Persist:Yes`**

  多目的コンフィギュレーション ピンがユーザー I/O として使用されないようにします。これは、Slave SelectMAP または Slave Serial モードがパーシャル リコンフィギュレーションで使用される場合に設定する必要があります。このオプションを CONFIG_MODE 制約と一緒に使用すると、適切なコンフィギュレーション ピンのセットがコンフィギュレーション後に使用されるように予約できます。CONFIG_MODE の値 (S_SELECTMAP、S_SERIAL など) については、『制約ガイド』を参照してください。

パーティション ベースのパーシャル リコンフィギュレーション フローでは BitGen オプションを使用しないでください。-r オプションでは相違ベースのフローがサポートされ、配線済みデザインに小さな変更を加えることができます。また、このオプションを使用すると、その変更点を比較してパーシャル ビット ファイルを構築できます。

これらも含めた BitGen オプションの詳細は、『コマンド ライン ツール ユーザー ガイド』の「BitGen」の章を参照してください。

レポート ファイル

NGDBuild、MAP、PAR、TRACE および BitGen のレポート ファイルには、リコンフィギャブル パーティションに特有の情報が含まれます。レポート ファイルには、次のようなファイルがあります。

- 「NGDBuild レポート」
- 「MAP レポート」
- 「PAR レポート」
- 「TRACE レポート」
- 「BitGen レポート」

次のサンプル レポートは、省略して表示しています。

NGDBuild レポート

NGDBuild レポートは、最上位レベルのスタティック パーティションも含め、どのパーティションがインプリメントされ、どのパーティションが保持されたかが表示されます。この例では、最上位レベルのスタティック パーティションが保持され、3 つのリコンフィギャブル パーティションがインプリメントされています。

```
Partition Implementation Status
-------------------------------

 Preserved Partitions:

  Partition "/top"


 Implemented Partitions:

  Partition "/top/reconfig_red" (Reconfigurable Module "red_fast"):
Attribute STATE set to IMPLEMENT.

  Partition "/top/reconfig_blue" (Reconfigurable Module "blue_fast"):
Attribute STATE set to IMPLEMENT.

  Partition "/top/reconfig_green" (Reconfigurable Module "green_fast"):
Attribute STATE set to IMPLEMENT.
```

MAP レポート

NGDBuild レポートと同様、MAP レポート (.mrp) にはすべてのパーティションがインプリメントされたことが示されますが、最上位レベルのスタティック パーティションだけは表示されません。

```
Section 9 - Area Group and Partition Summary
--------------------------------------------

Partition Implementation Status
-------------------------------

  Preserved Partitions:

    Partition "/top"


  Implemented Partitions:

    Partition "/top/reconfig_red" (Reconfigurable Module "red_fast"):
Attribute STATE set to IMPLEMENT.

   Partition "/top/reconfig_blue" (Reconfigurable Module "blue_fast"):
Attribute STATE set to IMPLEMENT.

    Partition "/top/reconfig_green" (Reconfigurable Module
"green_fast"):
Attribute STATE set to IMPLEMENT.
```

「Partition Resource Summary」には、デザインの各パーティションで使用されたリソース数がレポートされます。また、どのエリア グループがどのリコンフィギャブル パーティションに関連しているかもレポートされます。

次の例では、AREA GROUP pblock_reconfig_red がリコンフィギャブル パーティションの /top/reconfig_red と関連していることがわかります。

```
Partition Resource Summary:
---------------------------
Resources are reported for each Partition followed in parenthesis by resources
for the Partition plus all of its descendants.

Partition "/top":
  State=implement
  Slice Logic Utilization:
    Number of Slice Registers:             113 (188)
    Number of Slice LUTs:                  148 (274)
      Number used as logic:                146 (272)
      Number used as Memory:                 2 (2)
  Slice Logic Distribution:
    Number of occupied Slices:              60 (105)
    Number of LUT Flip Flop pairs used:    157 (288)
      Number with an unused Flip Flop:      44 out of    157   28%
      Number with an unused LUT:             7 out of    157    4%
      Number of fully used LUT-FF pairs:   106 out of    157   67%
  IO Utilization:
    Number of bonded IOBs:                  26 (26)
  Number of MMCM_ADV:                       1 (1)
  Number of OLOGICE1:                      17 (17)
  Number of STARTUP:                        1 (1)

Partition "/top/reconfig_blue" (Reconfigurable Module "Blue_Fast") (Area Group "AG_reconfig_blue"):
  State=implement
  Slice Logic Utilization:
    Number of Slice Registers:              25 (25)
    Number of Slice LUTs:                   42 (42)
      Number used as logic:                 42 (42)
  Slice Logic Distribution:
    Number of occupied Slices:              15 (15)
    Number of LUT Flip Flop pairs used:     44 (44)
      Number with an unused Flip Flop:      19 out of     44   43%
      Number with an unused LUT:             1 out of     44    2%
      Number of fully used LUT-FF pairs:    24 out of     44   54%
```

次の MAP レポートのセクションには、UCF ファイルで定義された物理エリア グループに含まれるリソースの使用率が % で表示されています。この例では、AG_reconfig_blue エリア グループにはスライス (LUT および FF) 用に関連付けられた範囲が 1 つ指定されています。AG_RP_green エリア グループにはブロック RAM とスライスの範囲が指定されています。

```
Area Group Information
----------------------

Area Group "AG_reconfig_blue"
  No COMPRESSION specified for Area Group "AG_reconfig_blue"
  RANGE:SLICE_X74Y0:SLICE_X83Y79
  Slice Logic Utilization:
    Number of Slice Registers:              25 out of  6,400    1%
    Number of Slice LUTs:                   42 out of  3,200    1%
      Number used as logic:                 42
  Slice Logic Distribution:
    Number of occupied Slices:              15 out of    800    1%
    Number of LUT Flip Flop pairs used:     44
      Number with an unused Flip Flop:      19 out of     44   43%
      Number with an unused LUT:             1 out of     44    2%
      Number of fully used LUT-FF pairs:    24 out of     44   54%
```

PAR レポート

NGDBuild レポートと MAP レポートと同様、PAR レポートにもどのパーティションがインプリメントされたかが表示されます。

```
Partition Implementation Status
-------------------------------

 Preserved Partitions:

  Partition "/top"


 Implemented Partitions:

  Partition "/top/reconfig_red" (Reconfigurable Module "red_fast"):
Attribute STATE set to IMPLEMENT.

  Partition "/top/reconfig_blue" (Reconfigurable Module "blue_fast"):
Attribute STATE set to IMPLEMENT.

  Partition "/top/reconfig_green" (Reconfigurable Module "green_fast"):
Attribute STATE set to IMPLEMENT.
```

TRACE レポート

TRACE ツールは、FPGA デザインのスタティック タイミング解析を実行するために使用されます。このツールは、タイミング検証とレポートの両方に使用されます。TRACE の使用方法については、『コマンド ライン ツール ユーザー ガイド』(UG628) の「TRACE」セクションを参照してください。

パーシャル リコンフィギュレーション デザイン フローは、常に完全なデザインと連動します。これにより、リコンフィギャブル モジュールの解析の際に、タイミング解析でスタティック領域に適用された制約を再利用できるようになります (つまり、スタティック領域のクロックに適用された PERIOD 制約により、リコンフィギャブル モジュールの該当パスの解析が実行されます)。スタティック ロジックは常に解析されます。

TRACE では複数の出力ファイルを生成できます。次の 3 つのファイルは、ユーザー定義の制約をデザインがどれくらい満たしているかを確認する場合に使用できます。

- **TWR** - ASCII 形式のタイミング レポート
- **TWX** - XML 形式のタイミング レポート
- **TSI** - ASCII 形式の制約相互作用レポート

TWR と TWX タイミング レポートは、各コンフィギュレーションをインプリメンテーションすると作成されます。別のオプションを使用したそれ以外のレポートが必要な場合は、TRACE をコマンド ラインから実行するか、PlanAhead ソフトウェアの該当するインプリメンテーション オプションを変更してから、インプリメンテーションを実行し直します。

リコンフィギャブル パーティションを含むデザインのスタティック タイミング解析を実行するのは、普通のデザインのスタティック タイミングを解析する場合と同じですが、手法が異なります。パーシャル リコンフィギュレーション デザインの場合、タイミング解析はコンフィギュレーションごとに実行する必要があります。

次は、TRACE コマンド ラインの例です。この例で使用されるオプションの詳細は、『コマンド ライン ツール ユーザー ガイド』(UG628) の「TRACE」の章を参照してください。

```
trce –v 10 –u 10 –tsi top.tsi –o top.twr –xml top.twx top top.pcf
```

タイミング レポートは、パーティション ピンを通るパスを検証するためにも使用できます。このロジックは、キーワード「PROXY」で検索できます。LUT 名に _PROXY が付いた名前は、LUT がプロキシ ロジックとして使用されることを示し、パーティション ピンがこのプロキシ ロジックに含まれることを意味します。

次の例では、これらの制約と共に、TPSYNC 制約が red.addr バスに適用されています。

```
PIN “red.addr(*)” TPSYNC = “group_RP_red_input”;
TIMESPEC TS_from_static_to_PP_input = TO “group_RP_red_input” 4.5 ns;
```

このパスのソースは、スタティック領域に含まれます。デスティネーションは、プロキシ ロジックとして挿入された LUT です。この特定パスのデスティネーション パスは、red.addr(11) です。これは、パーティション名が red で、ポート名が addr(11) であることを示しています。

この解析では、レジスタの clock-to-out タイムとネット遅延がパーティション ピンまで考慮されていることがわかります。パーティション ピンまでの遅延は、このパス解析では考慮されていません。

```
Timing constraint:TS_from_static_to_PP_input = MAXDELAY TO TIMEGRP
“group_RP_red_input” 4.5 ns;

 12 paths analyzed, 12 endpoints analyzed, 0 failing endpoints
 0 timing errors detected.(0 setup errors, 0 hold errors)
 Maximum delay is   1.111ns.
--------------------------------------------------------------------------------
Slack:                3.389ns (requirement - data path)
  Source:             count_34 (FF)
  Destination:        red/addr(11)_PROXY (LUT) (red.addr(11))
  Requirement:        4.500ns
  Data Path Delay:    1.111ns (Levels of Logic = 0)
  Source Clock:       gclk rising at 0.000ns

 Maximum Data Path: count_34 to RP_red/addr(11)_PROXY
   Location         Delay type      Delay(ns)  Physical Resource
                                               Logical Resource(s)
   ---------------------------------------  ----------------------
   SLICE_X47Y39.CQ  Tcko               0.326   count[34]
                                               count_34
   SLICE_X45Y37.A1  net (fanout=2)     0.785   count[34]
   ---------------------------------------  ----------------------
   Total                           1.111ns (0.326ns logic, 0.785ns route)
                                           (29.3% logic, 70.7% route)
```

45 ページの図 3-7 は、スタティック FF からパーティション ピンまでのパスを示しています。

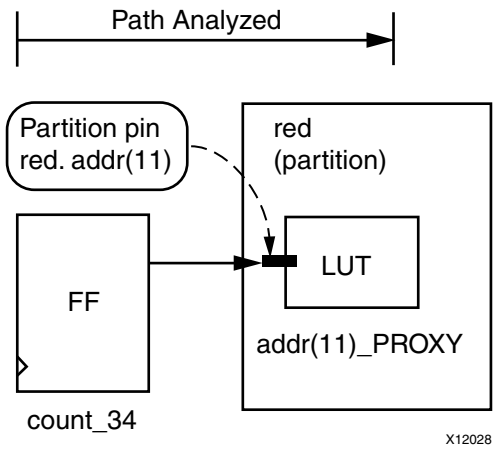

図 3-7 : スタティック FF からパーティション ピンまでのパス

次の例では、これらの制約と共に、TPSYNC 制約が red.d_out バスに適用されています。

```
PIN "red.d_out(*)" TPSYNC = "Bram_output_PPs";
TIMESPEC TS_from_PP_output_to_static = FROM "Bram_output_PPs" 5.0 ns;
```

このパスのソースは、リコンフィギャブル パーティションの出力のプロキシ ロジックです。デスティネーションは、スタティック領域の PAD です。この特定パスのソース名は red.d_out(5) で、red はパーティション名、d_out(5) はポート名を示しています。

次の解析では、プロキシ ロジックを介した伝搬時間と、出力バッファへのネット遅延、その後の出力バッファを介した PAD までの伝搬遅延が考慮されています。

```
Timing constraint:TS_from_PP_output_to_static = MAXDELAY FROM TIMEGRP
"Bram_output_PPs" 5.0 ns;

 8 paths analyzed, 8 endpoints analyzed, 0 failing endpoints
 0 timing errors detected.(0 setup errors, 0 hold errors)
 Maximum delay is   4.770ns.
--------------------------------------------------------------------------------
Slack:            0.230ns (requirement - data path)
 Source:          red/d_out(5)_PROXY (LUT) (red.d_out(5))
 Destination:     out_bram[5] (PAD)
 Requirement:     5.000ns
 Data Path Delay:4.770ns (Levels of Logic = 2)

 Maximum Data Path:U1_RP_Bram/d_out(5)_PROXY to out_bram[5]
   Location             Delay type    Delay(ns)  Physical Resource
                                                 Logical Resource(s)
                                                 (Partition Pin)
   -------------------------------------------  -------------------
   SLICE_X33Y38.B       Tilo             0.080   red/d_out(5)_PROXY
                                                 red/d_out(5)_PROXY
                                                 (red.d_out(5))
   G15.O                net (fanout=1)   2.514   out_bram_5_OBUF
   G15.PAD              Tioop            2.176   out_bram[5]
                                                 out_bram_5_OBUF
                                                 out_bram[5]
   -------------------------------------------  -------------------
  Total                                 4.770ns (2.256ns logic, 2.514ns rte)
                                                (47.3% logic, 52.7% route)
```

図 3-8 は、パーティション ピンからスタティック PAD までの解析パスを示しています。

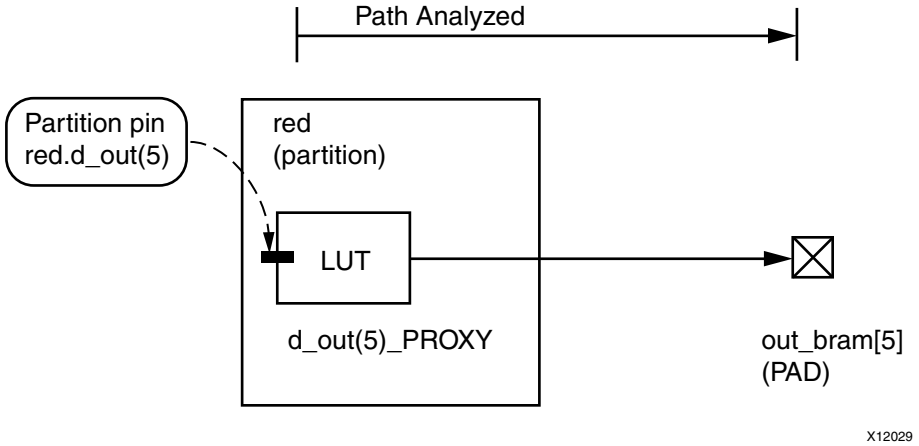

図 3-8 : パーティション ピンからスタティック PAD までパス

次の例では、PERIOD 制約が static_VGA_vgaclk2_i クロック信号に適用され、関連する PERIOD 制約が VGA_CLK クロック信号に適用されています (どちらもスタティック領域にあります)。

このパスのソースとデスティネーションはスタティック領域の FF (フリップフロップ) です。ソースとデスティネーション間のパスはプロキシ ロジックを通り、リコンフィギャブル パーティションを通った後、さらに多くのプロキシ ロジックへ戻り、最後にスタティック領域の FF に到着します。このパスの最初のパーティション ピンの名前 red.VGA_in7 は、パーティション名が red、ポート名が VGA_in7 であることを示しています。このパスの 2 つ目のパーティション ピンの名前 red.VGA_out7 は、パーティション名が red、ポート名が VGA_out7 であることを示しています。

次の解析ファイルの例では、パーティション ピンの伝搬遅延も含めたパス全体が考慮されています。このパスには違反がありますが、この違反はパーティション内にレジスタを追加すると回避できます。リコンフィギャブル パーティションを通る組み合わせパスは、2 つの LUT 遅延が追加されるだけでなく、第 7 章の「デカップリング機能」に示すロジック デカップリングが実行されないため、使用しないようにしてください。

```
Timing constraint:TS_static_VGA_vgaclk2_i = PERIOD TIMEGRP
“static_VGA_vgaclk2_i”  TS_static_VGA_pixel_clock_i PHASE 3.167 ns HIGH 50%;

 126 paths analyzed, 36 endpoints analyzed, 10 failing endpoints
 10 timing errors detected.(10 setup errors, 0 hold errors, 0 component switching limit
errors)
 Minimum period is  15.401ns.
--------------------------------------------------------------------------------
Slack:                  -0.451ns (req-(data path-clock path skew + uncer'ty))
  Source:               static_VGA/VGA_R_1[0] (FF)
  Destination:          static_DVI_IF/ODDR_DVI_DATA11 (FF)
  Requirement:          3.167ns
  Data Path Delay:      3.387ns (Levels of Logic = 2)
  Clock Path Skew:      0.084ns (1.427 - 1.343)
  Source Clock:         static_VGA/pixel_clock rising at 0.000ns
  Destination Clock:    VGA_CLK rising at 3.167ns
  Clock Uncertainty:    0.315ns

  Clock Uncertainty:          0.315ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):0.070ns
    Total Input Jitter (TIJ):   0.000ns
    Discrete Jitter (DJ):       0.458ns
    Phase Error (PE):           0.050ns

  Maximum Data Path: static_VGA/VGA_R_1[0] to static_DVI_IF/ODDR_DVI_DATA11
    Location             Delay type      Delay(ns)  Physical Resource
                                                    Logical Resource(s)
                                                    (Partition Pin)
    -------------------------------------------  -------------------
    SLICE_X25Y75.DQ      Tcko               0.326   VGA_R_bus_out[1]
                                                    static_VGA/VGA_R_1[0]
    SLICE_X25Y76.C6      net (fanout=8)     0.248   VGA_R_bus_out[1]
    SLICE_X25Y76.C       Tilo               0.080   red/VGA_out7_PROXY
                                                    red/VGA_in7_PROXY
                                                    (red.VGA_in7)
    SLICE_X25Y76.D5      net (fanout=1)     0.164   red/VGA_out7
    SLICE_X25Y76.D       Tilo               0.080   red/VGA_out7_PROXY
                                                    red/VGA_out7_PROXY
                                                    (red.VGA_out7)
    OLOGIC_X2Y39.D1      net (fanout=1)     2.192   VGA_R[7]
    OLOGIC_X2Y39.CLK     Todck              0.297   DVI_LCD_DATA11_c
                                                  static_DVI_IF/ODDR_DVI_DATA11
    -------------------------------------------  ---------------------
    Total                                 3.387ns (0.783ns logic, 2.604ns rte)
                                                    (23.1% logic, 76.9% rte)
```

48 ページの図 3-9 は、パーティション ピンを通る FF から FF までのパスを示しています。

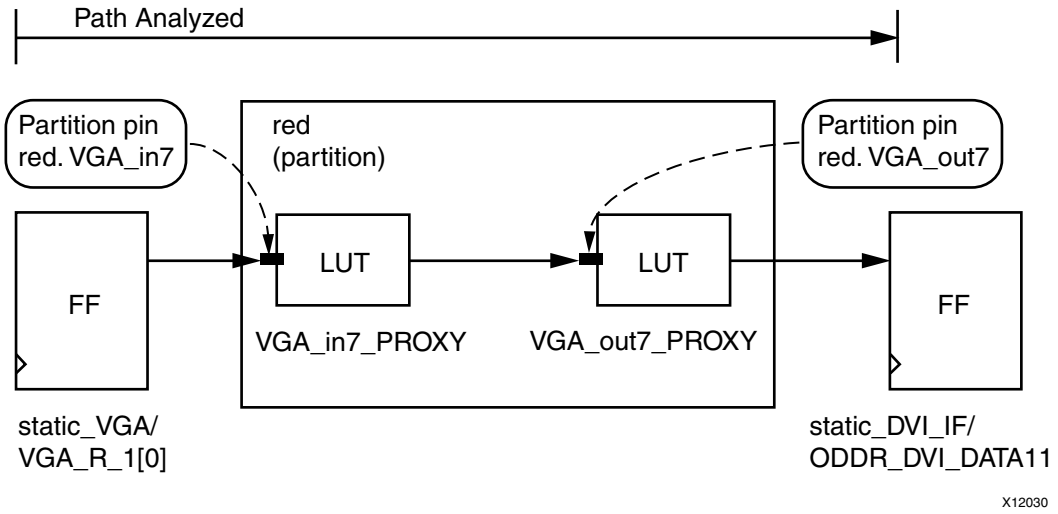

図 3-9 : パーティション ピンを通る FF から FF までのパス

BitGen レポート

BitGen を実行すると、各パーシャル ビット ファイルとフル ビット ファイルのレポート ファイルが作成されます。フル ビット ファイルのレポートには、フル ビット ファイルに含まれるリコンフィギャブル モジュールすべてがリストされ、ActiveReconfig = No 設定でそれがパーシャル ビット ファイルではないことが示されます。

```
...
Partition “/top/reconfig_red” (Reconfigurable Module “red_fast”)
Partition “/top/reconfig_blue” (Reconfigurable Module “blue_fast”)
Partition “/top/reconfig_green” (Reconfigurable Module “green_fast”)
...
Summary of Bitgen Options:
+----------------------+----------------------+
| ActiveReconfig     | No*          |
+----------------------+----------------------+
| Partial        | (Not Specified)*    |
+----------------------+----------------------+
...
* Default setting.
```

パーシャル ビット ファイルのレポートには、それがパーシャル ビット ファイルであることと、パーティションとリコンフィギャブル モジュールがそれぞれどれに関連しているのかが示されます。

```
...
Summary of Bitgen Options:
+----------------------+----------------------+
| ActiveReconfig     | Yes          |
+----------------------+----------------------+
| Partial        | reconfig_red      |
+----------------------+----------------------+
...
Creating bit stream for Partition “/top/reconfig_red”
   (Reconfigurable Module “red_fast”)
Creating bit map...
Saving bit stream in “fff_reconfig_red_red_fast_partial.bit”.
```

pr_verify

パーシャル リコンフィギュレーション デザインがハードウェアで動作するようにするには、すべてのコンフィギュレーション間でスタティック ロジックの配置配線が矛盾しないようにする必要があります。また、プロキシ ロジックも同じ位置に配置し、クロック スパイン配線も一致するようにする必要があります。pr_verify ユーティリティを使用すると、パーシャル リコンフィギュレーション デザイン用に作成された複数のコンフィギュレーションから配線済みの NCD ファイルが比較され、インポートされたリソースがすべて一致するかどうか確認されます。これらのリソースには、次が含まれます。

- グローバル クロック スパイン – 各グローバル クロックには、すべてのコンフィギュレーションで同じクロック領域内に配線されたクロック スパインが必要です。
- リージョナル クロック スパイン – Virtex-5 以外のアーキテクチャでは、すべてのコンフィギュレーションで各リージョナル クロックに同じクロック領域内に配線されたクロック スパインが必要です。詳細は、第 7 章の「リージョナル クロック」を参照してください。
- プロキシ ロジック – プロキシ ロジックは、論理的にはスタティック デザインの一部ですが、リコンフィギャブル パーティション用に割り当てられたエリア グループ内の同じ位置に配置する必要があります。
- インポートされたパーティション – コンフィギュレーション間でインポートされたパーティションはすべて同じ配置配線にする必要があります。スタティック部分と複数のコンフィギュレーションで使用されるリコンフィギャブル モジュールの両方が確認されます。
- パーティション インターフェイス - リコンフィギャブル パーティションには、それぞれ各コンフィギュレーションのリコンフィギャブル モジュールへの入力と出力に同じポートが使用される必要があります。

pr_verify の使用方法

pr_verify は、PlanAhead またはコマンド ラインのどちらかで実行できます。PlanAhead での実行方法については、第 4 章の「コンフィギュレーションの検証」を参照してください。

コマンド ライン構文

pr_verify [**-verbose**] <*design1*[.ncd]> <*design2*[.ncd]> [<*design*[.ncd]>] [**-o** <*outfile*>]

-verbose – すべてのメッセージがレポートされます。

-o <*outfile*> – 拡張子も含めた出力ファイル名を指定します。このオプションを使用しない場合は、デフォルトの pr_verify.log ファイルが作成されます。

<*design**[.ncd]> – 比較する少なくとも 2 つの NCD ファイルを入力します。

本書のデザイン例の場合、pr_verify コマンド ラインは次のようになります。

```
pr_verify -verbose ./FastConfig/FastConfig.ncd
./SlowConfig/SlowConfig.ncd ./FSFConfig/FSFConfig.ncd
./BlankConfig/BlankConfig.ncd
```

pr_verify ログ ファイル

上記のコマンド ライン例を実行すると、次のような pr_verify ログ ファイルが出力されます。

```
Command Line:/Xilinx/13.1/ISE_DS/ISE/bin/lin/unwrapped/pr_verify
./BlankConfig/BlankConfig.ncd ./FastConfig/FastConfig.ncd
./FSFConfig/FSFConfig.ncd ./SlowConfig/SlowConfig.ncd

Loading ./BlankConfig/BlankConfig.ncd:Mon Feb 14 14:53:16 2011
Loading ./FastConfig/FastConfig.ncd:Mon Feb 14 14:35:32 2011
Loading ./FSFConfig/FSFConfig.ncd:Mon Feb 14 14:47:54 2011
Loading ./SlowConfig/SlowConfig.ncd:Mon Feb 14 16 14:40:58 2011

----------------------------------------
Analyzing Designs:
  ./BlankConfig/BlankConfig.ncd
  ./FastConfig/FastConfig.ncd

  Number of matched proxy logic bels          = 54
  Number of matched external nets             = 33
  Number of matched global clock nets         = 4
  Number of matched Reconfigurable Partitions = 0

SUCCESS!

----------------------------------------
Analyzing Designs:
  ./FastConfig/FastConfig.ncd
  ./FSFConfig/FSFConfig.ncd

  Number of matched proxy logic bels          = 54
  Number of matched external nets             = 33
  Number of matched global clock nets         = 4
  Number of matched Reconfigurable Partitions = 2

SUCCESS!

----------------------------------------
Analyzing Designs:
  ./FSFConfig/FSFConfig.ncd
  ./BlankConfig/BlankConfig.ncd

  Number of matched proxy logic bels          = 54
  Number of matched external nets             = 33
  Number of matched global clock nets         = 4
  Number of matched Reconfigurable Partitions = 0

SUCCESS!

----------------------------------------
Analyzing Designs:
  ./FSFConfig/FSFConfig.ncd
  ./SlowConfig/SlowConfig.ncd

  Number of matched proxy logic bels          = 54
  Number of matched external nets             = 33
  Number of matched global clock nets         = 4
  Number of matched Reconfigurable Partitions = 1

SUCCESS!
```

```
----------------------------------------
Analyzing Designs:
  ./SlowConfig/SlowConfig.ncd
  ./BlankConfig/BlankConfig.ncd

  Number of matched proxy logic bels          = 54
  Number of matched external nets             = 33
  Number of matched global clock nets         = 4
  Number of matched Reconfigurable Partitions = 0

SUCCESS!

----------------------------------------
Analyzing Designs:
  ./SlowConfig/SlowConfig.ncd
  ./FastConfig/FastConfig.ncd

  Number of matched proxy logic bels          = 54
  Number of matched external nets             = 33
  Number of matched global clock nets         = 4
  Number of matched Reconfigurable Partitions = 0

SUCCESS!

/Xilinx/13.1/ISE_DS/ISE/bin/lin/unwrapped/pr_verify
./BlankConfig/BlankConfig.ncd ./FastConfig/FastConfig.ncd
./FSFConfig/FSFConfig.ncd ./SlowConfig/SlowConfig.ncd => PASS
```

ログ ファイルから、NCD ファイルが 1 度に 2 つ比較され、コンフィギュレーションの特定情報と矛盾するリソースが表示されていることがわかります。最後の行には、その run 全体がパス (PASS) したかエラー (FAIL) だったかが表示されます。

ログ ファイルには、次のリソース比較が表示されています。

- `Number of matched proxy logic bels`

  これは、これら 2 つのコンフィギュレーションの存在と位置の両方が同じプロキシ ロジックとして使用される LUT1 の数を示します。この数はすべての解析で同じになるはずです。

- `Number of matched external nets`

  これは、これら 2 つのコンフィギュレーションのリコンフィギャブル モジュールの入力または出力ポートの数を示します。この数はすべての解析で同じになるはずです。

- `Number of matched global clock nets`

  これは、これら 2 つのコンフィギュレーション間に常に配線されるグローバル クロックのネット数を示します。この数はすべての解析で同じになるはずです。

- `Number of matched Reconfigurable Partitions`

  これは、これらコンフィギュレーション両方で使用される同じインプリメンテーションのリコンフィギャブル モジュールの数を示します。この数はすべての解析で同じである必要はありません。たとえば、BlankConfig および FastConfig ではスタティックのみが共通しているので、これらのコンフィギュレーションの解析では一致するリコンフィギャブル パーティションは 0 と表示されます。ただし、FSFConfig と FastConfig にはスタティックが含まれ、Red_Fast と Green_Fast が共通しているので、一致するリコンフィギャブル パーティションは 2 つになります。

フローの違い

パーシャル リコンフィギュレーションのフローは、インプリメンテーション ツールを使用した標準的なフローとほとんど同じですが、安全なシリコン結果を作成するには、配置配線に制限を加える必要があります。これらの制限により、パフォーマンス、パッキング密度、インプリメンテーションの柔軟度などが影響を受けます。

表 3-1 : フローの違い

フロー	配置	配線
標準	デバイス制限以外の制限はありません。	デバイス制限以外の制限はありません。
パーティション	インポートされたロジックが最初に配置されます。インプリメントされたロジックが 2 つ目に配置されます。 エリア グループ要件はありません。	インポートされたロジックが最初に配線されます。 インプリメントされたロジックが 2 つ目に配置されます。
パーシャル リコンフィギュレーション	LOC 制約で強制的に位置を指定されない限り、リコンフィギャブル パーティションのエリア グループに配置できるのはリコンフィギャブル ロジックのみです。 配線制限は、配置段階で考慮されます。	リコンフィギャブル パーティションのエリア グループ外にまたがる配線リソースは、リコンフィギャブル ロジックには使用できません。 インポートされたロジックが最初に配線されます。 インプリメントされたロジックが 2 つ目に配置されます。

このようにフローに違いがあるため、標準フローまたはパーティション フローで問題なくインプリメントされたデザインでも、パーシャル リコンフィギュレーション フローでは同じタイミングや密度にならないことがあります。どれくらい違うかは、デザインによって異なります。

XILINX

第 4 章

PlanAhead サポート

この章では、PlanAhead ソフトウェアを使用したパーシャル リコンフィギュレーション デザインの作成手順について説明します。説明は、まず合成後から開始します。デザインは、RTL でコード記述され、第 3 章「ソフトウェア ツール フロー」の説明どおりに合成されているものとします。

このユーザー ガイドでは、PlanAhead の基礎知識があるものとして説明します。PlanAhead の基礎知識については、『PlanAhead ユーザー ガイド』(UG632) および『PlanAhead ソフトウェア チュートリアル : クイック フロー概要』(UG673) を参照してください。

パーシャル リコンフィギュレーション プロジェクトの作成

パーシャル リコンフィギュレーション プロジェクトを作成するには、次の手順に従います。

1. New Project ウィザードを起動し、プロジェクト名とディレクトリを指定し、合成済みネットリスト (EDIF または NGC) をインポートします。パーシャル リコンフィギュレーション プロジェクトは、PlanAhead ソフトウェアの RTL レベルでは開始できません。これをパーシャル リコンフィギュレーション プロジェクトとして定義するには、[Set PR Project] をオンにします。図 4-1 は、New Project ウィザードを示しています。

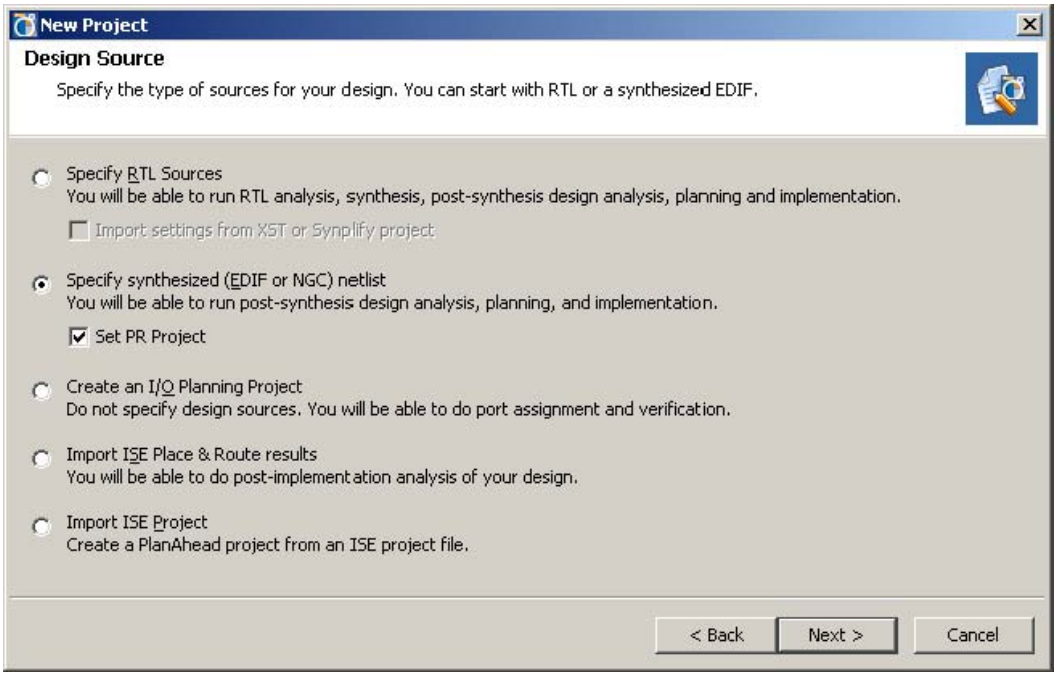

図 4-1 : New Project ウィザード

2. 最上位レベルのネットリストを指定します。ネットリストのディレクトリは、スタティック ロジックのほかのモジュールがあるディレクトリに設定する必要があります。この例では、スタティック ロジックはすべて top.ngc に含まれています。下位レベル (リコンフィギャブル) モジュールは後でインポートします。図 4-2 は、[New Project] → [Specify Top Netlist File] ダイアログ ボックスを示しています。

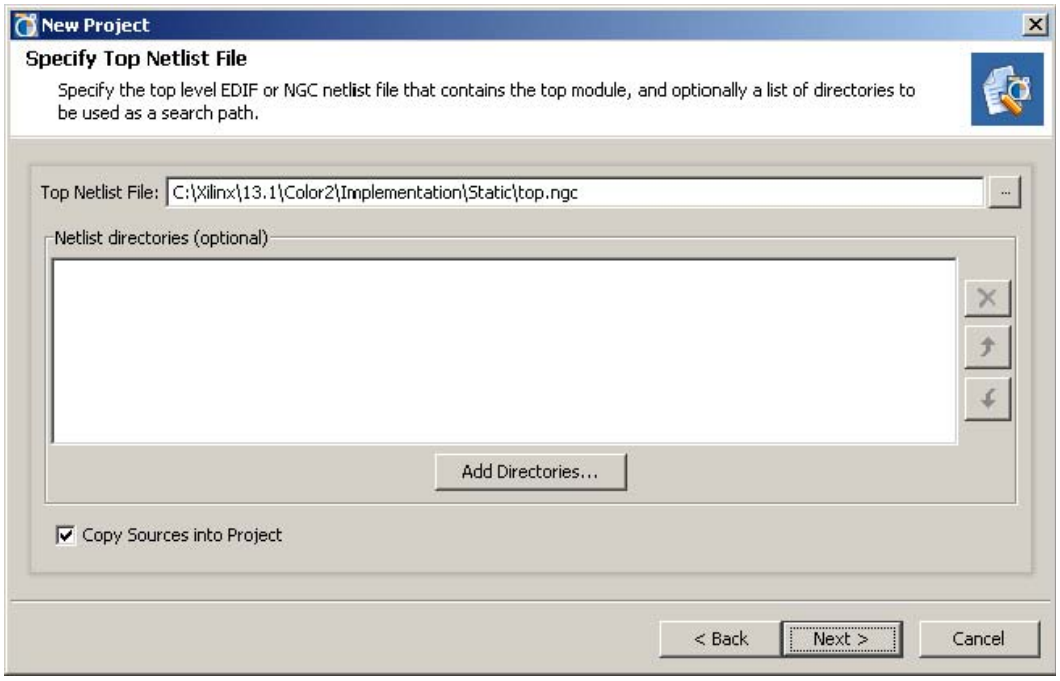

図 4-2 : スタティック ロジックのみを含むネットリストのインポート

3. I/O およびタイミング制約を含む最上位レベルの制約ファイルを追加します。複数の UCF ファイルを使用できます。PlanAhead ソフトウェアは、インプリメンテーションの実行前にすべての最上位レベルの UCF ファイルをモジュール レベルの UCF ファイルと関連付けます。図 4-3 は、[New Project] → [Constraint Files] ダイアログ ボックスを示しています。

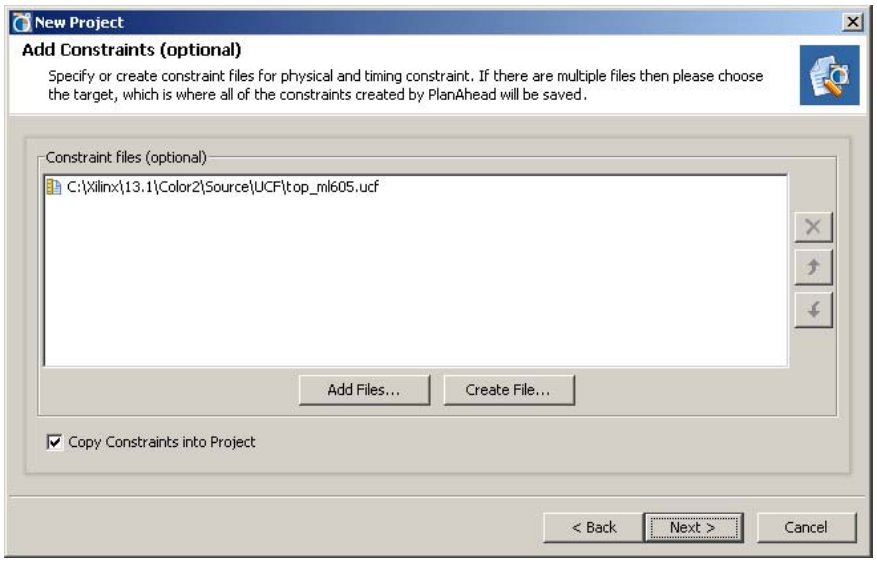

図 4-3 : [Constraints Files] ダイアログ ボックス

PlanAhead ソフトウェアは、ネットリストからターゲット デバイスを読み出します。

4. ターゲット デバイスが正しいかどうか確認し (必要であれば変更し)、[Next] をクリックします。
5. ウィザードの残りをクリックして、プロジェクトを作成します。

パーシャル リコンフィギュレーション プロジェクトとしてのプロジェクトの設定

プロジェクトを作成したときにパーシャル リコンフィギュレーション プロジェクトとしてプロジェクトを定義しなかった場合は、パーシャル リコンフィギュレーション プロジェクトとしてプロジェクトを定義するプロジェクト設定を使用して、パーシャル リコンフィギュレーション関連のコマンドを実行します。このオプションは、有効なパーシャル リコンフィギュレーション ライセンスがある場合にのみ表示され、XILINX 環境変数は古いバージョンの ISE® ツールのインストール ディレクトリをポイントしません。

プロジェクトをパーシャル リコンフィギュレーション プロジェクトとして設定するには、次の手順に従います。

- [Tools] → [Project Options] をクリックし、[General] タブで [Partial Reconfiguration Project] をオンにします。

プロジェクトをパーシャル リコンフィギュレーション プロジェクトとして設定すると、平坦な ISE インプリメンテーションには使用できなくなります。インターフェイスとオプションは、パーシャル リコンフィギュレーション ソフトウェアの機能を使用できるように変更され、平坦なデザインには不必要であった制限が含まれるようになります。

このオプションを選択すると、PlanAhead のインターフェイスがパーシャル リコンフィギュレーション デザイン用に変更されます。リコンフィギャブル パーティションとしてインスタンスを設定したり、インスタンスにリコンフィギャブル モジュールをさらに追加したりするその他のコマンドは、[Netlist] ビューのポップアップ メニューから実行できます。このオプションを選択すると、図 4-4 のように、右下のステータス バーが [Post-Synthesis Flow] から [Partial Reconfiguration Flow] に変わります。

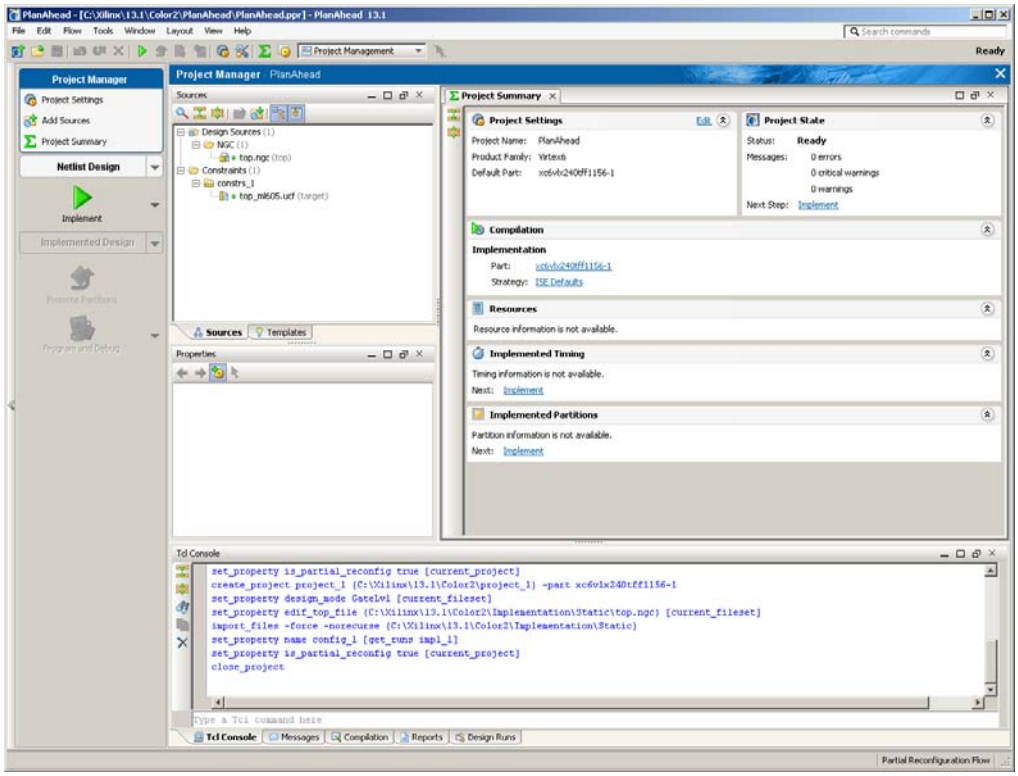

図 4-4 : PlanAhead のパーシャル リコンフィギュレーション プロジェクト

ネットリスト デザインを開く

PlanAhead で Project Manager ページが開きます。デザインを開始するには、まずネットリストをメモリに読み込む必要があります。Flow Manager で [Netlist Design] をクリックします。

ネットリストが読み込まれたら、未定義のモジュールがあることを示す警告メッセージが表示されます (図 4-5)。このメッセージは、インポートされたネットリストにデザイン全体が記述されていないことを示しています。リストされるモジュールがリコンフィギュレーションされるモジュールであるかどか確認してください。

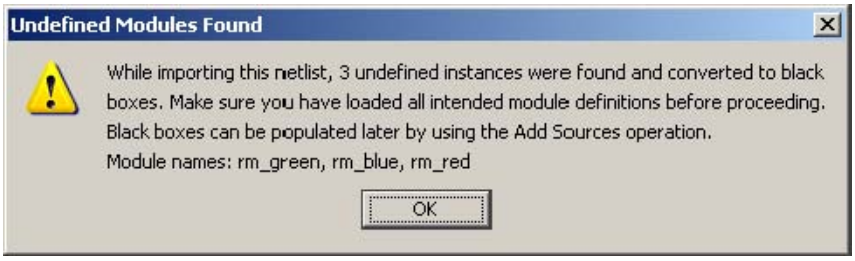

図 4-5 : 必ず表示される警告メッセージ

56 ページの図 4-4 のデザイン例の [Netlist] ビューでは、reconfig_blue、reconfig_green、reconfig_red という 3 つのブラック ボックス インスタンスにブラック ボックス アイコンが表示されています。これは、これらのブラック ボックスにネットリストが関連付けられていないからです。

その他の [Netlist] ビューのアイコンの意味については、『PlanAhead ユーザー ガイド』(UG632)を参照してください。

これらにリンクされたリコンフィギャブル モジュールのネットリストは、blue、green、red のそれぞれに Fast と Slow があります。

リコンフィギャブル インスタンスの定義

リコンフィギャブル パーティションは、下位レベル インスタンスを選択し、[Set Partition] コマンドを実行すると定義できます。

1. 図 4-6 に示すように [Set Partition] をクリックします。

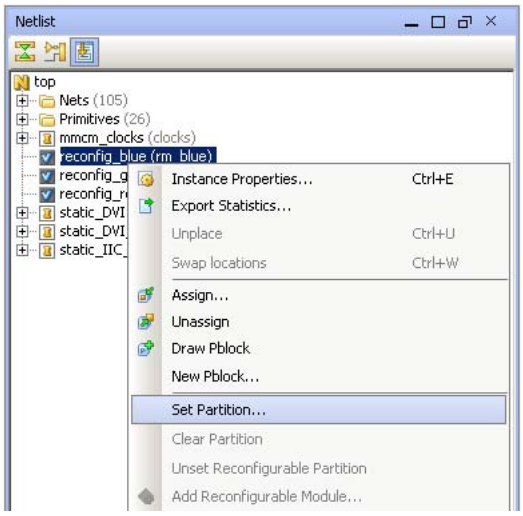

図 4-6 : パーティションをリコンフィギャブルとして設定

パーティションは、リコンフィギャブルまたは標準に設定できます。ここでは、このモジュールに関連付けられたネットリストがないので、リコンフィギャブルとしてしか定義できません。リコンフィギャブル パーティションには、読み込まれたリコンフィギャブル モジュールのネットリストを含めることができるほか、オプションでブラック ボックス モジュールとして定義することもできます。

2. 図 4-7 のように、モジュール名とそれが Fast か Slow かを示すように、リコンフィギャブル モジュールの名前を入力します。

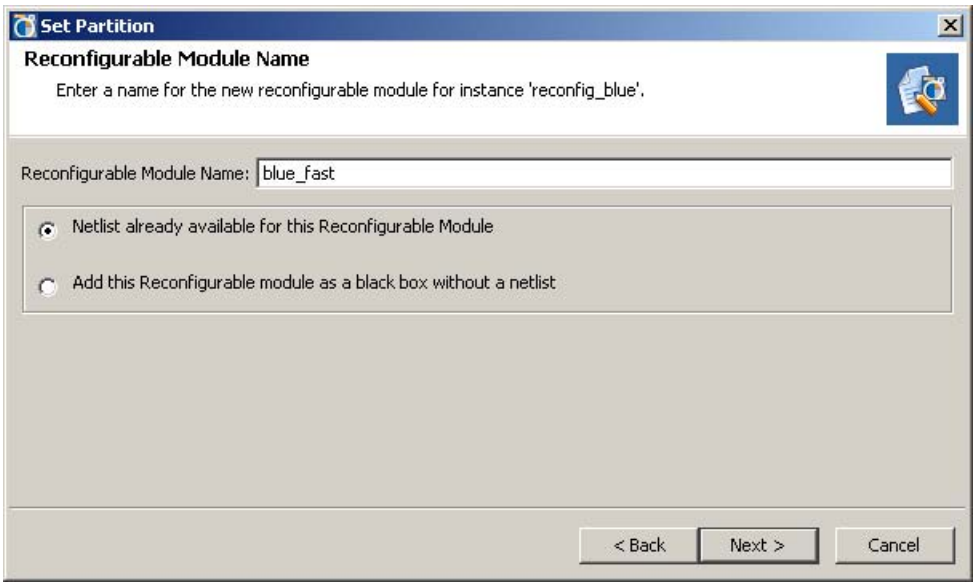

図 4-7 : リコンフィギャブル モジュールの命名

ネットリストが存在することを示す最初のオプションをオンにした場合、このモジュールのネットリストを指定するダイアログ ボックスが表示されます。1 つのリコンフィギャブル パーティションのすべての種類に同じネットリスト名を付ける必要があるので、ディレクトリ構造を使用してインスタンスを区別します。

3. [Set Partition] ダイアログ ボックスで NGC ファイルへのパスを指定します (図 4-8)。

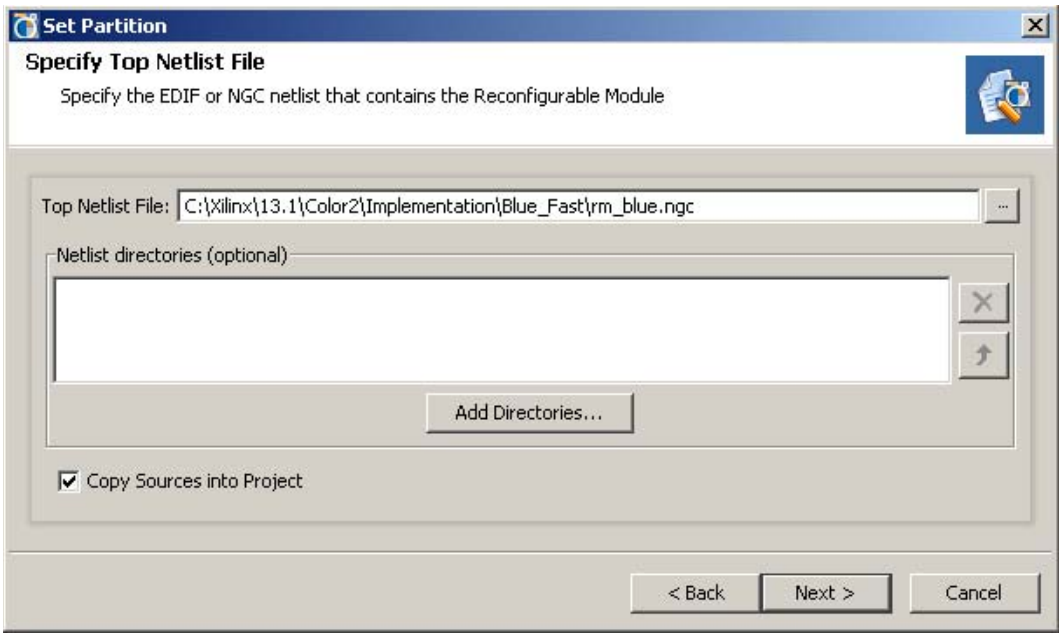

図 4-8 : リコンフィギャブル モジュール 1 つが追加されたリコンフィギャブル パーティション

ほかのディレクトリの別のネットリストを追加する場合は、検索パスをここで入力します。また、この特定のリコンフィギャブル モジュールの物理制約を含む制約ファイルは、次のダイアログ ボックスで指定できます。

リコンフィギャブル モジュールは、[Netlist] ビューのリコンフィギャブル パーティションの下に表示されます。

デザインには、複数のリコンフィギャブル パーティションを含むことができます。この場合、デザインのリコンフィギャブル パーティションそれぞれに対して [Set Partition] コマンドを実行する必要があります。このデザイン例では、fast バージョンのモジュールが各リコンフィギャブル パーティションに対して読み込まれます。`red`、`green`、`blue`

プロジェクトへのリコンフィギャブル モジュールの追加

図 4-9 に示すように、[Add Reconfigurable Module] コマンドを使用すると、リコンフィギャブル パーティションそれぞれに対してリコンフィギャブル モジュールをさらに追加できます。

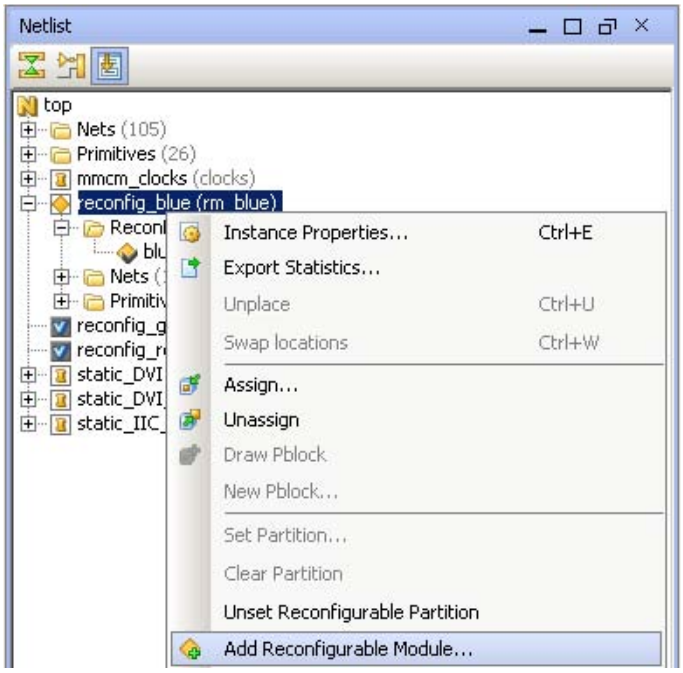

図 4-9 : リコンフィギャブル パーティションへのリコンフィギャブル モジュールの追加

このコマンドを使用してすべてのリコンフィギャブル モジュールをすべてのリコンフィギャブル パーティションに追加します。このデザイン例では、red、green、blue の slow バージョンが追加されています。

ブラック ボックス モジュールの追加

ブラック ボックス モジュールを定義することもできます。

1. この場合、同じ [Add Reconfigurable Module] コマンドを使用しますが、[black box] オプションを選択します。このモジュールにはネットリストが関連付けられていません (図 4-10)。

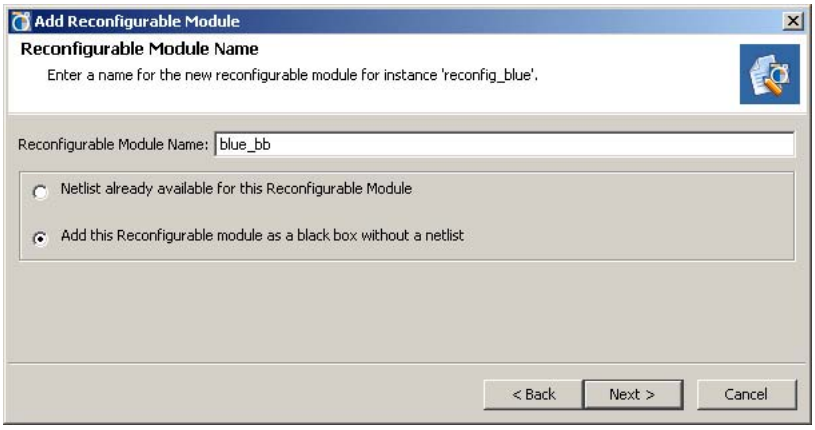

図 4-10 : リコンフィギャブル モジュールとしてのブラック ボックスの追加

リコンフィギャブル モジュールは、[Netlist] ビューのリコンフィギャブル モジュールの下の [Reconfigurable Modules] フォルダに追加されます。チェック マークは、リコンフィギャブル パーティションに対してアクティブなリコンフィギャブル モジュールを示します。

図 4-11 は、blue_fast がリコンフィギャブル パーティション reconfig_blue のアクティブなリコンフィギャブル モジュールであることを示しています。また、reconfig_blue は白い正方形の中に黄色のひし形が表示されたアイコンになっていますが、これはそのモジュールがリコンフィギャブル パーティションであることを示しています。グレーの正方形の中に黄色のひし形が表示されたアイコンになっている場合は、そのモジュールがブラック ボックスに定義されたリコンフィギャブル モジュールであることを示しています。

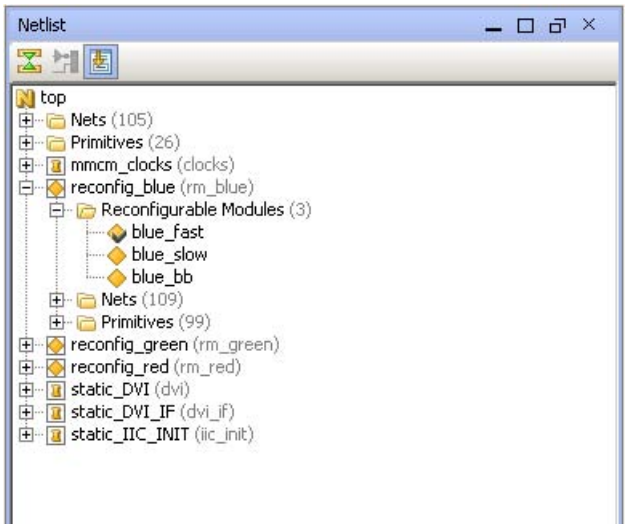

図 4-11 : すべてのリコンフィギャブル モジュールが追加されたリコンフィギャブル パーティション

2. ポップアップ メニューから [Set as Active Reconfigurable Module] コマンドを使用すると、リコンフィギャブル パーティションのアクティブ モジュールをいつでも変更できます。

   このコマンドを実行すると、選択したモジュールのネットリストが開いているワークスペースに読み込まれます (図 4-12)。

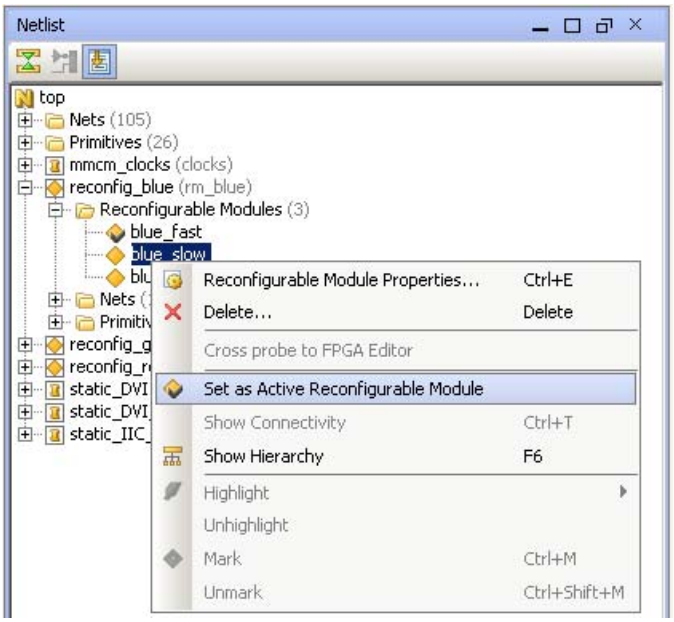

図 4-12 : アクティブ リコンフィギャブル モジュールの変更

デザイン ソースの管理

ソース ファイルを変更した場合は、新しいネットリストまたは制約を PlanAhead に読み込む必要があります。これらのファイルは、[Netlist Design] ビューの [Sources] タブですべて管理されます (図 4-13)。

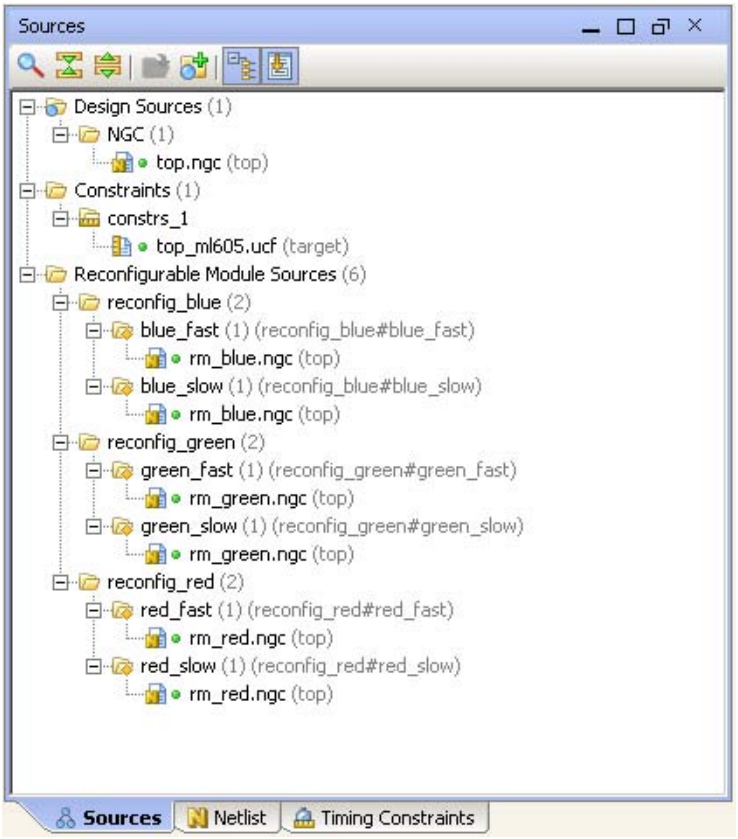

図 4-13 : [Sources] タブ

アップデートするネットリストを右クリックし、[Update File] を選択して、ネットリストを新しくします。PlanAhead では、この新規ネットリストをメモリに読み込むために、ソースを読み込み直すかどうか尋ねるメッセージが表示されます。

このプロセスでは、スタティック ロジックとリコンフィギャブル ロジック間のインターフェイスの変更はないと仮定されています。ポート リストが変更された場合は、新しいネットリストで新規プロジェクトを作成することをお勧めします。

パーシャル リコンフィギュレーション領域の定義

すべてのリコンフィギャブル パーティションに含まれるすべてのリコンフィギャブル モジュールのバージョンを PlanAhead ソフトウェアで定義したら、次はデザインの物理レイアウトを定義します。PlanAhead のメイン ツールバーで [Floorplanning] モードを選択して [Physical Constraints] タブと FPGA のフロアプラン ビューを開きます (図 4-14)。

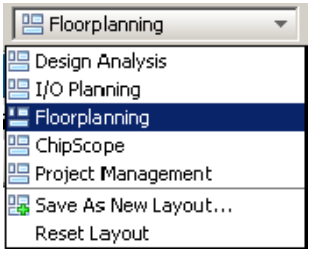

図 4-14 : PlanAhead ツールバーの [Floorplanning] モード

デバイスのリコンフィギャブル領域を定義するには、Pblock 長方形を作成する必要があります。長方形エリアを [Device] ビューで描画するには、[Set Pblock Size] コマンド ( ) を使用します。

メモ：デバイスに Pblock を自動的に配置する場合は、[Tools] → [Floorplanning] → [Place Pblocks] コマンドを使用しないでください。このコマンドを使用すると、インプリメンテーションに不向きな配置になってしまいます。

1. 65 ページの図 4-15 に示すように、[Physical Constraints] ビューで定義する Pblock を選択して、このコマンドを実行できるようにします。

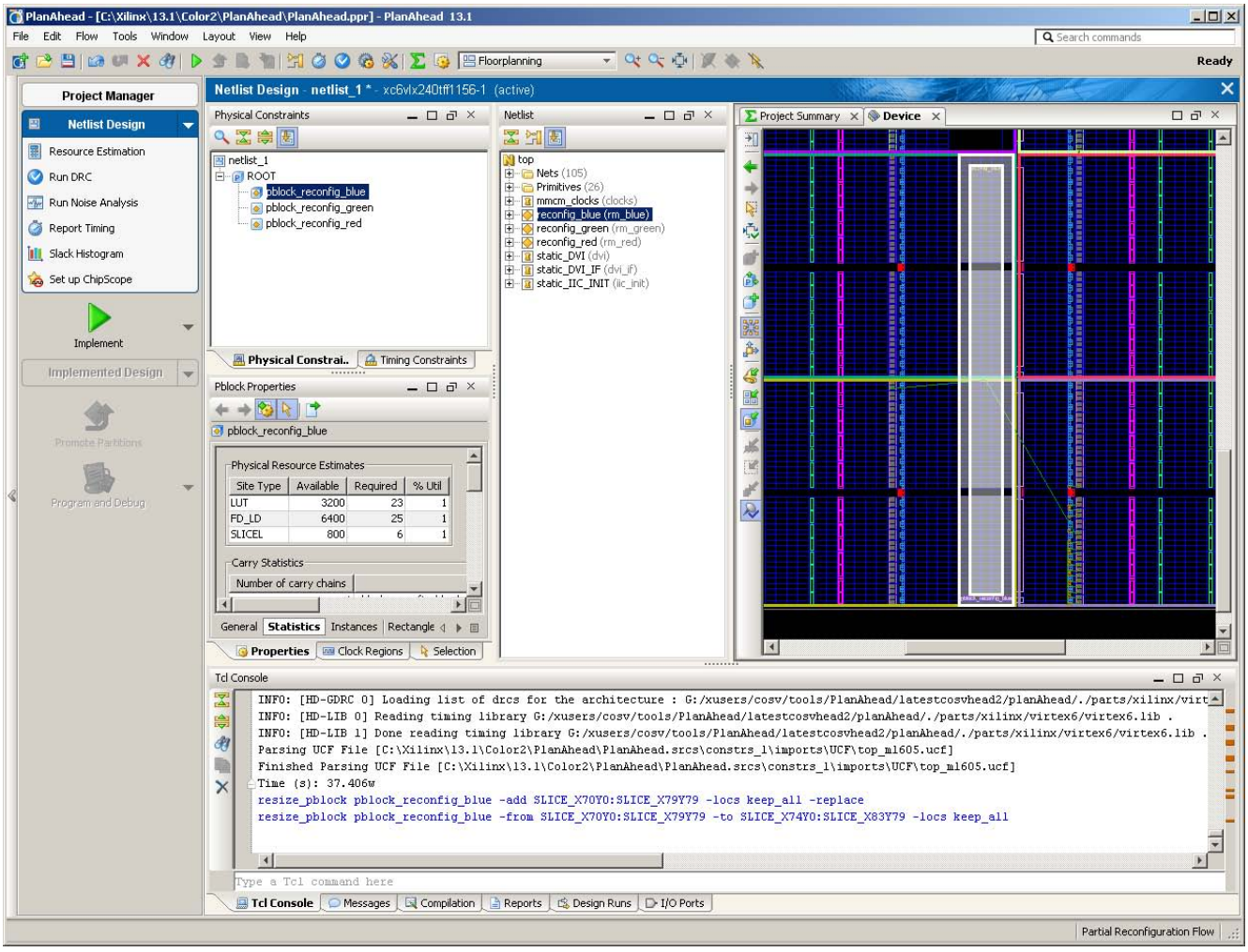

図 4-15 : リコンフィギャブル パーティションの Pblock の描画

[Device] ビューのクロック領域バウンダリは、リコンフィギャブル領域の形を描画する際のガイドとして使用できます。リコンフィギャブル領域のフロアプランの詳細は、第 3 章の「制約」および第 7 章の「リコンフィギャブル パーティション バウンダリの定義」を参照してください。Pblock が定義されると、PlanAhead ソフトウェアでその領域に制約を付けるリソースを選択するダイアログ ボックス (66 ページの図 4-16) が表示されます。

メモ : PlanAhead では、リコンフィギャブル パーティション内でエリア グループ サブモジュールは使用できません。

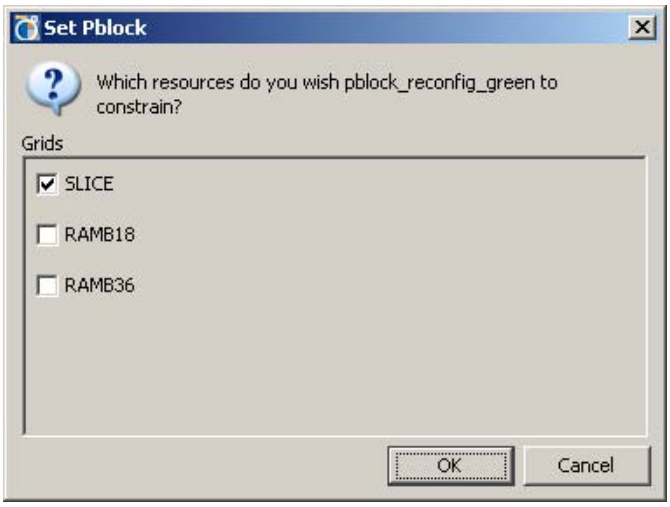

図 4-16 : Pblock を使用した範囲の定義

この選択をすると、リコンフィギャブル パーティションに対して一連の AREA_GROUP RANGE 制約が作成されます。

2. リコンフィギャブル モジュールに含まれないエレメントのチェック ボックスをオフにします。

   パーシャル ビット ファイルはここで選択した制約に基づいて作成されるので、余分なエレメントを含めると、ビット ファイルが不必要に大きくなります。

   [Pblock Properties] ビューの [General] タブ (図 4-17) には、含めることのできるほかのリソースが表示され、デザインに基づいてオンまたはオフにできます。

3. 対応するリコンフィギャブル モジュールすべてに含まれるロジック タイプそれぞれに対して、RANCE を定義します。

各リコンフィギャブル領域には、そこに配置するモジュール内に含まれるロジック タイプの RANGE を指定する必要があります。

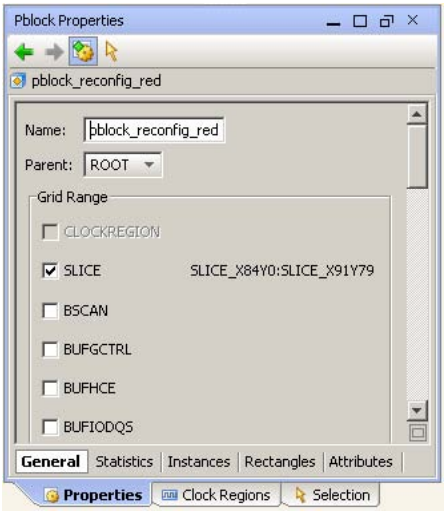

図 4-17 : リコンフィギャブル パーティションの RANGE 制約に適用可能なターゲット

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

パーシャル リコンフィギュレーション デザインの違反は、ザイリンクス開発のデザイン ルール チェック (DRC) 機能を使用すると検出できます。

1. [Tools] → [Run DRC] をクリックして表示されるダイアログ ボックスから、カテゴリ別になっている DRC をオンまたはオフにできます。
2. これらのチェックを定期的に実行し、デザインがパーシャル リコンフィギュレーションの原則に違反していないかどうかを確認してください。

図 4-18 は、パーシャル リコンフィギュレーションの DRC のリストを示しています。

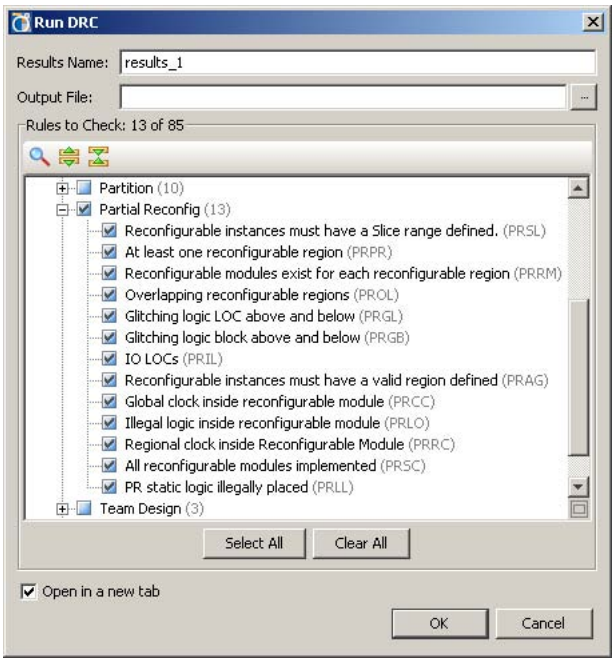

図 4-18 : パーシャル リコンフィギュレーションの [Run DRC] ダイアログ ボックス

[DRC Results] ビューにすべての警告およびエラーが表示されます。違反をクリックすると、図 4-19 に示す [Violation Properties] ビューにその詳細が表示されます。

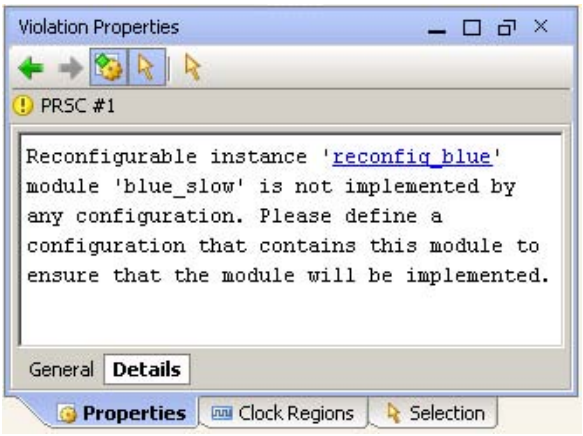

図 4-19 : DRC チェックの結果

一部のパーシャル リコンフィギュレーション DRC に違反するオブジェクトは、[Violation Properties] ビューでそのリンクをクリックすると、どこにあるのかわかります。

コンフィギュレーションの作成

すべてのモジュールおよび Pblock 範囲が定義されたら、コンフィギュレーションを定義およびインプリメントできます。

最初のコンフィギュレーションは、自動的に生成されます。PlanAhead の一番下の [Design Runs] タブをクリックし、config_1 をクリックします。[Implementation Run Properties] ダイアログ ボックスの下の [Partitions] タブに、このコンフィギュレーション用に選択されたリコンフィギャブル モジュールが表示されます (図 4-20 参照)。各リコンフィギャブル パーティションの最初のリコンフィギャブル モジュールが選択されますが、必要であれば変更できます。[General] タブのコンフィギュレーション名も変更できます。このデザインの場合、名前が config_1 から FFF に変更されています。

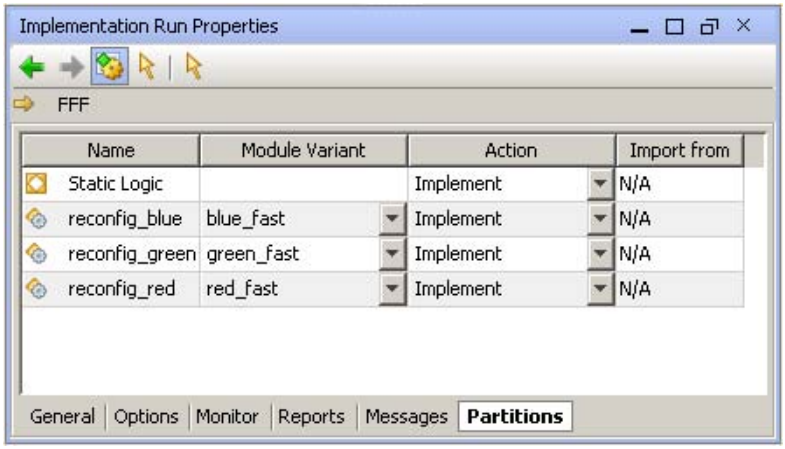

図 4-20 : コンフィギュレーションのリコンフィギャブル モジュールの定義

インプリメンテーションの run プロパティは、[Options] タブで選択するか、Flow Manager の [Implementation Settings] で選択すると変更できます。図 4-21 を参照してください。

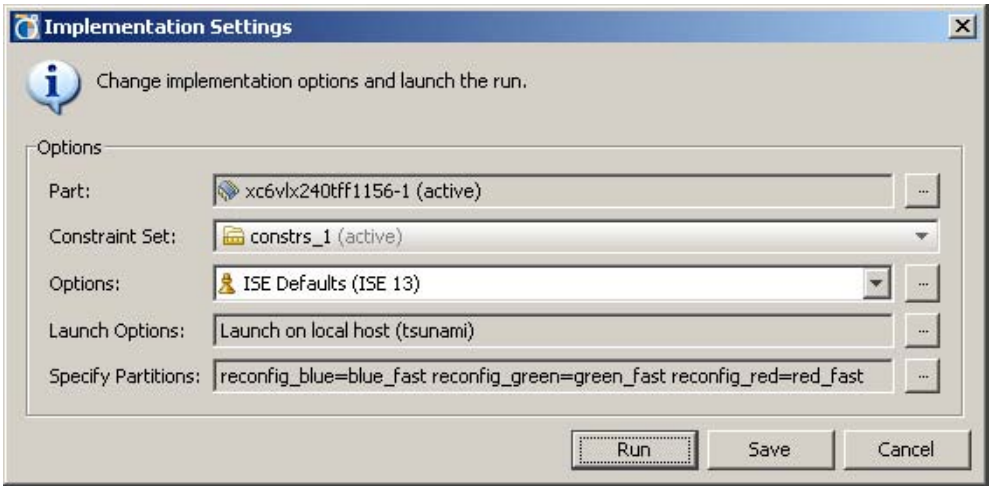

図 4-21 : インプリメンテーション プロパティの設定

[Configurations] ビュー ([Window] → [Configurations]) には、69 ページの図 4-22 のように、コンフィギュレーションとそれに含まれるリコンフィギャブル モジュール、ステータスなどが表示されます。

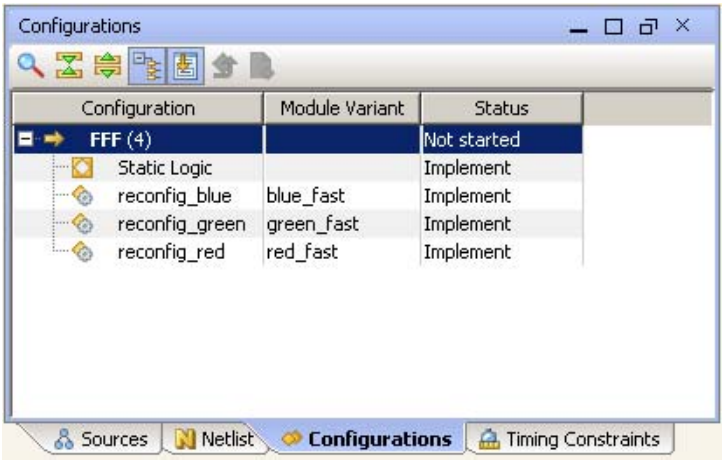

図 4-22 : レポートされる各コンフィギュレーションの詳細

Flow Manager の [Implement] の下の [Create New Implementation Runs] オプションをクリックするか、[Design Runs] ビューの [Create New Runs] ボタンをクリックすると、複数のコンフィギュレーションを作成できます (図 4-23 および図 4-24 を参照)。

図 4-23 : [Create New Implementation Runs Option] オプション

図 4-24 : [Create New Runs] ボタン

リコンフィギャブル モジュールとブラック ボックスをどのように組み合わせてもコンフィギュレーションが作成できます。コンフィギュレーションは、パーシャル リコンフィギュレーション デザインのどの段階ででも作成できます。[Partition Action] ボタンを使用し、各コンフィギュレーションに必要なリコンフィギャブル モジュールを選択します。

メモ：この時点でこれらの run は起動しないでください。

70 ページの図 4-25 は、[Create New Runs] ダイアログ ボックスを示しています。

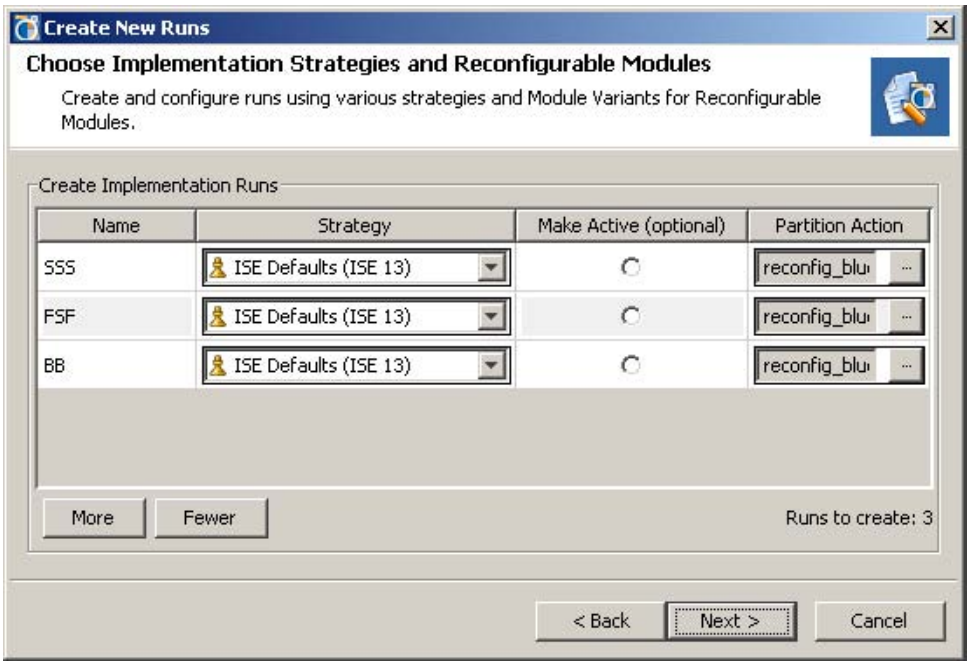

図 4-25 : 複数 run の作成

このデザイン例では、図 4-26 に示すように 4 つの独自のコンフィギュレーションが作成されます。

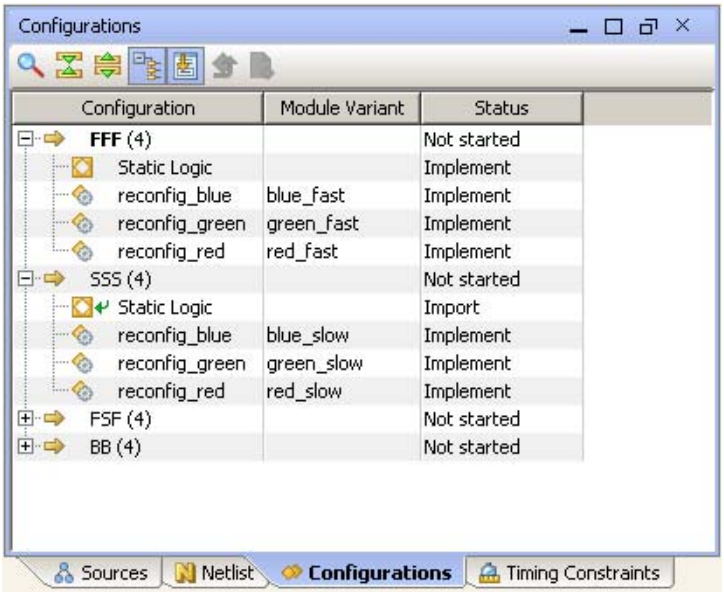

図 4-26 : 最初のコンフィギュレーション

コンフィギュレーションの制御

さまざまなコンフィギュレーションを確認するには、タイミング解析や回路図を使用したデザイン検索などの従来の PlanAhead ソフトウェアの解析機能を使用できます。

1. 図 4-27 のように [Configurations] ビューのポップアップ メニューから [Load Configuration] をクリックすると、解析用のネットリストが読み込まれます。

   これにより、[Netlist] ビューでそのコンフィギュレーションのリコンフィギャブル モジュールがアクティブになります。

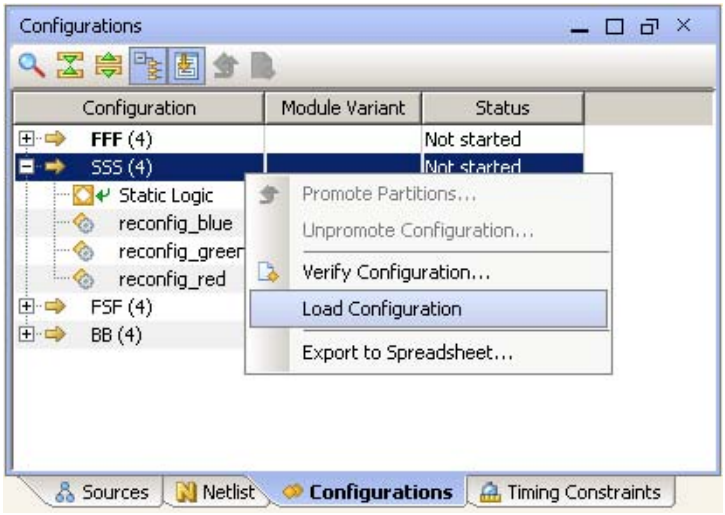

図 4-27 : 既存コンフィギュレーションの読み込み

インプリメンテーションおよび制約を設定し終えると、コンフィギュレーションをインプリメントできます。

2. [Design Runs] タブでコンフィギュレーションを右クリックし、[Launch Runs] をクリックします。

   Flow Manager で [Implement] ボタンをクリックしても、アクティブ デザインを起動することができます。図 4-28 は、コンフィギュレーションが実行されているところを示しています。

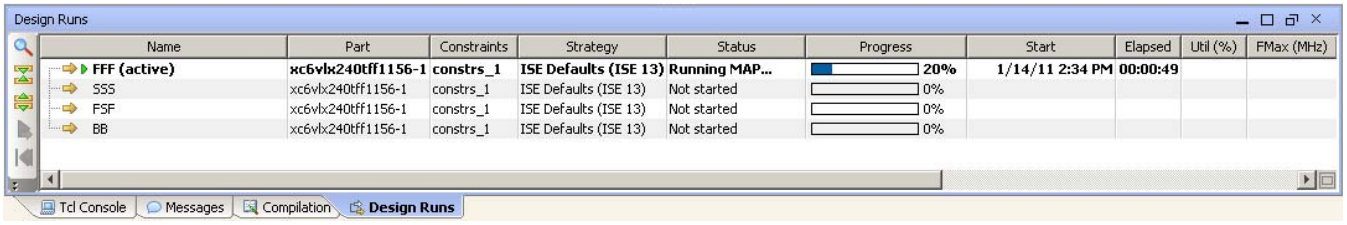

図 4-28 : コンフィギュレーションのインプリメント

3. コンフィギュレーションが問題なくインプリメントされると、その結果を今後のインプリメンテーションとコンフィギュレーションにインポートできるようにプロモートできます。72 ページの図 4-29 に示すダイアログ ボックスの [Promote Partitions] を使用して、コンフィギュレーションをプロモートします。

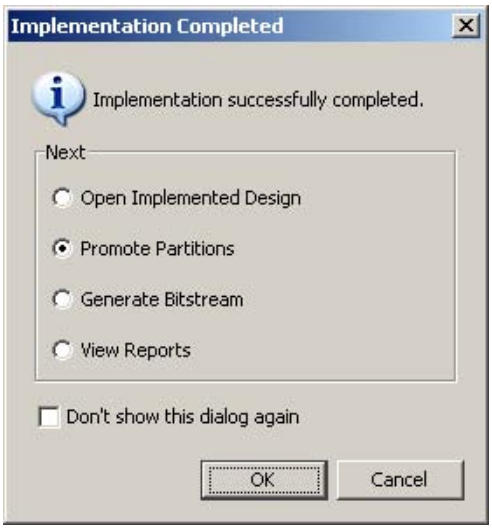

図 4-29 : [Implementation Completed] ダイアログ ボックス

図 4-30 に示す [Configurations] ビューのポップアップ メニューから [Promote Partitions] をクリックしてもコンフィギュレーションをプロモートできます。

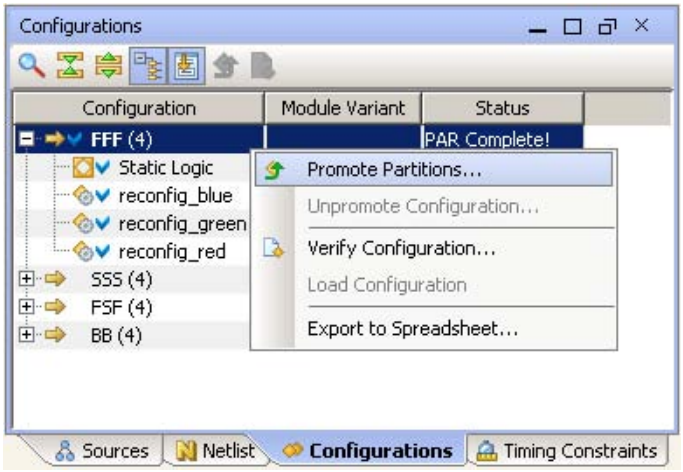

図 4-30 : コンフィギュレーションのプロモート

コンフィギュレーション同士は互いに依存しています。

- スタティック ロジックと各リコンフィギャブル モジュールは、それを使用する各コンフィギュレーションで同じである必要があります。
- すべてのコンフィギュレーションで同じスタティック ロジック インプリメンテーションを使用する必要があります。コンフィギュレーションの中には、同じリコンフィギャブル モジュールを共有することが可能なものもあります。
- コンフィギュレーションがプロモートされると、そのインプリメンテーションはコンフィギュレーションに含まれるすべてのモジュールで最適な結果として設定されます。
- コンフィギュレーションをプロモートまたはリセットすると、ほかのコンフィギュレーションも影響を受けることがあります。この場合、PlanAhead ソフトウェアで警告が表示されます (73 ページの図 4-31)。

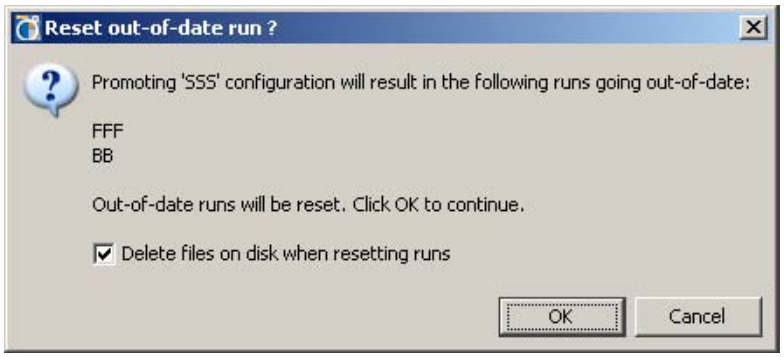

図 4-31 : 古いコンフィギュレーションのリセット

run がプロモートされたら、ほかのコンフィギュレーションのリコンフィギャブル モジュールのステータスがアップデートされます。

図 4-32 および 74 ページの図 4-33 は、コンフィギュレーション FFF がプロモートされたために、コンフィギュレーション SSS のステータス ロジックが Import に設定されたところを示しています。

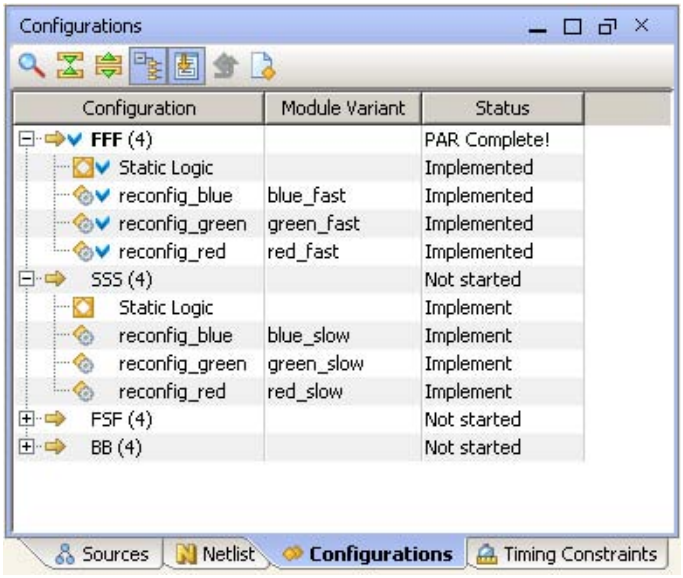

図 4-32 : コンフィギュレーション FFF のプロモート前

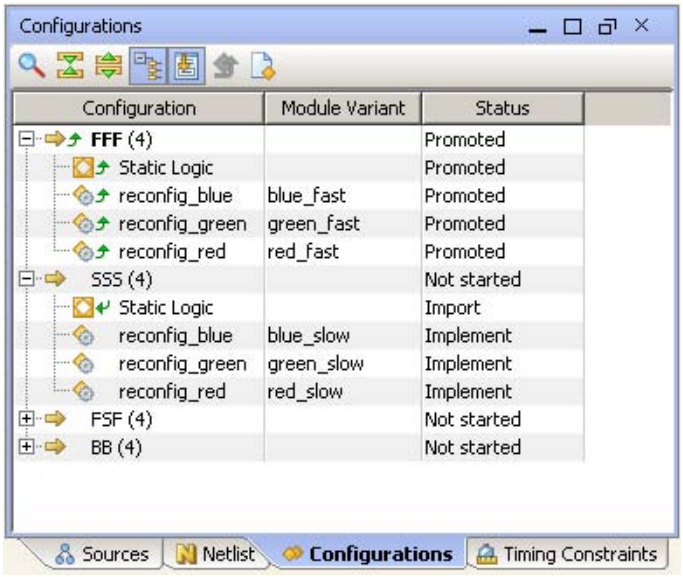

図 4-33 : コンフィギュレーション FFF のプロモート後

複数のコンフィギュレーションを一度にプロモートできます。モジュールはプロモートされた順にコンフィギュレーションからインポートされます。

図 4-34 のコンフィギュレーション FSF には、Static、reconfig_green、および reconfig_red が FFF からインポートされ、リコンフィギャブル モジュール reconfig_blue はコンフィギュレーション FFF にインプリメントされず、SSS からインポートされます。

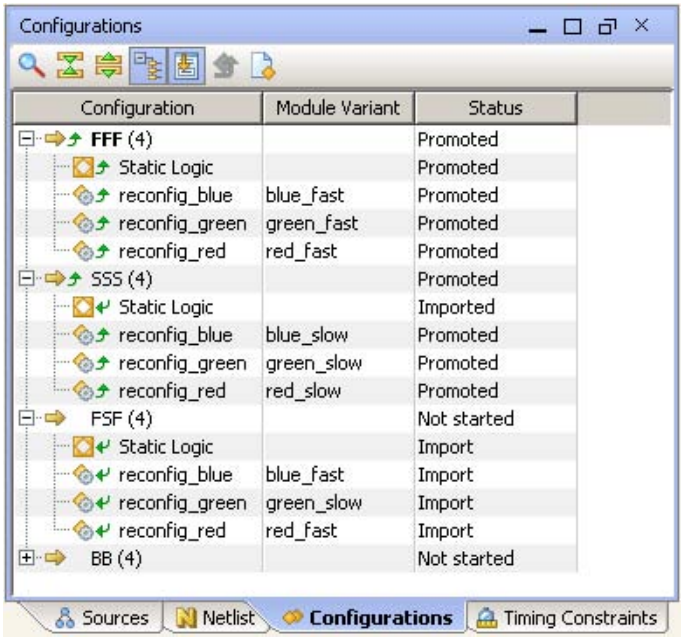

図 4-34 : プロモートされた複数コンフィギュレーション

スタティック ロジックおよびすべてのリコンフィギャブル モジュールがほかのコンフィギュレーションからインポートされる場合、コンフィギュレーションはプロモートできません。この例では、

FSF コンフィギュレーションが FFF と SSS の一部から構築されているので、FSF をプロモートする必要はありません。

リコンフィギャブル モジュールはインプリメントまたはインポートできるので、個別のリコンフィギャブル モジュールで試すことができます。このように柔軟性があるため、プロモートするのに最適なコンフィギュレーションが見つけやすくなっています。これは、図 4-35 の [Implementation Run Properties] からも実行できます。

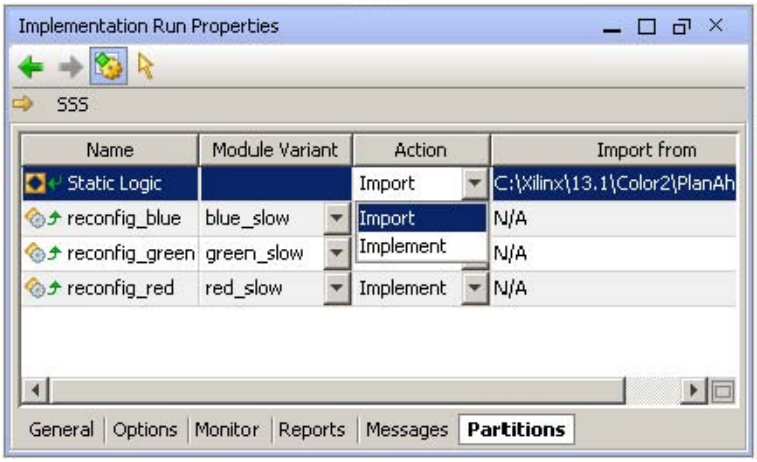

図 4-35 : [Action] の選択 ([Implement] または [Import])

次は、スタティックおよびリコンフィギャブル ロジックに対する74 ページの図 4-34 の [Status] フィールドのサマリを示しています。

- [Implement] (コンフィギュレーションの場合は [Not Started])

  モジュールは定義されたのに、インプリメントされていない状態です。インプリメンテーションが実行されると、そのモジュールに指定されたネットリスト、オプション、制約を使用して配置配線が最初から実行されます。

- [Import]

  モジュールは定義され、結果が別のコンフィギュレーションからコピーされます。インプリメンテーションが実行されると、配置配線でこのモジュールのプロモートされたディレクトリから結果がコピーされ、同じ結果が保持されます。

- [Implemented] (コンフィギュレーションの場合は [PAR Complete!])

  選択したコンフィギュレーションでモジュールの配置配線が問題なく終了したことを示しています。

- [Imported]

  モジュールがプロモートされた run から問題なくコピーされ、貼り付けられています。

- [Promoted]

  モジュールが完全なステータスになり、[Import] になっているほかのコンフィギュレーションで重複するモジュールがこのマスタ結果からインポートされます。

これらのインプリメンテーション run の結果は、次の PlanAhead プロジェクト ディレクトリにあります。`<project_name>.runs\<configuration_name>`

プロモートされた run は PlanAhead プロジェクト ディレクトリの別のフォルダにあります。`<project_name>.promote\<configuration_name>`

このデザイン例の場合、XFFF、XSSS、および XBB ディレクトリが FFF、SSS、FSF、BB に対して作成できます。使用されたモジュールはすべてほかのコンフィギュレーションからインプリメントされているので、FSF のプロモートは必要ありません。

コンフィギュレーションの検証

pr_verify は、インプリメント済みコンフィギュレーションのどの組み合わせでも呼び出す必要があるツールで、デザインのコンフィギュレーションのインプリメンテーションを確認します。

1. pr_verify は、[Configurations] ビューのポップアップ メニューから実行します (図 4-36)。これは、すべてのデザイン ルールが守られたかどうかを確認するパーシャル リコンフィギュレーション デザインの重要な手順です。

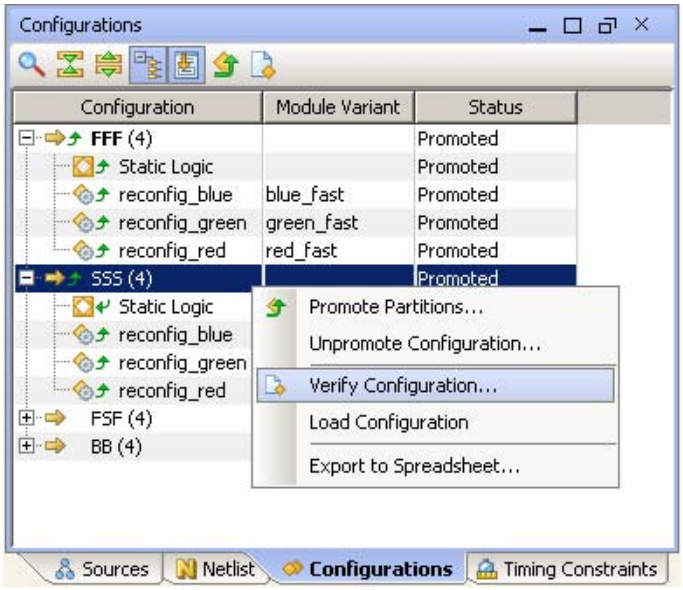

図 4-36：コンフィギュレーションの検証

2. 複数のコンフィギュレーションに対してダイアログ ボックスが表示されるので、出力ファイルを定義します (図 4-37)。

   すべてのコンフィギュレーションを検証して、ハードウェアで問題がないようにします。

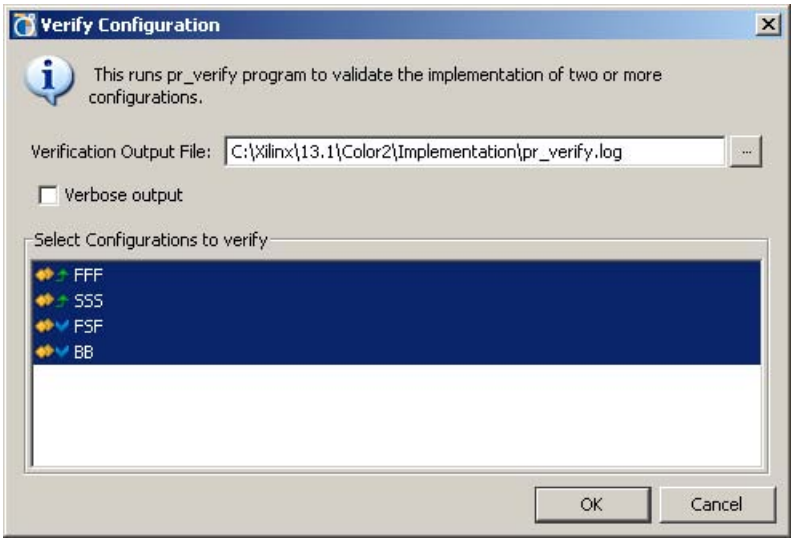

図 4-37 : 検証するコンフィギュレーションの選択

ワークスペースには、ログ ファイルも表示されます。pr_verify 中にエラーが検出されなければ、ビット ファイルを作成します。

ビット ファイルの生成

コンフィギュレーションが問題なくインプリメントされ、pr_verify ですべてのコンフィギュレーションが検証できたら、ビット ファイルを生成できます。

[Design Runs] ビューのポップアップ メニューから [Generate Bitstream] をクリックします (図 4-38)。

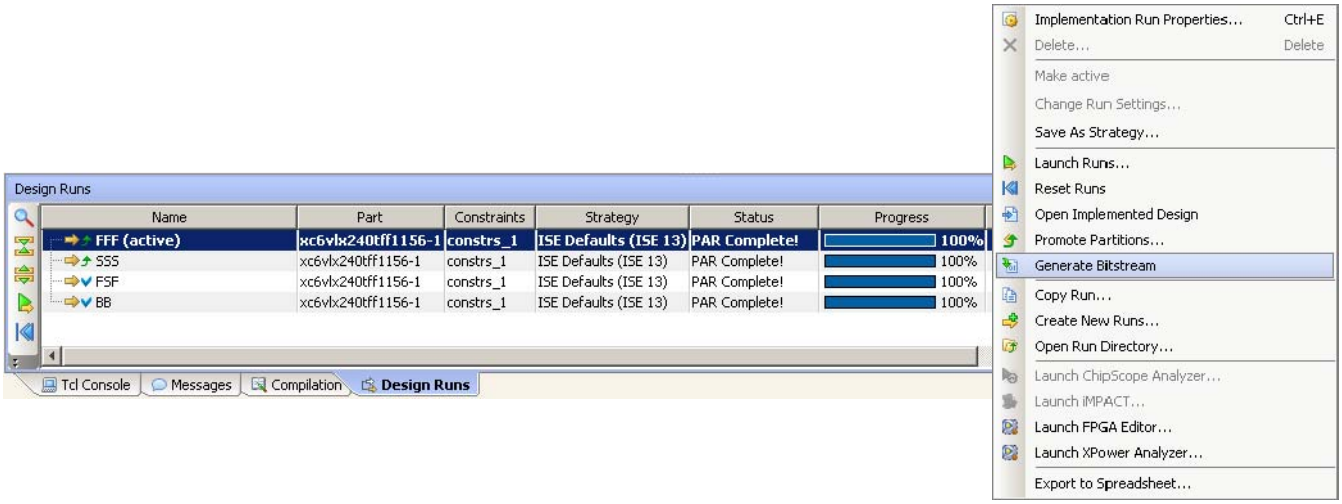

図 4-38 : ビット ファイルの生成

これにより、コンフィギュレーションのフル ビット ファイルと選択したコンフィギュレーションの各リコンフィギャブル モジュールのパーシャル ビット ファイルが生成されます。

メモ：ブロック RAM の内容をアップデートするのに Data2MEM プログラムを実行する必要がある場合は (EDK プロセッサ システムの場合など)、BitGen コマンドが -bd オプションで実行されるようにすると、ビットストリーム生成の一部として Data2MEM が実行されます。詳細については、第 7 章の「EDK との連動」を参照してください。

メモ：暗号化されたパーシャル ビット ファイル (bitgen -g encrypt で生成) は、Virtex-6 デバイスでサポートされます。この場合、各コンフィギュレーションに対して同じ NKY ファイルを指定して、暗号キーの値が同じになるようにする必要があります。暗号化されたパーシャル ビット ファイルは、Virtex-4 および Virtex-5 デバイスではサポートされません。

このデザイン例では、FFF コンフィギュレーションに対して次のビット ファイルが生成されます。

- `fff.bit`
- `fff_reconfig_blue_blue_fast_partial.bit`
- `fff_reconfig_red_red_fast_partial.bit`
- `fff_reconfig_green_green_fast_partial.bit`

複数のコンフィギュレーションを選択して、1 度にプロジェクト全体のフル ビット ファイルとパーシャル ビット ファイルすべてを作成することもできます。

フル ビット ファイルとパーシャル ビット ファイルはコンフィギュレーション別のディレクトリにそれぞれ出力されます。詳細は、「コンフィギュレーションの制御」を参照してください。

PlanAhead プロジェクトのディレクトリ構造

PlanAhead では、図 4-39 に示すように、すべてのファイル、コンフィギュレーション、インプリメンテーションなどのデザイン データが単純で構造的な方法で分類され保存されます。

図 4-39 : PlanAhead パーシャル リコンフィギュレーションのディレクトリ構造

この構造は、PlanAhead ソフトウェアの /project ディレクトリと類似しています。プロジェクトのネットリストと制約は、`<project>.srcs` ディレクトリにインポートされます。このディレクトリは、GUI と同じ構造で分類され、スタティック ロジックは sources_1 ディレクトリの下に、すべての RM ソースは最適な名前の付いたディレクトリの下に保存されます。ビット ファイルも含めたインプリメンテーション run は、該当するフロアプランおよびコンフィギュレーションの下の /PlanAhead.runs ディレクトリに保存されます。プロモートされたコンフィギュレーションは /PlanAhead.promote ディレクトリに保存され、X という文字がその前に追加されます。

XILINX®

第 5 章

コマンド ライン スクリプト

この章では、GUI を使用せずにツールセットからフローを自動化する方法についての手順と推奨情報を示します。

ザイリンクスでは、パーティション ベースのパーシャル リコンフィギュレーション デザインを定義してインプリメントする Tcl スクリプト例を提供しています。これらのスクリプトは、一般的なフローに使用でき、カスタム フローに合わせて修正できるテンプレートになっています。

これらのスクリプトを実行するには、Tcl シェルが使用できるようになっている必要があります。ほとんどの Linux で、Tcl シェルはデフォルトで /usr/bin ディレクトリにインストールされています。Tcl シェルがインストールされていない場合は、http://www.activestate.com/activetcl から無料でダウンロードできます。このマニュアルのスクリプトは、Tcl 8.4 を使用してテストされています。

Tcl スクリプト

- xpartition.tcl

  パーティション ベースのパーシャル リコンフィギュレーション デザインを定義しインプリメントします。3 つのほかの Tcl スクリプトを呼び出して、それらの関数を実行します。このスクリプトは完全なフローを実行するのに使用してください。

  - `gen_xp.tcl`

    各プロジェクトに必要なパーティション ファイルを作成および修正します。xpartition.tcl スクリプトから呼び出されます。

  - `implement.tcl`

    パーティション ベースのパーシャル リコンフィギュレーション コンフィギュレーションをインプリメントします。xpartition.tcl スクリプトから呼び出されます。

  - `export.tcl`

    必要なファイルをエクスポートし、パーティションを今後の run にインポートします。xpartition.tcl スクリプトから呼び出されます。

xpartition.tcl ファイルは data.tcl ファイルを引数として使用します。data.tcl ファイルには、パーティション定義、コンフィギュレーション、インプリメンテーションのオプションが含まれます。このファイルを使用すると、Tcl スクリプトを変更しなくてもデザインとそのオプションを変更できます。

次は、この Tcl スクリプトを呼び出すコマンド ラインの例です。これは、第 3 章「ソフトウェア ツール フロー」に説明されるようにパーシャル リコンフィギュレーション プロジェクトのルート フォルダから起動されます。

```
xtclsh .\Tools\xpartition.tcl .\Tools\data.tcl
```

data.tcl のフォーマット

data.tcl ファイルは、3 つの主なセクションに分割されます。data.tcl では、リストまたはアレイ宣言の外側でコメントを記述するのに # 記号が使用されます。リストおよびアレイのメンバーは、削除するか、リストやアレイの外側にコメントとして記述して、無視されるようにしてください。

Color2 サンプル デザインには、複数バージョンのデータ ファイルが含まれます。これらは、前述の xtclsh コマンドと同じように使用できます。これらのデータ ファイルは、リファレンスとして含まれ、必要に応じて修正することができます。

- **data.tcl** - 合成およびインプリメンテーションを実行します。スクリプト フローを最初から記述する場合に使用します。
- **data_synth.tcl** - 合成のみを実行します。コマンド ラインから合成を実行し、インプリメンテーションは PlanAhead ソフトウェアを使用する場合に便利です。
- **data_impl.tcl** - インプリメンテーションのみを実行します。合成が既に実行されていても、タイミング制約や物理制約の調整など、インプリメンテーションに少し変更が必要な場合などに便利です。

セクション 1 : プロジェクト オプションの設定

セクション 1 : このセクションでは、環境変数、パーツ、制約ファイル、パーティションおよびリコンフィギャブル モジュールなどを含む変数を設定できます。

```
# 1:environment variables for all configurations
set ::env(XIL_TIMING_ALLOW_IMPOSSIBLE) 1

# 2:part definition
set PART xc5vlx50t-3-ff1136

# 3:constraints file
set UCF ../../Source/UCF/top_ml505.ucf

# 4:Partition names
# These names must match the actual instance names in the design
set TOP_PART    /top
set RED_PART    ${TOP_PART}/reconfig_red
set GREEN_PART  ${TOP_PART}/reconfig_green
set BLUE_PART   ${TOP_PART}/reconfig_blue

# 5:RM names
set RED_FAST   Red_Fast
set RED_SLOW   Red_Slow
set RED_BB     Red_Blank
set GREEN_FAST Green_Fast
set GREEN_SLOW Green_Slow
set GREEN_BB   Green_Blank
set BLUE_FAST  Blue_Fast
set BLUE_SLOW  Blue_Slow
set BLUE_BB    Blue_Blank
set STATIC     Static
```

```
# 1:environment variables for all configurations
```

次のフォーマットを使用し、インプリメンテーションに必要な環境変数を定義します。これらの変数はすべてのコンフィギュレーションで使用されます。

```
set ::env(VARIABLE) value
```

2:part definition

インプリメンテーションでターゲットとされるパーツを定義します。

3:constraints file

制約ファイルを指定します。これは、すべてのコンフィギュレーションで使用されます。

4:Partition names

これらの名前は、デザインの実際のインスタンス名と一致する必要があります。デザインのすべてのパーティションは、リコンフィギャブルかどうかに関係なく、ここで定義する必要があります。これらの名前は、HDL のインスタンス名と一致する必要があります。

5:RM names

すべてのリコンフィギャブル モジュールを宣言します。これらは、ボトムアップ合成を実行したり、コンフィギュレーションを定義するのに使用されます。スタティックは宣言する必要はありません。

セクション 2 : 合成のモジュール指定とパーティション属性の定義

セクション 2 : このセクションでは、どのモジュールを合成するか定義し、パーティションがリコンフィギャブルかどうかを宣言します。

```
# 6:RM list
# Each RM in the list is synthesized with bottom-up synthesis.
# You must create a directory for each of the RMs in the list
set RMs [list $RED_FAST $RED_SLOW $GREEN_FAST $GREEN_SLOW $BLUE_FAST $BLUE_SLOW $STATIC]

# 7:Partition Attributes List
################################################################################
# Create the per-partition attributes list.This list must be called
# "PartitionAttrsList".The format is:
#  set PartitionAttrsList <partitionlist>
# where
#  <partitionlist> ::= { <partitionattrs> ...}
#  <partitionattrs> ::= { <partitionName> <attrslist> }
#  <attrslist>   ::= <namevalpair> ...
#  <namevalpair>  ::= { <attrName> <attrValue> }
################################################################################

set PartitionAttrsList {
 {/top {Reconfigurable false}}
 {/top/reconfig_red {Reconfigurable true}}
 {/top/reconfig_green {Reconfigurable true}}
 {/top/reconfig_blue {Reconfigurable true}}
}
```

6:RM list

Each RM in the list is synthesized with bottom-up synthesis.

You must create a directory for each of the RMs in the list

ボトムアップ合成を実行する必要のあるリコンフィギャブル　モジュールを指定します。合成は、指定順に実行されます。必要なディレクトリ構造については、後のセクションで説明されます。

7:Partition Attributes List

パーティションがリコンフィギャブルかどうか指定できます。3 つのリコンフィギャブル パーティション n では Reconfigurable が true に設定されていますが、top には何も設定されていないので、デフォルトの false になります。

セクション 3 : コンフィギュレーションの定義

セクション 3 : このセクションでは、各コンフィギュレーションの詳細とそのインプリメント順序を定義します。

```
# 8:Configuration Information
##############################################################################
# Create the per-configuration variables.The format is:
#   set CONFIG1DATA <ConfigList>
#   set CONFIG2DATA <ConfigList>
#     ...
#   set ALL_CFGS [list $CONFIG1DATA $CONFIG2DATA ...]
# where
#   <ConfigList>        ::= { <ConfigNamePair> <Settings> }
#   <ConfigNamePair>    ::= { 'ConfigName' <Name> }
#   <Settings>          ::= { 'Settings' <SettingsList> }
#   <SettingsList>      ::= <PartSettingsList> ...
#   <PartSettingsList>  ::= <partitionName> <namevalpair> ...
##############################################################################

# Configuration FastConfig settings.
# Everything is implemented; there is no import location

set CONFIG_FastConfig {
  {ConfigName FastConfig}
  {Settings
   {/top{State implement}}
   {/top/reconfig_red   {State implement}{NetlistDir Red_Fast}{ModName Red_Fast}}
   {/top/reconfig_green {State implement}{NetlistDir Green_Fast}{ModName Green_Fast}}
    {/top/reconfig_blue {State implement}{NetlistDir Blue_Fast}{ModName Blue_Fast}}
   }
}

# Configuration SlowConfig settings.
# Static is imported from the FastConfig

set CONFIG_SlowConfig {
  {ConfigName SlowConfig}
  {Settings
   {/top{State import} {ImportLocation ../XFastConfig}}
   {/top/reconfig_red   {State implement}{NetlistDir Red_Slow}{ModName Red_Slow}}
   {/top/reconfig_green {State implement}{NetlistDir Green_Slow}{ModName Green_Slow} }
   {/top/reconfig_blue  {State implement}{NetlistDir Blue_Slow}{ModName Blue_Slow}}
  }
}
```

```
# Configuration FSFConfig settings.
# All 4 partitions are imported.

set CONFIG_FSFConfig {
  {ConfigName FSFConfig}
  {Settings
   {/top{State import} {ImportLocation ../XFastConfig} }
   {/top/reconfig_red  {State import}{ImportLocation ../XFastConfig}{NetlistDir Red_Fast}
{ModName Red_Fast}}
   {/top/reconfig_green {State import}{ImportLocation ../XFastConfig}{NetlistDir
Green_Fast} {ModName Green_Fast}}
   {/top/reconfig_blue  {State import}{ImportLocation ../XSlowConfig}{NetlistDir
Blue_Slow} {ModName Blue_Slow}}
  }
}

# Configuration BlankConfig settings.

set CONFIG_BlankConfig {
  {ConfigName BlankConfig}
  {Settings
   {/top{State import} {ImportLocation ../XFastConfig} }
   {/top/reconfig_red  {State implement}{NetlistDir Red_Blank}{ModName Red_Blank}}
  {/top/reconfig_green {State implement}{NetlistDir Green_Blank}{ModName Green_Blank}}
   {/top/reconfig_blue {State implement}{NetlistDir Blue_Blank}{ModName Blue_Blank}}
  }
}

# 9:List of configurations in order of implementation
# finally, build the list of all the configuration data.
# This list will drive the implementation of all configurations,
# in the order they are listed
set ALL_CFGS [list $CONFIG_FastConfig $CONFIG_SlowConfig $CONFIG_FSFConfig
$CONFIG_BlankConfig]
#set ALL_CFGS [list $CONFIG_BlankConfig]
```

8.コンフィギュレーション情報

このセクションは次を含め、各コンフィギュレーションを定義します。

- リコンフィギャブル モジュールに含まれるもの
- リコンフィギャブル モジュールをインポートするかインプリメントするか
- どこからインポートするか

フォーマットは次のとおりです。

```
set CONFIG_<config_name> {
 {ConfigName <config_name>}
 {Settings
{<partition_name>} {State <"implement"|"import">} > {ImportLocation
<directory to import from> } {NetlistDir <directory where RM netlist is
located>} {ModName <name of netlist file>}
 }
}
```

`ImportLocation` は、そのパーティションの State が import に設定されている場合にのみ必要です。`NetlistDir` は、合成が Tcl スクリプトの外で実行された場合にのみ `ModName` と異なります。

最初のコンフィギュレーションでは、インポートするプロモート済みのイメージがまだないので、すべてのパーティションがインプリメントされます。インプリメンテーションが終了すると、すべてのコンフィギュレーションが X<*config_name*> にエクスポートされます。これは、ほかのコンフィギュレーションのインポート ディレクトリにも使用できます。

9:All configurations with implementation order

```
# This list drives the implementation of all configurations,
# in the order they are listed
```

このリストの順序は重要です。パーティションは、最初のインプリメンテーションが終わるまでインポートできません。

セクション 4 : インプリメンテーション オプション

セクション 4 : このセクションでは、インプリメンテーション オプションを変更する変数を設定できます。

```
10:Implementation options
# set the optional implementation data flags.
# The format of the optional data is:
# RUN_RM_SYNTH=NO if the design has no modules to be synthesized bottom-up
# NGDBUILD_TOP=<top_path> is path to pre-existing top module for Ngdbuild
# NGDBUILD_SEARCH=<search_path ...> a string containing search path directories
# NGDBUILD_OPTS=<ngdbuild_command_line_options> optional cmd line options for Ngdbuild
# RUN_MAP=NO if you do not want to run Map
# MAP_OPTS=<map_command_line_options> optional command line options for Map
# RUN_PAR=NO if you do not want to run PAR
# PAR_OPTS=<par_command_line_options> optional command line options for Par
# RUN_BITGEN=NO if you do not want to generate bitstreams
array set IMPLEMENTATION_DATA { \
                RUN_RM_SYNTH NO \
        }
```

変数は、次の通りです。

- `SYNTH_TOOL xst/synplify_pro`

  ボトムアップ合成を実行する際にどの合成ツールを使用するか指定します。この場合、該当する合成プロジェクトが Synth ディレクトリに必要です。

- `RUN_RM_SYNTH YES/NO`

  リコンフィギャブル モジュール リストのすべてのモジュールでボトムアップ合成を実行するかどうか設定します。最初のインプリメンテーションでは Yes にする必要があり、HDL が変更されるまで NO に変更する必要があります。デフォルトは YES です。

- `NGDBUILD_TOP <path_to_top_level_netlist>`

  スタティック ロジックが既に合成されている場合は、リコンフィギャブル モジュールを使用して合成を実行するよりも、この変数を使用してパスを指定します。この変数は、`RUN_RM_SYNTH` が `NO` に設定される場合、またはスタティックがリコンフィギャブル モジュール リストに含まれない場合に設定する必要があります。

- `NGDBUILD_SEARCH <search_directories_for_NGDBUILD>`

  NGDBuild のマクロ検索パスがコア ネットリストのあるディレクトリを指定するように設定します。スペースで区切って { } で囲めば、複数のディレクトリを指定できます。UNIX タイ

プのスラッシュ (/) を、Tcl 命名規則に従い、Windows と Linux の両方で使用する必要があります。

- `RUN_NGDBUILD YES/NO`

  NGDBuild をすべてのインプリメンテーションで実行するかどうか制御します。

- `RUN_MAP YES/NO`

  MAP をすべてのインプリメンテーションで実行するかどうか制御します。

- `RUN_PAR YES/NO`

  PAR をすべてのインプリメンテーションで実行するかどうか制御します。

- `RUN_BITGEN YES/NO`

  BitGen をすべてのインプリメンテーションで実行するかどうか制御します。

各インプリメンテーション プロセスにも、それぞれカスタマイズされたコマンド ライン オプションがあります。現在のソフトウェアでは、カスタマイズされたオプションは、すべてのコンフィギュレーションに対して設定されます。コマンド ライン ツールをカスタマイズするには、次の 3 つの変数を使用します。3 つすべての変数のデフォルトでは、デフォルトのインプリメンテーション オプションが使用されます。使用可能なコマンド ライン オプションの詳細は、『コマンド ライン ツール ユーザー ガイド』(UG628) を参照してください。

- `NGDBUILD_OPTS <ngdbuild_options>`

  オプションの NGDBuild コマンド ライン オプション

- `MAP_OPTS <map_options>`

  オプションの MAP コマンド ライン オプション

- `PAR_OPTS <par_options>`

  オプションの PAR コマンド ライン オプション

これらのオプションは、すべてのコンフィギュレーションに使用されます。特定のコンフィギュレーションに対して別のコマンド ライン オプションを指定するには、–f オプションを使用して、各ディレクトリのコマンド ファイルを選択します。次に例を示します。

```
MAP_OPTS=<-f ./map.opt>
```

この場合、各インプリメンテーションに対してディレクトリの map.opt ファイルを探して、ファイルに含まれるオプションを使用します。-f オプションの詳細は、『コマンド ライン ツール ユーザー ガイド』(UG628) を参照してください。

推奨フロー

現在のところ、これらのスクリプトでは pr_verify は実行されませんが、今後のリリースで実行されるようにするかどうか調査中です。このため、デバイスのコンフィギュレーション前に生成されたビットストリームを使用して pr_verify を実行する必要があります。

この手順を含めたフローには、次が推奨されます。

1. Tcl スクリプトを使用し、**`bitgen`** も含めた完全なフローを実行します。
2. デバイスをコンフィギュレーションする前に pr_verify コマンド ラインを実行します。ログ ファイルで PASS とレポートされたら、生成されたビットストリームを使用できます。

pr_verify の実行の詳細については、第 4 章の「コンフィギュレーションの検証」を参照してください。

必要なファイルとディレクトリ構造

Tcl スクリプトには、独自のディレクトリが必要です。ソース ファイルはすべて Source ディレクトリにあり、コンフィギュレーションは ディレクトリにインプリメントされ、スタティックおよび各 RM は Synth ディレクトリで合成され、フローを実行するスクリプトはすべて Tools ディレクトリに保存されます。図 5-1 は、ディレクトリ構造の例を示しています。

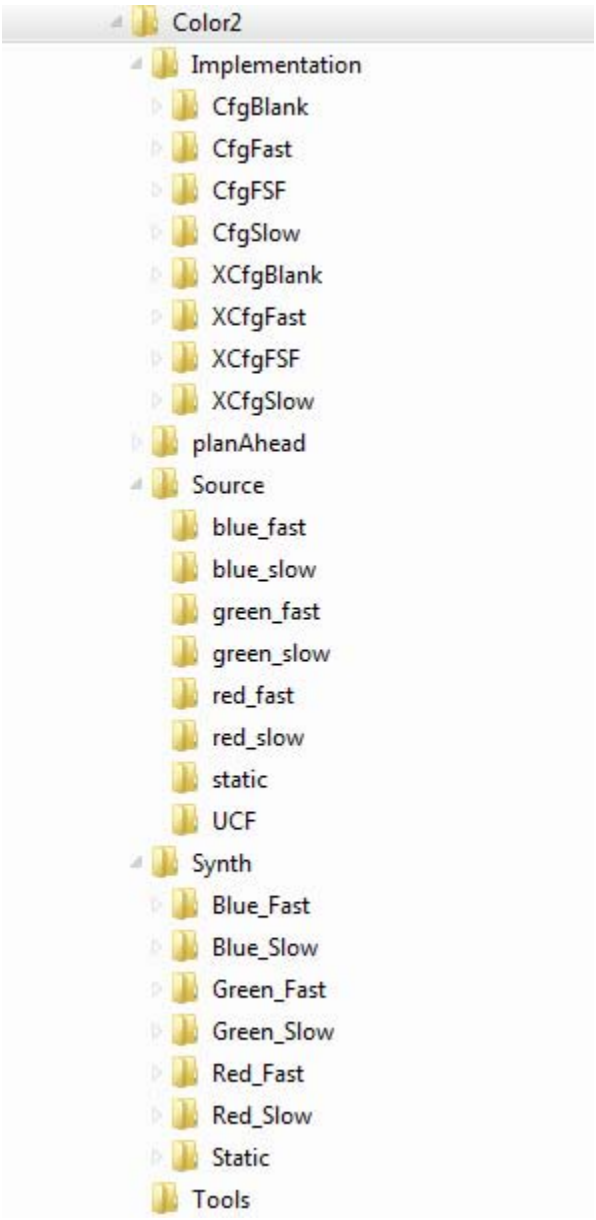

図 5-1 : サンプル スクリプトに必要なディレクトリ構造

これらのディレクトリのうちのどれかがない場合、内容が生成されたかどうかに関係なく、スクリプトがエラーになることがあります。

スクリプトが実行されたら、コンフィギュレーションのディレクトリにそれらを移動し、インプリメンテーションを実行します。レポート ファイルは、デバッグに必要です。

合成 RM ディレクトリ

RUN_RM_SYNTH オプションが YES に設定される場合、リストの各リコンフィギャブル モジュールのディレクトリに合成入力ファイル (.xst および .prj) が含まれている必要があります。

XST ファイルには、合成 run のコマンド ライン オプションが含まれます。XST コマンド ライン オプションの詳細は、『XST ユーザー ガイド (Virtex-6、Spartan-6、および 7 シリーズ用)』(UG687) を参照してください。

次は、XST ファイルの例です。

```
run
-ifn red.prj
-ifmt mixed
-ofn red
-ofmt NGC
-p xc5vlx50t-3-ff1136
-top red
-opt_mode Speed
-opt_level 1
-power NO
-iuc NO
-keep_hierarchy NO
-netlist_hierarchy as_optimized
-rtlview Yes
-glob_opt AllClockNets
-read_cores YES
-write_timing_constraints NO
-hierarchy_separator /
-bus_delimiter <>
-case maintain
-slice_utilization_ratio 100
-bram_utilization_ratio 100
-dsp_utilization_ratio 100
-reduce_control_sets off
-verilog2001 YES
-fsm_extract YES
-fsm_encoding Auto
-safe_implementation No
-fsm_style lut
```

XST ファイルでは、入力ファイルとして適切な PRJ ファイルが指定されます。PRJ ファイルには、リコンフィギャブル モジュールのすべての HDL ファイルと、ソースをコンパイルするための言語とライブラリが指定されます。次に例を示します。

```
verilog work "../../Source/red_fast/led_fast.v"
verilog work "../../Source/red_fast/red_fast.v"
```

XST および PRJ ファイル両方の例は、『XST ユーザー ガイド (Virtex-6、Spartan-6、および 7 シリーズ用)』(UG687) で確認できるほか、ISE® Design Suite からも生成されます。

この例の場合、必要なディレクトリは、Red_Fast、Red_Slow、Red_Blank、Green_Fast、Green_Slow、Green_Blank、Blue_Fast、Blue_Slow、Blue_Blank、および Static です。NGDBUILD_TOP 変数が使用され、$STATIC がリコンフィギャブル モジュールのリストから削除される場合、/Static ディレクトリは必要ありません。

RUN_RM_SYNTH オプションが NO に設定される場合、各リコンフィギャブル モジュールのディレクトリに各モジュールのネットリストが必要です。

コンフィギュレーション ディレクトリ

これらのディレクトリには特定の内容は必要ありませんが、ディレクトリはインプリメンテーションが実行されるために作成する必要があります。上記の例の場合は、CfgFast、CfgSlow、CfgFSF および CfgBlank ディレクトリです。

エクスポート ディレクトリ

エクスポート ディレクトリは、スクリプトで作成され、完了したインプリメンテーションを含むコンフィギュレーションが含まれます。名前は、コンフィギュレーション名 (X<*config_name*>) に基づいて命名されます。この例では、XCfgFast、XCfgSlow、XCfgFSF、XCfgBlank になります。これらのディレクトリのファイルは、スクリプトが実行されるたびに上書きされます。解析または比較の実行結果を保存する場合は、別の新しいディレクトリにコピーしておきます。

XILINX®

第 6 章

FPGA デバイスのコンフィギュレーション

この章では、パーシャル ビット ファイルを使用して FPGA デバイスをコンフィギュレーションする際のシステム デザインに関する注意事項と、パーシャル リコンフィギュレーションを使用する FPGA のアーキテクチャ機能について説明します。

パーシャル リコンフィギュレーションのほとんどの側面が標準のフル コンフィギュレーションと同じなので、このセクションではパーシャル リコンフィギュレーションにのみ存在する側面について説明します。

SelectMAP、シリアル、JTAG、または ICAP (Internal Configuration Access Port) のコンフィギュレーション ポートどれでもパーシャル ビットストリームを読み込むのに使用できます。

SelectMAP または Serial モードを使用してパーシャル ビット ファイルを読み込むには、最初のデバイス コンフィギュレーションの後に使用できるよう、これらのピンを予約しておく必要があります。これには、UCF 制約の CONFIG_MODE (幅は 16 か 32 を選択するためのみに必要) と bitgen -g persist オプションを使用します。

パーシャル ビットストリームには、パーシャル リコンフィギュレーションに必要なすべてのコンフィギュレーション コマンドとデータが含まれます。コンフィギュレーション フレームのアドレス指定情報はパーシャル ビットストリームに含まれるので、パーシャル ビットストリームを FPGA に読み込むのに、リコンフィギャブル モジュールの物理的な位置を知っておく必要はありません。パーシャル ビットストリームは、FPGA デバイスの間違ったパーツには送信できません。

パーシャル リコンフィギュレーション コントローラは、不揮発性メモリからパーシャル ビットストリームを取り出し、コンフィギュレーション ポートへそれを駆動します。このパーシャル リコンフィギュレーション制御ロジックは、外部デバイス (プロセッサなど) かリコンフィギュレーションする FPGA デバイスのファブリックのいずれかに配置できます。ユーザーのデザインした内部パーシャル リコンフィギュレーション コントローラは、パーシャル ビットストリームを ICAP インターフェイスを介して読み込みます。内部パーシャル リコンフィギュレーション制御回路は、スタティック デザインのほかのロジックと同様、パーシャル リコンフィギュレーション プロセスを使用して割り込みなしに動作します。

内部コンフィギュレーションには、カスタム ステート マシンか MicroBlaze™ プロセッサや PowerPC® 405 プロセッサ (PPC405) などのエンベデッド プロセッサのいずれかを含めることができます。

ザイリンクスでは、パーシャル リコンフィギュレーション デザインのデバッグをしやすくするために、iMPACT™ ツールでJTAG ポートを介して FPGA デバイスにフル ビットストリームおよびパーシャル ビットストリームを読み込むことができるようにしています。

ビットストリームのコンフィギュレーション ポートへの読み込みについては、次を参照してください。

- 『Virtex-4 FPGA コンフィギュレーション ユーザー ガイド』(UG071)

- 『Virtex-5 FPGA コンフィギュレーション ユーザー ガイド』(UG191)
- 『Virtex-6 FPGA コンフィギュレーション ユーザー ガイド』(UG360)

コンフィギュレーション モード

パーシャル リコンフィギュレーションは、次のコンフィギュレーション モードでサポートされています。

- **ICAP**

  ユーザー コンフィギュレーション ソリューションには最適なインターフェイスです。ICAP コントローラのインスタンシエーションと ICAP インターフェイスを駆動するロジックが必要になります。

- **JTAG**

  素早いテストやデバッグに最適なインターフェイスです。JTAG をサポートするザイリンクス コンフィギュレーション ケーブルを使用し、iMPACT または ChipScope Analyzer から駆動できます。

- **Slave SelectMAP** または **Slave Serial**

  フル コンフィギュレーションおよびパーシャル リコンフィギュレーションを同じインターフェイスで実行するのに最適なインターフェイスです。

マスタ モードは、コンフィギュレーション メモリを一掃する IPROG のため、直接はサポートされません。

フル ビット ファイルのダウンロード

デジタル システムの FPGA デバイスは、フル ビット ファイルを PROM またはマイクロプロセッサによる汎用メモリ空間から直接ダウンロードして、パワー オン リセット後にコンフィギュレーションされます。フル ビット ファイルには、FPGA デバイスをリセットし、完全なデザインでコンフィギュレーションし、そのビット ファイルが破損していないかどうかを検証するために必要な情報がすべて含まれます。図 6-1 は、このプロセスを示しています。

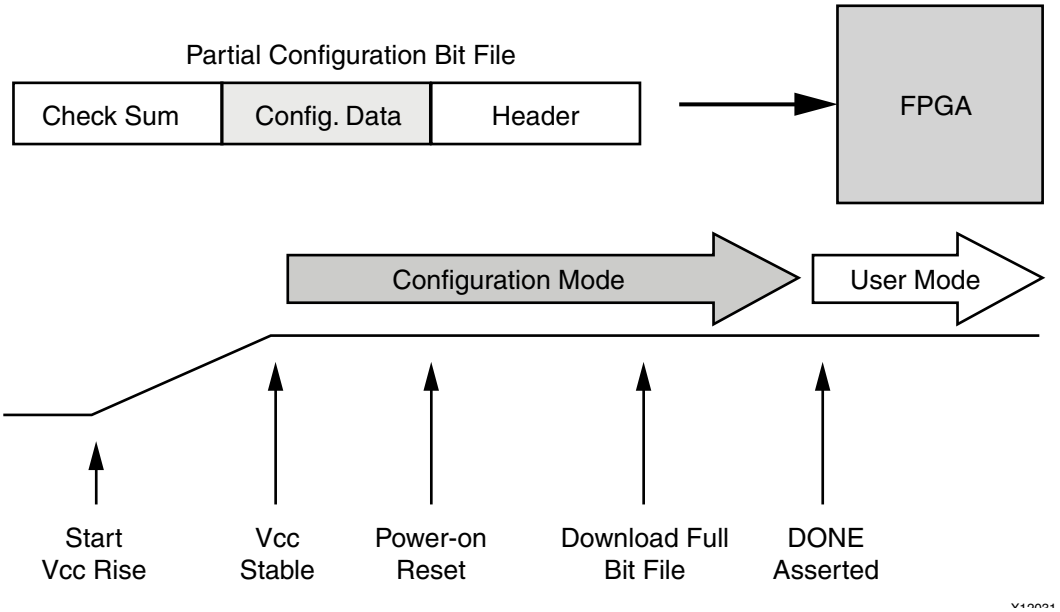

図 6-1 : フル ビット ファイルを使用したコンフィギュレーション

最初のコンフィギュレーションが完了して検証されると、FPGA デバイスがユーザー モードになり、ダウンロードしたデザインが動作し始めます。ビット ファイルの破損が検出されると、DONE 信号がアサートにならず、FPGA デバイスもユーザー モードにならず、デザインが動作し始めることはありません。

パーシャル ビット ファイルのダウンロード

部分的にリコンフィギュレーションされた FPGA デバイスは、パーシャル ビット ファイルが読み込まれている間、ユーザー モードになります。これにより、リコンフィギャブル部分が変更されている間、リコンフィギュレーションされない FPGA ロジック部分は動作し続けることができます。図 6-2 は、このプロセスを示しています。

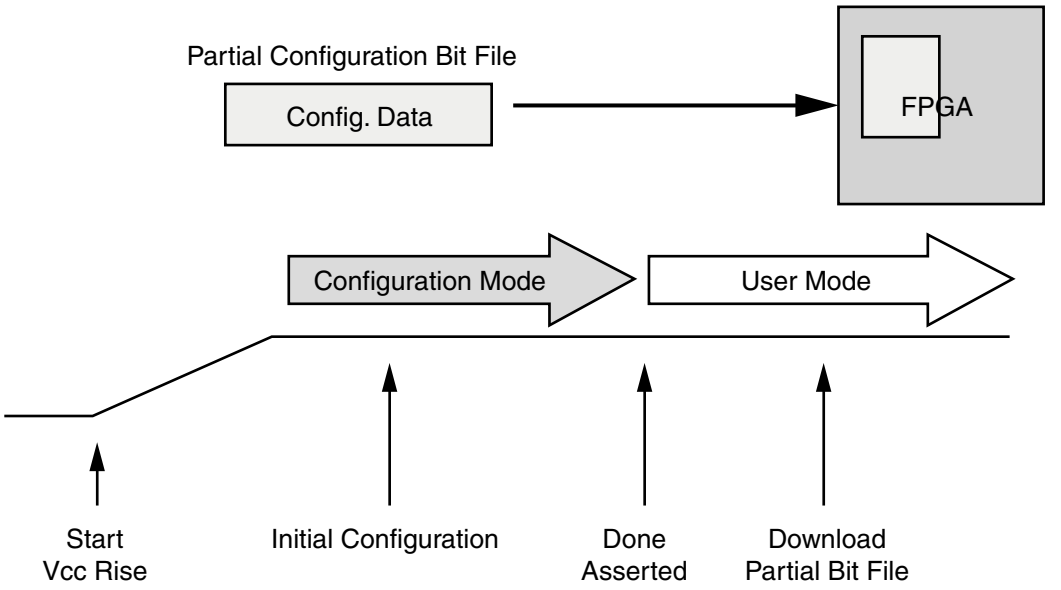

図 6-2 : パーシャル ビット ファイルを使用したコンフィギュレーション

パーシャル ビット ファイルには、ヘッダがなく、FPGA デバイスをユーザー モードにするスタートアップ シーケンスもありません。ビット ファイルには、基本的にフレーム アドレスとコンフィギュレーション データ、最終的なチェックサム値のみが含まれます。パーシャル ビット ファイルの情報が専用モードまたは ICAP を使用してすべて FPGA デバイスに送信される場合、完了を示す外部 DONE ピンはアサートされません。

コンフィギュレーションが終了したときにデータが送信されたかどうかは、ユーザーがモニタする必要があります。パーシャル ビット ファイルの終わりには、ビット ファイルが完全に送信されたことをコンフィギュレーション エンジンに知らせる DESYNCH ワード (0000000D) が含まれます。NO OP コマンドのパディング後にこのワードを含めると、DESYNCH に到達した後、すべてのコンフィギュレーション データが既にデバイスを介してターゲット フレームに送信されたことを確認できます。完全なパーシャル ビット ファイルがコンフィギュレーション ポートに送信され次第、リコンフィギュレーション領域をアクティブな使用のために解放しておくことをお勧めします。

FPGA デバイスをコンフィギュレーションするためのシステム デザイン

パーシャル ビット ファイルはフル ビット ファイルと同じ方法で FPGA デバイスにダウンロードできます。どのパーシャル ビット ファイルをダウンロードするかは外部マイクロプロセッサで決定されます。外部マクロプロセッサは、外部のメモリ空間にあり、パーシャル ビット ファイルを JTAG、SelectMAP またはシリアル インターフェイスなどの標準的な FPGA コンフィギュレーション ポートに指定します。FPGA デバイスでは、受信を指定する特定の命令がなくても、パーシャル ビット ファイルが正しく処理されます。

通常は、フル ビット ファイルをダウンロードする前に FPGA コンフィギュレーション インターフェイスの INIT または PROG 信号をアサートしておきます。ただし、パーシャル ビット ファイルをダウンロードする前は、パーシャル ビット ファイルではなく、フル ビット ファイルの送信が指定されているのでアサートしないでください。

イネーブル信号のホールドおよびクロックのディスエーブルなど、パーシャル ビット ファイルが送信されることを作業中デザインに知らせる動作は、専用の FPGA コンフィギュレーション ピン

を使用するのではなく、デザイン内で指定する必要があります。図 6-3 は、マイクロプロセッサを使用したコンフィギュレーション プロセスを示しています。

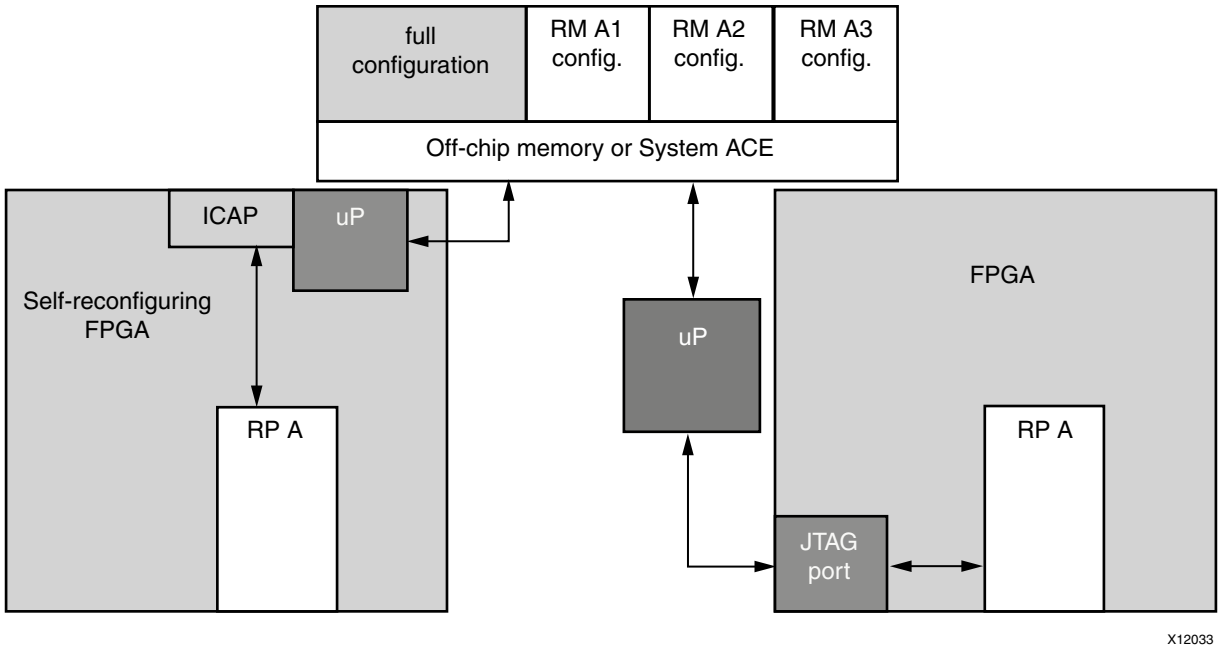

図 6-3 : マイクロプロセッサを使用したコンフィギュレーション

パーシャル リコンフィギュレーションでは、標準的なコンフィギュレーション インターフェイスだけでなく、Internal Configuration Access Port (ICAP) によるコンフィギュレーションもサポートされます。ICAP プロトコルは SelectMAP と同じです。詳細は、該当する FPGA デバイスのコンフィギュレーション ユーザー ガイドを参照してください。ICAP ライブラリ プリミティブは、FPGA デザインの HDL コードでインスタンシエートできるので、コンフィギュレーション ポートに送信される前に、パーシャル ビット ファイルの解析と制御が実行できます。パーシャル ビット ファイルは汎用 I/O またはギガビット トランシーバを介して FPGA デバイスにダウンロードでき、FPGA ファブリックの ICAP に配線できます。

ICAP には、暗号化された Virtex-6 のパーシャル ビット ファイルのパーシャル リコンフィギュレーションの場合は、8 ビット バスのみを使用する必要があります。暗号化が使用される場合、外部コンフィギュレーション ポートを介してリコンフィギュレーションはできません。

パーシャル ビット ファイルのインテグリティ

フル ビット ファイルと比べると、パーシャル ビット ファイルのエラー検出と復元には独自の要件があります。フル ビット ファイルが FPGA デバイスに読み込まれる際にエラーが検出されると、FPGA デバイスがユーザー モードになることはありません。エラーは、破損デザインがコンフィギュレーション メモリに読み込まれ、特定信号がエラー状態を示すためにアサートされると、検出されます。FPGA デバイスはユーザー モードにはならないので、破損デザインがアクティブになることはありません。設計者は、エラーが検出された場合は、異なるビット ファイルをダウンロードするなど、コンフィギュレーション エラーから回復するためのシステム ビヘイビアを決めます。

パーシャル ビット ファイルのダウンロードでは、エラー検出と復元のためにこの手法は使用できません。パーシャル ビット ファイルが読み込まれる際、FPGA デバイスは定義どおり既にユーザー モードになっています。コンフィギュレーション回路ではビット ファイルが読み込まれた後にのみエラー検出がサポートされるので、破損パーシャル ビット ファイルがアクティブになることがあり、長時間動作したままにしておくと、FPGA デバイスにダメージを与えることもあります。

CRC エラーがパーシャル リコンフィギュレーション中に検出されると、FPGA の INIT_B ピンがアサートされます (この場合、INIT_B は CRC エラーを示すために Low になっていす)。システムが最初のコンフィギュレーション中に CRC エラーの INIT_B をモニタした場合は、パーシャル リコンフィギュレーション中の CRC エラーが同じ反応を引き起こすことに注意してください。FPGA 内からの CRC エラーがあるかどうかを検出するには、ICAP ブロックを介して CRC ステータスをモニタします。ステータス レジスタ (STAT) は、CRC_ERROR フラグ (ビット 0) をアサートすることでパーシャル ビット ファイルに CRC エラーがあることを示します。

考慮する必要のあるパーシャル ビット ファイルのエラーには、データ エラーとアドレス エラーの 2 つがあります (パーシャル ビット ファイルは基本的にアドレスとデータの情報です)。

データ部分にエラーがある場合は、簡単に復元できます。新しいパーシャル ビット ファイル (空白のパーシャル ビット ファイルでも) を読み込んで、破損を修復します。

パーシャル ビット ファイルのアドレス部分にエラーがある場合は、簡単には復元できません。破損が FPGA のスタティック部分を変更してしまう可能性があります。この場合、安全に復元するには、新しいフル ビット ファイルをダウンロードして、スタティック ロジックのステートを確定するしか方法はありませんが、FPGA デバイス全体をリセットする必要があります。

多くのシステムは複雑な復元メカニズムを必要としません。これは、FPGA デバイス全体のリセットが重要ではなかったり、パーシャル ビット ファイルがローカルに保存されるからです。この場合、ビット ファイルの破損の可能性はそれほどありません。パーシャル ビット ファイルをラジオ リンクに送信するような、ビット ファイルが破損してしまう危険性のあるシステムには、問題を軽減するためのデザイン回路を含める必要があります。この場合、パーシャル ビット ファイルが ICAP に読み込まれてデバイスが部分的にリコンフィギュレーションされる直前に、ファイルを FPGA ファブリックでローカルに処理する方法があります。

FPGA デザインのスタティック ロジックには、パーシャル ビット ファイルが ICAP に送信される前に解析する回路を含めることができます。エラーが検出されると、パーシャル リコンフィギュレーションが停止し再試行されるか、既知の問題のないパーシャル ビット ファイルが代わりに読み込まれます。図 6-4 は、このプロセスを示しています。

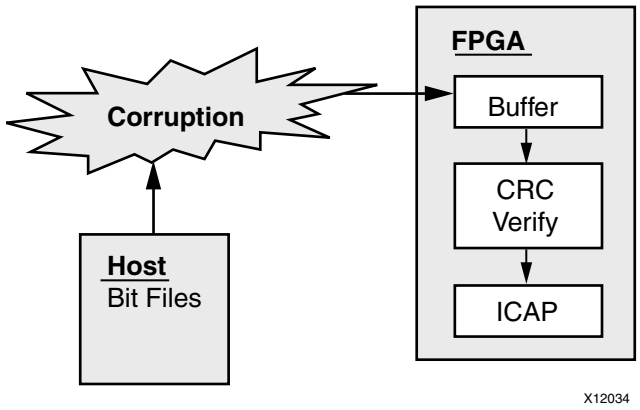

図 6-4 : パーシャル ビット ファイルのエラー検出

パーシャル ビット ファイルには、インテグリティをチェックするために使用できる CRC 情報が含まれます。または、ユーザーがカスタムの CRC 情報を生成してパーシャル ビット ファイルと共に送信することもできます。この方法は、第 2 章「よく使用されるアプリケーション」に説明される「非対称鍵暗号化方式」のアプリケーションと類似しています。

パーシャル ビットストリームの CRC チェック

パーシャル ビットストリームはアクティブ デザインに読み込まれたり、ビルトイン CRC チェックがビットストリームの最後まで実行されなかったりするため、ビットストリーム データが FPGA に読み込まれる前にそれを確認する CRC チェッカーをインプリメントすることをお勧めします。この問題の完全なソリューションには、ソフトウェア ソリューションとハードウェア ソリューションの両方が必要です。ソフトウェア ソリューションでは、ブロックまたはデータ フレームの CRC 値が計算され、ビットストリームにその CRC 値が挿入されます。ハードウェア ソリューションでは、CRC 値が計算し直され、ビットストリームに埋め込まれたソフトウェア値と比較されます。

このソリューションは、格納されたビット ファイルのインテグリティに潜在的なリスクがある場合にのみ必要です。たとえば、パーシャル ビット ファイルのシステムへのリモート アップロードや放射線アップセットになりやすい宇宙用アプリケーションなどで必要になります。

これらのソリューションは、次の図で示されています。

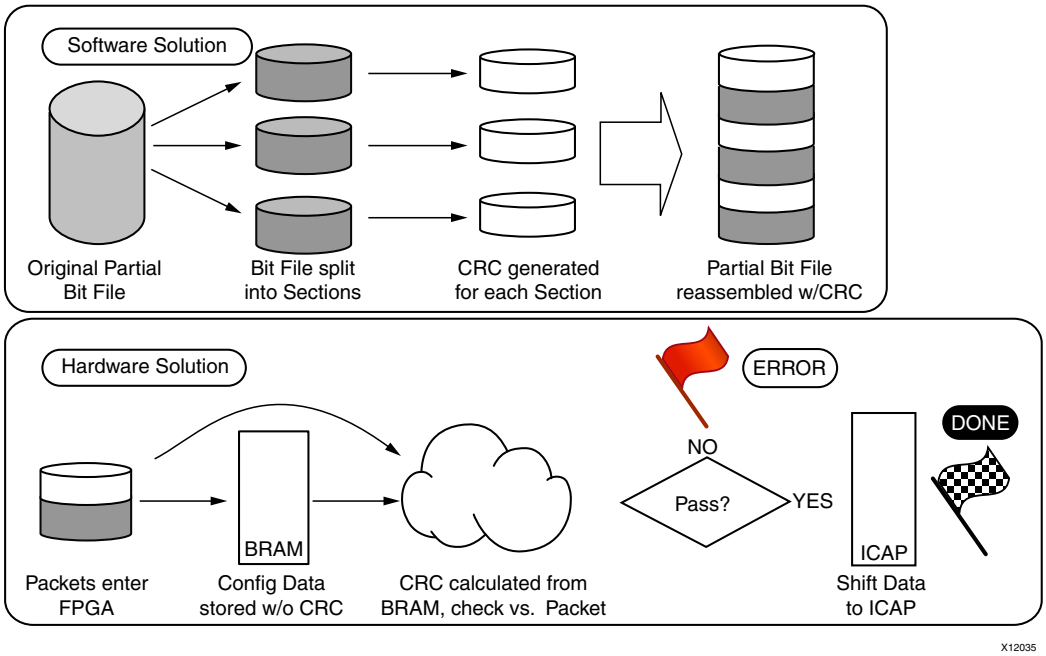

図 6-5 : パーシャル リコンフィギュレーション デザインの CRC チェック

この図の上半分は、ソフトウェア ソリューションの概念を示しています。これは、スクリプトを使用してインプリメントできます。また、将来のソフトウェア リリースに含まれる BitGen でもソリューションを提供する予定です。

この図の下半分は、ハードウェア ソリューションの概念を示しています。将来のソフトウェア リリースでは、BitGen のソリューションと連動するようなリファレンス デザイン/IP コアを提供する予定です。

これと同じようなソリューションを使用して CRC エラーが検出された場合、データを再送信して問題を修正する方法はユーザー自身で見つける必要があります。データ破損は破損データが読み込まれる前にわかるので、スタティック ロジックをリコンフィギュレーションする必要はありません。

コンフィギュレーション フレーム

Virtex デバイス内のユーザー プログラマブル機能は、電源投入時にコンフィギュレーションされる必要のある揮発性メモリ セルで制御されます。これらのメモリ セルは、「コンフィギュレーション メモリ」と総称され、LUT 式、信号配線、IOB 電圧規格、およびその他デザインのすべての側面を定義します。

Virtex アーキテクチャのコンフィギュレーション メモリは、デバイスの周囲にタイル状に並んだフレームに含まれます。これらのフレームは、デバイスのコンフィギュレーション メモリ空間で最も小さなアドレス指定可能なセグメントなので、すべての動作がコンフィギュレーション フレーム全体に影響します。各デバイスのコンフィギュレーション フレーム数は、該当する FPGA デバイス ファミリのコンフィギュレーション ユーザー ガイド (Virtex-4 の場合は表 7-1、Virtex-5 の場合は表 6-1、Virtex-6 の場合は表 6-22) を参照してください。

リコンフィギャブル フレームは、これらのコンフィギュレーション フレーム上に構築され、パーシャル リコンフィギュレーションを実行する最小の構築ブロックです。

- Virtex-6 の基礎領域は高さ 40 CLB X 幅 1 CLB
- Virtex-5 の基礎領域は高さ 20 CLB X 幅 1 CLB
- Virtex-4 の基礎領域は高さ 16 CLB X 幅 1 CLB

ブロック RAM、IOB、I/O エレメント (ILOGIC、OLOGIC、IODELAY、DSP48) などの別のエレメント タイプにも、同様の基礎領域があります。これらの基礎領域のサイズを検証するには、PlanAhead™ ソフトウェアのフロアプラン機能を使用してください。

PlanAhead のマニュアルの「フレーム」および上記の「リコンフィギャブル フレーム」は、デバイスのコンフィギュレーション ユーザー ガイドの「コンフィギュレーション フレーム」とは同じ意味で使用されていません。フレームは、パーシャル リコンフィギュレーションの [PR Statistics] タブに示されるように、最小のリコンフィギャブル構築ブロックであり、これより小さく分割はできません。1 つのリコンフィギャブル フレームよりも小さいエリア グループを選択した場合でも、フレーム全体がリコンフィギュレーションされます。

リコンフィギャブル パーティションに対応する Pblock を描画すると、そのパーティションの詳細が [Pblock Properties] ビューに表示されます。[Statistics] タブには、その Pblock に含まれるフレーム (領域) 数とリコンフィギャブル パーティションのビットストリーム サイズの概算が表示されます。Pblock のサイズを変更すると、この情報もそれに合わせて変更されます。

コンフィギュレーション時間

コンフィギュレーションの速度は、パーシャル ビット ファイルとコンフィギュレーション ポートのバンド幅に直接関連します。表 6-1 は、Virtex アーキテクチャのコンフィギュレーション ポートのバンド幅を示しています。

表 6-1 : Virtex アーキテクチャのコンフィギュレーション ポートの最大バンド幅

コンフィギュレーション モード	最大クロック レート	データ幅	最大バンド幅
ICAP	100 MHz	32 ビット	3.2 Gbps
SelectMAP モード	100 MHz	32 ビット	3.2 Gbps

表 6-1 : Virtex アーキテクチャのコンフィギュレーション ポートの最大バンド幅

コンフィギュレーション モード	最大クロック レート	データ幅	最大バンド幅
シリアル モード	100 MHz	1 ビット	100 Mbps
JTAG	66 MHz	1 ビット	66 Mbps

PlanAhead の [PR Statistics] タブにレポートされるリコンフィギャブル パーティションのビットストリームのサイズは、作成されるパーシャル ビット ファイルのサイズの正確な概算です。この数値はバイトで表示されるので、ビットストリームのサイズをビットで算出する場合は、これを 8 倍にします。

例 : Virtex-5 デバイスの小さいパーシャル ビット ファイルには、200 スライスにまたがる領域が含まれ、この領域はリコンフィギャブル フレーム 5 個 (幅 5 CLB X 高さ 20 CLB = 100 CLB) が含まれるように描画されています。コンフィギュレーション時間は、PlanAhead ソフトウェアで表示されるビットストリーム サイズ (29,520 バイトまたは 236,160 ビット) を使用して、RBT (rawbits) ファイルが生成される前に概算できます。SelectMAP モードまたは ICAP を使用した場合、このパーシャル ビット ファイルは次の時間で読み込むことができます。

```
236,160 ビット / 3,200,000,000 bps = 0.0000738 秒
```

または約 73.8 マイクロ秒パーシャル ビット ファイルのサイズはフレーム数と共に増加するので、コンフィギュレーション時間はかなり直線的に上がりますが、フレームの位置および内容によって少しずつ異なります。また、最後のフレームが読み込まれた後に少量のオーバーヘッドもあります。

ビットストリームの正確な長さは、BitGen で -b オプションを使用すると、RBT ファイルに表示されます。この数値とバンド幅を使用して、コンフィギュレーション時間の合計を計算します。上記の例で作成されたビットストリームのヘッダは、次のような RBT ヘッダになります。実際のコンフィギュレーション時間は、約 75.6 マイクロ秒です。

```
Xilinx ASCII Bitstream
Created by Bitstream L.46
Design name: FFF_routed.ncd;UserID=0xFFFFFFFF
Architecture: virtex5
Part:     5vlx50tff1136
Date:     Mon Feb 14 14:00:59 2011
Bits:     242016
11111111111111111111111111111111
```

コンフィギュレーションのデバッグ

ICAP インターフェイスは、JTAG や Slave SelectMAP などのほかのコンフィギュレーション手段が使用される場合でも、コンフィギュレーション プロセスをモニタするために使用できます。実際に、コンフィギュレーションのステータスは読み出し命令がなくても ICAP の O ポートから自動的に出力されます。

ICAP ブロックの O ポートは 32 ビット バスですが、最下位バイトのみが使用されます。下位バイトのマップは、次のようになります。

表 6-2 : ICAP の O ポートのビット

ビット番号	ステータス ビット	説明
O[7]	CFGERR_B	コンフィギュレーション エラー (アクティブ Low) 0 = コンフィギュレーション エラーが発生 1 = コンフィギュレーション エラーなし
O[6]	DALIGN	同期ワードの受信 (アクティブ High) 0 = 同期ワードの受信なし 1 = インターフェイス ロジックで同期ワードの受信あり
O[5]	RIP	リードバックを実行中（アクティブ High） 0 = 処理中のリードバックなし 1 = リードバックを処理中
O[4]	IN_ABORT_B	ABORT を実行中（アクティブ Low） 0 = 停止を処理中 1 = 処理中の停止なし
O[3:0]	1	予約済み

このバイトの最上位ニブルがステータスをレポートします。これらのステータス ビットは、同期ワードが受信されたか、コンフィギュレーション エラーが発生されたかを示します。次の表は、これらの状況に対する値を示しています。

表 6-3 : ICAP 同期ビット

O[7:0]	同期ワード	CFGERR
9F	同期なし	CFGERR なし
DF	同期あり	CFGERR なし
5F	同期あり	CFGERR
1F	同期なし	CFGERR

図 6-6 は、フル コンフィギュレーションの終了後、CRC エラーを含むパーシャル リコンフィギュレーションが実行され、最後に問題のないパーシャル リコンフィギュレーションが実行されたところを示しています。上記の表と下記の説明から、ICAP の O ポートを使用してコンフィギュレーション プロセスを監視する方法がわかります。CRC エラーが発生した場合は、これらの信号をコンフィギュレーション ステート マシンで使用して、エラーから回復することができます。これらの信号は、ChipScope でもデバッグ目的でコンフィギュレーション エラーを検出するために使用できます。この情報を使用すると、ChipScope でもパーシャル リコンフィギュレーションのさまざまなポイントを確認できます。

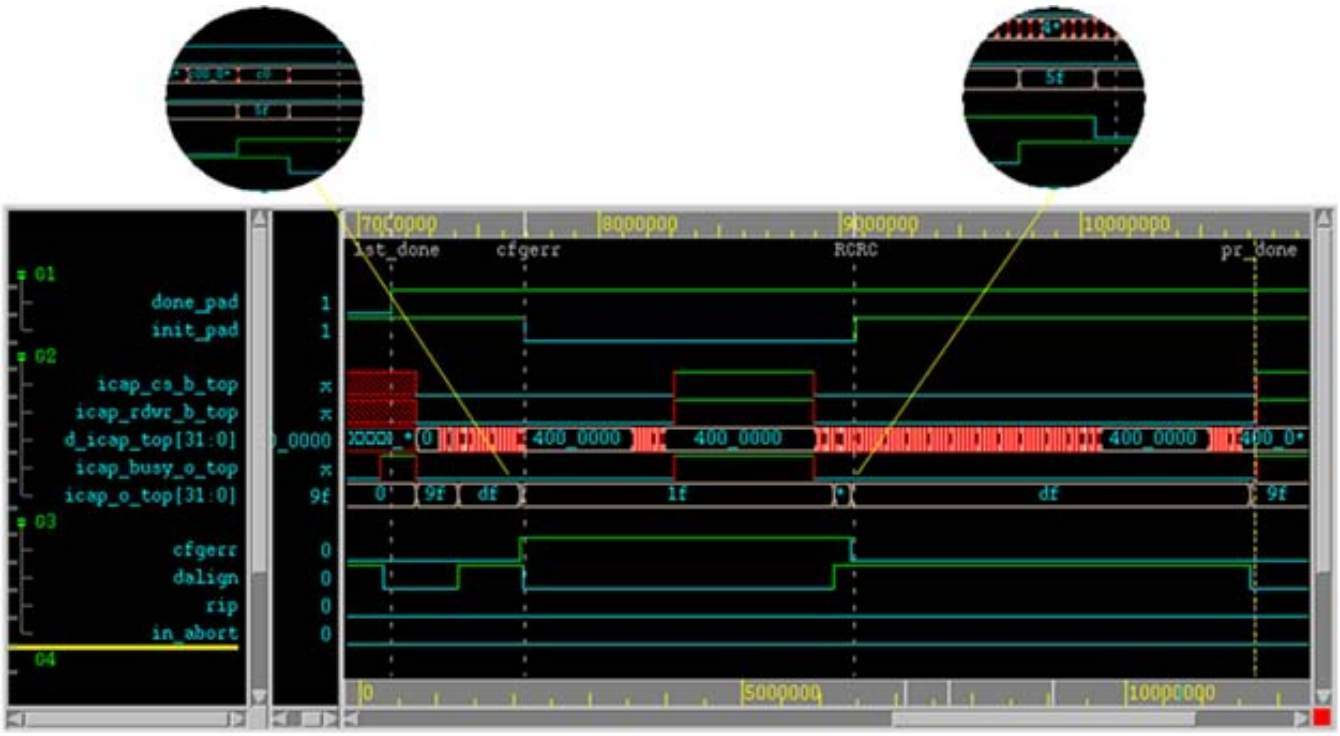

図 6-6 : パーシャル リコンフィギュレーションの ChipScope 表示

ChipScope で表示されるマーカーはそれぞれ次を示しています。

- **1st_done**

  最初のフル ビットストリームのコンフィギュレーションが終了したことを示します。DONE ピン (波形の done_pad) は High になります。

- **cfgerr**

  パーシャル ビットストリームの読み込み中に CRC エラーが検出されたことを示します。ステータスは O[31:0] (波形の icap_o_top[31:0]) で確認できます。

  - Icap_o_top[31:0] は 0x9F から開始します。
  - SYNC ワードを確認すると、Icap_o_top[31:0] は 0xDF に変わります。
  - CRC エラーを検出すると、Icap_o_top[31:0] は 1 サイクル間 0x5F になったあと、0x1F に切り替わります。
  - INIT_B ピンは Low になります (波形の init_pad)。

- **RCRC**

  パーシャル ビットストリームが再び読み込まれたことを示します。RCRC コマンドは cfgerr ステータスをリセットし、INIT_B ピン (波形の init_pad) のプルダウンを削除します。

  - SYNC ワードが確認されると、Icap_o_top[31:0] は 0x1F から 0x5F に変更されます。
  - RCRC コマンドが受信されると、Icap_o_top[31:0] は 0x5F から 0xDF に変更されます。

- **pr_done**

  パーシャル リコンフィギュレーションが問題なく終了したことを示します。

  - DESYNC コマンドが受信され、コンフィギュレーション エラーが検出されない場合、Icap_o_top[31:0] は 0xDF から 0x9F に変更されます。

パーシャル リコンフィギュレーションではパーシャル ビット ファイルすべてが読み込まれるまで CRC が実行されないので、破損データが既に FPGA に読み込まれてしまっている可能性があることに注意してください。破損がアドレス ビットで発生している場合は、スタティック ロジックも破損している可能性があります。ステータスは INIT_B コンフィギュレーション レジスタ ビットで示されます。高い信頼度のシステムが必要な場合は、パーシャル ビットストリームをコンフィギュレーション インターフェイスに送信する前に CRC チェックを必ず実行してください。読み込み前にパーシャル ビットストリームで CRC チェックを実行する方法については、この章の「パーシャル ビットストリームの CRC チェック」セクションを参照してください。

CRC エラーが発生したら、コンフィギュレーション インターフェイスはデフォルトでデバイスのフル リコンフィギュレーションを命令しようとします。これは、通常は推奨されません。これを避けるには、第 3 章の「ビット ファイルの生成」の方法に従ってください。

XILINX®

第 7 章

デザインの注意事項

この章では、パーシャル リコンフィギュレーション用のデザイン要件およびザイリンクス FPGA デザイン ソフトウェア ツール内のパーシャル リコンフィギュレーション機能について説明します。®

ザイリンクスの FPGA デバイスのパーシャル リコンフィギュレーション機能を利用するには、まずデザイン仕様をよく解析し、パーシャル リコンフィギュレーション デザインに関する要件、特徴、制限などを考慮する必要があります。これにより、デザイン プロセスおよびデバッグ プロセスの両方が簡素化され、潜在的な問題を回避できます。

デザイン階層

良いデザイン階層手法を使用することで、FPGA デザインにパーシャル リコンフィギュレーションを実行する際の複雑さや困難な点を軽減できます。明確なデザイン インスタンス階層により、物理制約およびタイミング制約がシンプルになります。スタティックとリコンフィギャブル ロジック間のバウンダリの信号にレジスタを付けると、タイミング クロージャが容易になります。ロジックは、同じ階層レベルにグループごとにまとめる必要があります。

これらは、よくあるデザイン手法ですが、標準の FPGA デザインでは使用されないことが多々あります。部分的にリコンフィギャブルなデザインでは、これらのデザイン規則に完全に従う必要はありませんが、規則に従わないことで悪影響の出ることもあります。パーシャル リコンフィギュレーションは便利なデザイン手法ですが、デザインが複雑になるため、特にハードウェアでデバッグする場合に問題となることもあります。

デザイン階層の商才については、次を参照してください。

- 『Repeatable Results with Design Preservation』 (WP362)
- 『階層デザイン手法ガイド』 (UG748)

リコンフィギャブル モジュール内のデザイン エレメント

すべてのロジックがリコンフィギュレーションできるわけではありません。グローバル ロジックおよびクロック リソースは、リコンフィギュレーション中も動作可能な状態にしておくだけでなく、フル デバイス コンフィギュレーションの終わりに起こる初期化シーケンスの影響を受けるように、スタティック領域に配置する必要があります。スタティック ロジックに配置する必要のあるロジックは次のとおりです。

- クロック修正ブロック (PLL、PMCD、DCM)
- 一部のクロック バッファ (BUFG)

  BUFR は、この章の「クロック規則」セクションで説明されるように、制限はありますが、リコンフィギャブル モジュールに配置できます。

- ザイリンクスでは、デバイス特有のブロックはスタティック ロジックに配置することをお勧めしています。これらのエレメントはリコンフィギュレーション後も問題なく動作する可能性はありますが、シリコン テストは実行されていません。これらのエレメントに含まれるのは、BSCAN、CAPTURE、DCIRESET、FRAME_ECC、ICAP、KEY_CLEAR、STARTUP、および USR_ACCESS です。

ロジックのパック

一緒にパックする必要のロジックは、スタティックまたはリコンフィギャブルに関係なく、同じグループに配置する必要があります。たとえば、I/O レジスタは I/O ポートと一緒のグループにします。パーティション バウンダリは、最適化の障害になります。プロキシ ロジックの挿入により、不適切な結果になったり、達成できない配線になったりするので、階層バウンダリを選択する際には注意が必要です。

リコンフィギャブル モジュールの I/O

デバイス ピンはリコンフィギャブル モジュール内に配置できるので、リコンフィギュレーションできます。リコンフィギャブル モジュールには、内部ロジックをパッケージ ピンに接続するのに必要な I/O 回路 (IBUF および OBUF など) が必ず含まれ、そのポートは名前別にスタティック ロジックにのみ接続されている必要があります。

つまり、I/O 機能にはモジュール内にのみ制約を付ける必要がありますが、完全なデザインのポート リストは最上位レベルのデザイン記述に残ったままになります。サブモジュール I/O のほかの要件は、次のとおりです。

- **HDL**
  - リコンフィギャブル パーティションに含まれる各リコンフィギャブル モジュールには、同じセットのモジュール ポートが必要です。
  - 外部ポート宣言は、すべて最上位レベルで作成される必要があります。
  - 自動 IOB 挿入は、最上位レベルの合成ではオン、モジュール レベルの合成ではオフにします。
  - 合成ツールでは、サブモジュールのピンの宣言が別の方法で処理されます。XST および Synplify の例は、この後のサブセクションに記述しています。
- **UCF**
  - リコンフィギャブル モジュールに配置されるすべての I/O には、ロケーション制約が必要です。ロケーション制約は、さまざまなモジュール間で一貫性するように、最上位レベルの UCF に記述する必要がありますが、必要であれば、サブモジュールの UCF に記述することもできます。
  - たとえば、PlanAhead でリコンフィギャブル モジュールをフロアプランして、リコンフィギャブル パーティション内の 1 つのリコンフィギャブル モジュールの I/O の位置を変更した場合、そのリコンフィギャブル モジュール (アクティブ リコンフィギャブル モジュール) のみの位置が UCF で変更されます。この後、そのリコンフィギャブル パーティションのそれ以外のリコンフィギャブル モジュールすべてで、その I/O に新しい位置へ制約を付ける必要があります。
  - リコンフィギャブル領域には、IOB サイトを含む AREA_GROUP RANGE 制約と、必要に応じて ILOGIC または OLOGIC などのほかのタイプの制約を含める必要があります。

次のコード例では、port_in ポートと port_out ポートが最上位レベルからリコンフィギャブル モジュールの I/O ロジックへ接続され、clk、reset、data_in、data_out ポートが最上位レベルの I/O

ロジックに接続されています。スタティックからサブモジュールをインスタンシエートするには、port_in および port_out へのリファレンスを含める必要があり、サブモジュールには該当ポートへの I/O ロジックのインスタンシエーションを含める必要があります。

- **XST**
  - XST では、BUFFER_TYPE 属性を使用して I/O バッファを挿入するか、挿入しないかを指定できます。XST を使用するのに推奨されるフローでは、合成ツールで最上位レベルに I/O を挿入し (デフォルト)、この属性を使用して、最上位レベルで I/O バッファを受信しないことを I/O に示します。この属性は最上位レベル ポートの定義でポートに適用します。IBUF および OBUF などの I/O コンポーネントは、サブモジュールの HDL でインスタンシエートする必要があります。

次のコードは、XST の Verilog および VHDL の例です。

```
//XST Verilog example
//top level HDL
module top (clk, reset, data_in, data_out, port_in, port_out);
 input clk, reset, data_in
 (* buffer_type = “none” *) input port_in;
 output data_out
 (* buffer_type = “none” *) output port_out;
...


--XST VHDL example
–-top level HDL
...
entity top is
port(
   clk    : in std_logic;
   reset   : in std_logic;
   data_in  : in std_logic;
   data_out : out std_logic;
   port_in  : in std_logic;
   port_out : out std_logic
   );

attribute buffer_type: string;
attribute buffer_type of port_in  : signal is "none";
attribute buffer_type of port_out  : signal is "none";
end my_rm;
...
```

- **Synplicity**
  - Synplicity では、この機能が syn_black_box および black_box_pad_pin 制約を使用してサポートされます。ここでの方法は異なり、すべてのポートは最上位レベルにリストされたままですが、Synplicity にはブラック ボックス モジュールの定義で I/O バッファがサブモジュール (まだ最上位レベルの HDL にあり) で検出されたことが知らされます。XST の場合と同様、リコンフィギャブル I/O ロジックは各リコンフィギャブル モジュールでインスタンシエートされる必要があります。次のコード例は、Synplicity の Verilog ファイルと VHDL ファイルをそれぞれ示しています。

```
//Synplicity Verilog example
//reconfigurable module declaration within Top level HDL
module my_rm (clk, reset, data_in, data_out, port_in, port_out)
   /*synthesis syn_black_box black_box_pad_pin="port_in, port_out"*/;
 input clk, reset, data_in;
 input port_in;
 output data_out;
 output port_out;
endmodule
--Synplicity VHDL example
–-reconfigurable module entity declaration within Top level HDL
...
entity my_rm is
port(
   clk    : in std_logic;
   reset   : in std_logic;
   data_in  : in std_logic;
   data_out : out std_logic;
   port_in  : in std_logic;
   port_out : out std_logic
   );

attribute syn_black_box : boolean;
attribute syn_black_box of my_rm: component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of my_rm: component is "port_in,
port_out";

end my_rm;
...
```

```
--Synplicity VHDL example
--reconfigurable module entity declaration within Top level HDL
...
entity my_rm is
port(
      clk       : in std_logic;
      reset     : in std_logic;
      data_in   : in std_logic;
      data_out : out std_logic;
      port_in   : in std_logic;
      port_out  : out std_logic
     );

attribute syn_black_box : boolean;
attribute syn_black_box of my_rm: component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of my_rm: component is "port_in, port_out";

end my_rm;
...
```

IOB での I/O レジスタのパック

入力および出力レジスタは、できるだけ関連する入力または出力バッファと同じパーティションに含めることをお勧めします。これにより、レジスタが I/O ロジックに接続されると、インプリメンテーション ツールで認識されるようになります。レジスタと関連バッファ間にパーティション バウンダリがあると、インプリメンテーション ツールではパーティション バウンダリを超える場合認識されないので、I/O ロジックにレジスタが正しく配置されません。

正しく配置されない場合、次の規則に従っていると、インプリメンテーション ツールでこの問題が処理されます。

- レジスタには、UCF 制約の IOB=FORCE を指定する必要があります。これにより、パーティション バウンダリを介してレジスタが I/O バッファに接続されていることがツールで認識され、I/O ロジックのレジスタ (ILOGIC/OLOGIC) を配置できるようになります。IOB=FORCE を使用すると、レジスタが I/O ロジックに配置できない場合にインプリメンテーションでエラーになります。これは、レジスタが I/O ロジックに配置される必要のある場合 (レジスタのクロックが BUFIO から供給される場合やインターフェイスのタイミングが固定遅延を必要とする場合など) に必要な機能です。このような場合、map -pr b オプションを使用してもフラット フローの場合やバッファとレジスタが同じパーティションにある場合と違い、I/O ロジックにレジスタが配置されません。
- IOB=FORCE 制約はレジスタのインスタンス名にする必要があります (INST "rp_module/out1_ff" IOB=FORCE;)。この制約はレジスタの出力または入力ネットには適用しないでください。
- リコンフィギャブル パーティションの AREA_GROUP 制約には、入力/出力レジスタが配置される I/O ロジックを含める必要があります。たとえば、レジスタに接続される I/O バッファに関連する ILOGIC/OLOGIC を含む RANGE 値を必ず設定する必要があります。I/O ロジック サイトがリコンフィギャブル パーティションの AREA_GROUP には属さない場合、ツールではそのサイトを使用できないようになります (またはパーシャル ビットストリームにそのサイトが含まれません)。
- リコンフィギャブル パーティションの出力ポートには PARTITION_PIN_DIRECT_ROUTE 制約が設定され、バッファとレジスタ間にプロキシ ロジックが挿入されないようになります (これにより、レジスタが I/O ロジックにパックされるのを回避できます)。また、これにより、このリコンフィギャブル パーティションに関連するすべてのリコンフィギャブル モジュールに同じ IOB=FORCE 制約が設定され、このリコンフィギャブル パーティションのブラックボックス リコンフィギャブル モジュールの生成機能がオフになります。

デザイン インスタンスの階層

最も簡潔なのは、リコンフィギャブル パーティションを最上位レベル モジュールにインスタンシエートする方法です。各リコンフィギャブル パーティションはそれぞれ 1 つのインスタンスに対応している必要があります。インスタンスには、それが関連付けられた複数のモジュールが含まれます。

リコンフィギャブル モジュールのサブモジュール

リコンフィギャブル モジュールのロジックはすべて同じディレクトリに保存される必要があります。リコンフィギャブル モジュールにサブモジュール ネットリスト ファイルが必要な場合、それらのファイルがルートのリコンフィギャブル モジュール ネットリストと同じローカル フォルダにないと、PlanAhead™ ソフトウェアで読み込まれません。PlanAhead では、各コンフィギュレー

ションを制約付けし、インプリメントするために、リコンフィギャブル モジュールすべての内容が必要になります。

ほかのネットリスト (たとえば、IP コア ネットリスト) をほかのディレクトリから統合する必要のある場合は、ngcbuild ユーティリティを使用すると、リコンフィギャブル モジュールをパーシャル リコンフィギュレーション プロジェクトで簡単に参照されるように 1 つのネットリストにまとめることができます。NGCBuild では、EDIF や NGC ソース ファイルが ngdbuild で有効なオプション セット (-sd および -uc) と共に読み込まれ、1 つの制約がアノテートされた NGC ファイルが出力されます。

クロック規則

このセクションでは、グローバル クロックとリージョナル クロックの規則について、それぞれ説明します。

グローバル クロック

特定のリコンフィギャブル パーティションに含まれるリコンフィギャブル モジュールすべてのクロック情報は最初のインプリメンテーション段階ではわからないので、パーシャル リコンフィギュレーション ツールではそのリコンフィギャブル パーティションのパーティション ピンを駆動する各 BUFG 出力を AREA GROUP で囲んだすべてのクロック領域にあらかじめ配線 (前配線) します。つまり、これらのクロック領域のクロック スパインは、リコンフィギャブル パーティションがその領域にロードを持つかどうかに関係なく、スタティック ロジックでは使用できない可能性があります。

図 7-1 の例では、配置前に icap_clk が X0Y1、X0Y2、X0Y3 というクロック領域に配線されているので、スタティック ロジックはその領域のそれ以外のクロック スパインを使用できます。

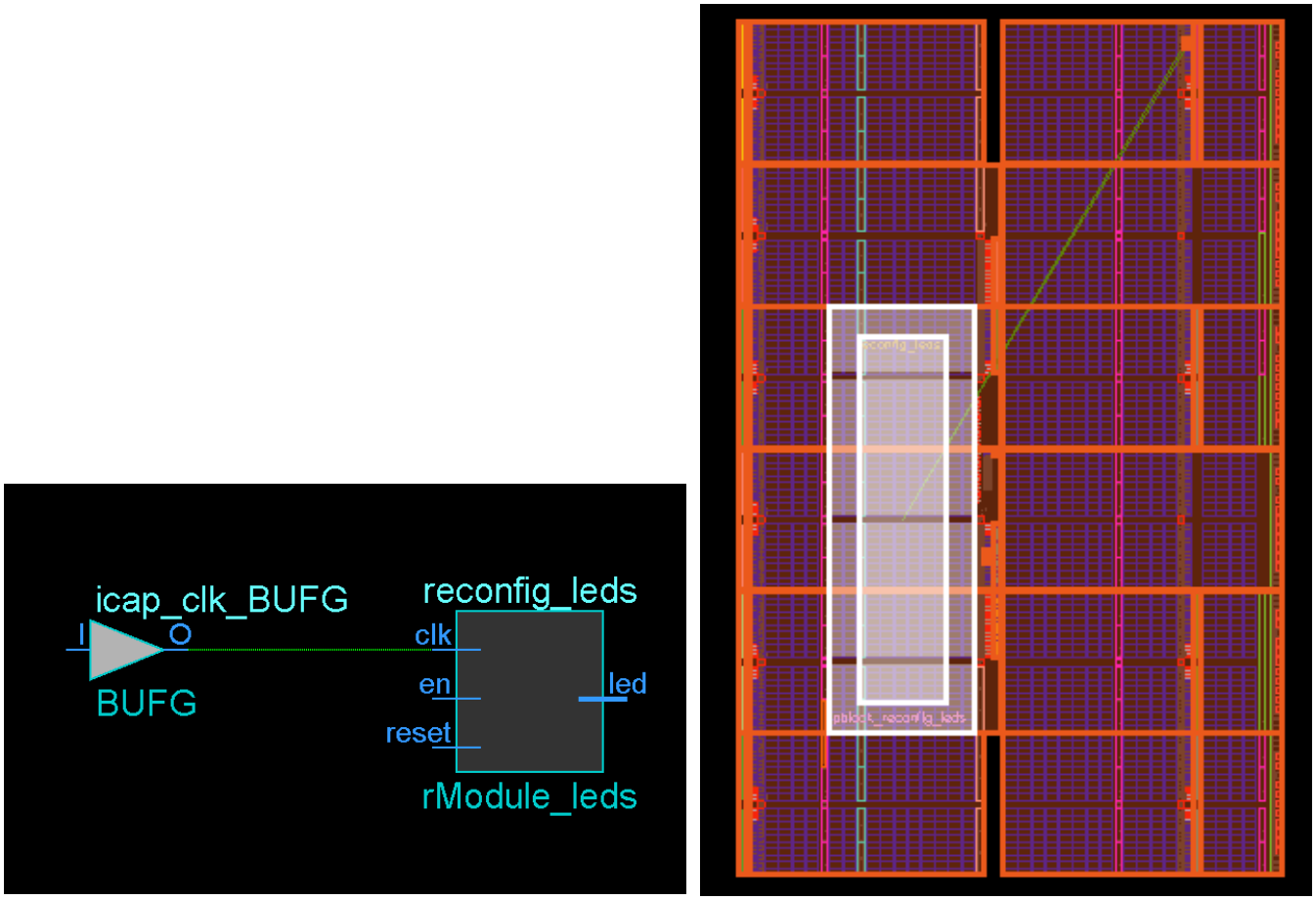

図 7-1 : グローバル クロックからリコンフィギャブル パーティションへの前配線

リコンフィギャブル パーティションを駆動するグローバル クロック数が多い場合は、完全なクロック領域を囲むエリア グループを作成して、スタティック ロジックを配置配線しやすくしてください。各領域のクロック スパインの数については、http://japan.xilinx.com/support/documentation からターゲット デバイスのユーザー ガイドを参照してください。

リージョナル クロック

パーティション ピンを駆動するスタティック ロジックの BUFR には、プロキシ ロジックが挿入されます。このロジックは、スタティック ロジックからリコンフィギャブル パーティションまでのその他すべてのネットの場合と同じように、異なるコンフィギュレーション間の定留点を保証するために必要です。プロキシ ロジックはリージョナル クロックに遅延を追加します。このため、タイミングに悪影響が出ることがあります。

タイミングの悪化を防ぐには、BUFR とそれらのロードすべてが、スタティックでもリコンフィギャブルでも同じパーティション内になるように制約を付けてください。これにより、リージョナル クロックのロード間のスキューを最小限に抑えることができ、タイミングを満たす可能性が上がります。

リージョナル クロックには、3 つのサポートされる Virtex® アーキテクチャそれぞれで異なる制限があります。

- **Virtex-4**

  リージョナル クロックにはプロキシ ロジックは挿入されず、リージョナル クロック スパインは前配線されます。スパインは、そのクロック スパインを必要とする各クロック領域へ前配線されます。たとえば、リージョナル クロックが 2 つのクロック領域部分を含むリコンフィギャブル パーティションのポートを駆動する場合、リージョナル クロック スパインは両方のクロック領域でそのリージョナル クロック用に予約されます。各クロック領域で使用できるリージョナル クロック スパインは 2 つだけなので、リージョナル クロックで駆動されるリコンフィギャブル パーティションの領域ができるだけクロック領域のバウンダリに対応するようにしてください。また、BUFR は特定の BUFR 位置に指定されるように制約を付けてください。リージョナル クロックの制限については、『Virtex-4 FPGA ユーザー ガイド』(UG070) の「クロック リソース」の章を参照してください。

- **Virtex-5**

  プロキシ ロジックはスタティック とリコンフィギャブル パーティション間のリージョナル クロックに挿入されます。プロキシ ロジックはリージョナル クロックに遅延を追加します。このため、タイミングに悪影響が出ることがあります。タイミングの悪化を防ぐには、BUFR とそれらのロードすべてが、スタティックでもリコンフィギャブルでも同じパーティション内になるように制約を付けてください。これにより、リージョナル クロックのロード間のスキューを最小限に抑えることができ、タイミングを満たす可能性が上がります。また、BUFR は特定の BUFR 位置に指定されるように制約を付けてください。その他すべてのリージョナル クロックの制限が適用されます。リージョナル クロックの制限については、『Virtex-5 FPGA ユーザー ガイド』(UG190) の「クロック リソース」の章を参照してください。

- **Virtex-6**

  リージョナル クロックにはプロキシ ロジックは挿入されず、リージョナル クロック スパインは前配線されます。スパインは、そのクロック スパインを必要とする各クロック領域へ前配線されます。たとえば、リージョナル クロックが 2 つのクロック領域部分を含むリコンフィギャブル パーティションのポートを駆動する場合、リージョナル クロック スパインは両方のクロック領域でそのリージョナル クロック用に予約されます。

  Virtex-6 は、同じクロック領域内に複数の BUFR 列を持つ最初のアーキテクチャです。パーシャル リコンフィギュレーションでは、1 つのクロック領域内で使用されるすべての BUFR が同じパーティションに含まれる必要があります。たとえば、外部の BUFR が内部 BUFR と同じクロック領域で使用される場合、どちらもスタティック内に含まれるか、同じリコンフィギャブル パーティション内に含まれている必要があります。リージョナル クロックの制限については、『Virtex-6 FPGA クロック リソース ユーザー ガイド』(UG362) を参照してください。

アクティブ Low のリセットとクロック イネーブル

ザイリンクス FPGA アーキテクチャには、制御信号 (リセットまたはクロック イネーブル) にローカル インバーターがありません。

次は、リセットを例に説明していますが、クロック イネーブルでも同じです。

デザインでアクティブ Low のリセットが使用される場合、信号を反転させるのに LUT を使用する必要があります。すべてのアクティブ Low のリセットを使用するパーティションのないデザインでは、複数の LUT が推論されますが、これらは 1 つの LUT にまとめて、I/O エレメントへプッシュすることができます (LUT は削除されます)。high と low の両方を使用するパーティションのないデザインでは、LUT インバーターをデザインに残った 1 つの LUT にまとめることができますが、配線およびリセット ネットのタイミングにはあまり影響しません (LUT の出力はグローバル リソースに置いたままにできます)。ただし、パーティションにアクティブ Low のリセットを使用するデザインの場合、統合も分割もできないパーティション内部で、推論されるインバーターを取得可能です。これにより、グローバル リソースにリセットを置くことは不可能になるので、リセット タイミングが悪くなることがあり、デザインが既に混線している場合は配線問題も発生することがあります。

これを回避する最適な方法は、アクティブ Low の制御信号を使用しないことですが、AXI (Advanced eXtensible Interface) インターフェイスを含む IP コアを使用する場合など、それが無理なこともあります。こういった場合は、アクティブ Low のリセットを最上位レベルの信号に割り当て、その新しい信号をデザインのどこかで使用します。

例 :

```
reset_n <= !reset;
```

すべての場合に reset_n 信号を使用します。信号またはポートに !reset は割り当てないでください。

この結果、LUT はデザイン全体のリセット ネット用にのみ推論されるので、デザインのパフォーマンスにはあまり影響しません。

デカップリング機能

リコンフィギャブル ロジックは FPGA デバイスの動作中に変更できるので、リコンフィギャブル モジュールの出力に接続されるスタティック ロジックは、パーシャル リコンフィギュレーション中のリコンフィギャブル モジュールからのデータを無視する必要があります。リコンフィギャブル モジュールは、パーシャル リコンフィギュレーションが完了してリコンフィギュレーションされたロジックがリセットされるまで、有効なデータを出力しません。この問題を回避するのによく使用されるのは、リコンフィギャブル モジュールからのすべての出力信号 (インターフェイスのスタティック側) にレジスタを付ける方法です。イネーブル信号を使用すると、完全にリコンフィギュレーションされるまでロジックを隔離できます。

スタティック部分には、データおよびインターフェイス管理に必要なロジックが含まれている必要があり、ハンドシェークやインターフェイス ディスエーブル (バス構造で無効なトランザクションを回避するために必要な場合あり) などの機能をインプリメントできます。パーシャル リコンフィギュレーション モジュールのダウンタイム パフォーマンスの影響 (リコンフィギュレーション中またはリコンフィギュレーション後のパーシャル リコンフィギュレーション モジュールに含まれる共通リソースの使用不能状態) について考慮する際にも便利です。

リコンフィギュレーションが終了したら、リコンフィギュレーション済みのロジックのローカル リセットをアサートし、問題のない既知の開始ステートになるようにする必要があります。フル デバイス コンフィギュレーションと異なり、ロジックを初期ステートに強制的に指定する GSR (グロー

バル セット リセット) や GTS (グローバル トライステート) などの専用ファンクションはありません。リコンフィギュレーション フレームの周囲のロジックはリコンフィギュレーション中に動作するので、新規ロジックが使用のために解除されていても、ステートや動作は予測できません。これは汎用ファブリック ロジックと同様、I/O ロジックにも該当します。

デザインのリビジョン チェック

第 6 章「FPGA デバイスのコンフィギュレーション」の説明のように、パーシャル ビットストリームには、プログラム情報とその他の情報が少し含まれます。ビットストリームのターゲット ロケーションは指定する必要はありませんが (ダイ ロケーションはそれがビット ファイルの一部であることを指定すると決定されます)、ハードウェアではこのパーシャル ビットストリームと現在動作中のデザインとの互換性についてはチェックされません。パーシャル ビットストリームをリコンフィギャブル モジュールの異なるリビジョンを使用してインプリメントされなかったスタティック デザインに読み込むと、予測不可能な動作になることがあります。

パーシャル ビットストリームには、特定のデザイン、リビジョンおよびモジュールなどを示す独自の識別子を先頭に付けておくことをお勧めします。識別子はコンフィギュレーション コントローラーで認識され、そのパーシャル ビットストリームと既存デザインとの互換性が確認されます。不一致が検出されると、互換性のないビットストリームはコンフィギュレーション メモリーに読み込まれる前に拒否されます。この機能は必ずデザインに含める必要があり、『PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration』(XAPP887) で説明されるように、暗号解読と CRC チェックと連動します。

リコンフィギャブル パーティション バウンダリの定義

パーシャル リコンフィギュレーションは、フレームごとに実行されます。このため、パーシャル ビット ファイルが作成される場合は、それっぞれ別の数のコンフィギュレーション フレームが使用されます。パーティションの物理領域が定義されると、PlanAhead ソフトウェアでは消費されるリコンフィギャブル領域数と対応するビットストリーム容量の概算がレポートされます。PlanAhead の概算は、2 ～ 3 % の誤差内で計算されます。

パーティション バウンダリは、リコンフィギャブル フレーム バウンダリと揃える必要はありませんが、揃っている場合は最も効率的な配置配線結果になります。スタティック ロジックは次の条件下にある限り、リコンフィギュレーションされるフレーム内に含めることができます。

- Pblock で定義されるエリア グループの外部にある (LOC 制約で内部に強制的に指定されない場合)、および
- ブロック RAM、分散 (LUT) RAM または SRL などのダイナミック エレメントを含まない場合

スタティック ロジックがリコンフィギュレーションされたフレーム内に配置され、そのスタティック ロジックの機能はそのまま記述し直され、グリッチがないことが保証されている場合

T や L 型などの規格外の形のパーティションは、使用できますが、お勧めしません。これは、配線リソースはこれらの領域内に完全に含まれる必要があり、このような形の領域には配置配線しにくいからです。パーティションのバウンダリ同士の接着も可能ですが、お勧めしません。これは、分離することで、潜在的な配線の制限に関する問題を回避できることがあるからです。リコンフィギャブル パーティションのネストや重複 (パーティション内のパーティション) はできません。デザイン ルール チェック ([Tools] → [Run DRC]) を実行すると、パーシャル リコンフィギュレーション プロジェクトのパーティションおよび設定に問題がないかどうか確認できます。

作成されるパーシャル ビット ファイルは、ユーザーの設定した AREA_GROUP RANGE 制約に基づいています。できるだけ小さい容量のビット ファイルを生成するには、複雑な構造やエラーを避け、リコンフィギャブル パーティションのリコンフィギャブル モジュールすべてに存在するエレメントに対してのみ AREA_GROUP RANGE 制約を定義します。PlanAhead を使用する場合は、[Pblock Properties] → [General] で不必要なエレメント タイプをオフにします (67 ページの図 4-18)。

各物理的リコンフィギャブル フレームに含めることのできるリコンフィギャブル パーティションは 1 つだけです。

リコンフィギャブル フレームはリコンフィギュレーション可能な最小サイズの物理領域で、複数のリコンフィギャブル パーティションからのロジックは含めることができません。複数のリコンフィギャブル パーティションからのロジックが含まれると、不正なリコンフィギャブル モジュールからの情報で領域がリコンフィギュレーションされてしまうので、競合が発生します。ソフトウェアは、このような潜在的な問題を回避するように設計されています。

プロキシ ロジック

パーティション ピンはスタティックおよびリコンフィギャブル ロジック間のインターフェイスとして定義されます。この定義に必要な特定のロジックまたはタグはありません。これらはソフトウェアで自動的に処理されます。ほとんどの場合、このノードを表すために、このインターフェイス ポイントに LUT1 が挿入されます。この LUT はスタティック ロジックの階層レベルにあるので、すべてのコンフィギュレーションで同じ論理および物理ロケーションにあります。物理ロケーション自体は接続されるリコンフィギャブル パーティション内にあるので、リコンフィギュレーションではリコンフィギャブル モジュール内部のロジックがこの既知のインターフェイス ポイントに接続されます。

第 3 章の「制約」に示すとおり、プロキシ ロジックには UCF で制約を付けることができます。pr2ucf ユーティリティを使用すると、インプリメントされたコンフィギュレーションからのプロキシ ロジックすべての制約を生成できます。プロキシ ロジックのロケーション制約を指定する必要はありません。このセクションには、個々のおよびグループのパーティション ピンのタイミング制約を設定するための情報も含まれます。

制御された配線

接続には、1 つのデザイン エレメントから別のデザイン エレメントまでの特別な配線リソースを必要とするものがあります。パーティション バウンダリがこの接続にまたがる場合に、プロキシ ロジックがそのパスに挿入されるとエラーが発生します。このような場合、インプリメンテーション ツールにプロキシ ロジックの挿入を実行しないように指示する必要があります。これには、PARTITION_PIN_DIRECT_ROUTE という UCF 制約を該当する配線のパーティション ピンに適用します。次は、そのエラー メッセージと UCF 構文の例です。

```
ERROR:NgdBuild:1319 - Detected a controlled route on Partition Pin
  'aurora_201_i.TX_CLK_OUT'.The connection between GTP_DUAL pin
  'aurora_201_i/aurora_mod_i/GTP_DUAL_INST.TXOUTCLK0' in Partition
  '/fpga_chip_top/aurora_201_i' and
 DCM_ADV pin 'aurora_201_clk_mod_i/clock_divider_i.CLKIN' in Partition
  '/fpga_chip_top' should reside within the same Partition.If this is
  not possible, create the following constraint and run pr_verify to
  validate that this route is the same in each configuration:

 PIN aurora_201_i.TX_CLK_OUT PARTITION_PIN_DIRECT_ROUTE = TRUE;
```

メッセージに記述されるように、制御された完全な配線 (ソース エレメントとデスティネーション エレメントのペア) は、各リコンフィギャブル モジュール内にないと、ワイヤが混線することがあります。このため、制御された配線はブラック ボックスのリコンフィギャブル モジュールには使用できません。

ブラック ボックス

パーシャル リコンフィギュレーション ソフトウェアでは、ブラック ボックスをリコンフィギャブル モジュールとしてインプリメントできます。これは、フル コンフィギュレーション ビット ファイルの容量を削減する効率的な方法で、最初のコンフィギュレーション時間も削減できます。ブラック ボックスのパーティションを作成するには、ネットリスト ファイルを関連付けせずにリコンフィギャブル モジュールを作成します。PlanAhead には、このソースがブラック ボックス モジュールとして表示されます。

ブラック ボックスに論理記述で制約が付けられたユーザー ロジックが含まれなくても、物理領域は完全に空ではありません。上記の「プロキシ ロジック」セクションに示すように、LUT1 は各パーティション ピンごとにインターフェイスとしてリコンフィギャブル パーティションへ挿入されます。これらのプロキシ LUT はリコンフィギャブル領域内にある必要があるので、領域外の接続と共にブラック ボックスに表示されます。

BitGen の圧縮機能 を使用すると (-g compress)、ビット ファイルの容量を削減できることがあります。このオプションは反復するコンフィギュレーション フレーム構造を検索し、ビット ファイルに格納する必要のあるコンフィギュレーション データ量を削減します。これにより、コンフィギュレーション時間も削減されます。この圧縮オプションが配線済みのパーシャル リコンフィギュレーション デザインに使用されると、すべてのビット ファイル (フルおよびパーシャル) が圧縮されたビット ファイルとして作成されます。このオプションは、ブラック ボックス リコンフィギャブル モジュールを含むパーシャル リコンフィギュレーション デザインを構築する際に使用すると特に有効です。

モジュール レベルの制約ファイル

デザイン全体に適切な制約を付けるには、デザインのスタティック部分とリコンフィギャブル部分の両方に制約を設定する必要があります。これには、複数の方法があります。スタティック ロジックは、最上位レベルのネットリストと PlanAhead ソフトウェアまたは Tcl スクリプトで使用されるメインの UCF の制約で制御されます。I/O ロケーション制約のような、リコンフィギャブル パーティションのあらゆるパターンで共有される制約は、最上位レベルの UCF に含める必要があります。

特定のリコンフィギャブル モジュールにのみ適用される制約は、次の 3 つのいずれかの方法で指定される必要があります。

- ネットリストの一部として指定

  合成ツールではデザイン ネットリスト内に制約を含めるので、これらの制約はそのファイルの残りの内容と共に読み込まれます。

- リコンフィギャブル モジュール ネットリストUCF 内で指定

  ネットリストをリコンフィギャブル モジュールとして PlanAhead に読み込む際に、UCF を指定することができます (図 7-2)。この UCF の制約はモジュール レベルを対象とするので、リコンフィギャブル　モジュール内のインスタンスへのリファレンスはインスタンスへのフル階層パスにできません。

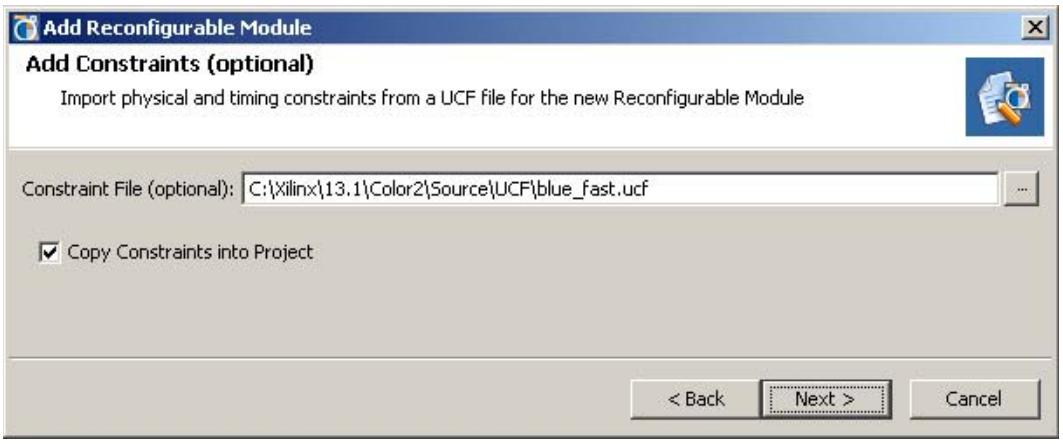

図 7-2 : RM ネットリストと共に指定する UCF ファイル

- ngcbuild を使用してリコンフィギャブル モジュール ネットリストと統合する UCF 内で指定

  NGCBuild をコマンド ラインで実行すると、ネットリストと制約を統合できます。詳細は、105 ページの「デザイン階層」を参照してください。
  この方法は、1 つのネットリストに UCF からの情報を含めるために使用できます。この UCF の制約もモジュール レベルを対象とするので、リコンフィギャブル モジュール内のインスタンスへのリファレンスはインスタンスへのフル階層パスにできません。

インプリメンテーション ストラテジ

FPGA デザインを最適化する際には、トレードオフがあります。パーシャル リコンフィギュレーションも同様です。パーティションは最適化に対する障害であり、リコンフィギャブル フレームには特定のレイアウト制約が必要とされます。リコンフィギャブル デザインを構築するには、余分なコストがかかります。タイミングおよびエリアのニーズに対する追加オーバーヘッドは、デザインによって異なります。影響を最小限に抑えるには、このガイドで説明されるデザインの注意事項に従うようにしてください。

リコンフィギャブル デザインのコンフィギュレーションを構築する際は、インプリメンテーションに選択する最初のコンフィギュレーションを最も困難なコンフィギュレーションにしてください。各リコンフィギャブル パーティションのリコンフィギャブル モジュールで選択した物理領域に適切なリソース (特にブロック RAM、DSP48、I/O などのリソース) が必ず含まれるようにし、各リコンフィギャブル パーティションで一番要求の多いリコンフィギャブル モジュールを選択します。次に続くコンフィギュレーションのリコンフィギャブル モジュールすべてがそれよりも小型か遅い場合、それらの要求が満たしやすくなります。すべてのリコンフィギャブル モジュールのニーズを満たすために、タイミング バジェットを作成する必要があります。

インプリメンテーション中の配置配線問題の回避方法については、第 3 章の「配置配線問題のデバッグ」を参照してください。

シミュレーションと検証

パーシャル リコンフィギュレーション デザインのコンフィギュレーションは、それ自体で完全なデザインです。標準的なシミュレーション、タイミング解析、検証方法のすべてがパーシャル リコンフィギュレーション デザインでサポートされます。パーシャル リコンフィギュレーション自体はシミュレーションできません。

高速トランシーバの使用

ザイリンクスの高速トランシーバ (GT11、GTP、GTX) には、これらのピンの多くに対する専用接続があります。専用接続では、これらのピンに接続される I/O が汎用 I/O とは別の方法で処理される必要があります。ツールで直接接続を認識させるには、トランシーバと関連するすべての I/O ロジックが同じパーティション内になるように制約を設定する必要があります。これには、すべてのパッド、バッファ、トランシーバ ロジックが含まれます。

その他のザイリンクス ツールとの連動

このセクションには、次の内容が含まれます。

ChipScope Pro との連動

ChipScope™ Pro Aalyzer は、ロジック アナライザ、バス アナライザ、仮想 I/O ソフトウェア コアを直接デザインに挿入し、エンベデッド ハード プロセッサおよびソフト プロセッサを含むすべての内部信号またはノードを表示できます。デザインの計装には、次のいずれかを使用します。

- ザイリンクス CORE Generator™ ソフトウェア
- ChipScope Core Inserter

どちらのツールもパーシャル リコンフィギュレーションに対応していますが、それぞれ制限があります。

ザイリンクス CORE Generator ソフトウェアを使用する場合、ネットリスト ベースのコアを作成し、デザインにインスタンシエートします。リコンフィギャブル パーティションのバウンダリが変更されない限り、これらのコアを簡単にインスタンシエートして、該当のデザイン部分をデバッグできます。すべての ChipScope コアがデザインのスタティック部分に配置される場合、これは簡単に管理できます。ICON コアは、BUFG および BSCAN エレメントの両方を含むので、スタティック ロジックに残しておく必要があります。

ILA または VLO コアがリコンフィギャブル パーティションにインスタンシエートされる場合は、別の方法を実行する必要があります。フロアプランの結合領域には、必要なエレメントすべて (必要なファンクションを構築するのに十分なブロック RAM) を含めないと、ChipScope コアがインプリメントできません。また、36 ビット制御バスはスタティック領域の ICON コアからリコンフィギャブル パーティションのデバッグ コアまで接続される必要があります。この要件のサイズおよび物理ロケーションの場合は、リコンフィギャブル パーティションに多大な影響を及ぼす可能性があります。

複数領域 (スタティックおよびリコンフィギャブル) で信号をデバッグする必要がある場合、デバッグはできますが、適切な信号 (データ、トリガ、または制御バス) がリコンフィギャブル パーティションから最上位レベルに送信される必要があります。これには、各リコンフィギャブル モジュールごとにパーティション インターフェイスを修正する必要があります。このストラテジは、CORE Generator フローでのみサポートされます。

ChipScope Core Inserter ソフトウェアでは、HDL ソースではなく、ネットリスト自体でデザインが修正されます。このフローは PlanAhead でサポートされますが、プローブ ポイントはスタティック ロジック内の信号にのみ制限されます。リコンフィギャブル モジュールのロジックにプローブ ポイントを設定すると、パーティション インターフェイスが変更されるので、設定できないことを示すメッセージが表示されます。

System Generator for DSP および CORE Generator との連動

ザイリンクスのアドバンス ツールおよび IP またはサードパーティ ソースを使用する場合も、ChipScope Pro ソフトウェアの場合と同様の規則に従う必要があります。これらのツールはデザインを HDL またはネットリスト レベルで構築および変更するので、パーシャル リコンフィギュレーション フローに必要なボトムアップ合成手法とうまく連動するようになっています。リコンフィギャブル領域の定義 (適切なエレメントが含まれるようにする) とリコンフィギャブル パーティ

ションのタイミングには注意が必要ですが、これらの一般的な要件以外は、どちらのツールもパーシャル リコンフィギュレーションと問題なく連動します。

アドバンス ツールおよび IP を使用したパーシャル リコンフィギュレーションの場合は、これらのデザイン ブロックの内容に注意が必要です。グローバル クロックまたはクロック修正ロジック (BUFG、DCM、PLL、etc) はリコンフィギュレーションされるモジュールには配置できません。

ブロックにリコンフィギャブルではないデザイン エレメントが含まれる場合は、ChipScope ICON コアのように、一部のブロックをスタティック ロジックに残しておく必要があります。

EDK との連動

EDK で開発したプロセッサ デザインのパーシャル リコンフィギュレーション フローについては、『PlanAhead ソフトウェア チュートリアル : プロセッサ ペリフェラルのパーシャル リコンフィギュレーション』(UG744) を参照してください。このチュートリアルは、次のパーシャル リコンフィギュレーションの Web サイトからダウンロードできます。

http://japan.xilinx.com/tools/partial-reconfiguration

EDK とパーシャル リコンフィギュレーションの連動の詳細 :

- PlanAhead プロジェクトを作成し、EDK で開発したデザインの最上位レベル ネットリストを指定する場合、implementation ディレクトリ (../implementation/top_level_filename.ngc) のネットリストではなく、synthesis ディレクトリ (../synthesis/top_level_filename.ngc) の最上位レベル ネットリストを指定します。

  リコンフィギャブル モジュールとして使用するネットリストはすべて、PlanAhead の起動前に EDK の implementation ディレクトリから削除しておく必要があります。削除されたネットリストは EDK の synthesis ディレクトリの最上位レベル ネットリストから呼び出されるので、PlanAhead でこれらのネットリストをブラックボックスとして処理するかどうかの選択ができるようになります。PlanAhead でブラックボックスを作成できるようにしたら、削除されたネットリストと同じポート定義を使用して EDK の外部でネットリストを作成し、これらのネットリストを PlanAhead を使用して新しいリコンフィギャブル モジュールとして追加します。

- EDK プロセッサ システム デザインのビット ファイルを生成する場合は、ビット ファイルで Data2MEM プログラムを実行し、コンパイル済みのソフトウェア プログラムを使用してブロック RAM の内容をアップデートする必要があります。PlanAhead 環境で実行する場合、Data2MEM プログラムを直接呼び出すリンクはありません。ただし、BitGen の -bd オプションを使用すると、BitGen で Data2MEM を直接呼び出すことができます。PlanAhead で [Generate Bitstream] コマンドを実行すると、使用可能な BitGen オプションを含むダイアログ ボックスが表示されます。-bd オプションは、このリストに表示されます。-bd オプションの値フィールドには、EDK で生成した ELF ファイルを参照して指定できます。

  Data2MEM を別に実行しなくても、BitGen コマンド ラインからこのオプションを使用することもできます。次は、そのコマンドの例です。

```
bitgen -bd <path_to_ELF_file>/executable.elf
```

パーシャル リコンフィギュレーションのデザイン チェックリスト

パーシャル リコンフィギュレーションを使用するデザインの場合は、次の項目について考慮してください。

- グローバル クロック バッファまたはクロック修正ブロック (DCM、MMCM、PLL) を使用していますか。
  - グローバル クロック バッファおよびクロック修正ブロックは、スタティック ロジックで使用する必要があります。
    - 詳細は、この章の「リコンフィギャブル モジュール内のデザイン エレメント」を参照してください。
    - グローバル クロック インプリメンテーションの詳細は、「グローバル クロック」を参照してください。
- リージョナル クロック バッファを使用していますか。
  - リージョナル クロック バッファの要件は、アーキテクチャによって異なります。
    - リージョナル クロックの要件については、「グローバル クロック」を参照してください。
- デバイス機能ブロック (BSCAN、CAPTURE、DCIRESET、FRAME_ECC、ICAP、KEY_CLEAR、STARTUP、USR_ACCESS) を使用していますか。
  - デバイス機能ブロックは、スタティック ロジックで使用することをお勧めします。
    - 詳細は、この章の「リコンフィギャブル モジュール内のデザイン エレメント」を参照してください。
- 一緒にパックされるすべてのロジックは同じリコンフィギャブル パーティションにある必要がありますか。
  - 一緒にパックされるロジックはすべて同じリコンフィギャブル パーティション/リコンフィギャブル モジュールにある必要があります。
    - 詳細は、この章の「ロジックのパック」を参照してください。
- クリティカル パスが同じパーティションに含まれますか。
  - リコンフィギャブル パーティションの境界により、最適化およびパッキングが制限されることがあるので、クリティカル パスは同じパーティション内に含める必要があります。
    - 詳細は、この章の「ロジックのパック」を参照してください。
- リコンフィギャブル モジュールに I/O が含まれますか。
  - リコンフィギャブル モジュールの I/O の要件はさまざまです。
    - 詳細は、この章の「リコンフィギャブル モジュールの I/O」を参照してください。
- リコンフィギャブル モジュールの出力にデカップリング ロジックを作成しましたか。
  - リコンフィギュレーション中は、リコンフィギャブル パーティションの出力は中間ステートになるので、データ破損にならないようにデカップリング ロジックを使用する必要があります。
    - 詳細は、この章の「デカップリング機能」を参照してください。
- デザインに高速トランシーバが含まれますか。
  - 高速トランシーバには、特定の要件があります。
    - この要件については、この章の「高速トランシーバの使用」を参照してください。

- パーシャル リコンフィギュレーション デザインに ChipScope Pro Analyzer を使用していますか。
  - ChipScope Pro はパーシャル リコンフィギュレーションに使用できますが、一部の要件を満たしている必要があります。
    - 詳細は、この章の「ChipScope Pro との連動」を参照してください。
- パーシャル リコンフィギュレーション デザインに System Generator for DSP または CORE Generator を使用していますか。
  - System Generator と CORE Generator はパーシャル リコンフィギュレーションに使用できますが、一部の要件を満たしている必要があります。
    - 詳細は、この章の「System Generator for DSP および CORE Generator との連動」を参照してください。
- パーシャル リコンフィギュレーション デザインに EDK を使用していますか。
  - EDK はパーシャル リコンフィギュレーションに使用できますが、一部の要件を満たしている必要があります。
    - 詳細は、この章の「EDK との連動」を参照してください。
    - 詳細は、『PlanAhead ソフトウェア チュートリアル : プロセッサ ペリフェラルのパーシャル リコンフィギュレーション』(UG744) を参照してください。
      このチュートリアルは、次のパーシャル リコンフィギュレーションの Web サイトからダウンロードできます。http://japan.xilinx.com/tools/partial-reconfiguration.htm
- 暗号化されたパーシャル ビット ファイルを Virtex-4 および Virtex-5 デバイスで使用する必要がありますか。
  - 暗号化されたパーシャル ビット ファイルは、Virtex-4 および Virtex-5 デバイスではサポートされません。
    - 詳細は、付録 A の「既知の制限」を参照してください。
    - Virtex-5 用に暗号化されたパーシャル ビット ファイルを構築する方法については、ザイリンクス アプリケーション ノート 『PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration』(XAPP887) を参照してください。
- ブロック RAM の内容をアップデートする必要がありますか。
  - パーシャル ビットストリームでは Data2MEM はサポートされません。
    - 詳細は、付録 A の「既知の制限」を参照してください。
- リコンフィギャブル パーティションすべてにザイリンクス ガイドラインに従ったエリア グループが設定されていますか。
  - リコンフィギャブル パーティションのエリア グループには要件が複数あります。
    - 詳細は、第 3 章の「エリア グループ制約」を参照してください。
- リコンフィギャブル パーティションのエリア グループを効率的な方法で作成しましたか。
  - パーシャル リコンフィギュレーションはフレーム ベースで実行されるので、推奨される作成方法を使用してください。
    - 詳細は、この章の「リコンフィギャブル パーティション バウンダリの定義」を参照してください。
- すべてのコンフィギュレーションで pr_verify を実行しましたか。
  - pr_verify は、すべてのコンフィギュレーションに一致するインポート済みリソースが含まれていることを確認するために使用されます。

- 詳細は、第 3 章の「pr_verify」を参照してください。

- デバイス特有のコンフィギュレーション要件を認識していますか。
  - デバイス ファミリにはそれぞれ独自のコンフィギュレーション要件があります。
    - 第 6 章「FPGA デバイスのコンフィギュレーション」を参照してください。
    - 詳細は、該当するデバイス ファミリのコンフィギュレーション ユーザー ガイドを参照してください。

    http://japan.xilinx.com/support/documentation/index.htm.

付録 A

既知の問題と制限

この付録では、13.1 のパーシャル リコンフィギュレーション ソフトウェアの既知の問題および制限についてリストします。

既知の問題

パーシャル リコンフィギュレーションの既知の問題のリストについては、アンサー 35019 を参照してください。

既知の問題には、次のようなものがあります。

- Tcl スクリプトのスペルミスがレポートされない

  Color2 デザインに含まれるサンプルの Tcl スクリプトを使用する場合、インスタンス名 (コンフィギュレーション、リコンフィギャブル モジュールおよびパスなどの名前) はユーザー デザインを反映するように変更しておく必要があります。名前のスペルが間違っていても、間違っていることを示すエラー メッセージは表示されませんので、合成およびインプリメンテーション中にファイル名や設定が間違っていないかどうか、レポート ファイルを確認してください。

既知の制限

既知の制限には、次のようなものがあります。

- パーシャル リコンフィギュレーション ソフトウェアでは Spartan® デバイス ファミリはサポートされません。
- PlanAhead では、リコンフィギャブル パーティション内でエリア グループ サブモジュールは使用できません。
- 暗号化されたパーシャル ビット ファイル (bitgen -g encrypt で生成) は、Virtex-4 および Virtex-5 デバイスでは直接サポートされません。Virtex-5 用に暗号化されたパーシャル ビット ファイルを構築する方法については、ザイリンクス アプリケーション ノート 『PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration』(XAPP887) を参照してください。

  暗号化されたパーシャル ビット ファイルは、Virtex-6 デバイスではサポートされます。この場合、各コンフィギュレーションに対して同じ NKY ファイルを指定して、暗号キーの値が同じになるようにする必要があります。

  8 ビット バスの場合にのみ、暗号化された Virtex-6 のパーシャル ビット ファイルのパーシャル リコンフィギュレーションには ICAP を使用する必要があります。暗号化が使用される場合は、外部コンフィギュレーション ポートを介してリコンフィギュレーションはできません。
- PlanAhead では、Data2MEM プログラムを直接実行して、ブロック RAM の内容 (EDK プロセッサ システムなど) をアップデートすることはできません。ただし、BitGen コマンドを -bd

オプションで実行すると、ビットストリーム生成の一部として Data2MEM を実行できます。詳細については、第 7 章の「EDK との連動」を参照してください。

- 双方向パーティション ピンはサポートされません。スタティックおよびリコンフィギャブル ロジック間のインターフェイスには一方向ピンのみを使用する必要があります。

XILINX®

付録 B

パーシャル リコンフィギュレーション アップグレード ガイド

この付録では、9.2.04i モジュール デザイン早期アクセス (EA) パーシャル リコンフィギュレーションで作成したデザインをパーティション ベースの ISE® 13 にアップグレードする方法について、手順別に説明しています。

ISE 13 のパーシャル リコンフィギュレーション デザイン フローは基本的に ISE 12 のデザイン フローと同じで、アップグレード方法も同じです。

早期アクセス版と製品版の違い

アップグレード可能なデザイン

Virtex®-4 およびそれ以降のデバイスをターゲットにする早期アクセス (EA) デザインは、ISE 13 用にアップグレードできます。この場合、ISE 13 で PlanAhead™ プロジェクトを新規に作成する必要があります。このプロジェクトを作成するには、単純に第 4 章「PlanAhead サポート」の方法に従ってください。

バス マクロのインスタンシエーションの必要なし

バス マクロ (BM) は必要なくなりました。パーティション ピンは自動的に管理されるようになっています。この自動化に伴い、バス マクロの機能の一部が変更されています。同期および非同期のバス マクロ両方が早期アクセスで使用できます。バウンダリにレジスタを付けて最適な階層デザイン手法に従い、リコンフィギャブル ロジックをデカップリングするためには、HDL にレジスタを追加して、同期バス マクロに含まれている出力レジスタの機能を書き換えることができます。

必ずパーティション バウンダリにレジスタを付け、これらのレジスタと共にイネーブルを使用するようにしてください。リコンフィギュレーション中のこれらの領域の動作は不明なので、リコンフィギュレーション中にロジックの出力が使用されると、デザインが破損してしまうこともあります。このため、イネーブルを使用してバウンダリにレジスタを付けて、リコンフィギュレーション中はリコンフィギャブル領域をディスエーブルしておく必要があります。

パーシャル リコンフィギュレーション特有の環境変数の廃止

早期アクセスの場合、さまざまな環境変数を設定する必要がありましたが、ISE 12 からはその必要はなくなっています。早期アクセス用に設定した環境変数はすべて設定を解除しておいてください。

MODE 制約の廃止

早期アクセスの場合、どのエリア グループがリコンフィギャブルかをツールで指定する必要がありました。これは、UCF に追加される専用の制約 (MODE=RECONFIG) で処理されていましたが、これらの制約は必要なくなりました。この機能は、PlanAhead の [Set Reconfigurable] オプションを使用すると指定できるようになり、この結果、xpartition.pxml に Reconfigurable=TRUE という情報が追加されるようになりました。

NGDBuild -modular オプションの廃止

パーシャル リコンフィギュレーション デザインを実行していることを NGDBuild に伝える必要はなくなっています。これは、xpartition.pxml ファイルで処理されるようになりました。詳細は、次のセクションを参照してください。

パーティション情報の xpartition.pxml ファイルへの保存

ISE 13 では、PXML ファイルでパーティション特有の情報が管理されます。このファイルは xpartition.pxml という名前で、この名前は変更できません。このファイルは ASCII XML 形式で、インプリメンテーションごとに作成されます。ほとんどのパーシャル リコンフィギュレーション特有の情報 (AREA GROUP RANGE 制約用に保存されるすべての情報) は、xpartition.pxml ファイルに含まれます。ツールでは自動的に xpartition.pxml ファイルがチェックされます。リコンフィギャブル パーティションを含むデザインには、xpartition.pxml ファイルが必ず必要で、少なくとも 1 つのパーティションが定義されている必要があります。このファイルがない場合、デザインは「フラット」なデザインとして処理されます。

xpartition.pxml ファイルは PlanAhead で生成され、ユーザーは変更できません。デザインをインプリメントするのにザイリンクス HD Tcl スクリプトを使用する場合は、インプリメンテーション スクリプトが実行されたときにこのファイルが作成されます。

コマンド ライン オプション は Tcl フローのみ

早期アクセスの場合、ツールはコマンド ラインから直接実行できました。ツールは 13.1 でもコマンド ラインから実行できますが、ISE 13 ツールがデザインをパーシャル リコンフィギュレーション デザインとして処理する前に、PMXL ファイルが必要になる点が違います。このため、フローを Tcl でスクリプト記述して、PMXL ファイルを生成する必要があります。

メモ : ザイリンクス HD Tcl スクリプトを使用する場合、まず run を作成するときに [Generate Scripts Only] オプションを使用して基本的な「フラット フロー」のスクリプトを複数生成しておくことができます。リコンフィギャブル パーティションのプロモート、インプリメント、インポートなどの機能を実行するザイリンクス HD Tcl スクリプトを記述する場合は、第 5 章「コマンド ライン スクリプト」を参照してください。

UCF は NGDBuild でのみ必要

早期アクセスの場合、変換 (Translate) 後インプリメンテーション プロセス (MAP およびプロセス) に UCF が必要でしたが、ダウンストリーム インプリメンテーション プロセスに必要な情報はすべてデザインのデータベース ファイルに含まれるようになったので、UCF は必要なくなりました。

完全デザインのタイミング制約の管理

ISE 13 では完全なデザインをインプリメントするので、タイミング制約とタイミング バジェットを作成する必要があります。タイミング管理については、第 3 章「ソフトウェア ツール フロー」を参照してください。

BUFR にパーティション ピンが必要 (Virtex-5)

早期アクセスの場合、BUFR には制限が複数ありましたが、ネットワークはバス マクロを必要としませんでした。ISE 13 では、クロック領域の前配線要件を満たすためにパーティション ピンが BUFR ネットワークに追加されています。これは、Virtex-5 にのみ該当します。

デザインのアップグレード

早期アクセス デザインは、ISE 13 用に簡単にアップグレードできます。まず、HDL でバス マクロを削除または置き換え、該当するネットリストを再生成 (再合成) します。ネットリストが正しく設定されたら、ISE 13 で新しい PlanAhead プロジェクトを作成します。9.2.04i PlanAhead のプロジェクトを 13.1 PlanAhead に直接アップグレードはしないでください。

バス マクロの削除

デザインをアップグレードするには、まず HDL でバス マクロを削除します。バス マクロを削除する方法は 2 つあります。

- バス マクロのインスタンシエーションの削除
  - 利点 :HDL をきれいなままで残せる
  - 不利点 :時間がかかりすぎる。また、すべてのインスタンスに対して実行する必要がある
- バス マクロの再定義
  - 利点 :大量のバス マクロを置換する最も速い方法
  - 不利点 :バス マクロのインスタンシエーションがデザイン中に残ったままになる

バス マクロを削除できず、バス マクロの NMC ファイルを削除すると、変換 (NGDBuild) プロセスで次のようなエラー メッセージが表示されます。

```
ERROR:NgdBuild:604 - logical block 'my_RP/my_BM_GENERATE[7].my_BM'
with type 'busmacro_xc5v_async_enable' could not be resolved.A pin
name misspelling can cause this, a missing edif or ngc file, case
mismatch between the block name and the edif or ngc file name, or the
misspelling of a type name.Symbol 'busmacro_xc5v_async_enable' is
not supported in target 'virtex5'.
```

VHDL バス マクロの削除

バス マクロのインスタンシエーションのみの削除

次の例では、非同期バス マクロが使用されています。この例では、バス マクロの削除を簡単に説明するため、バス マクロの入力を直接バス マクロの出力に接続していますが、これは実際には必要なく、1 つのネットワークでバス マクロ入力とバス マクロ出力が置換できます。逆に、複数のバス マクロには関連する制御ロジックが含まれるので、これらのバス マクロ タイプが保存されるには、入力信号と出力信号の両方を必要です。これは、制御ロジックが 2 つの信号のインターフェイスになるからです。

バス マクロを再定義する方法については、後述します。

手順 1 : すべてのバス マクロからコンポーネント宣言を削除します。

例 – 削除するVHDL バス マクロ宣言

```
component busmacro_xc5v_async is
  port (
    input0  : in  std_logic;
    input1  : in  std_logic;
    input2  : in  std_logic;
    input3  : in  std_logic;
    output0 : out std_logic;
    output1 : out std_logic;
    output2 : out std_logic;
    output3 : out std_logic
  );
end component;
```

手順 2 :バス マクロのインスタンシエーションを 1:1 の信号マップ代入文に置き換えます。

例 – 古い VHDL バス マクロのインスタンシエーション

```
Control1_0_BM : busmacro_xc5v_async
  port map (
    input0  => MY_ADDR_SPACE,
    input1  => PLB_SAValid,
    input2  => PLB_rdPrim,
    input3  => PLB_wrPrim,
    output0 => MY_ADDR_SPACE_pr,
    output1 => PLB_SAValid_pr,
    output2 => PLB_rdPrim_pr,
    output3 => PLB_wrPrim_pr
    );
```

例 – バス マクロの新しい VHDL 置換文 (1:1 の代入文)

```
MY_ADDR_SPACE_pr <= MY_ADDR_SPACE;
PLB_SAValid_pr <= PLB_SAValid;
PLB_rdPrim_pr <= PLB_rdPrim;
PLB_wrPrim_pr <= PLB_wrPrim;
```

これは簡潔な非同期バス マクロですが、バス マクロの置換方法をわかりやすく示しています。バス マクロには、制御ロジック付きのバス マクロおよび同期バス マクロもあります。これらのバス マクロは、レジスタ推論と必要であれば制御ロジック (イネーブル、クロック イネーブルなど) と置換する必要があります。次は、制御ロジック付きの非同期バス マクロの例です。

例 – 古い VHDL バス マクロのインスタンシエーション (イネーブル付き)

```
Control2_0_BM : busmacro_xc5v_async_enable
  port map (
    input0  => Sl_addrAck_pr,
    input1  => Sl_SSize_pr(0),
    input2  => Sl_SSize_pr(1),
    input3  => Sl_wait_pr,
    enable0 => busmacro_enable,
    enable1 => busmacro_enable,
    enable2 => busmacro_enable,
    enable3 => busmacro_enable,
    output0 => Sl_addrAck,
    output1 => Sl_SSize(0),
```

```
      output2 => Sl_SSize(1),
      output3 => Sl_wait
      );
```

例 – バス マクロの新しい VHDL 置換文 (イネーブル付き)

```
  Sl_addrAck <= Sl_addrAck_pr and busmacro_enable;
  Sl_SSize(0) <= Sl_SSize_pr(0) and busmacro_enable;
  Sl_SSize(1) <= Sl_SSize_pr(1) and busmacro_enable;
  Sl_wait <= Sl_wait_pr and busmacro_enable;
```

バス マクロの再定義

バス マクロは、古いバス マクロと同じ名前で新規に作成したネットリストと置換できます。この方法は、同期バス マクロに推奨されます。これは、同期バス マクロがロジックのデカップリングに直接使用できるからです。ロジック デザインは同じになるので、パーシャル リコンフィギュレーションを再び有効にするタスクはかなり単純化されています。

HDL からのバス マクロと同じインターフェイスでネットリストを作成し、内部代入を必要なだけ定義します。合成中は、I/O バッファの挿入がディスエーブルになるようにしてください (たとえば XST の場合、このオプションは [Add I/O Buffers] (-iobuf) です)。

メモ : これらのロジック モジュールは、置換されたバス マクロがリコンフィギャブル パーティションの入力であろうが出力であろうが、スタティック ロジック内になります。

例 – busmacro_xc5v_async 用に再定義された VHDL バス マクロ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity busmacro_xc5v_async is
    Port ( input0  : in   STD_LOGIC;
           input1  : in   STD_LOGIC;
           input2  : in   STD_LOGIC;
           input2  : in   STD_LOGIC;
           output0 : out  STD_LOGIC;
           output1 : out  STD_LOGIC;
           output2 : out  STD_LOGIC;
           output3 : out  STD_LOGIC);
end busmacro_xc5v_async;
architecture Behavioral of busmacro_xc5v_async is
begin
  output0 <= input0;
  output1 <= input1;
  output2 <= input2;
  output3 <= input3;
end Behavioral;
```

この方法は、一見大変なようですが、デザイン全体にバス マクロが何百も使われているような場合、インスタンスごとに変更を加える必要がないので、これが最も簡単で速い方法になります。これらのバス マクロを再定義し始めると、モジュールの問題が修正できるため、変更がそのタイプのバス マクロすべてと矛盾しなくなります。次は、制御ロジック付きの非同期バス マクロの例です。

例 – busmacro_xc5v_async_enable 用に再定義されたイネーブル付き VHDL バス マクロ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity busmacro_xc5v_async_enable is
    Port ( input0  : in   STD_LOGIC;
           input1  : in   STD_LOGIC;
           input2  : in   STD_LOGIC;
```

```
                input2  : in   STD_LOGIC;
                input2  : in   STD_LOGIC;
                enable1 : in   STD_LOGIC;
                enable2 : in   STD_LOGIC;
                enable3 : in   STD_LOGIC;
                output0 : out  STD_LOGIC;
                output1 : out  STD_LOGIC;
                output2 : out  STD_LOGIC;
                output3 : out  STD_LOGIC);
end busmacro_xc5v_async_enable;
architecture Behavioral of busmacro_xc5v_async_enable is
begin
  output0 <= input0 and enable0;
  output1 <= input1 and enable1;
  output2 <= input2 and enable2;
  output3 <= input3 and enable3;
end Behavioral;
```

Verilog バス マクロの削除

Verilog の場合も VHDL と同じフローになりますが、Verilog フローにはモジュール宣言が含まれない点が異なります。VHDL のフローに従います。ただし、使用するのは Verilog 構文です。

PlanAhead プロジェクトを 13.1 で作成します。

このプロジェクトを作成するには、第 4 章の「パーシャル リコンフィギュレーション プロジェクトの作成」の手順に従ってください。

バス マクロの再定義プロセスを使用する場合、PlanAhead でプロジェクトを作成する際に、バス マクロ置換ネットリストをスタティック ロジック ソースとして含める必要があります。

まとめ

モジュール デザイン早期アクセス パーシャル リコンフィギュレーション ツールを使用して作成およびインプリメントしたデザインは、簡単にパーティション ベースの ISE 13 用デザインに変換できます。バス マクロは削除または置換する必要があり、ロジックのデカップリングを考慮する必要があります。また、モジュール デザイン特有のオプションは削除できます。これにより、時間をかけずに、最新のパーシャル リコンフィギュレーション ソフトウェアで、デザインをインプリメントできるようになっています。

XILINX

付録 C

その他のリソース

追加資料は、次のザイリンクス Web サイトを参照してください。

http://japan.xilinx.com/literature

シリコン、ソフトウェア、IP に関する問題をアンサー データベースで検索したり、テクニカル サポートのウェブ ケースを開くには、次のザイリンクス Web サイトにアクセスしてください。

http://japan.xilinx.com/support

パーシャル リコンフィギュレーション デザインの構築に関するその他の情報は、次を参照してください。

- 『コマンド ライン ツール ユーザー ガイド』(UG628) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/devref.pdf
- 『制約ガイド』(UG625) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/cgd.pdf
- 『Differencing Method for Partial Reconfiguration』(XAPP290) : http://japan.xilinx.com/support/documentation/application_notes/xapp290.pdf
- 『Fast Configuration of PCI Express Technology through Partial Reconfiguration』(XAPP883) *:* http://japan.xilinx.com/support/documentation/application_notes/xapp883_Fast_Config_PCIe.pdf
- 『階層デザイン手法ガイド』*(UG748)* : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/Hierarchical_Design_Methodolgy_Guide.pdf
- 『Partial Reconfiguration of Virtex FPGAs in ISE 12』(WP374) : http://japan.xilinx.com/support/documentation/white_papers/wp374_Partial_Reconfig_Virtex_FPGAs.pdf
- 『PlanAhead ソフトウェアチュートリアル : プロセッサペリフェラルのパーシャルリコンフィギュレーション』(UG744) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_Tutorial_Reconfigurable_Peripheral.pdf
- 『PlanAhead ユーザー ガイド』(UG632) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_UserGuide.pdf
- 『PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration』(XAPP887) : http://japan.xilinx.com/support/documentation/application_notes/xapp887_PRC_EPRC.pdf
- 『PlanAhead ソフトウェアチュートリアル : パーシャルリコンフィギュレーション フローの概要』(UG743) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_Tutorial_Partial_Reconfiguration.pdf

- 『Repeatable Results with Design Preservation』(WP362) *:*
  http://japan.xilinx.com/support/documentation/white_papers/wp362.pdf
- 『Virtex-6 FPGA コンフィギュレーション ユーザー ガイド』(UG360) :
  http://japan.xilinx.com/support/documentation/user_guides.htm
- 『Virtex-5 FPGA コンフィギュレーション ユーザー ガイド』(UG191) :
  http://japan.xilinx.com/support/documentation/user_guides.htm
- 『Virtex-4 FPGA コンフィギュレーション ユーザー ガイド』(UG071) :
  http://japan.xilinx.com/support/documentation/user_guides.htm
- 『XST ユーザーガイド (Virtex-4、Virtex-5、Spartan-3 および CPLD デバイス用)』(UG627) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/xst.pdf
- 『XST ユーザーガイド (VVirtex-6、Spartan-6 および 7 シリーズ デバイス用)』(UG687) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/xst_v6s6.pdf