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 (v13.3) 2011 年 10 月 19 日

XILINX®

本資料は英語版 (v 13.3) 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。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com までお知らせください。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。

改訂履歴

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	改訂内容
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
2011 年 7 月 6 日	13.2	ISE 13.2 リリース用にアップデート： • 7 シリーズのパーシャル リコンフィギュレーション サポートに関する記述と、パーシャル リコンフィギュレーションが 7 シリーズ デバイスにどのように適用されるかの情報を追加 • パーシャル リコンフィギュレーション フローにおける BitGen の `-g USR_ACCESS` オプションの使用について記述 • スタックド シリコン インターコネクト (SSI) デバイスでパーシャル リコンフィギュレーション フローに適用される規則について記述 • リージョナル クロックの記述を 7 シリーズ デバイスに適用するようアップデート • パーシャル リコンフィギュレーションで使用可能なコンフィギュレーション モードに BPI および SPI モードを追加 (7 シリーズ デバイスのみ) • 13.2 リリース用に既知の問題と既知の制限をアップデート • 付録 C 「その他のリソース」の資料の順序を変更
2011 年 10 月 19 日	13.3	ISE 13.3 リリース用にアップデート： • 第 4 章「PlanAhead サポート」で 13.3 PlanAhead デザイン フローの手順を修正 • 第 4 章「PlanAhead サポート」のダイアログ ボックスの図を 13.3 リリース用にアップデート • 第 6 章の最大バンド幅の表 (表 6-1) に BPI および SPI モードを追加 • 第 1 章の「デザイン要件とガイドライン」にリコンフィギュレーション ロジックに対してローカル リセットを送信する必要があることを追記 • コマンド ライン ユーザーが前にプロモートしたコンフィギュレーションに関するデータを保存する方法を追加。詳細は、第 3 章の「パーティションとインポート」を参照

目次

第 6 章 : FPGA デバイスのコンフィギュレーション

第 7 章 : デザインの注意事項

付録 A : 既知の問題と制限

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

付録 C : その他のリソース

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 デザインに変更を加えることです。このガイドでは、「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 BIT ファイルに変換されます。『差分ベースのパーシャル リコンフィギュレーション』(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、このガイドでは説明しません。その他のマニュアルは、付録 C「その他のリソース」を参照してください。

このガイドの対象は、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
- 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
- ISE Design Suite 13.3 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、Kintex™-7、Virtex-7、および Virtex-7 485XT 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 BIT ファイル) を読み込むことで、動作中の FPGA デザインを変更できます。フル BIT 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 BIT ファイルをダウンロードすると、デバイスのリコンフィギュレーションされない部分で実行されているアプリケーションに影響を与えることなく、FPGA のリコンフィギャラ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。

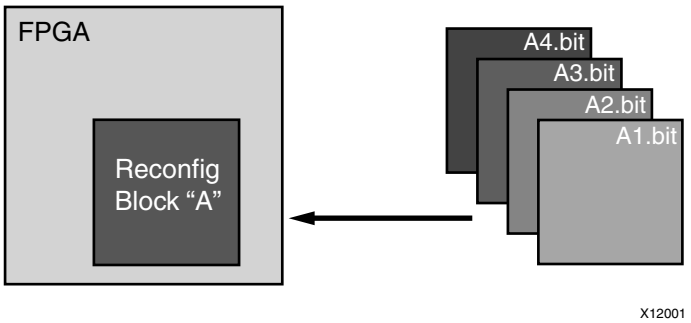

図 1-1 : パーシャル リコンフィギュレーションの基本的概念

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` のいずれかのパーシャル BIT ファイルをダウンロードすると変更できます。FPGA デザイン ロジックは、リコンフィギャラブル 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ラブル ロジックを表しています。パーシャル BIT 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。リコンフィギャラブル ロジックはパーシャル BIT 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、さまざまな理由から有益です。

次のような利点があります。

- 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でき、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を柔軟に選択できます。
- デザインを保護したまま新技術をイネーブルにできます。
- FPGA のフォールト トレランスを改善します。
- コンフィギャラブルな計算を促進します。

サイズ、重さ、消費電力、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を使用せずにインプリメントすることができない新しいタイプの FPGA デザインも可能にします。

用語

次に、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特有の用語を示します。

ボトムアップ合成

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。モジュールの境界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。最上位ロジックは、パーティションにブラックボックスを使用して合成する必要があります。

コンフィギュレーション

リコンフィギャラブル パーティションごとに 1 つのリコンフィギャラブル 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 BIT ファイルを生成するほか、リコンフィギャラブル モジュールごとに 1 つのパーシャル BIT 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

FPGA コンフィギュレーション メモリ空間の中でアドレス指定可能な最小のセグメントです。リコンフィギャラブル フレームは、これらの最下位エレメントから構築されます。

フレーム

フレーム (このガイドでは「コンフィギュレーション フレーム」を除く) は、FPGA デバイス内の最小のリコンフィギャラブル領域を表します。リコンフィギャラブル 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)

基本的には 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 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の一部を変更することを示します。

パーティション

階層の境界でユーザーにより定義されたデザインの論理セクションで、デザインの再利用に使用されます。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

スタティック ロジックとリコンフィギャラブル ロジック間の論理および物理接続です。すべてのリコンフィギャラブル 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

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ラブル 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。

リコンフィギャラブル ロジック

リコンフィギャラブル モジュールの一部である論理エレメントで、パーシャル BIT 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をリコンフィギュレーションできます。

リコンフィギャラブル モジュール (RM)

リコンフィギャラブル 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ラブル パーティションに対して、複数のリコンフィギャラブル モジュールがあることもあります。

リコンフィギャラブル パーティション (RP)

インスタンスをリコンフィギュレーション可能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ラブル 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ラブル パーティションの場合、「リコンフィギャラブル パーティション」という用語は「インスタンス」と同じ意味で使用されることがあります。

スタティック ロジック

リコンフィギャラブル 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、リコンフィギャラブル パーティションがリコンフィギュレーションされているときも、常にアクティブです。スタティック ロジックは、「最上位ロジック」とも呼ばれます。

ISE 13.3 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、パーシャル リコンフィギュレーション プロジェクトを開始する前に、まず次の要件を理解しておく必要があります。

各項目の詳細は、このユーザー ガイドの後のセクションで説明します。

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、Kintex-7、Virtex-7、および Virtex-7 485XT
  - Virtex-4、Virtex-5、Virtex-6 デバイスはすべてサポートされています。
  - Artix™-7、Virtex-7 XT、Virtex-7 HT、および Zynq™-7000 デバイスは、ISE ソフトウェアの今後のリリース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は、PlanAhead™ およびコマンド 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ラ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/クロック領域の境界にアライメントしておくと、効率的です。

- ボトムアップ合成 (複数のネットリスト ファイルを作成) とリコンフィギャラブル 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ラ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、デカップリング ロジックを使用してください。
  - リコンフィギャラブル エレメントが FPGA 外にある場合、デカップリングはオフチップで実行する必要があります。
- グローバル セット/リセット (GSR) コマンドはパーシャル リコンフィギュレーションの後には実行されないので、リコンフィギュレーションされるロジックに対してローカル リセットを実行して、問題のない既知の状態で開始されるようにしておく必要があります。
- 標準的なタイミング制約がサポートされるほか、必要に応じて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。
- パーシャル リコンフィギュレーション 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 BIT ファイルを FPGA 内で配布するか、システム デザインの一部として配布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 オプションをパーシャル リコンフィギュレーション 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。
- **-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を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化する必要のある高度なクロック ゲーティング機能が使用されるの、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。
- リコンフィギャラブル パーティションには、パーティションにインプリメントされるさまざまなリコンフィギャラブル モジュールで使用されるすべてのピンが含まれている必要があります。このため、モジュールによっては一部の入力または出力が未使用になりますが、これは予測されることであり、パーシャル リコンフィギュレーションの柔軟性に組み込まれています。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 ツールでそれ示すメッセージが表示されますが、これらのメッセージは無視しても問題ありません。未使用の出力は 1 に接続されます。リコンフィギャラブル 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ピンが含まれることがあるので、パーシャル リコンフィギュレーション フローでは、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『再現可能な結果を活用したデザインの保持』(WP362) 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ます。

  通常、次のようになります。

  - クロック周波数が 10% 低下
  - パック密度が 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、パーシャル リコンフィギュレーション デザインを処理するために実行時間が長くなる場合があります
- リコンフィギャラ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変更ロジックは、スタティック領域に含める必要があります。
    - 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、デザインのスタティック領域に含める必要があります。
- リコンフィギャブル パーティションへのグローバル クロック リソースには、デバイスとこれらのリコンフィギャブル パーティションに占有されるクロック領域によって、制限があります。詳細は、第 7 章「デザインの注意事項」を参照してください。
- 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 の制限が適用されることがあります。具体的な例は、次のとおりです。
  - ChipScope ICON (BUFG)
  - グローバル バッファーの付いた EDK ブロック
  - MIG コントローラー (MMCM)
- スタティック領域とリコンフィギャラブル領域の間には、専用配線がある場合以外は、双方向インターフェイスを使用することはできません。たとえば、スタティック ロジックである最上位 I/O パッドに配線されているリコンフィギャラブル領域の IOBUF などの I/O バッファーは、双方向インターフェイスを介してリコンフィギャラブル領域とスタティック ロジックをまたぐことがあります。
- 専用の暗号化は、7 シリーズおよび 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 BIT ファイルを変更でき、FPGA 内のハードウェア暗号化エンジンで暗号化のニーズに対応できます。
- 7 シリーズおよび 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最後に専用の CRC 機能はありますが、パーシャル BIT ファイルの整合性は、BIT ファイル配布機能の一部として挿入される IP コアを使用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 : パーシャル リコンフィギュレーションのデータ インテグリティおよびセキュリティ コントローラー』(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 サービスにより、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、マイクロプロセッサの機能と同様に、FPGA ハードウェア リソースを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、さまざまな例を使用して、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、消費電力、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。時間に依存しないファンクションをリコンフィギャラブル モジュールとして識別、分離、インプリメントし、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。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するのを避けるには、図 2-1 に示すように、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。

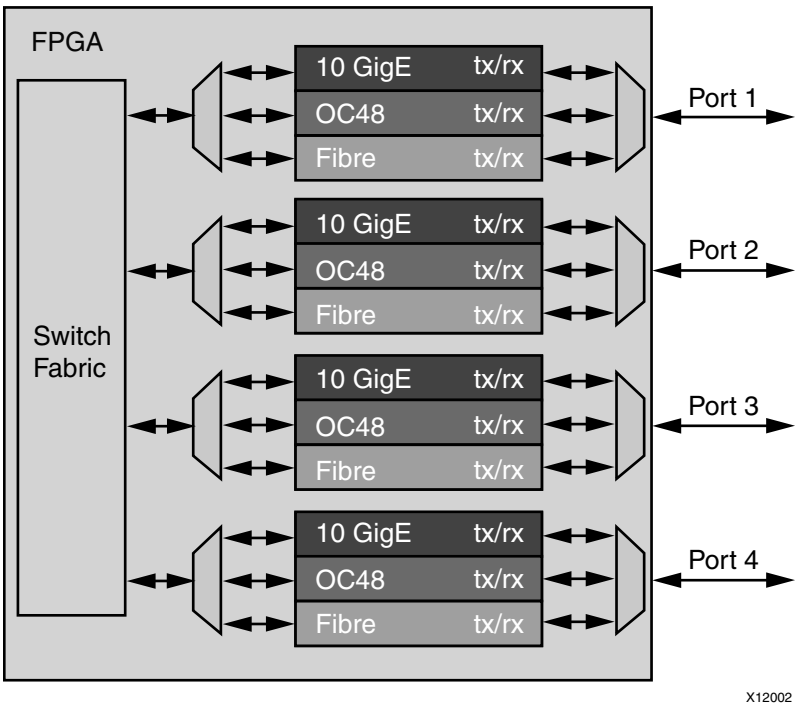

図 2-1 : パーシャル リコンフィギュレーションを使用しない場合のネットワーク スイッチ

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。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ラブル 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。

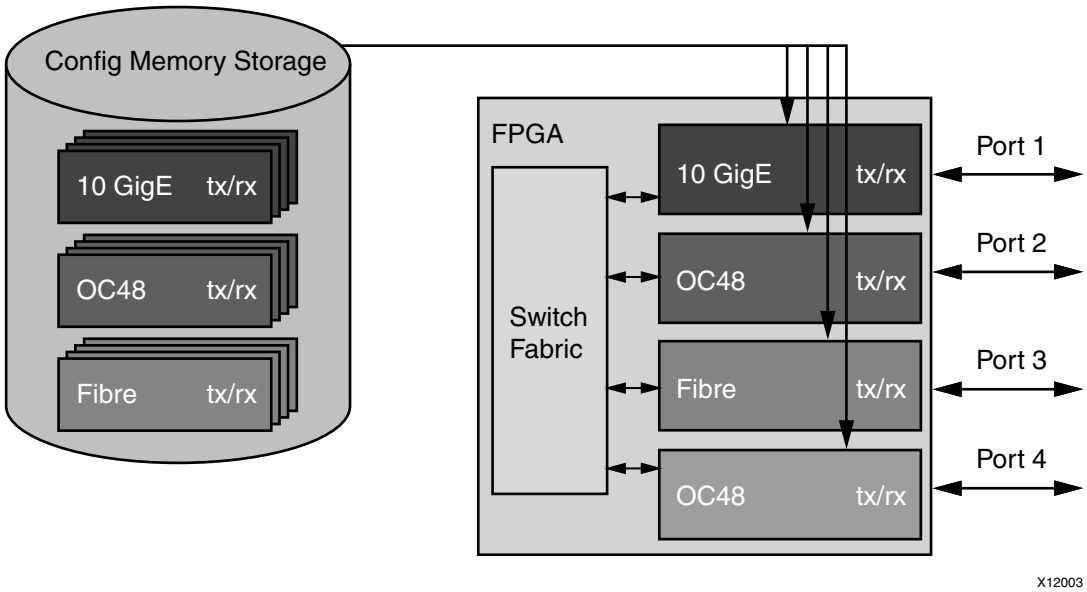

図 2-2 : パーシャル リコンフィギュレーションを使用した場合のネットワーク スイッチ

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 SDR (Software Defined Radio) で機能を多重分割すると、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、図 2-2 に示すように、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) に影響を与えることなく、新規プロトコルをいつでもサポートできます。新しい規格がポートに読み込まれても、それ以外のポートにはまったく影響はありません。追加される規格は、すべてを設計し直さなくても、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 モジュールを作成し、ポートでエラーが発生した場合に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、各プロトコルのターゲットとなる独自の物理ロケーションに対して、独自のパーシャル BIT ファイルを生成する必要があります。パーシャル BIT 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 BIT 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改善により、BIT ファイルを異なる物理ロケーションに配置でき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規格を使用して、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 デバイスを PCIe バスのペリフェラルにすることができるので、システム ホストで PCIe 接続を介して FPGA をコンフィギュレーションできます。パワーオン リセット後は、FPGA デバイスをフル BIT 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 BIT ファイルには PCIe インターフェイスと ICAP (Internal Configuration Access Port) への接続しか含まれていない場合もあります。

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、この最初のデバイス読み込みにかかるコンフィギュレーション時間も短縮でき、FPGA コンフィギュレーションで PCIe の列挙仕様を満たしやすくなります。

これで、図 2-3 に示すように、システム ホストで PCIe ポートを介してパーシャル BIT 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。

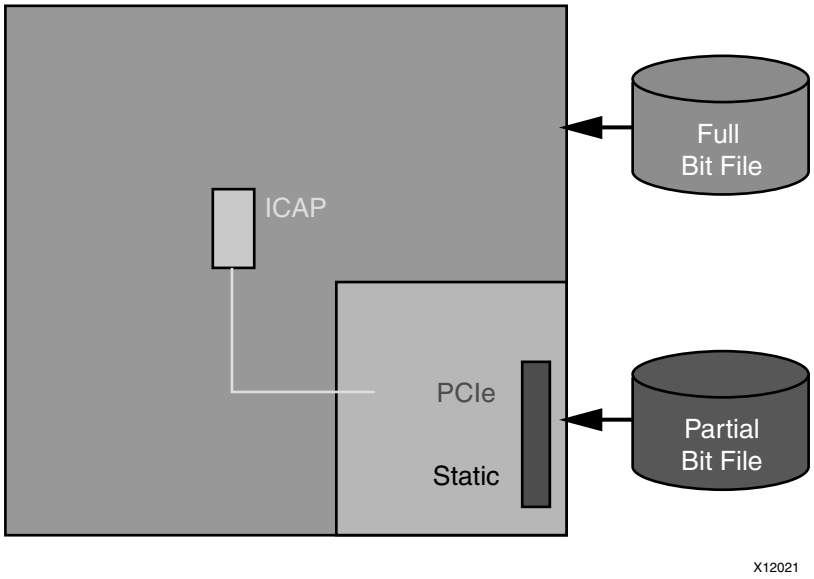

図 2-3 : PCIe インターフェイスを使用したコンフィギュレーション

PCIe 規格には、要求を処理できない場合でも、要求の受信を確認するためのペリフェラル (この場合は FPGA デバイス) 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。そのため、FPGA デバイスはリコンフィギュレーション中でも PCIe コマンドに応答できるようになっています。この使用例は、アプリケーション ノート XAPP883 『パーシャル リコンフィギュレーションを使用した PCI Express テクノロジの高速コンフィギュレーション』に掲載されています。このアプリケーション ノートには、ML605 評価ボードをターゲットとするリファレンス デザインが含まれています。

ダイナミック リコンフィギャラブル 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すばやく変更できます。図 2-4 では、パケットにパーシャル BIT ファイルを含むヘッダーがあるか、特殊パケットにパーシャル BIT ファイルが含まれています。パーシャル BIT ファイルが処理されると、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 BIT ファイルの定義済みライブラリではなく、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。

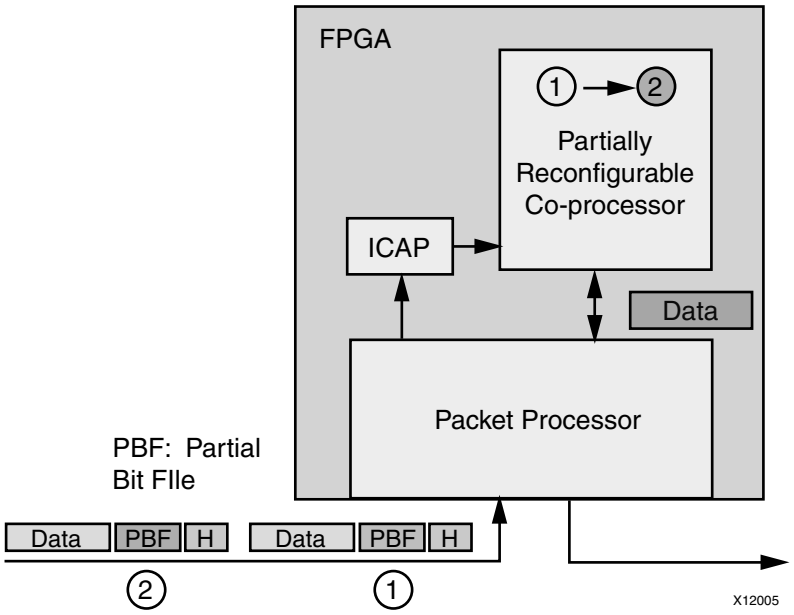

図 2-4 : ダイナミック リコンフィギャラブル パケット プロセッサ

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。パーシャル リコンフィギュレーションと非対称暗号を組み合わせることにより、FPGA コンフィギュレーション ファイルを非常に安全な方法で保護できます。非対称鍵暗号の詳細は、「公開鍵暗号」を参照してください。

図 2-5 では、青いボックスのすべてのファンクションを FPGA の物理パッケージ内に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ー外部には出せません。

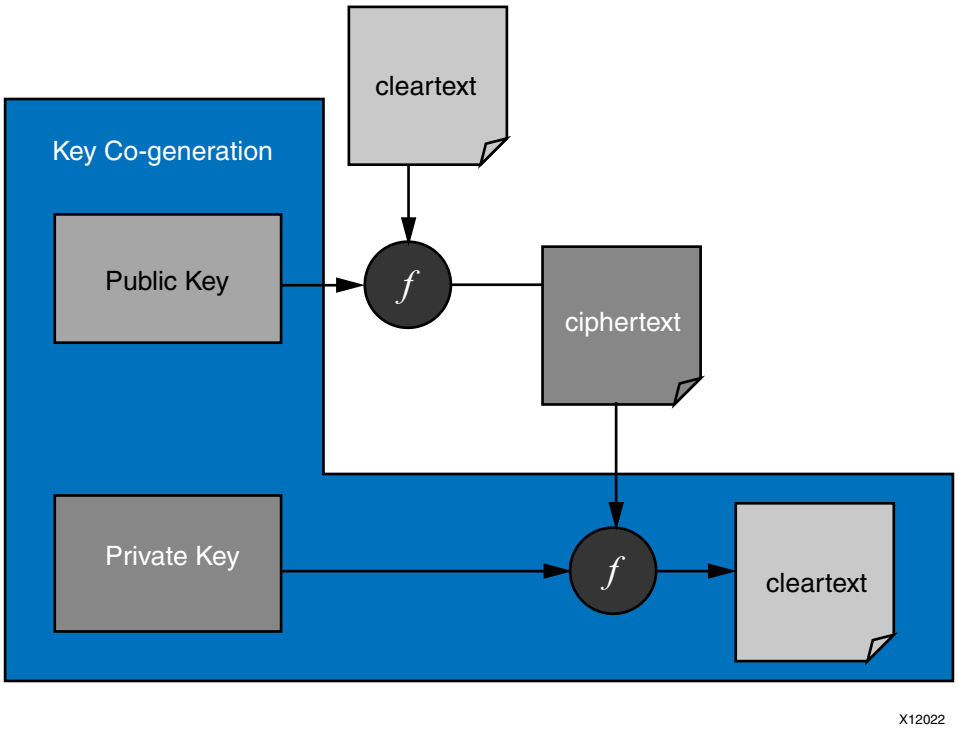

図 2-5 : 非対称鍵暗号化方式

このデザインを実際にインプリメントする場合、最初の BIT 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 BIT 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 BIT 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 BIT 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 が部分的にリコンフィギュレーションされます。

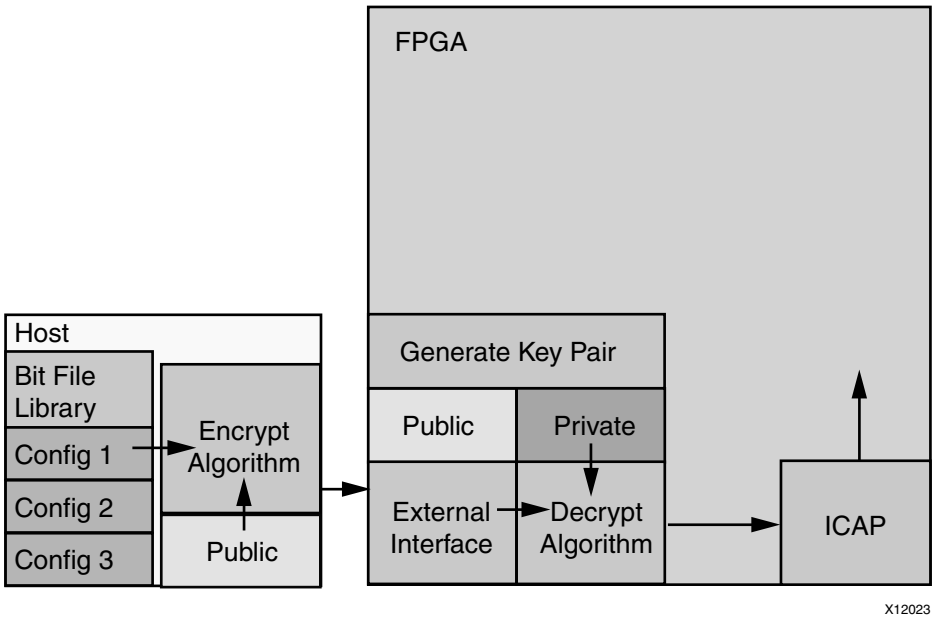

図 2-6 : 暗号化されたパーシャル BIT ファイルの読み込み

パーシャル BIT ファイルが FPGA デザインの大部分を占めるようにし、スタティック デザインのロジックが FPGA リソースのわずかしか使用しないようにすることも可能です。

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 デバイスがパワーオン リセット後などに同じパーシャル BIT ファイルを使用してコンフィギュレーションする場合、同じ BIT 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 デバイスの電源が失われると、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、秘密鍵を格納する各レジスタ ビットを、物理的にリモートの領域および無関連の領域に手動で配置できます。暗号化機能の例は、アプリケーション ノート XAPP887 『PRC/EPRC : パーシャル リコンフィギュレーションのデータ インテグリティおよびセキュリティ コントローラー』を参照してください。このアプリケーション ノートには、Virtex-5 および Virtex-6 のサンプル デザインが含まれています。

まとめ

パーシャル リコンフィギュレーションは、サイズ、重量、消費電力、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はインプリメントできない新しいタイプの FPGA デザインも可能にします。

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、パーシャル リコンフィギュレーション FPGA をサポートするシステムを構築し、パーシャル リコンフィギュレーション デザインを構成する方法、および制約の適用を説明します。

パーシャル リコンフィギュレーション FPGA デザインをインプリメントする方法は、ロジックを共有する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。複数デザイン間で共通ロジックが使用されるようにするため、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。

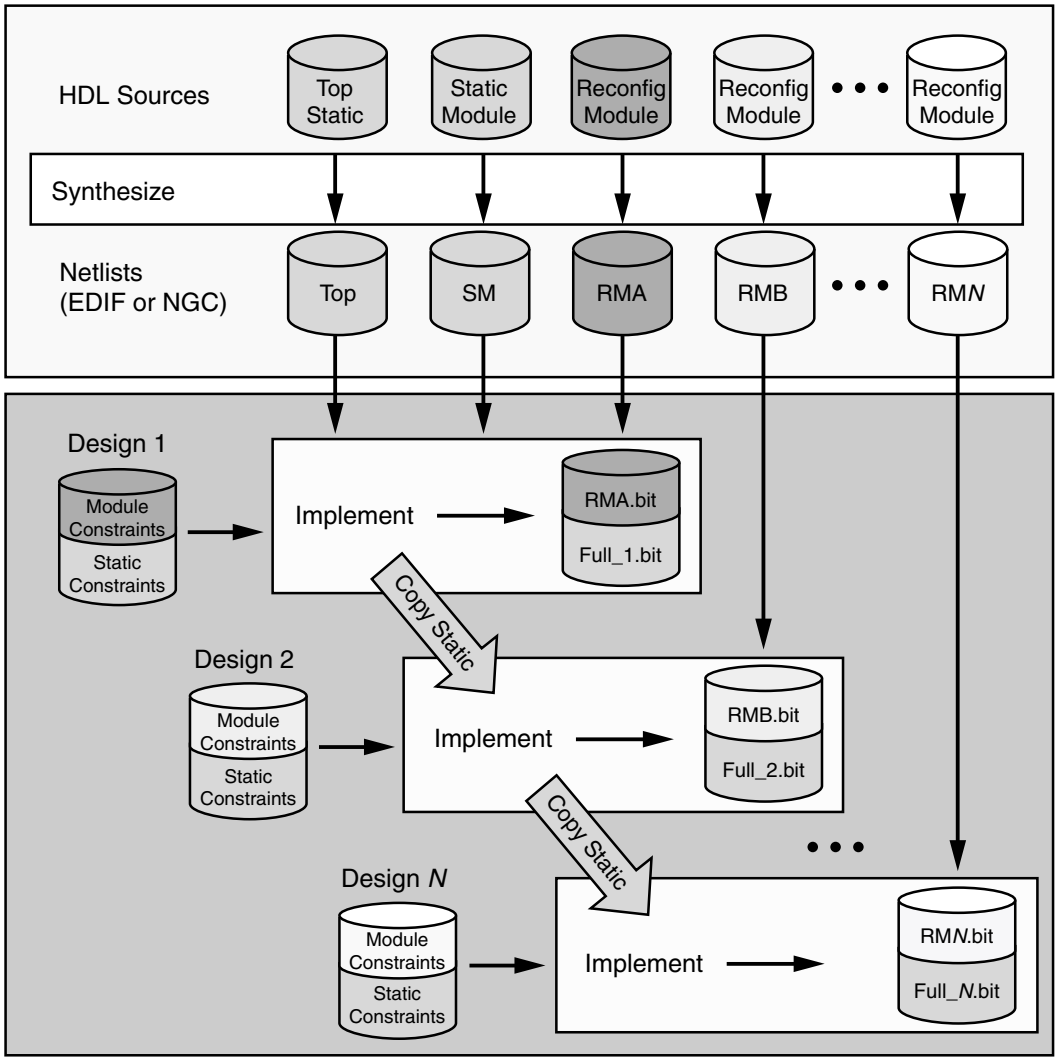

図 3-1 : パーシャル リコンフィギュレーション ソフトウェア フローの概要

一番上の薄いグレーのボックスは、HDL ソースを合成して各モジュールのネットリストを作成する段階を示しています。その後、適切なネットリストを各デザインにインプリメントし、そのコンフィギュレーション用のフル BIT ファイルとパーシャル BIT ファイルを生成します。スタティック ロジックは、最初のインプリメンテーションからのものがその後のすべてのデザイン インプリメンテーションで共有されます。

サンプル デザインの構造

このガイドでは、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、原色の赤、青、および緑と、それらを異なる割合で混合した非原色のカラー バーを DVI サポート モニターに表示します。パーシャル リコンフィギャラブル モジュールは、赤、青、緑のモジュールです。各モジュールには、赤、青、緑ごとに高速および低速のものがあります。この速度は、LED がデモ ボードで点滅する速さを示しています。このデザインは、Virtex®-6 ML-605 評価版プラットフォームをターゲットとしています。

参照デザインのデザイン ファイルは、次のサイトからダウンロードできます。

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

図 3-2 は、階層ネットリストを図示しています。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中でも通常の動作状態を保ちます。Red、Blue、Green は、それぞれ赤、青、緑のリコンフィギャラブル 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、それぞれ高速と低速があります。

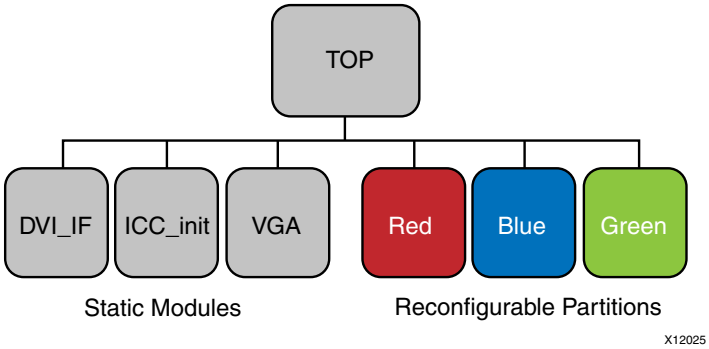

図 3-2 : Color2 デザイン階層

次は、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デザイン ソースの階層とリコンフィギャラブル 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、リコンフィギャラブル 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、ロジックは含まれません。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インプリメントするロジックが含まれます。

サンプル プロジェクト ファイルの構造

パーシャル リコンフィギュレーション 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。

プロジェクトには、各リコンフィギャラブル パーティションに複数のリコンフィギャラブル 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、各リコンフィギャラブル 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 デザインで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ラブル 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ラブル 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ラブル 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 ....... 最上位スタティック モジュール
    DVI_IF...... 下位スタティック モジュール
    IIC_init....."
    VGA ......."
   red_fast......Red のリコンフィギャラブル モジュール
   red_slow....."
   blue_fast .....Blue のリコンフィギャラブル モジュール
   blue_slow ....."
   green_fast.....Green のリコンフィギャラブル 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 BIT ファイル
```

```
 SlowConfig....."
 FSFConfig ....."
 BlankConfig .... 3 つの色のブラック 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 BIT 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

リコンフィギャラブル 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「リコンフィギャラブル モジュールの I/O」を参照してください。

スタティック モジュールは、一緒に合成して 1 つのネットリストを生成するか、個別に合成して複数のスタティック ネットリストを生成できます。NGDBuild でスタティック モジュールとリコンフィギャラブル モジュールがまとめられ、リコンフィギャラブル 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ラブル 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 モジュールまたはリコンフィギャラブル モジュールの合成には、異なるオプションを使用できます。

次に、Color2 サンプル デザインで最低限生成されるネットリストを示します。

```
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ラブル　モジュールを呼び出すことができなくなります。また、各リコンフィギャラブル 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。

リコンフィギャラブル モジュールの各インスタンシエーションには、固有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ラブル モジュールがどのリコンフィギャラブル パーティションに関連付けられているかが正しく判断されます。

実際には、各リコンフィギャラブル モジュールのネットリスト名は同一なので、各ネットリストが別のディレクトリに配置されている必要があります。

```
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 リコンフィギュレーション 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ラブル パーティションごとに 1 つのリコンフィギャラブル 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のコンテキストで実行されます。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、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ラブル モジュールをインプリメントするには、リコンフィギャラブル モジュールの可能な組み合わせのサブセットを選択し、それらを固有のデザインとしてインプリメントする必要があります。各固有のインプリメンテーションは、「コンフィギュレーション」と呼ばれます。

各リコンフィギャラブル 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ラブル モジュールにしておくことができます。そのため、Color2 デザインでインプリメント可能なリコンフィギャラブル モジュールとパーシャル BIT ファイルは、次のとおりです。

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

各リコンフィギャラブル パーティションに 3 つの選択肢があり、このデザインには 3 つのリコンフィギャラブル パーティションがあるので、コンフィギュレーションとして定義できる組み合わせは 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 BIT ファイルはそれ以外のリコンフィギャラブル 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 デザインで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るので、すべてのリコンフィギャラブル 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。必要に応じて、さらにコンフィギュレーションを作成して、独自のフル BIT ファイル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できます。この場合、3 つのリコンフィギャラブル 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 BIT 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がある場合があります。パーシャル リコンフィギュレーション デザインのデザイン レベルのシミュレーションと検証フローは、通常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 UCF ファイルがすべてのコンフィギュレーションで共有され、すべてのスタティック制約が同じになるようにすることができます。

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、あるタイミング制約を red_fast にのみ存在するパスに設定する場合、この制約を上記の最初のコンフィギュレーションにのみ適用できます。これには、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みます。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ラブル 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ラブル モジュールに統合されないようにしたり、リコンフィギャラブル 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、リコンフィギャラブル 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ラブル パーティションに 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”;
```

各リコンフィギャラブル領域に AREA_GROUP RANGE 制約を少なくとも 1 つ定義して、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、通常パーシャル リコンフィギュレーション領域に含まれるスライスを定義するスライス範囲です。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。リコンフィギャラブル モジュールにブロック RAM、I/O、その他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 制約を設定するにはいくつかの要件がありますが、それらは PlanAhead で管理できます。

- AREA_GROUP RANGE 制約はリコンフィギャラブル パーティション領域のサイズと形を定義するので、リコンフィギャラブル パーティションごとに必要です。
- リコンフィギャラブル パーティション内に配置されるリコンフィギャラブル 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 トランシーバー (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 制約が必要です。
- リコンフィギュレーションしないエレメント、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ラブル パーティションを複数の AREA_GROUP RANGE 制約で定義する場合、これらは隣接している必要があります。
- リコンフィギャラブル パーティションの AREA_GROUP RANGE 制約は、別のリコンフィギャラブル 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 (minX, minY) から右上 (maxX, maxY) までを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;
  ```

- スライス、ブロック RAM、DSP48、IOB、MGT などほとんどのロジック タイプは、リコンフィギャラブル パーティションに含めることができます。DCM、PLL、PMCD のようなクロック変更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 モジュール内に含める必要があります。リコンフィギャラブル パーティションの規則の詳細は、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 の境界に設定される必要があります (CLB は分割しない)。この規則に従うと、AREA_GROUP RANGE 制約に Virtex®-5 デバイスの CLB 全体が含まれます。
  - AREA_GROUP のスライス範囲の横軸 (minX) は常に偶数
  - AREA_GROUP スライス範囲の横軸 (maxX) は常に奇数

  この規則に従うと、リコンフィギャラブル パーティションの RANGE が Virtex-5 デバイスの CLB の境界に設定されます。この場合、リコンフィギャラブル 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ラブル 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 の座標は (minX, minY) および (maxX, maxY) で、奇数または偶数のどちらでもかまいません。AREA_GROUP のブロック RAM 範囲は、PlanAhead または FPGA Editor で表示して確認できます。

  30 ページの図 3-3 に、AREA_GROUP RANGE の例を示します。
- スタックド シリコン インターコネクト (SSI) テクノロジでインプリメントされる Virtex-7 デバイスでは、リコンフィギャラブル パーティションの AREA_GROUP RANGE がスーパー ロジック領域 (SLR) 境界を越えることができます。AREA_GROUP の大きさおよび形には厳しい制限は特にありませんが、特に SRL 境界を横切る場合は、RANGE を SLR 内のクロック領域に縦方向に揃えることをお勧めします。

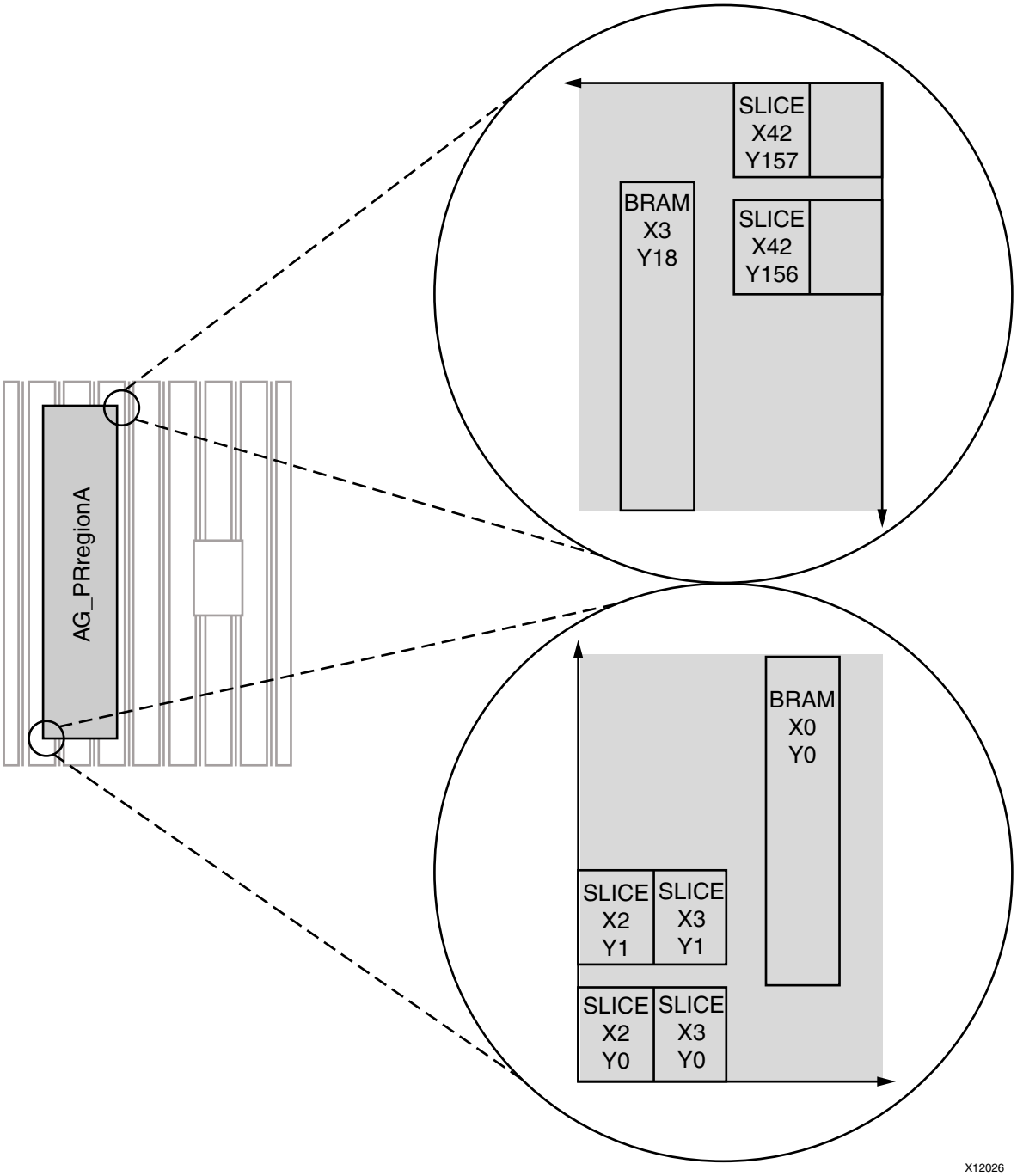

図 3-3 : パーシャル リコンフィギュレーション領域のスライス範囲と BRAM 範囲

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、各リコンフィギャラブル 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、リコンフィギャラブル 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ラブル モジュールを配置するのに十分な大きさである必要があります。使用するスライス数が最も多いリコンフィギャラブル モジュールと、使用する BRAM 数が最も多いリコンフィギャラブル モジュール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ラブル ロジック間のポートの境界に「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ラブル パーティション間の異なるリコンフィギャラブル 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ラブル パーティションの境界を始点、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。

注記：パーティション ピンは、リコンフィギャラ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、32 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ラブル モジュールのボックスは論理範囲を表しており、物理範囲またはフロアプランとはかぎりません。

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ラブル 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ラブル 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

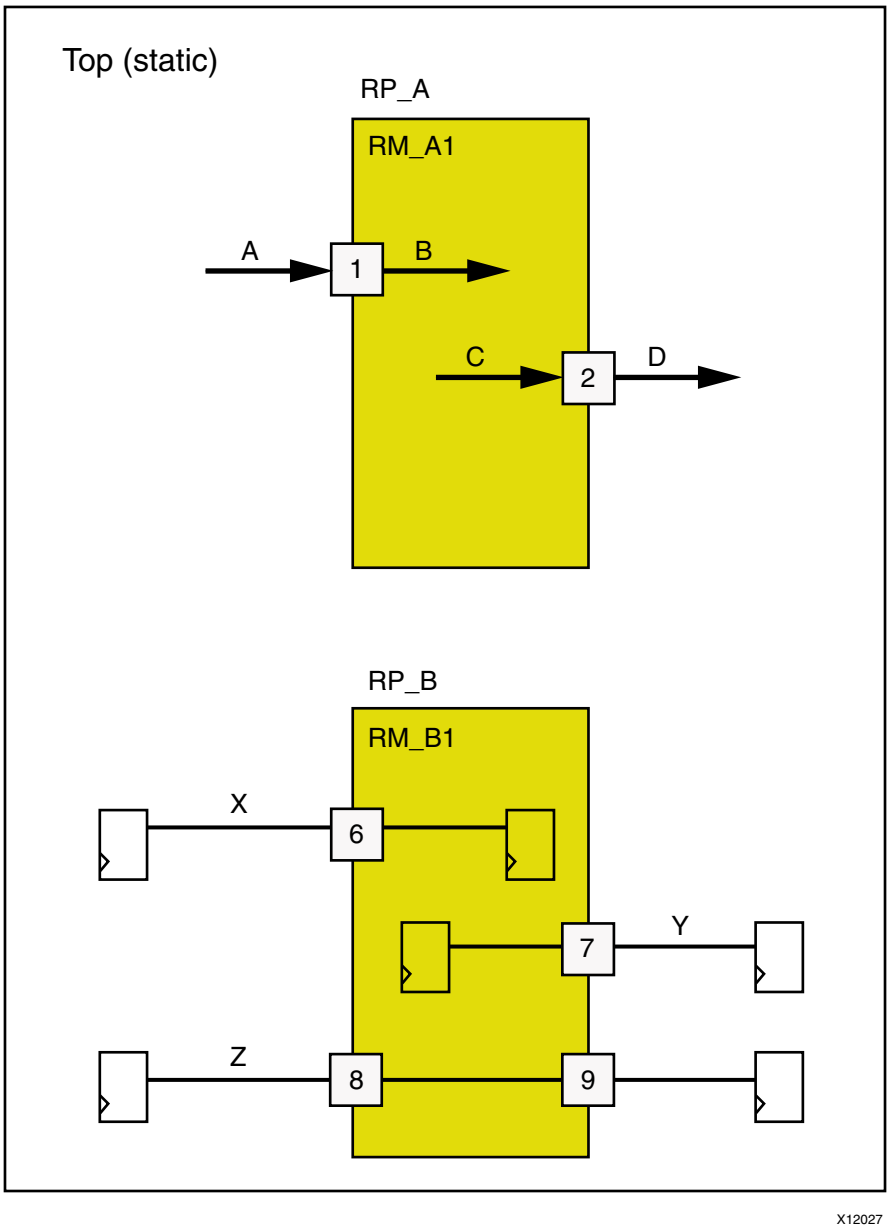

図 3-4 : リコンフィギャラブル パーティションに入出力されるタイミング パス

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ラブル モジュールへの入力とリコンフィギャラブル モジュールからの出力に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 制約を使用すると、これらのパスに PERIOD 制約を使用するよりも包括的に制約を適用できます。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、リコンフィギャラブル 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ラブル 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、次のように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ラブル ネットすべて (上記のパス A と 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ラブル 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 と 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、リコンフィギャラブル パスへのパスとリコンフィギャラブル 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では、タイミングの点ではリコンフィギャラブル モジュールの 1 つのみが考慮されます。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ラブル モジュールに対してタイミング マージンが十分でなく、後でインプリメントしたときにタイミングが満たされない可能性があります。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ラブル モジュールとは別に制約できるので、スタティック領域と各リコンフィギャラブル 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、次のように制約が設定されます。

```
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ラ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 パフォーマンスが得られない可能性があります。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 エラーが発生することがあります。

リコンフィギャラブル パーティションの境界を通過するグローバル クロックを除くすべての信号がレジスタを介するようにし、タイミング制約を簡略化して、タイミング制約が満たされる可能性が向上されるようにします。ただし、パッドがリコンフィギャラブル パーティションの同期コンポーネントに直接接続されている場合は、OFFSET 制約を使用してパスを正しく制約します。

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ラブル パーティションの area_group 範囲内のサイトに物理的に固定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、必ずしも固定する必要はありませんが、固定しておくと、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、最終手段として、タイミング制約を使用した自動配置を実行した後にのみ実行してください。次の UCF コマンドは、パーティション ピンをサイトに物理的に固定します。

```
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 ICAP (Internal Configuration Access Port) を使用する場合、タイミング制約を使用すると、このインターフェイスで達成可能なパフォーマンスを理解しやすくなります。

7 シリーズおよび Virtex-6 の ICAP のタイミング制約

7 シリーズおよび 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 コンポーネントが該当するタイミング グループに追加されていれば、その他に制約は必要ありません。

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 制約は 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、例外制約の NET MAXDELAY を設定する必要があります。

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;
```

この例では、to_icap および from_icap ネットワークは任意の幅のバスです。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、ネット遅延のみに適用されます。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はないので、タイム グループには追加できません。このため、TPSYNC 制約を使用して ICAP を同期コンポーネントにすることはできません。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する必要があります。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 ロケーション制約は、`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が、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配置を保持するためには必要ありません。

パーティション ピンは、リコンフィギャラブル領域内に物理的に配置されていても、論理的にはスタティック ロジックの一部なので、パーティション ピンに設定する制約は最上位 UCF に含める必要があります。パーティション ピンの配置を FPGA Editor 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

少なくとも 1 つのコンフィギュレーションに対して最初のインプリメンテーションを実行した後、Constraints Editor を使用してパーティション ピン グループおよびタイミング制約を作成できます。

デザイン ファイルを指定するダイアログ 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 ソフトウェア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 の [Constraint Type] リストに [Group Constraints] があります。[By Combinatorial Pins] をダブルクリックし、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] フィールドで [Partition Pins] を選択すると、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。

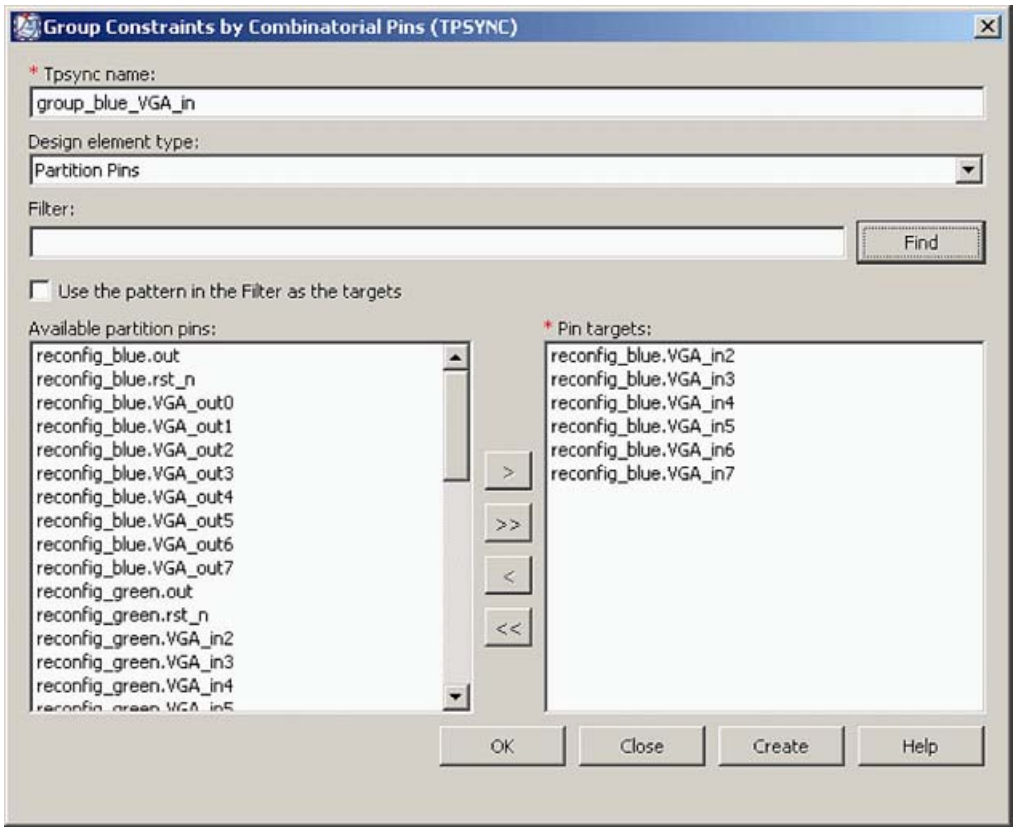

図 3-5 : Constraints Editor でのパーティション ピン グループの作成

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] → [Add Sources] をクリックし、[Add or Create Constraints] をオンにして、Constraints Editor でアップデートした UCF を指定して制約をインポートします。

PlanAhead のリコンフィギャラブル 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 デザインを効率的に管理する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 デザインに制約を設定するのは複雑なので、PlanAhead を使用してこれらの制約を管理するには計画が必要です。

リコンフィギャラブル 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 UCF** : 制約は PlanAhead プロジェクト作成前に存在し、1 つまたは複数の最上位 UCF ファイルに含めます。これらの制約は、リコンフィギャラブル 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ラブル モジュールに適用されます。リコンフィギャラブル モジュール ロジックに関連する制約は、最上位 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、リコンフィギャラブル 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。

- **リコンフィギャラブル モジュールの UCF**：制約は PlanAhead プロジェクト作成前に存在し、リコンフィギャラブル モジュール レベルの UCF ファイルに含めます。これらの制約の階層はリコンフィギャラブル モジュール階層に特定のものです (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ラブル モジュールに同じ制約が必要な場合は、その制約を各リコンフィギャラブル モジュール UCF に複製する必要があります。この方法は、リコンフィギャラブル モジュール特定の制約を追加する際に推奨される方法です。
- **GUI 手法**：制約は、PlanAhead プロジェクトを作成した後に PlanAhead の GUI または Tcl コマンドを使用して作成します。リコンフィギャラブル モジュール特定の制約は、制約の作成時にアクティブなリコンフィギャラブル 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ラブル モジュール UCF に追加されます (最上位のターゲット UCF には含まれない)。その代わり、これらの制約をユーザー定義のリコンフィギャラブル モジュール UCF 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ラブル モジュールをアップデートすることをお勧めします。

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ラブル 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ラブル モジュール制約をリコンフィギャラブル モジュール用の UCF ファイルに含めます。リコンフィギャラブル モジュール制約を最上位 UCF に含めたり、GUI を使用してリコンフィギャラブル モジュールの UCF を作成すると、動作が不正になることがあります。

PlanAhead UCF の既知の問題

- 最上位 UCF にリコンフィギャラブル 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ラブル モジュールが該当するリコンフィギャラブル 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] ビューをいったん閉じ、リコンフィギャラブル モジュールのネットリストを追加してから、開き直してください。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、リコンフィギャラブル 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、run を開始する前に [Netlist Design] ビューをいったん閉じてから開き直してください。

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 ソフトウェアでロジックがこれらの属性に従って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 BIT ファイルにインプリメントされる方法を指定します。リコンフィギャラブル モジュールにはリコンフィギャラ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。これは、モジュールの最終的なインプリメンテーションに大きく影響します。

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デザインからインポート (保持) 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元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、この結果をエクスポート (プロモート) すると、この後のインプリメンテーションでは、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 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、その後の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。

インプリメンテーション ツールは前の結果を使用してパーティション情報をインポートします。PlanAhead では、プロモートされたコンフィギュレーション データが自動的に管理されます。コマンド ライン ユーザーの場合もユーザー自身が簡単に管理できるようになっています。コンフィギュレーションの繰り返しが完了したら、これらのファイルが上書きされないように、単にコンフィギュレーションを安全なディレクトリにコピーします。このコンフィギュレーションをインポートしたら、xpartition.pxml ファイルの ImportLocation をこのディレクトリに設定します。このディレクトリのインポート ファイルには、xpartition.pxml と *_prev_* ファイルがすべて含まれます。PlanAhead プロジェクトでは、これらのファイルの名前はそれぞれ <design>_prev_built.ngd、<design>_prev_mapped.ncd、<design>_prev_mapped.ngm および <design>_routed_prev_routed.ncd のようになります。また、各段階のレポート ファイルはすべて維持され、保存されたコンフィギュレーションを記述するのに使用できます。

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。各コンフィギュレーションには、それぞれのデザイン ディレクトリに独自の PXML ファイルがあります。

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。
- リビジョン管理の必要に応じてソースとして扱うことができます。

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、リコンフィギャラブル 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 属性が保持され、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。

注記：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 ロジックをインポートする 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 つのリコンフィギャラブル モジュールすべてをインポートする 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 および 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、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。次に、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例を示します。

```
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、リコンフィギャラブル パーティションを通して配線できます。
- リコンフィギャラブル モジュール内の配線は、そのリコンフィギャラブル 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された配線より優先されます。

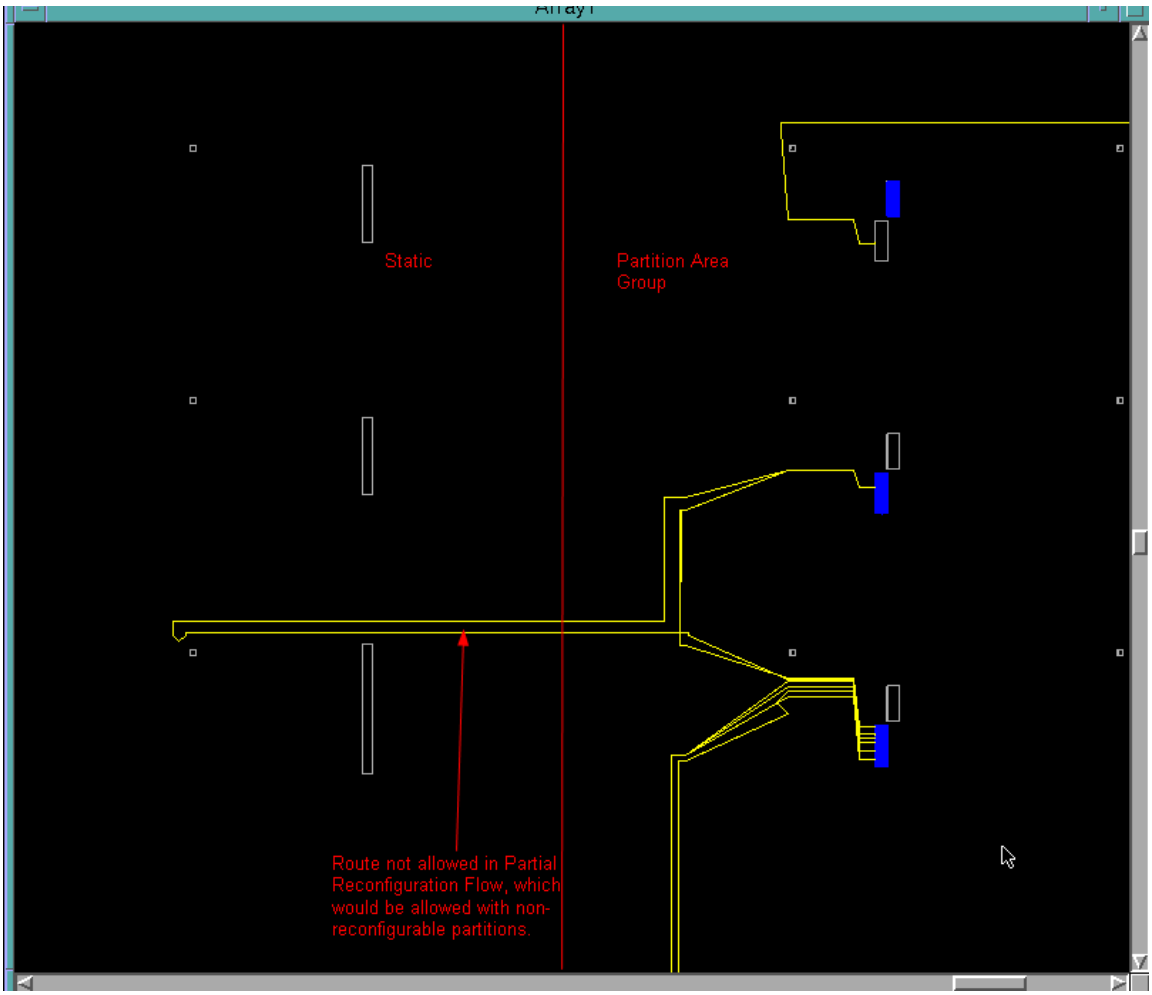

図 3-6 : パーシャル リコンフィギュレーションの配線制限

そのため、配置配線問題をデバッグする際は次の点を考慮する必要があります。

- リコンフィギャラブル パーティションのエリア グループは、フラット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、リコンフィギャラブル 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>
```

リコンフィギャラブル 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、リコンフィギャラブル パーティションがこの変更で問題なく配置配線された場合、エリア グループをそのリコンフィギャラブル 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。

BIT 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 BIT ファイルとパーシャル BIT ファイルの両方が生成されます。パーシャル BIT 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ラブル 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 BIT ファイルが自動的に生成されます。フル BIT 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、サンプル デザインの 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
```

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

```
fff_reconfig_blue_blue_fast_partial.bit
```

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

```
fff_reconfig_green_green_fast_partial.bit
```

(リコンフィギャラブル パーティション `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**

  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する)。

- **-g Binary:Yes**

  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ー情報を削除したものと同じ) を生成します。BIT ファイルにはさまざまな長さのヘッダー情報が含まれるので (ワード境界にあるとはかぎらない)、カスタム コンフィギュレーション インターフェイスでは BIN ファイルを使用する方が適切です。

- **-g ConfigFallback:Disable**

  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) は発生した場合に、フル デバイス コンフィギュレーションがトリガーされないようにします。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**

  これがデフォルトです。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、パーシャル リコンフィギュレーションにスレーブ SelectMAP、スレーブ シリアル、または BPI モードを使用する場合に設定する必要があります。このオプションは CONFIG_MODE 制約と共に使用して、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』(UG625) 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 の **-r** 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た場合、パーシャル BIT ファイルを生成する際にその変更点が比較されます。

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』(UG628) 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、リコンフィギャラブル 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、次のものがあります。

- 「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ラブル 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

MAP レポート (.mrp) には、NGDBuild レポートと同様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ラブル 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 がリコンフィギャラブル 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 グループにスライス (LUT および FF) の範囲が 1 つ関連付けられています。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。

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、リコンフィギャラブル モジュールのタイミング解析で、スタティック領域に適用された制約が再利用されます。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、リコンフィギャラブル モジュールの該当する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。

リコンフィギャラブル 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 ピンが始点のパス、終点のパス、およびパーティション ピンを通過す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。このパスのデスティネーション名は 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 ピンを介した遅延は、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)
```

49 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。

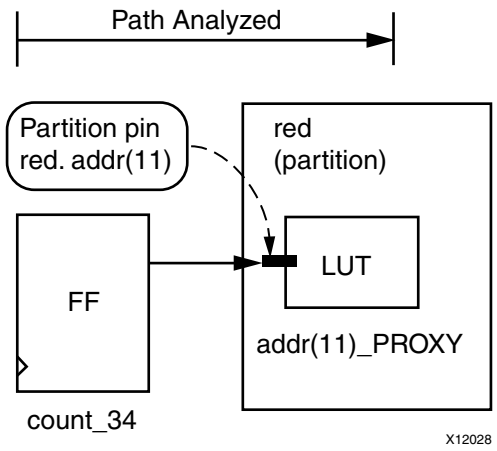

図 3-7 : スタティック FF からパーティション ピンまでのパス

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、リコンフィギャラブル 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。このパスのソース名は red.d_out(5) で、パーティション名が red、ポート名が d_out(5) であることを示しています。

次の解析では、プロキシ ロジックを介した伝搬時間と、出力バッファーへのネット遅延、出力バッファーを介した 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。

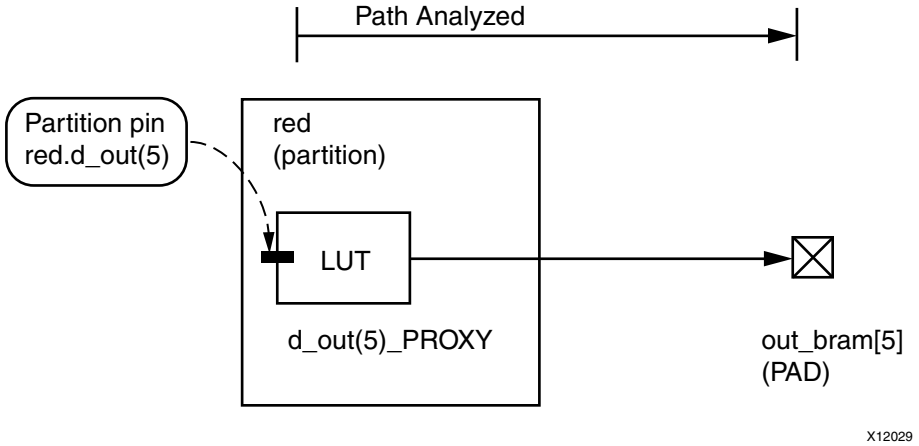

図 3-8 : パーティション ピンからスタティック PAD までのパス

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内)。

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、リコンフィギャラブル パーティションを通った後、さらに多くのプロキシ ロジックを通過して、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。リコンフィギャラブル 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)
```

52 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。

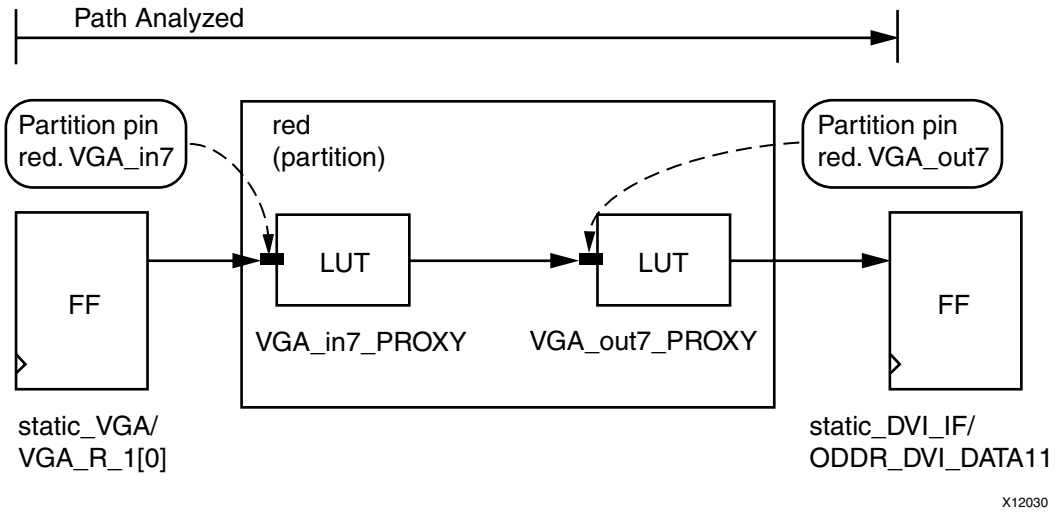

図 3-9 : パーティション ピンを通る FF から FF までのパス

BitGen レポート

BitGen を実行すると、各パーシャル BIT ファイルとフル BIT ファイルのレポート ファイルが作成されます。フル BIT ファイルのレポートには、フル BIT ファイルに含まれるリコンフィギャラブル 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 BIT 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 BIT ファイルのレポートには、それがパーシャル BIT ファイルであることと、関連付けられているパーティションとリコンフィギャラブル モジュールが示されます。

```
...
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の配線済み 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、リコンフィギャラブル 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 : インポートされたパーティションは、コンフィギュレーション間で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ラブル 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 : リコンフィギャラブル 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ラブル モジュールへの入力と出力に同じポートを使用する必要があります。

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。

このユーザー ガイドのサンプル デザインの場合、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/13.3/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/13.3/ISE_DS/ISE/bin/lin/unwrapped/pr_verify
./BlankConfig/BlankConfig.ncd ./FastConfig/FastConfig.ncd
./FSFConfig/FSFConfig.ncd ./SlowConfig/SlowConfig.ncd => PASS
```

ログ ファイルに示されるように、NCD ファイルが 2 つずつ比較され、コンフィギュレーションの特定情報とリソースが不一致であるものが検出されます。最後の行には、その実行全体の結果 (PASS または FAIL) が示されます。

ログ ファイルには、次のリソース比較が示されます。

- `Number of matched proxy logic bels`

  プロキシ ロジックとして使用される LUT1 で、2 つのコンフィギュレーションに存在しており、位置が同じものの数を示します。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`

  2 つのコンフィギュレーションのリコンフィギャラブル 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`

  2 つのコンフィギュレーションで配線が同じグローバル クロックのネットの数を示します。この数はすべての解析で同じになるはずです。

- `Number of matched Reconfigurable Partitions`

  2 つのコンフィギュレーションで使用され、インプリメンテーションが同じリコンフィギャラブル 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ラブル パーティションは 0 と表示されます。`FSFConfig` と `FastConfig` では、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ラブル 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、シリコンに安全な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。これらの制限は、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性などに影響します。

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	まずインポートされたロジックが配置されます。次にインプリメントされたロジックが配置されます。 エリア グループ要件はありません。	まずインポートされたロジックが配線されます。 次にインプリメントされたロジックが配線されます。
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しない限り、リコンフィギャラブル 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ラブル ロジックのみです。 配線制限は、配置段階で考慮されます。	リコンフィギャラブル 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、リコンフィギャラブル ロジックには使用できません。 まずインポートされたロジックが配線されます。 次にインプリメントされたロジックが配線されます。

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 フローではインプリメントできなかったり、同じタイミングや密度が得られ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、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」の説明に従って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、[Specify synthesized (EDIF or NGC) netlist] をオンにします。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。[Enable Partial Reconfiguration] をオンにして、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。

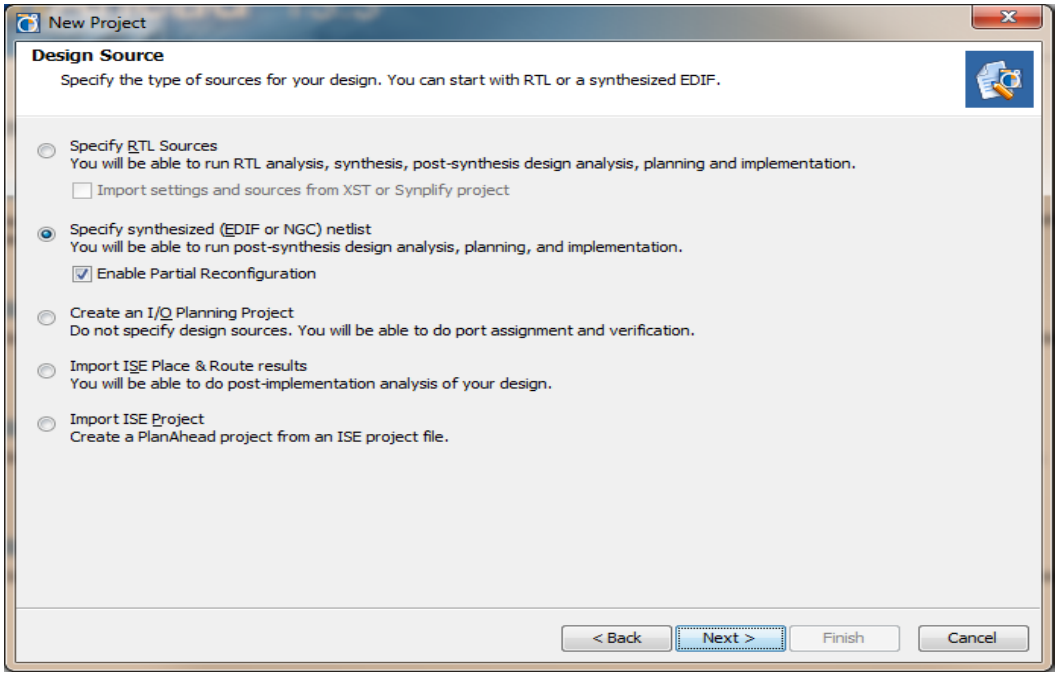

図 4-1 : New Project ウィザード

2. スタティック ロジックに関連するすべてのネットリスト ソースを追加します。各ファイルまたはすべてのディレクトリを追加できますが、ソースはすべてスタティック ロジックのみが含まれている状態のものである必要があります。リコンフィギャブル モジュールのソースは、後で追加します。この例では、すべてのスタティック ロジックが top.ngc に含まれ、最上位ソースとして識別されています。図 4-2 は、New Project ウィザードの [Add Netlist Sources] ページを示しています。

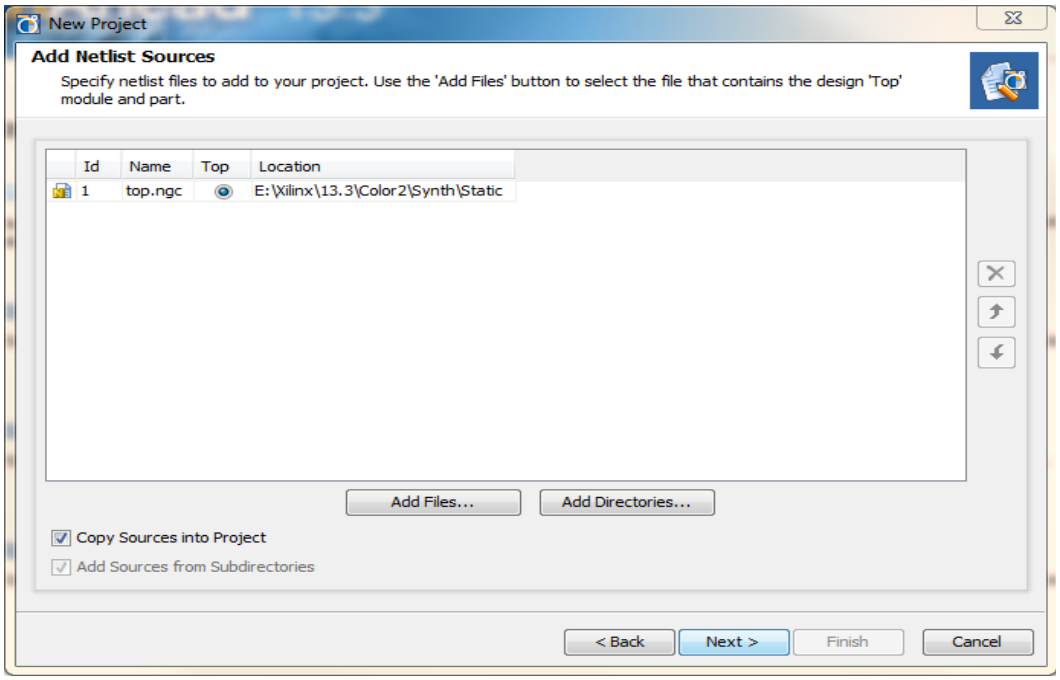

図 4-2 : スタティック ロジックのみを含むネットリストの追加

3. I/O およびタイミング制約を含む最上位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 ソフトウェアでは、インプリメンテーションの実行前に、すべての最上位 UCF ファイルとモジュール レベルの UCF ファイルが連結されます。図 4-3 は、New Project ウィザードの [Add Constraints] ページを示しています。

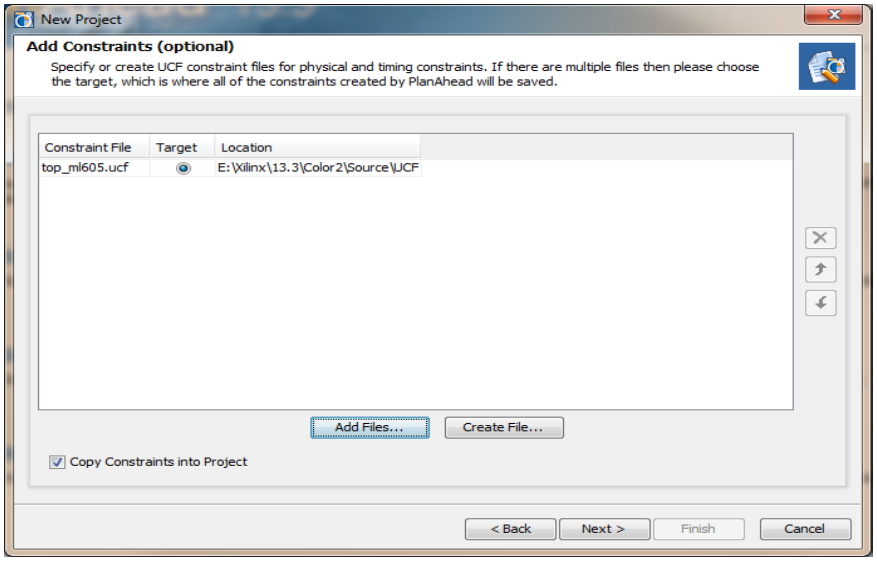

図 4-3 : New Project ウィザードの [Add Constraints] ページ

ターゲット デバイスは、ネットリストから読み出されます。

4. ターゲット デバイスが正しいかどうかを確認し、必要であれば変更して、[Next] をクリックします。
5. ウィザードの残りのページで [Next] をクリックし、最後のページで [Finish] 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、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できるようにします。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 環境変数は以前のバージョンの ISE® ツールのインストール ディレクトリには設定されません。

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 Settings] をクリックし、[General] ページで [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、フラットな 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、フラット デザインには不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。リコンフィギャラブル 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ラブル 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 に示すように、右下のステータス バーの表示が [Post-Synthesis Flow] から [Partial Reconfiguration Flow] に変わります。

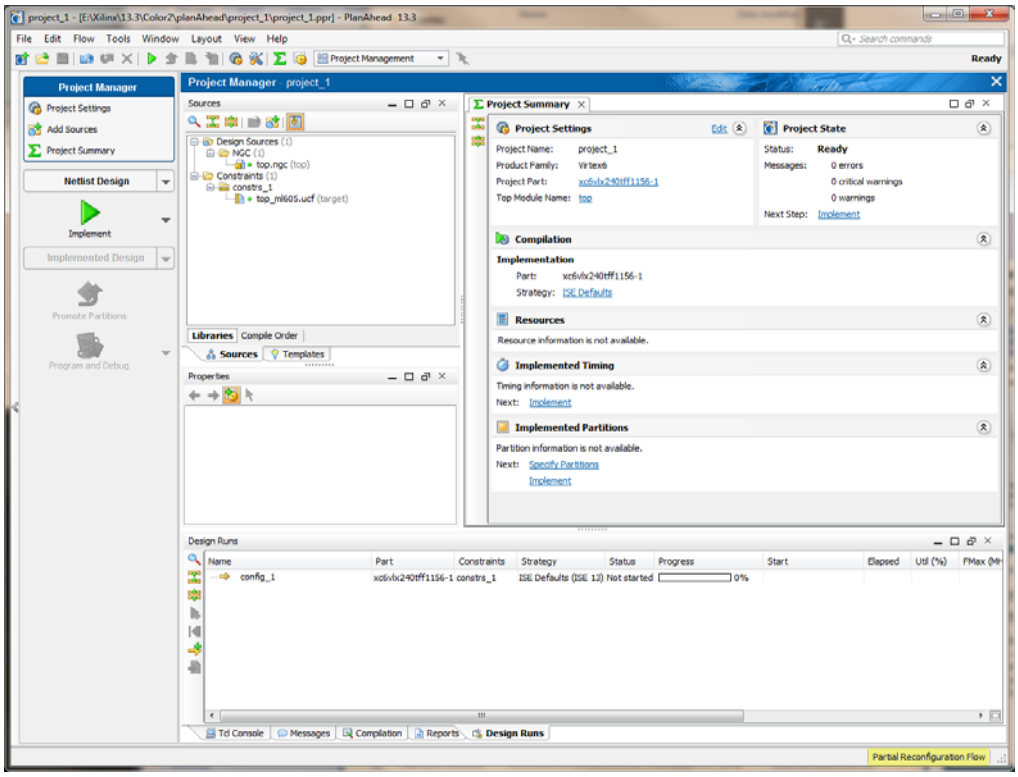

図 4-4 : PlanAhead のパーシャル リコンフィギュレーション プロジェクト

ネットリスト デザインを開く

PlanAhead で Project Manager レイアウトが開きます。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ると、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ャラブル モジュールであることを確認してください。

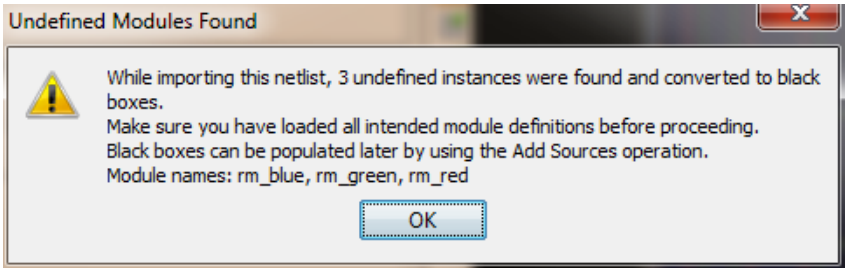

図 4-5 : 必ず表示される警告メッセージ

図 4-4 に示すサンプル デザインの [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ラブル モジュールのネットリストには、blue、green、red のそれぞれに Fast と Slow があります。

リコンフィギャラブル インスタンスの定義

リコンフィギャラブル パーティションを定義するには、下位インスタンスを右クリックし、[Set Partition] をクリックします。

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

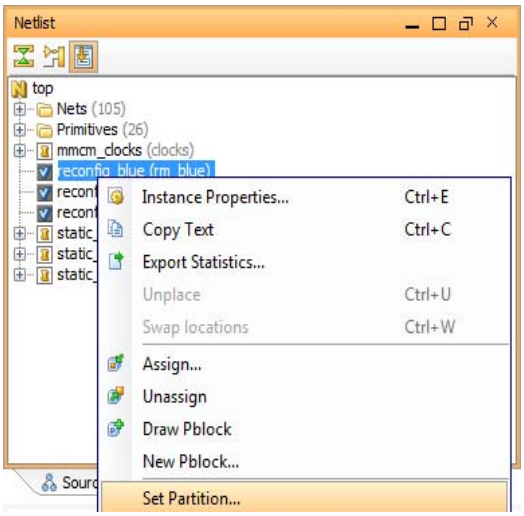

図 4-6 : パーティションをリコンフィギャラブルとして設定

パーティションは、リコンフィギャラブルまたは通常のパーティションに設定できます。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、リコンフィギャラブルとしてしか定義できません。リコンフィギャラブル パーティションには、リコンフィギャラブル モジュールのネットリストを読み込むことができるほか、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 に示すように、モジュールと Fast か Slow かが識別できるように、リコンフィギャラブル モジュールの名前を入力します。

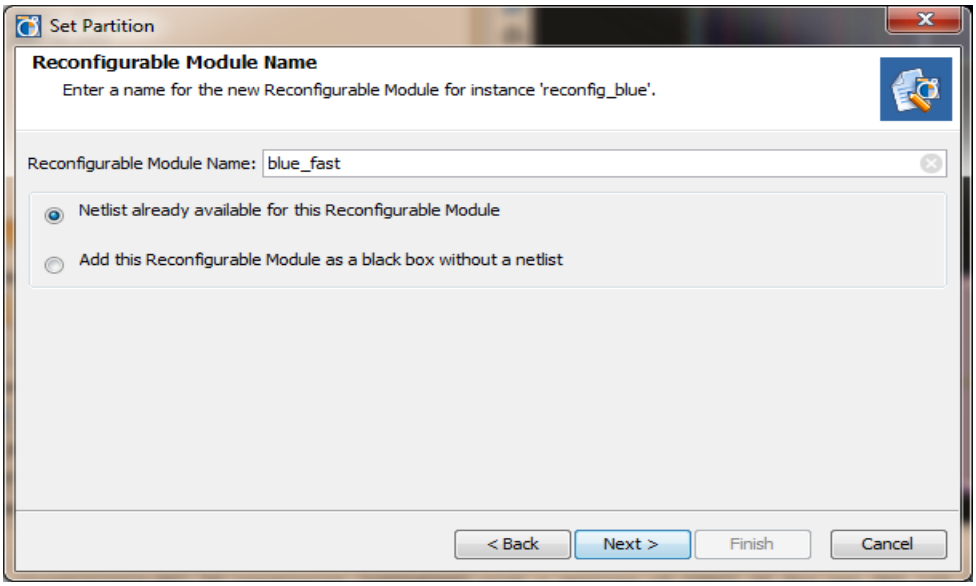

図 4-7 : リコンフィギャラブル モジュールの命名

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ラブル 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)。

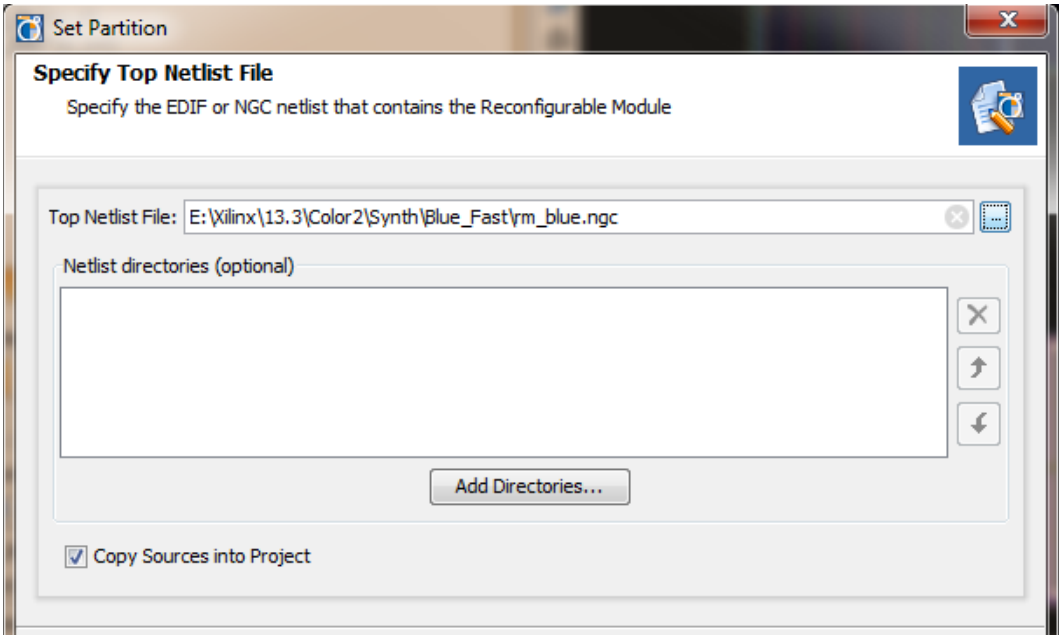

図 4-8 : リコンフィギャラブル モジュール 1 つが追加されたリコンフィギャラブル パーティション

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。このリコンフィギャラブル 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。

リコンフィギャラブル モジュールは、[Netlist] ビューのリコンフィギャラブル 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ラブル パーティションを含めることができます。この場合、デザインの各リコンフィギャラブル パーティションに対して [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、各リコンフィギャラブル パーティション (`red`、`green`、`blue`) に対して `fast` バージョンのモジュールが読み込まれます。

プロジェクトへのリコンフィギャラブル 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、リコンフィギャラブル パーティションにリコンフィギャラブル モジュールをさらに追加できます。

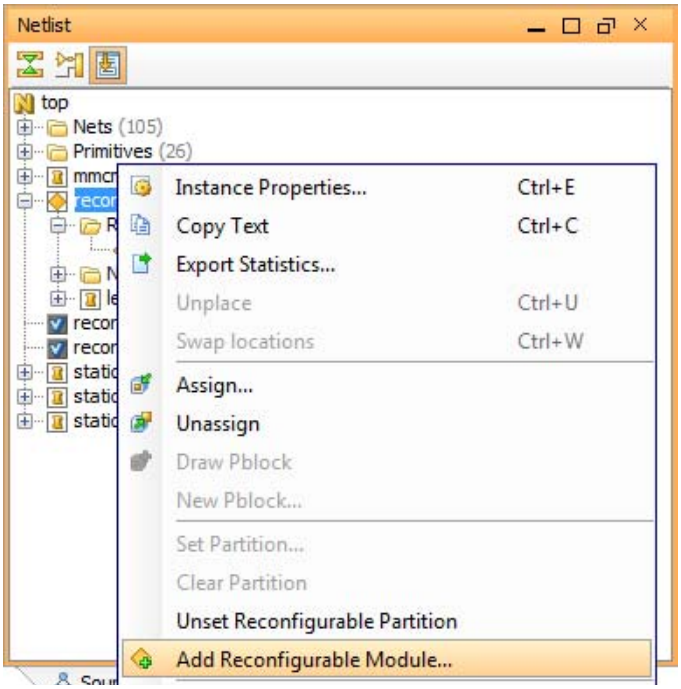

図 4-9 : リコンフィギャラブル パーティションへのリコンフィギャラブル モジュールの追加

このコマンドを使用してすべてのリコンフィギャラブル モジュールをすべてのリコンフィギャラブル パーティションに追加します。このサンプル デザインでは、`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. [Add Reconfigurable Module] コマンドを使用し、[Add this Reconfigurable Module as a black box without a netlist] をオンにします。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)。

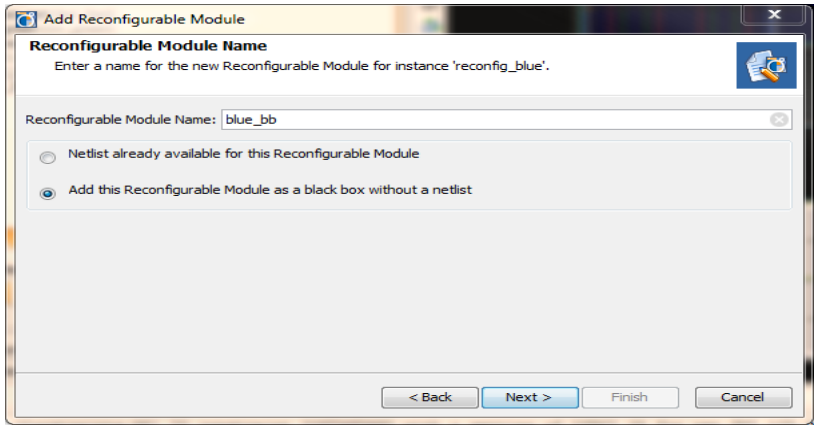

図 4-10：リコンフィギャラブル モジュールとしてのブラック ボックスの追加

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

図 4-11 は、`blue_fast` がリコンフィギャラブル パーティション `reconfig_blue` のアクティブなリコンフィギャラブル 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ラブル 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ラブル モジュールであることを示しています。

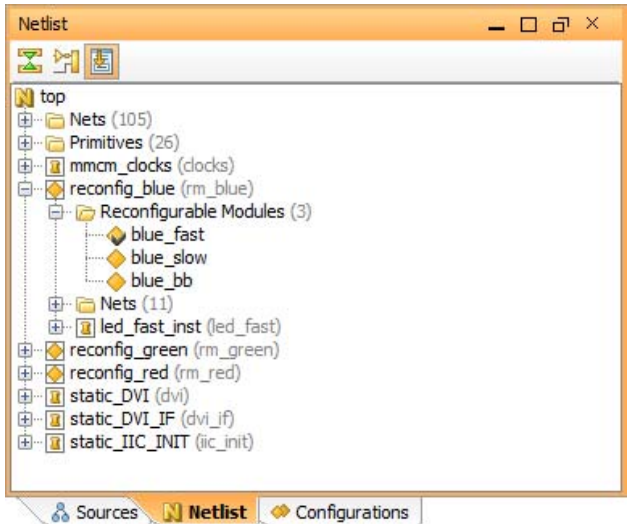

図 4-11：すべてのリコンフィギャラブル モジュールが追加されたリコンフィギャラブル パーティション

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、リコンフィギャラブル 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)。

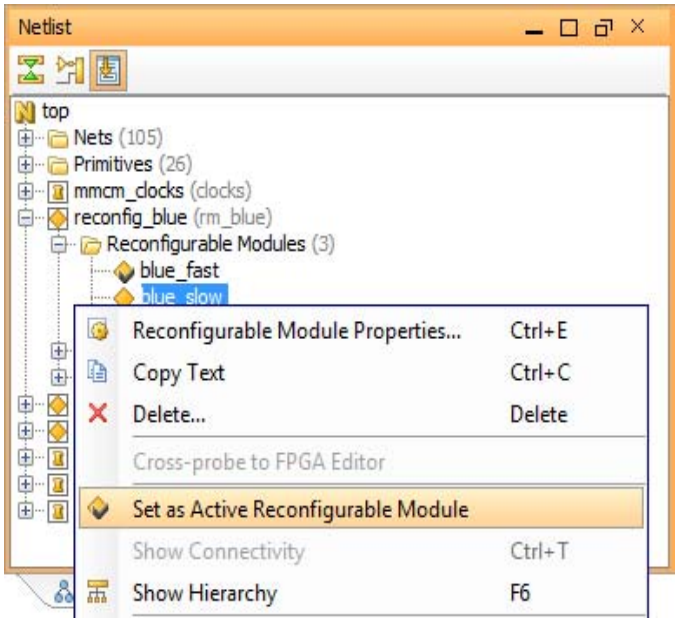

図 4-12 : アクティブ リコンフィギャラブル モジュールの変更

デザイン ソースの管理

ソース ファイルに変更を加えた場合、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)。

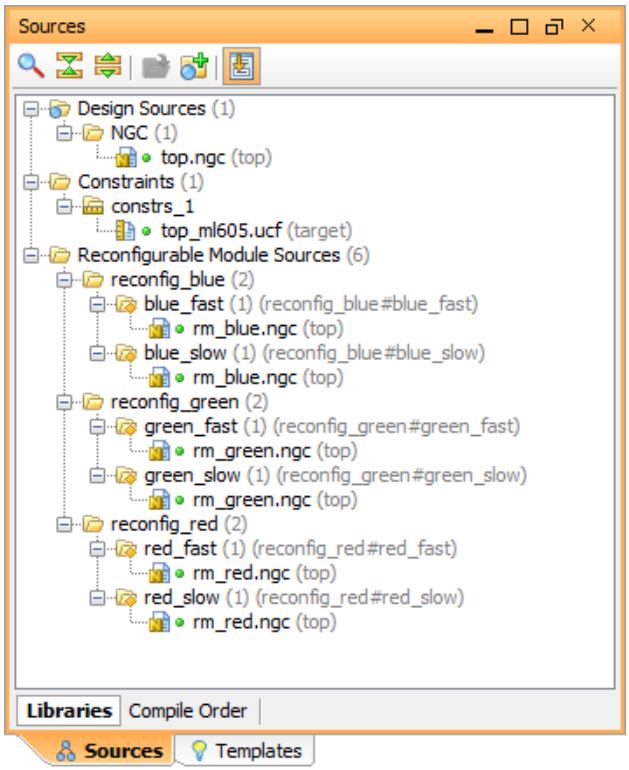

図 4-13 : [Sources] タブ

アップデートするネットリストを右クリックし、[Update File] をクリックして、新しいネットリストを読み込みます。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 (スタティック パーティションまたはアクティブ リコンフィギャラブル モジュールの一部である場合)。

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ラブル ロジック間のインターフェイスの変更はないと想定されます。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ラブル パーティションに含まれるすべてのリコンフィギャラブル 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、次に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] ビュー レイアウトを選択して [Physical Constraints] ビューと FPGA のフロアプラン ビューを開きます (図 4-14)。

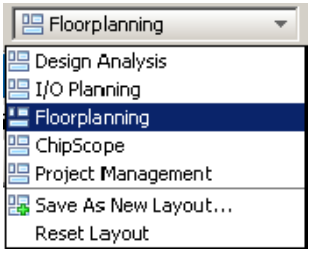

図 4-14 : PlanAhead ツールバーの Floorplanning ビュー レイアウト

デバイスのリコンフィギャラ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] ボタン ( ) をクリックします。

注記：[Tools] → [Floorplanning] → [Place Pblocks] コマンドを使用してデバイスに Pblock を自動配置しないでください。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. 68 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、[Set Pblock Size] ボタンを使用できるようにします。

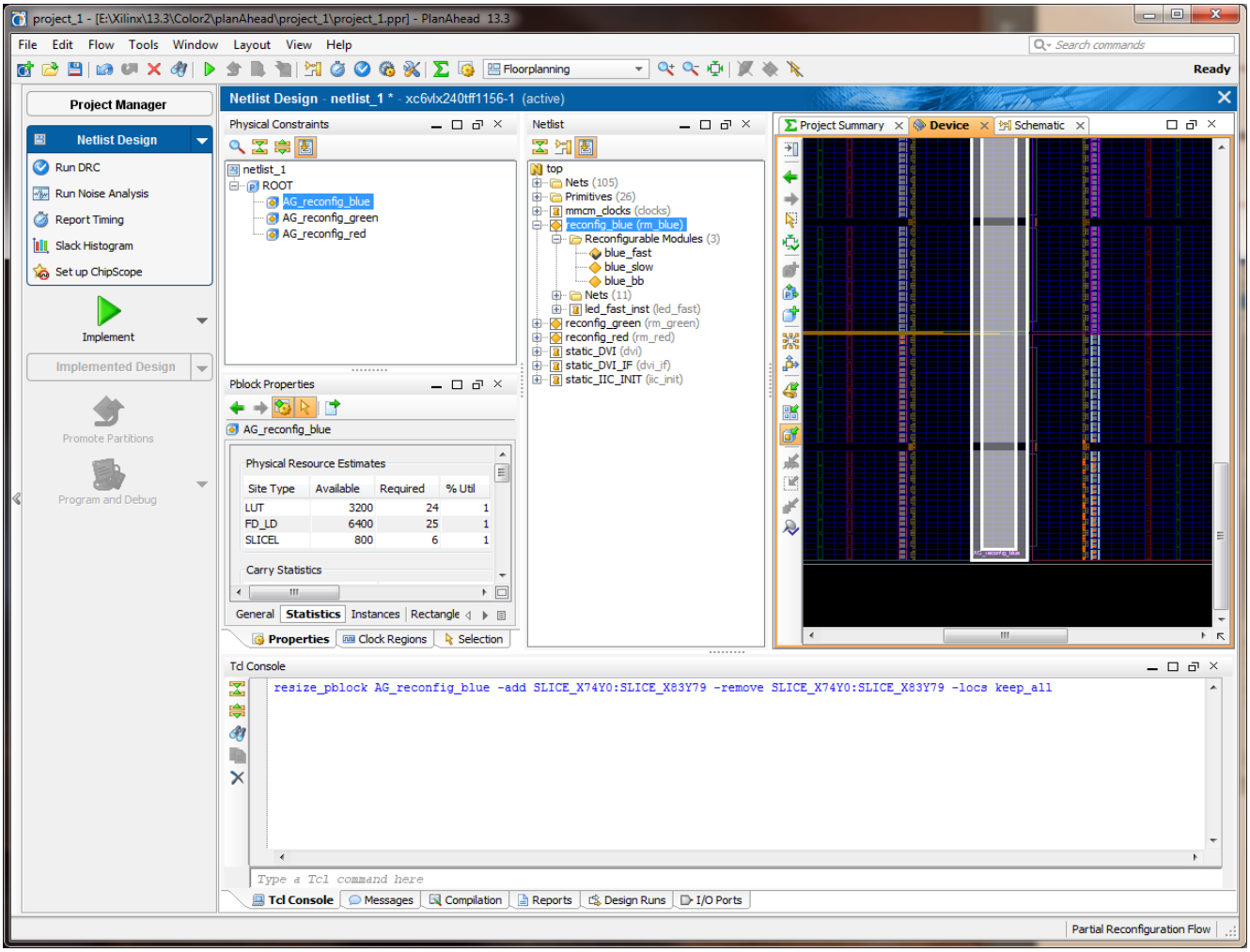

図 4-15 : リコンフィギャラブル パーティションの Pblock の描画

[Device] ビューのクロック領域の境界を、リコンフィギャラブル領域を描画する際のガイドとして使用できます。リコンフィギャラ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「リコンフィギャラブル パーティション境界の定義」を参照してください。Pblock が定義されると、その領域に制約するリソースを選択するダイアログ ボックス (69 ページの図 4-16) が表示されます。

注記 : PlanAhead ソフトウェアでは、リコンフィギャラブル 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。

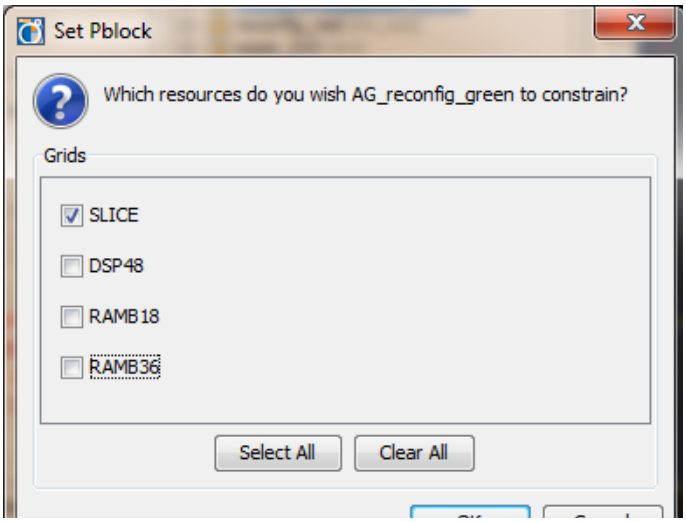

図 4-16 : Pblock を使用した範囲の定義

このように選択すると、リコンフィギャラブル 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. リコンフィギャラブル 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 BIT 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、BIT 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ラブル モジュールに含まれる各ロジック タイプに対して、RANGE を定義します。

各リコンフィギャラブル領域には、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 : リコンフィギャラブル 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 ボックスで、カテゴリ別に表示されている DRC をオンまたはオフにします。
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ます。

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。

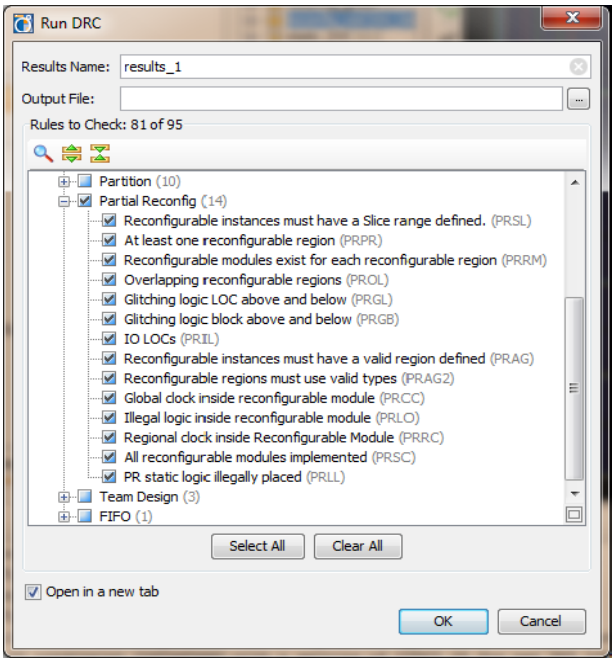

図 4-18 : パーシャル リコンフィギュレーションの [Run DRC] ダイアログ ボックス

[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 に示すように [Violation Properties] ビューにその詳細が表示されます。

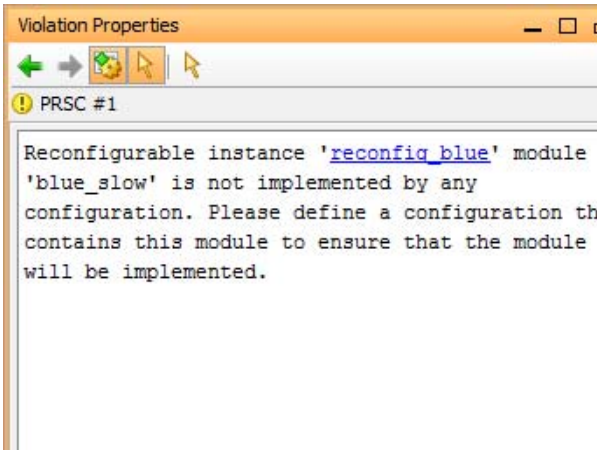

図 4-19 : DRC チェックの結果

[Violation Properties] ビューでパーシャル リコンフィギュレーション DRC に違反するオブジェクトのリンクをクリックすると、どこにあるのかを確認できます。

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 範囲を定義したら、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 の下部にある [Design Runs] ビューをクリックし、`config_1` をクリックします。[Implementation Run Properties] ビューの [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ラブル モジュールが表示されます (図 4-20 参照)。各リコンフィギャラブル パーティションの最初のリコンフィギャラブル 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。

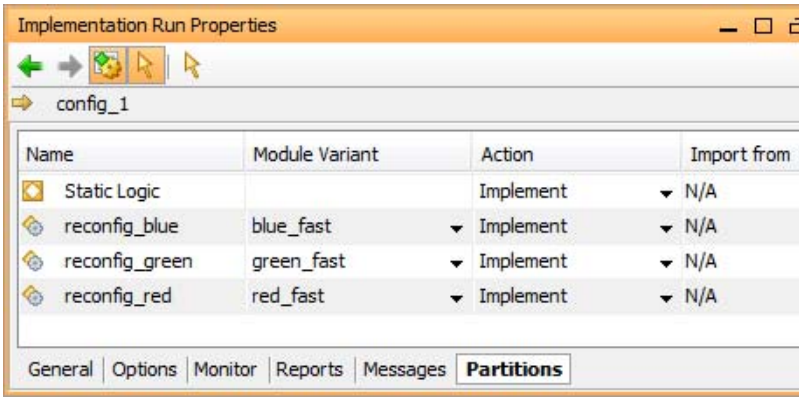

図 4-20 : コンフィギュレーションのリコンフィギャラブル モジュールの定義

インプリメンテーション run のプロパティは、[Implementation Run Properties] ビューの [Options] タブで変更するか、または Flow Manager で [Implement] ボタンの右側の矢印をクリックして [Implementation Settings] をクリックすると変更できます。図 4-21 を参照してください。

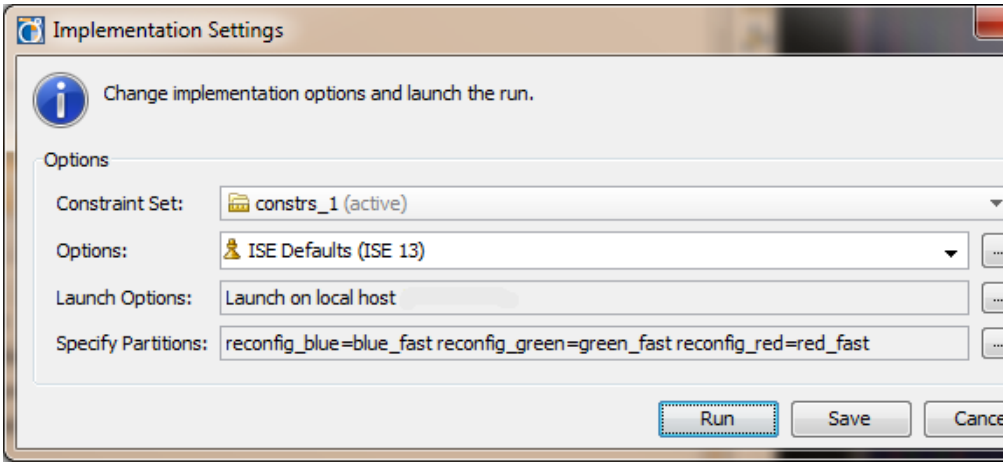

図 4-21 : インプリメンテーション プロパティの設定

[Configurations] ビュー ([Window] → [Configurations]) には、図 4-22 に示すように、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ラブル モジュール、ステータスなどが表示されます。

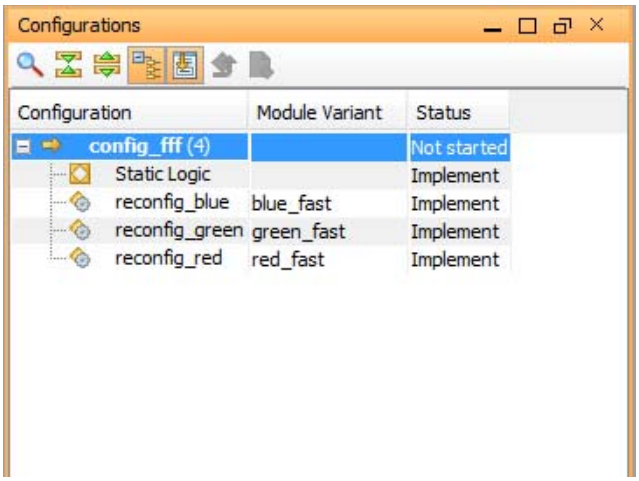

図 4-22 : レポートされる各コンフィギュレーションの詳細

Flow Manager の [Implement] ボタンの右側の矢印をクリックして [Create New Implementation Runs] 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)。

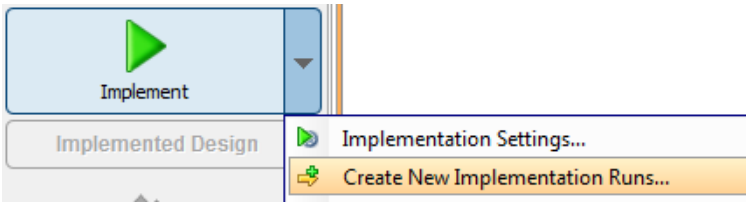

図 4-23 : [Create New Implementation Runs Option] オプション

図 4-24 : [Create New Runs] ボタン

コンフィギュレーションは、リコンフィギャラブル モジュールとブラック ボックスを任意に組み合わせて作成できます。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ラブル モジュールを選択します。

注記 : この時点でこれらの run は起動しないでください。

73 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。

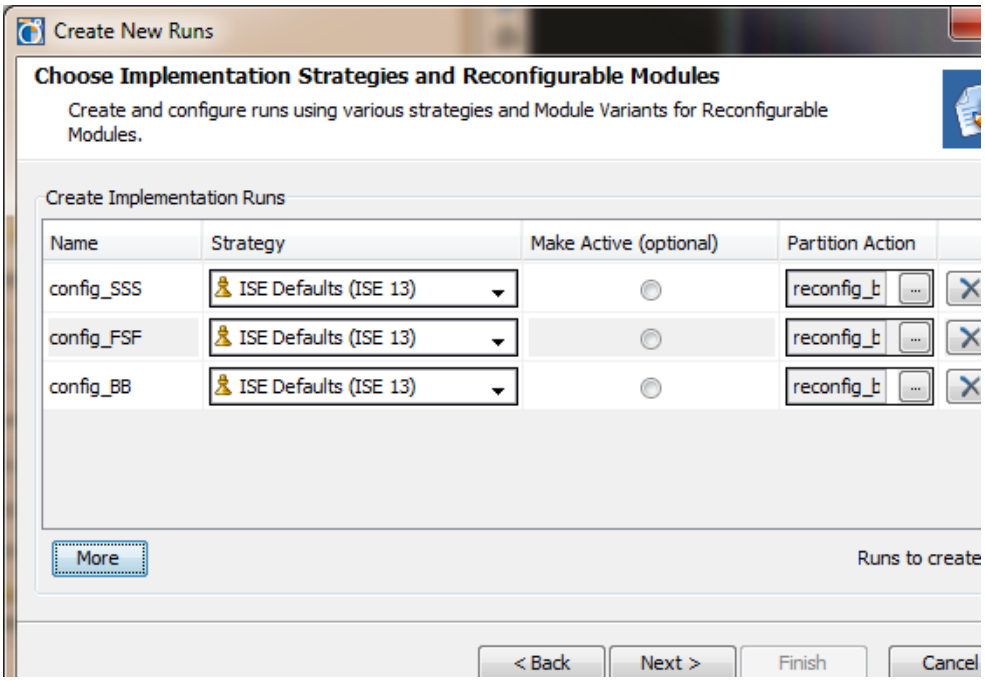

図 4-25 : 複数の run の作成

このサンプル デザインでは、図 4-26 に示すように 4 つのコンフィギュレーションが作成されています。

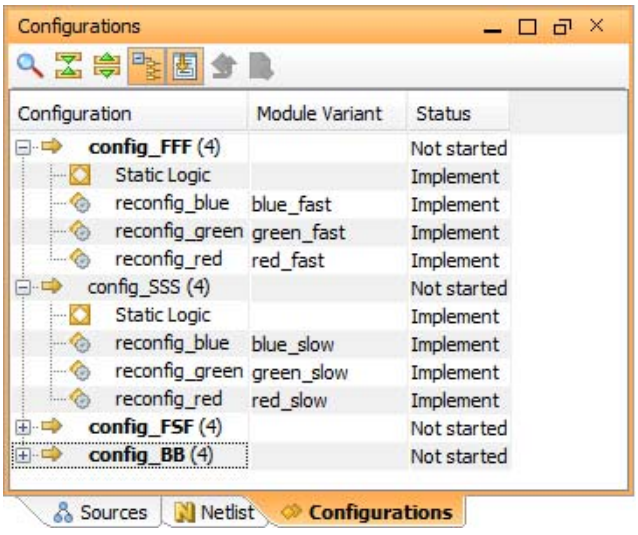

図 4-26 : 最初のコンフィギュレーション

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 に示すように、[Configurations] ビューでコンフィギュレーションを右クリックして [Load Configuration] をクリックし、解析用にネットリストを読み込みます。

   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ラブル モジュールがアクティブになります。

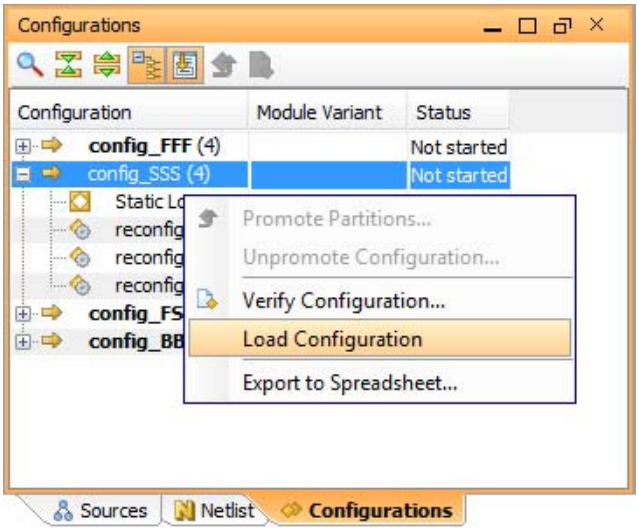

図 4-27 : 既存コンフィギュレーションの読み込み

インプリメンテーションおよび制約を設定したら、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] ビュー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。75 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。

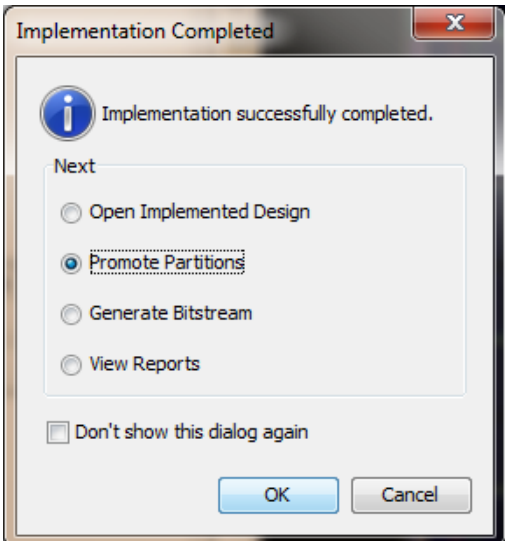

図 4-29 : [Implementation Completed] ダイアログ ボックス

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。

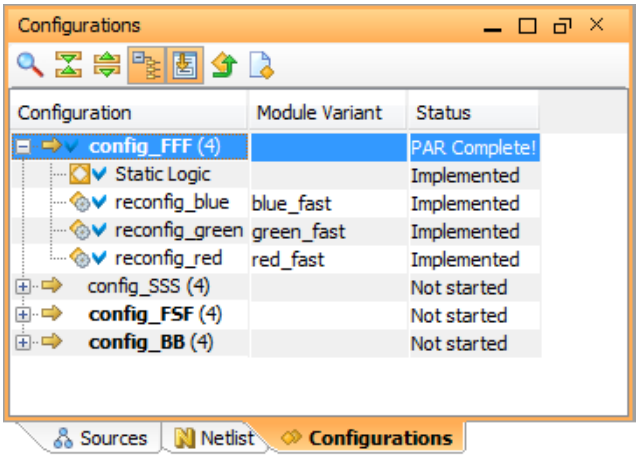

図 4-30 : コンフィギュレーションのプロモート

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ラブル 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ラブル 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の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メッセージが表示されます (76 ページの図 4-31)。

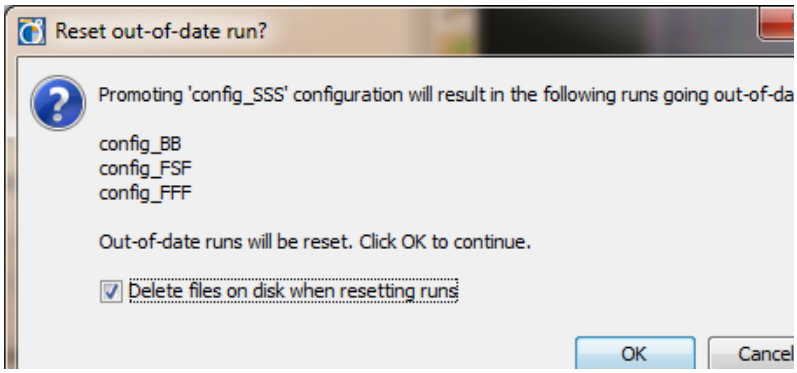

図 4-31 : 最新でないコンフィギュレーションのリセット

run をプロモートすると、ほかのコンフィギュレーションのリコンフィギャラブル 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 77 ページの図 4-33 は、コンフィギュレーション FFF をプロモートしたため、コンフィギュレーション SSS のスタティック ロジックのステータスが Import に設定されたところを示しています。

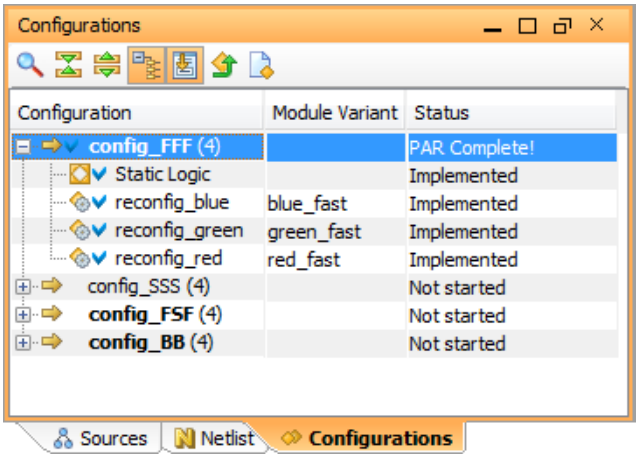

図 4-32 : コンフィギュレーション FFF のプロモート前

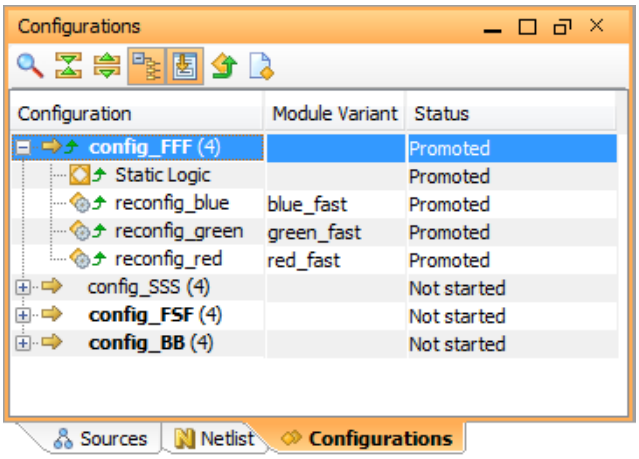

図 4-33 : コンフィギュレーション FFF のプロモート後

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_blue、および reconfig_red が FFF からインポートされ、リコンフィギャラブル モジュール reconfig_green はコンフィギュレーション FFF にインプリメントされていないので、SSS からインポートされます。

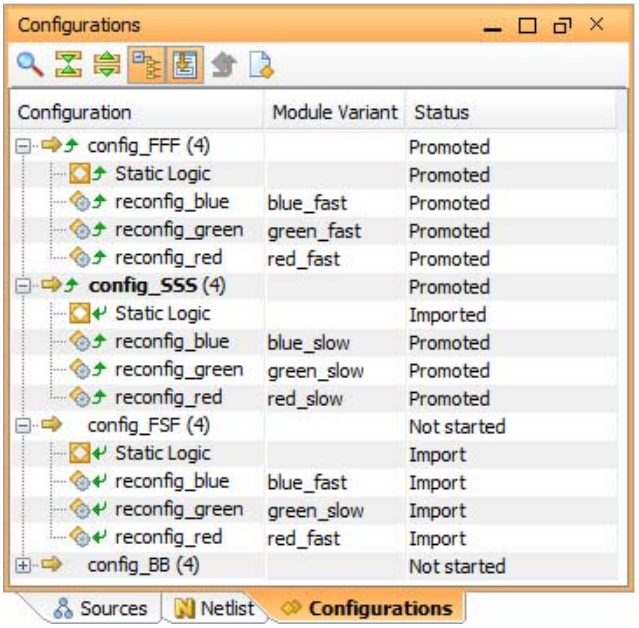

図 4-34 : プロモートされた複数コンフィギュレーション

スタティック ロジックおよびすべてのリコンフィギャラブル 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ている場合、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は FFF と SSS 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。

リコンフィギャラブル 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、異なるリコンフィギャラブル モジュールを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。インプリメントするかインポートするかは、図 4-35 に示す [Specify Partitions] ダイアログ ボックスから設定できます。

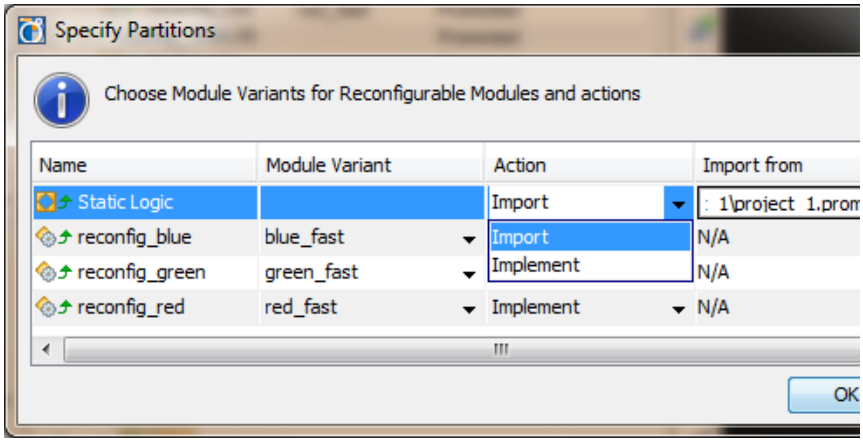

図 4-35 : [Action] の選択 ([Implement] または [Import])

スタティック ロジックおよびリコンフィギャラブル ロジックに対して 77 ページの図 4-34 の [Status] フィールドに示されるステータスは、次のとおりです。

- [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ていますが、インプリメントされていません。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ており、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ています。

- [Promoted]

  モジュールはプロモートされており、ほかのコンフィギュレーションに含まれる重複するモジュールで [Import] に設定されているものには、この結果が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、PlanAhead プロジェクト ディレクトリの次のフォルダーにあります。
<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ーにあります。
<project_name>`.promote\`*<configuration_name>*

このデザイン例の場合、FFF、SSS、FSF、BB に対して XFFF、XSSS、および XBB というディレクトリが作成されます。FSF は、使用されるモジュールが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、プロモートする必要はありません。

コンフィギュレーションの検証

`pr_verify` は、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検証するため、すべてのインプリメント済みコンフィギュレーションに対して呼び出す必要があります。

1. [Configurations] ビューでコンフィギュレーションを右クリックして [Verify Configuration] をクリックし、`pr_verify` を起動します (図 4-36)。これは、すべてのデザイン ルールにしたがっているかどうかを確認するため、パーシャル リコンフィギュレーション デザインで重要な手順です。

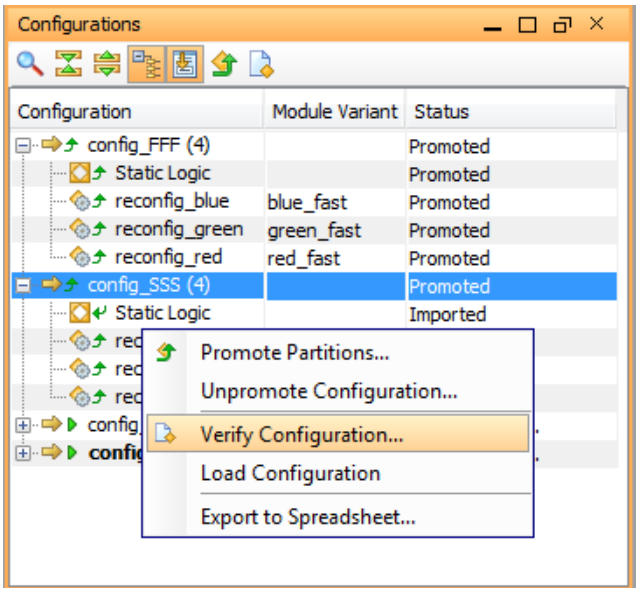

図 4-36 : コンフィギュレーションの検証

2. 検証するコンフィギュレーションを選択するダイアログ ボックスが表示されます (図 4-37)。コンフィギュレーションを選択し、出力ファイルを指定します。

   ハードウェアで問題が発生しないように、すべてのコンフィギュレーションを検証する必要があります。

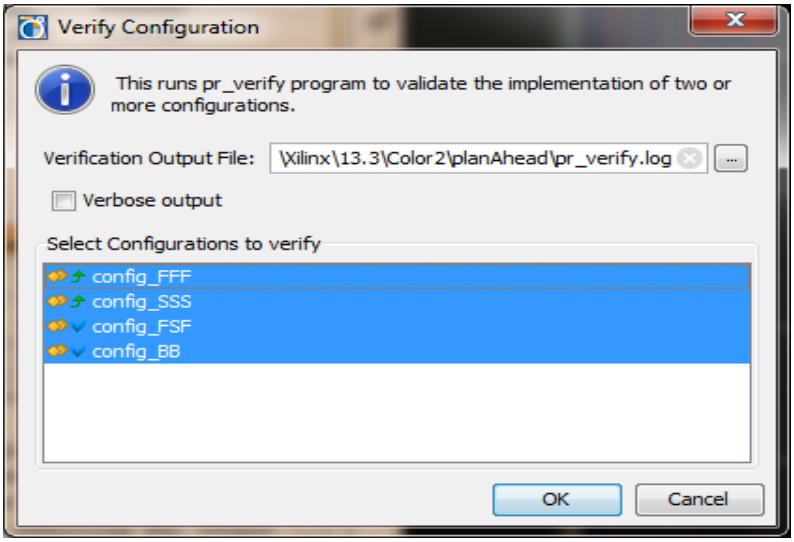

図 4-37 : 検証するコンフィギュレーションの選択

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、BIT ファイルを作成します。

BIT 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されたら、BIT 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)。

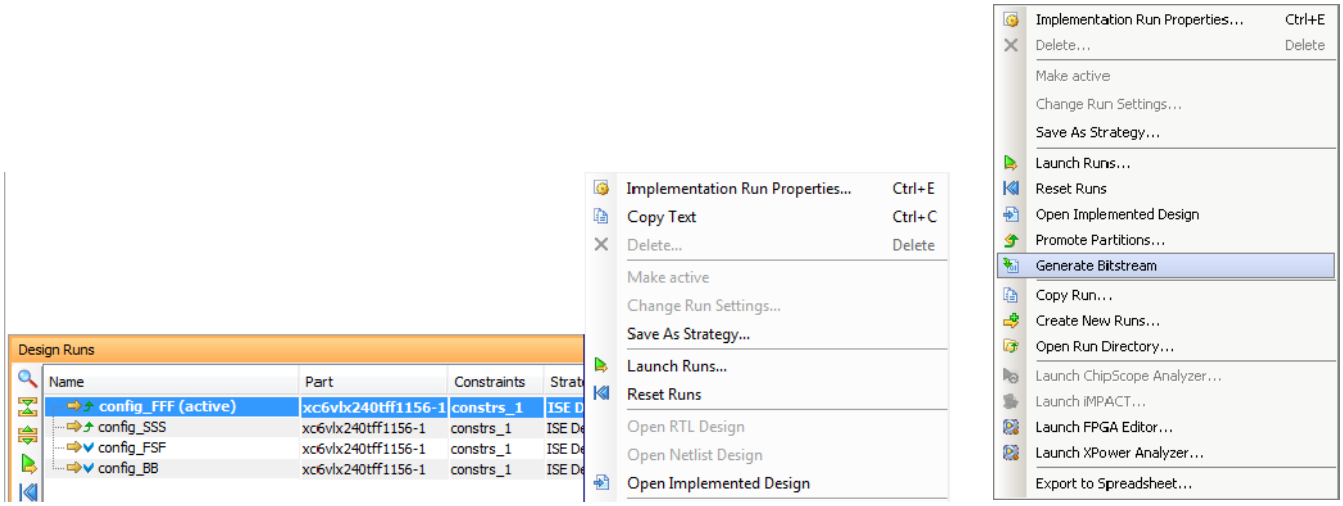

図 4-38 : BIT ファイルの生成

これにより、コンフィギュレーションのフル BIT 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ラブル モジュールのパーシャル BIT ファイルが生成されます。

注記 : 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 コマンドを **-bd** オプションを使用して実行すると、ビットストリーム生成の一部として Data2MEM が実行されます。詳細は、第 7 章の「EDK との連動」を参照してください。

注記 : 暗号化されたパーシャル BIT 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 BIT 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。

このサンプル デザインでは、FFF コンフィギュレーションに対して次の BIT 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 BIT ファイルとパーシャル BIT ファイルすべてを作成することもできます。

フル BIT ファイルとパーシャル BIT 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 の表示と同様に構成されており、スタティック ロジックは `sources_1` ディレクトリの下に、リコンフィギャラブル モジュール ソースは適切な名前の付いたディレクトリの下に保存されます。BIT 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と推奨事項を示します。

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、またカスタム フローに合わせて修正できます。

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 のディストリビューションでは、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。このガイドのスクリプトは、Tcl バージョン 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 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変更します。`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」で説明されているように、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 フォルダーから起動します。

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

data.tcl のフォーマット

data.tcl ファイルは、4 つのセクションに分割できます。data.tcl では、リストまたはアレイ宣言外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が無視されるようにするには、削除するか、リストまたはアレイ外にコメントとして記述してください。

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、必要に応じて変更できます。

- 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を PlanAhead ソフトウェアを使用して実行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、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

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ラブル 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するデバイスを定義します。

`# 3:constraints file`

制約ファイルを指定します。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、リコンフィギャラ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ラブル 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

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ラ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ラブル 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ラブルかどうか指定できます。3 つのリコンフィギャラブル パーティションでは 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

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:Configuration information

このセクションは、各コンフィギュレーションを定義します。

- リコンフィギャラブル モジュールに含まれるもの
- リコンフィギャラブル 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 ディレクトリとしても使用できます。

```
# 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

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、次のとおりです。

- 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。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

  リコンフィギャラブル 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 にし、次に HDL が変更されるまで NO に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、リコンフィギャラブル モジュールを使用して合成を実行するのではなく、この変数を使用してパスを指定できます。この変数は、RUN_RM_SYNTH が NO に設定されている場合、またはスタティックがリコンフィギャラブル 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 のマクロ検索パスとしてコア ネットリストのあるディレクトリを指定します。複数のディレクトリを指定するには、スペースで区切って { } で囲みます。Tcl 命名規則に従い、Windows と Linux の両方で UNIX タイプのスラッシュ (/) を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 オプションを使用できます。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。これら 3 つ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>
```

この例では、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` ファイルに含まれるオプションが使用されます。**`-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。そのため、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、特定のディレクトリ構造が必要です。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 Implementation 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リコンフィギャラブル モジュールは 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。

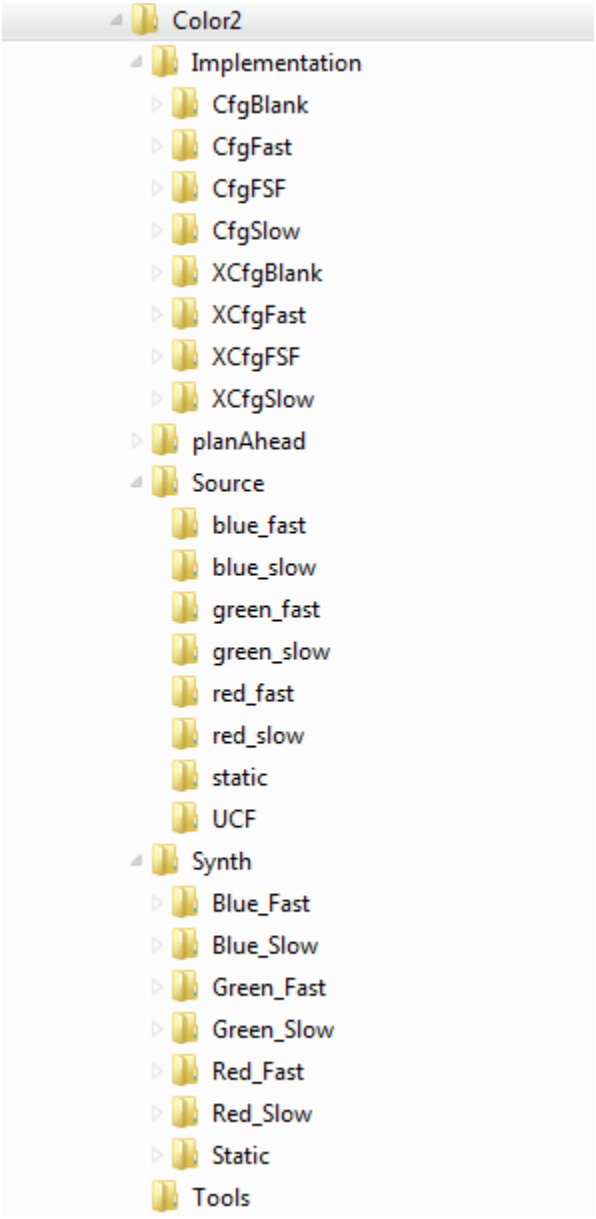

図 5-1 : サンプル スクリプトに必要なディレクトリ構造

これらのディレクトリのいずれかが存在しない場合、内容が生成されたかどうかにかかわらず、スクリプトでプロセスを実行できない可能性があります。

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。

合成リコンフィギャラブル モジュールのディレクトリ

RUN_RM_SYNTH オプションが YES に設定されている場合、リストの各リコンフィギャラブル 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 ファイルでは、リコンフィギャラブル 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) に掲載されているほか、ISE® Design Suite からも生成されます。

このサンプル デザインでは、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 がリコンフィギャラブル 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 に設定されている場合、各リコンフィギャラブル 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

コンフィギュレーションのディレクトリには特定の内容は必要ありませんが、インプリメンテーションを実行す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*>) に基づいて付けられます。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。解析または比較のため実行結果を保存する場合は、別の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 BIT 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、FPGA のアーキテクチャでパーシャル リコンフィギュレーションに有益な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は標準のフル コンフィギュレーションと同じなので、ここではパーシャル リコンフィギュレーションのみに関連する事項を説明します。

パーシャル ビットストリームは、SelectMAP、シリアル、BPI (7 シリーズ デバイスのみ)、SPI (7 シリーズ デバイスのみ)、JTAG、または ICAP (Internal Configuration Access Port) のコンフィギュレーション ポートを使用して読み込むことができます。

SelectMAP、シリアル、または BPI モードを使用してパーシャル BIT 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 16 または 32 を選択するためにのみ必要) 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、リコンフィギャラブル 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部分には送信されることはありません。

パーシャル リコンフィギュレーション コントローラーにより、不揮発性メモリからパーシャル ビットストリームが取り出され、それがコンフィギュレーション ポートに駆動されます。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) または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 コントローラーは、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 プロセス中は割り込みなしで動作します。

内部コンフィギュレーションには、カスタム ステート マシンまたは 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 デザインをデバッグしやすくするため、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、次のユーザー ガイドのコンフィギュレーション インターフェイスの章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)
- 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』(UG470)

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 コントローラー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。

  注記 : スタックド シリコン インターコネクト (SSI) テクノロジでインプリメントされる Virtex-7 デバイスでは、ICAP を使用するパーシャル リコンフィギュレーションはマスター スーパー ロジック領域 (SLR) からのみサポートされます。

- JTAG

  簡単な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。

- スレーブ SelectMAP またはスレーブ シリアル

  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。

- BPI または SPI

  7 シリーズ デバイスは、BPI または SPI コンフィギュレーション インターフェイスを使用してリコンフィギュレーションできます。

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

フル BIT 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、パワー オン リセット後に、PROM から直接またはマイクロプロセッサにより汎用メモリ空間からフル BIT ファイルをダウンロードすることにより、コンフィギュレーションされます。フル BIT ファイルには、FPGA デバイスをリセットし、完成したデザインでコンフィギュレーションし、BIT 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 に、このプロセスを示します。

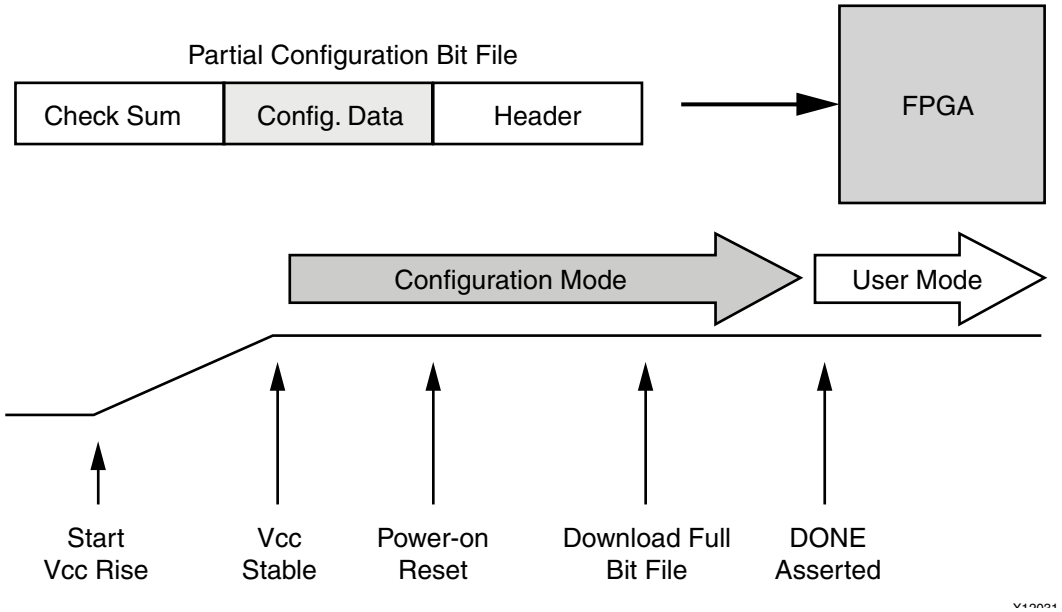

図 6-1 : フル BIT ファイルを使用したコンフィギュレーション

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。BIT ファイルの破損が検出されると、DONE 信号はアサートされず、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 BIT 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る FPGA デバイスは、パーシャル BIT 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、リコンフィギャラ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。

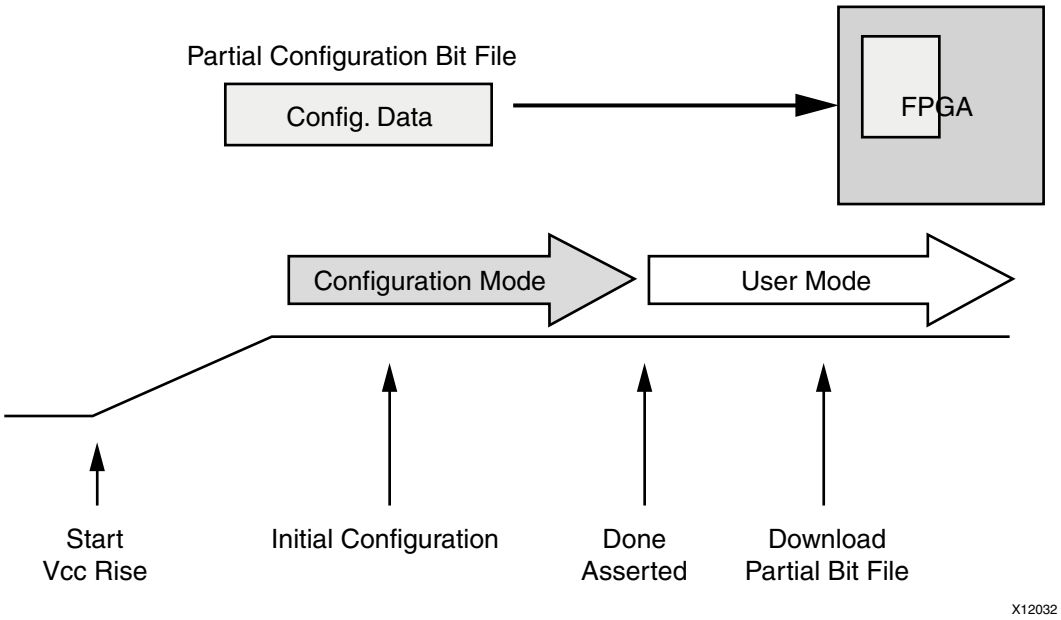

図 6-2 : パーシャル BIT ファイルを使用したコンフィギュレーション

パーシャル BIT ファイルには、ヘッダーがなく、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。BIT 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 BIT 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、ユーザーがモニターする必要があります。パーシャル BIT ファイルの終わりには、BIT ファイルが完全に送信され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。このワードは NO OP コマンドのパディング後に配置されており、DESYNCH に到達すれば、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ていることが確実になります。完全なパーシャル BIT ファイルがコンフィギュレーション ポートに送信されたら、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できます。

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 BIT ファイルはフル BIT 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 BIT 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 BIT ファイルを JTAG、SelectMAP、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 ポートに送信します。FPGA デバイスでは、パーシャル BIT ファイルを受信するための特別な命令がなくても、パーシャル BIT ファイルが正しく処理されます。

通常は、フル BIT 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。これらの信号をアサートすると、パーシャル BIT ファイルではなくフル BIT ファイルの送信が指定されるので、パーシャル BIT ファイルをダウンロードする前にはアサートしないでください。

イネーブル信号を保持したりクロックをディスエーブルにしたりするなど、パーシャル BIT ファイルが送信されることを動作中のデザインに通知する操作は、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、デザイン内で実行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。

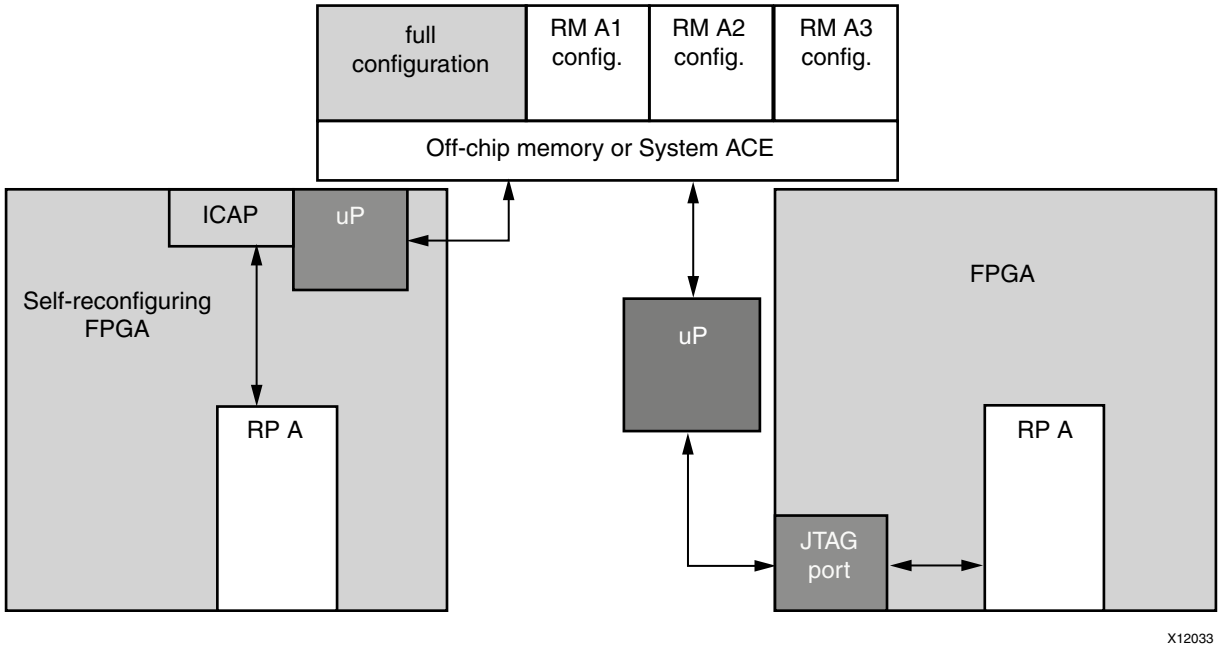

図 6-3 : マイクロプロセッサを使用したコンフィギュレーション

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、ICAP (Internal Configuration Access Port) 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 BIT ファイルを解析および制御できます。パーシャル BIT ファイルは汎用 I/O またはギガビット トランシーバーを介して FPGA デバイスにダウンロードでき、FPGA ファブリックの ICAP に転送されます。

暗号化された 7 シリーズおよび Virtex-6 のパーシャル BIT 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 8 ビット バスで使用する必要があります。暗号化を使用する場合、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することはできません。

パーシャル BIT ファイルの整合性

フル BIT ファイルと比べると、パーシャル BIT 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 BIT 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。エラーが検出された場合は、異なる BIT 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動作を設計者が決定します。

パーシャル BIT ファイルのダウンロードでは、エラー検出と復元にこの手法は使用できません。パーシャル BIT ファイルが読み込まれる際、FPGA デバイスは既にユーザー 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 BIT 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 BIT 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て Low になります。システムで最初のコンフィギュレーション中の CRC エラーを検出するため INIT_B をモニターするようにしている場合、パーシャル リコンフィギュレーション中の CRC エラーでも処理が実行されることに注意してください。FPGA 内に 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ーします。パーシャル BIT ファイルに CRC エラーがあると、ステータス レジスタ (STAT) の CRC_ERROR フラグ (ビット 0) がアサートされます。

考慮する必要のあるパーシャル BIT 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 BIT 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 BIT ファイル (または空のパーシャル BIT ファイル) を読み込んで、破損を修復します。

パーシャル BIT 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 BIT 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、この場合 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 BIT ファイルがローカルに保存されるからです。この場合、BIT ファイルの破損の可能性はほとんどありません。パーシャル BIT ファイルをラジオ リンクを介して送信する場合など、BIT 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 BIT ファイルが ICAP に読み込まれてデバイスがパーシャル 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 BIT 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 BIT 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。

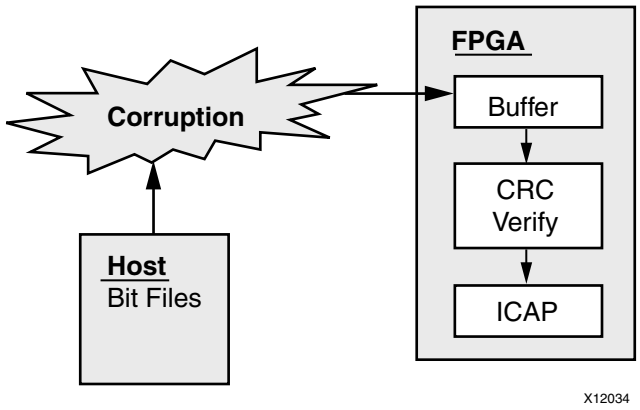

図 6-4 : パーシャル BIT ファイルのエラー検出

パーシャル BIT ファイルには、整合性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 BIT 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、ビルトイン CRC チェックはビットストリームの最後まで実行されないので、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 BIT ファイルの整合性に問題が発生する可能性がある場合にのみ必要です。たとえば、パーシャル BIT ファイルを現場のシステムにリモートでアップロードする場合や、放射線の影響を受ける可能性のある宇宙用アプリケーションなどで必要になります。

図 6-5 に、このようなソリューションの概略図を示します。

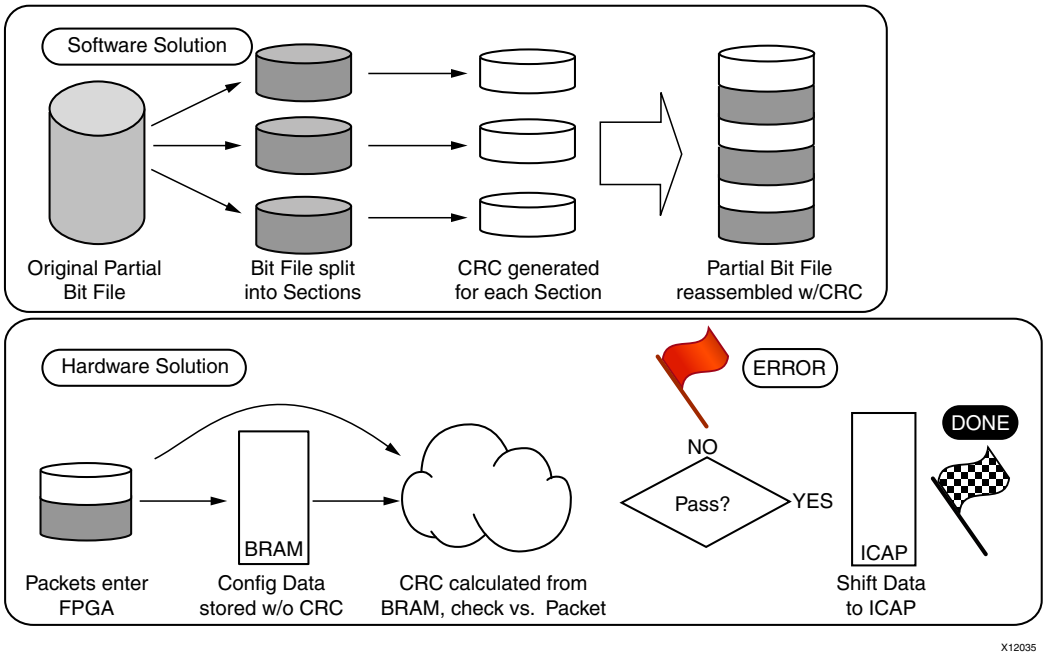

図 6-5 : パーシャル リコンフィギュレーション デザインの CRC チェック

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 および 7 シリーズ FPGA 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 および 7 シリーズ FPGA 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配置されています。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、すべての操作をコンフィギュレーション フレーム全体に対して実行する必要があります。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。7 シリーズ デバイスにはこの情報はありません。

リコンフィギャラブル 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。

- 7 シリーズ FPGA のベース領域は高さ 50 CLB X 幅 1 CLB
- Virtex-6 FPGA のベース領域は高さ 40 CLB X 幅 1 CLB
- Virtex-5 FPGA のベース領域は高さ 20 CLB X 幅 1 CLB
- Virtex-4 FPGA のベース領域は高さ 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、同様のベース領域があります。これらのベース領域のサイズを調べるには、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「リコンフィギャラブル 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」とは意味が異なります。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ラブル構築ブロックであり、これ以上分割できません。1 つのリコンフィギャラブル 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。

リコンフィギャラブル 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ラブル 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 BIT 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 および Kintex™-7 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	100MHz	32 ビット	3.2Gbps
SelectMAP モード	100MHz	32 ビット	3.2Gbps
シリアル モード	100MHz	1 ビット	100Mbps
JTAG	66MHz	1 ビット	66Mbps
BPI (7 シリーズのみ)	100MHz	16 ビット	1.6Gbps
SPI (7 シリーズのみ)	100MHz	4 ビット	400Mbps

PlanAhead の [PR Statistics] タブにレポートされるリコンフィギャラブル 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 BIT ファイルのサイズの正確な予測です。この数値はバイト数なので、ビットストリームのサイズをビット数に変換するには、この数値を 8 倍します。

例 : Virtex-5 デバイス用の小さいパーシャル BIT ファイルがあり、200 個のスライスにまたがる領域が含まれ、リコンフィギャラブル 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るとします。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 (ロービット) 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 BIT ファイルを読み込むのにかかる時間は次のとおりです。

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

つまり、約 73.8 マイクロ秒です。パーシャル BIT 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も、フレームの位置および内容によって多少ずれることもありますが、ほぼ直線的に増加します。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があります。

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の合計時間を算出します。上記の例で作成されたビットストリームのヘッダーは、次のような RBT ヘッダーになります。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 プロセスをモニターするために使用できます。実際、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を受信
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 = 停止を実行中 1 = 実行中の停止なし
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 でパーシャル リコンフィギュレーションのさまざまな地点をキャプチャすることも可能です。

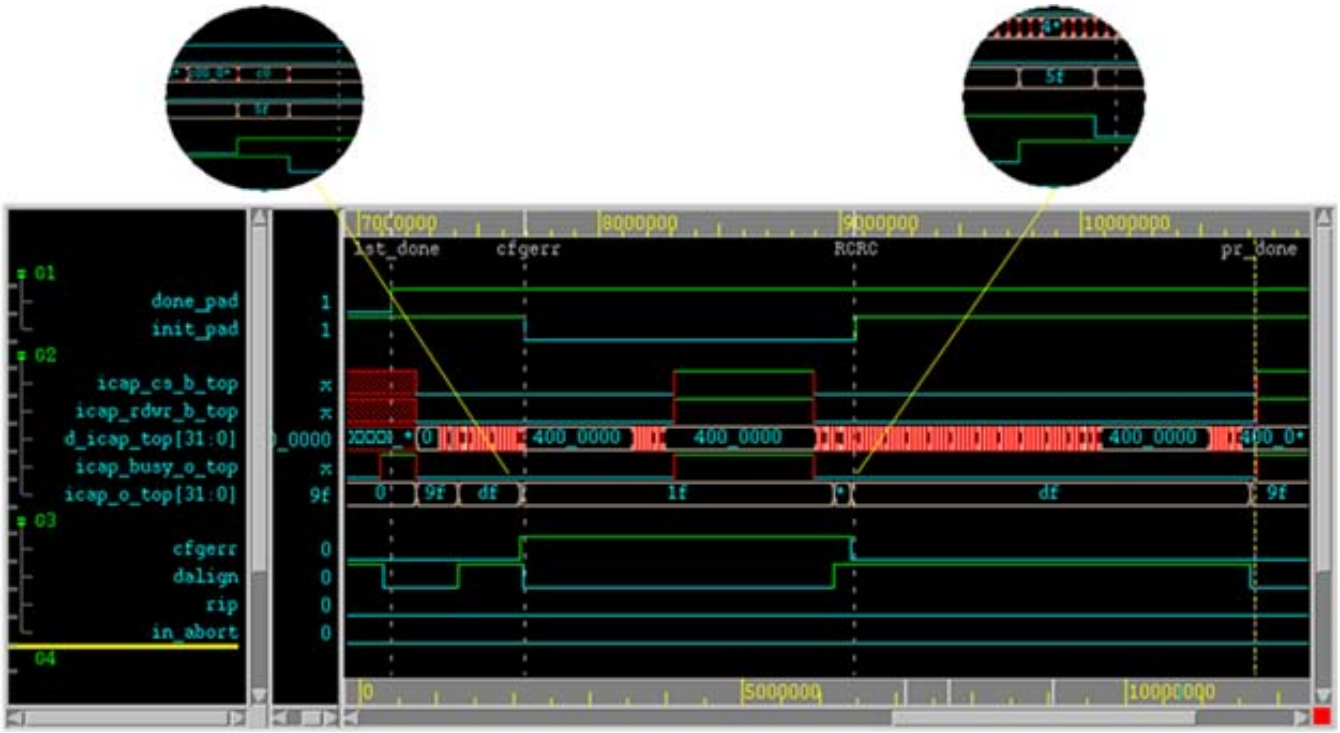

図 6-6 : パーシャル リコンフィギュレーションの ChipScope 表示

ChipScope で表示されるマーカーは、次のとおりです。

- **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 ワードが確認されると、Icap_o_top[31:0] は 0xDF に変わります。
  - CRC エラーが検出されると、Icap_o_top[31:0] は 1 サイクル間 0x5F になった後、0x1F に変わります。
  - 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 に変わります。
  - RCRC コマンドが受信されると、Icap_o_top[31:0] は 0x5F から 0xDF に変わります。

- **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 に変わります。

パーシャル リコンフィギュレーションではパーシャル BIT 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。信頼度の高い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 エラーが発生すると、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命令を発行しようとしますが、これは通常は推奨されません。これを回避するには、第 3 章の「BIT ファイルの生成」の推奨事項に従ってください。

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に特有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略化され、発生する可能性のある問題を回避できます。

デザイン階層

適切なデザイン階層手法を使用することで、FPGA デザインでパーシャル 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 ロジックとリコンフィギャラブル ロジックの境界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、よく知られてい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。パーシャル リコンフィギュレーション デザインでは、これらのデザイン規則に厳密に従う必要はありませんが、規則に従わないことによる悪影響は大きくなります。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。

デザイン階層の詳細は、次を参照してください。

- 『再現可能な結果を活用したデザインの保持』(WP362)
- 『階層デザイン手法ガイド』(UG748)

リコンフィギャラブル モジュール内のデザイン エレメント

すべてのロジックを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、フル デバイス コンフィギュレーションの最後に初期化シーケンスが実行されるようにするため、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があり、リコンフィギャラブル パーティションに配置すべきでないロジックは、次のとおりです。

- クロック変更ブロック (PLL、PMCD、DCM、MMCM、PHASER)
- 一部のクロック バッファー (BUFG、BUFGCTRL、BUFGMUX、BUFHCE、BUFMRCE)

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

- デバイス機能ブロック (BSCAN、CAPTURE、DCIRESET、DNA_PORT、FRAME_ECC、ICAP、KEY_CLEAR、STARTUP、FRAME_ECC、および USR_ACCESS)

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ラブルに関係なく、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。パーティションの境界は、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、配線不可能な状況になることがあるので、階層の境界を選択する際には注意が必要です。

リコンフィギャラブル モジュールの I/O

デバイス ピンはリコンフィギャラブル 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。リコンフィギャラブル 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、OBUF、IOBUF など) を必ず含め、そのポートは名前のみでスタティック ロジックに接続する必要があります。

つまり、I/O 機能はモジュール内に含める必要がありますが、完全なデザインのポート リストは最上位デザイン記述に残します。サブモジュール I/O のその他の要件は、次のとおりです。

- HDL
  - リコンフィギャラブル パーティションに含まれる各リコンフィギャラブル 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。
  - 外部ポートは、すべて最上位で宣言する必要があります。
  - 自動 IOB 挿入は、最上位の合成ではオン、モジュール レベルの合成ではオフにします。
  - 異なる合成ツールでは、サブモジュールのピン宣言の処理方法が異なります。XST および Synplify の例は、この後のサブセクションに示します。
- UCF
  - リコンフィギャラブル 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するように、最上位 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 でリコンフィギャラブル モジュールをフロアプランして、リコンフィギャラブル パーティション内の 1 つのリコンフィギャラブル 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ラブル モジュール (アクティブ リコンフィギャラブル 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ラブル パーティションのそれ以外のリコンフィギャラブル モジュールすべてで、その I/O を新しい位置に制約する必要があります。
  - リコンフィギャラブル領域には、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` ポートが最上位からリコンフィギャラブル 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` への参照を含め、サブモジュールに該当するポートへの 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 バッファーを挿入するかしないかを指定できます。XST を使用した場合に推奨されるフローでは、デフォルトで最上位 I/O が挿入されるようにし、最上位で I/O バッファーを挿入すべきでない I/O に、この属性を使用して I/O バッファーが挿入されないように設定します。この属性は、最上位のポート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;
...
```

- Synopsys
  - Synopsys 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。方法は異なり、すべてのポートは最上位にリストされたままですが、Synopsys にブラック ボックス モジュールの定義で I/O バッファーがサブモジュール (まだ最上位 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、リコンフィギャラブル I/O ロジックは各リコンフィギャラブル モジュールでインスタンシエートする必要があります。次に、Synplicity の Verilog ファイルと VHDL ファイルの例を示します。

```
//Synopsys 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
--Synopsys 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;
...
```

```
--Synopsys 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 への 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ー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ー間にパーティションの境界があると、インプリメンテーション ツールでパーティションの境界を超えるものは認識されないので、I/O ロジックにレジスタが正しく配置されません。

入力および出力レジスタを関連する入力または出力バッファーと同じパーティションに含めることができない場合、次の規則に従っていれば、インプリメンテーション ツールでこの状況が処理されます。

- レジスタに、UCF 制約の IOB=FORCE を指定します。これにより、パーティションの境界を越えてレジスタが I/O バッファーに接続されていることがツールで認識され、レジスタ (ILOGIC/OLOGIC) が I/O ロジックに配置されます。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が発生します。これは、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、レジスタを I/O ロジックに配置する必要がある場合に有益です。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ーとレジスタが同じパーティションにある場合と異なり、レジスタは I/O ロジックに配置されません。
- `IOB=FORCE` 制約は、レジスタのインスタンス名に設定します (`INST "rp_module/out1_ff" IOB=FORCE;`)。この制約をレジスタの出力または入力ネットに設定しないでください。
- リコンフィギャラブル パーティションの `AREA_GROUP` 制約に、入力/出力レジスタを配置する I/O ロジックを含めます。たとえば、レジスタに接続される I/O バッファーに関連する ILOGIC/OLOGIC を含む `RANGE` 値を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ラブル パーティションの `AREA_GROUP` に含まれない場合、そのサイトはツールで使用できず、パーシャル ビットストリームに含まれません。
- リコンフィギャラブル パーティションの出力ポートに `PARTITION_PIN_DIRECT_ROUTE` 制約を設定し、バッファーとレジスタの間にプロキシ ロジックが挿入されないようにします。バッファーとレジスタの間にプロキシ ロジックが挿入されると、レジスタが I/O ロジックにパックされません。また、このように設定するとこのリコンフィギャラブル パーティションに関連するすべてのリコンフィギャラブル 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ラブル パーティションのブラック ボックス リコンフィギャラブル 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、リコンフィギャラブル パーティションを最上位モジュールにインスタンシエートする方法です。各リコンフィギャラブル 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。

リコンフィギャラブル モジュールのサブモジュール

リコンフィギャラブル 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する必要があります。リコンフィギャラブル 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リコンフィギャラブル モジュール ネットリストのルートと同じローカル フォルダーにないと、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し、インプリメントするため、リコンフィギャラブル モジュールすべての内容が必要です。

ほかのネットリスト (IP コア ネットリストなど) をほかのディレクトリから統合する必要がある場合は、ngcbuild ユーティリティを使用すると、リコンフィギャラブル 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できるように 1 つのネットリストにまとめることができます。NGCBuild では、ngdbuild で有効なオプション (-sd および -uc を含む) をすべて使用でき、EDIF または NGC ソース ファイルを入力して制約がアノテートされた 1 つの NGC ファイルを生成できます。

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説明します。

グローバル クロック

特定のリコンフィギャラブル パーティションに含まれるすべてのリコンフィギャラブル モジュール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ラブル 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 に含まれる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、リコンフィギャラブル パーティションのロードがその領域にあるかどうかにかかわらず、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。

どのクロック領域 (どのリコンフィギャブル パーティション) にも前もって配線可能なグローバル クロックの数は、使用されるデバイス ファミリによって異なります。各クロック領域に対するクロック スパインの数もさまざまで、Virtex-4 の場合は最大 8、Virtex-5 の場合は 10、Virtex-6 および 7 シリーズの場合は 12 です。これらの数は、スタティックおよびリコンフィギャブル ロジックの両方が含まれます。たとえば、Virtex-7 デバイスで 1 つのクロック領域にグローバル クロックが 3 つ配線される場合、この領域を含む RP では、これらの 3 つの最上位クロックだけでなく、合計で 9 個のグローバル クロックを使用できます。

図 7-1 の例では、配置前に icap_clk がクロック領域 X0Y1、X0Y2、X0Y3 に配線されるので、スタティック ロジックで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。

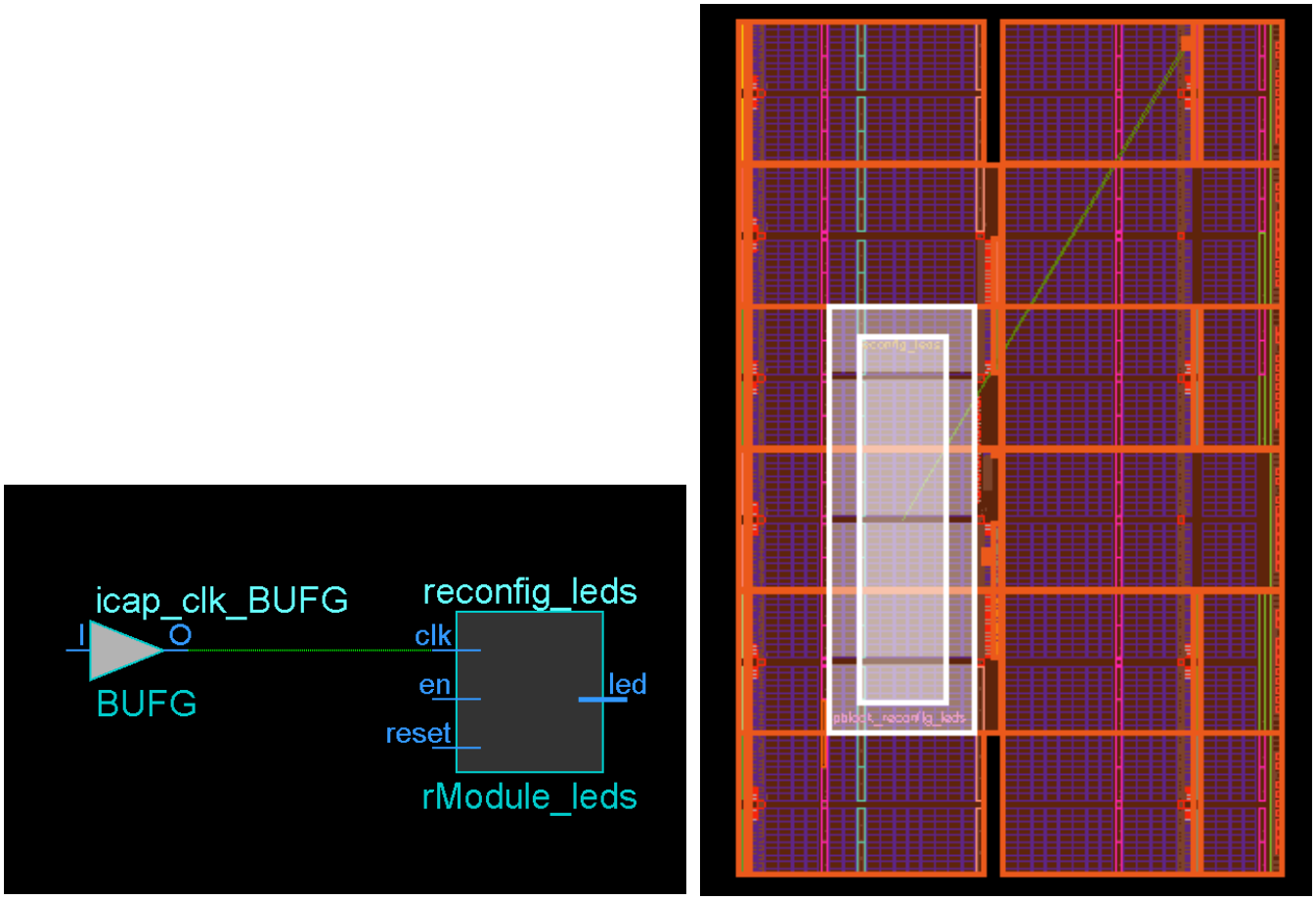

図 7-1 : グローバル クロックからリコンフィギャラブル パーティションへの前配線

リコンフィギャラブル 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ラブル パーティションに接続されるその他すべてのネットと同様に、異なるコンフィギュレーション間で同じ位置に配置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、タイミングが悪化することがあります。

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を、スタティックかリコンフィギャラブルかにかかわらず、同じパーティション内に制約します。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、サポートされる FPGA アーキテクチャによって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 つのクロック領域の一部を含むリコンフィギャラブル 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ラブル パーティションの領域ができるだけクロック領域の境界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設定してください。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

  スタティックとリコンフィギャラブル パーティションにまたがるリージョナル クロックに、プロキシ ロジックが挿入されます。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、タイミングが悪化することがあります。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を、スタティックかリコンフィギャラブルかにかかわらず、同じパーティション内に制約します。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設定してください。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ません。BUFR がスタティック ロジックにあり、そのロードがリコンフィギャラブル パーティションにある場合、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 つのクロック領域の一部を含むリコンフィギャラブル 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 BUFR と内部 BUFR が同じクロック領域で使用される場合、両方をスタティック内に含めるか、同じリコンフィギャラブル パーティション内に含める必要があります。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。

- Kintex-7 および Virtex-7

  7 シリーズの BUFR は、それ以前のアーキテクチャとは異なり、配置されているクロック領域内のロードしか駆動できず、複数領域サポートはありません。BUFR をリコンフィギャラブル パーティションに配置する場合は、そのロードもすべてそのリコンフィギャラブル パーティションに配置する必要があります。スタティック ロジック内の BUFR は、スタティック ロジック内のロードおよびプロキシ ロジックが挿入されていないリコンフィギャラブル パーティション内のロードを駆動でき、リージョナル クロック スパインは前配線されます。

7 シリーズ アーキテクチャには、BUFR に加え、マルチ リージョナル クロック バッファー (BUFMRCE) も含まれています。BUFMRCE は、リコンフィギャラブル パーティションのロードを駆動でき、プロキシ ロジックが挿入されないという点は BUFR と似ていますが、リコンフィギャラブル パーティション内には配置できず、スタティック ロジック内に配置する必要があります。リージョナル クロックの制限については、『7 シリーズ 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 エレメントに配置できます (LUT は削除)。アクティブ High とアクティブ 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、リセット ネットの配線およびタイミングにはそれほど影響せず、LUT の出力はグローバル リソースに配置したままにできます。パーティションでアクティブ Low のリセットを使用するデザインの場合、パーティション内でインバーターを推論させることは可能ですが、取り出したり統合したりすることはできません。そのため、リセットをグローバル リソースに配置することは不可能となるので、デザインが既に密集している場合にリセットのタイミングが悪くなり、配線問題が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。その場合は、アクティブ Low のリセットを最上位信号に割り当て、その新しい信号をデザインで使用します。

例 :

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

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

これにより、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

リコンフィギャラブル ロジックは FPGA デバイスの動作中に変更されるので、パーシャル リコンフィギュレーション中は、リコンフィギャラブル モジュールの出力に接続されるスタティック ロジックでリコンフィギャラブル モジュールからのデータを無視する必要があります。リコンフィギャラブル 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、リコンフィギャラブル 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して、リコンフィギュレーションが完了するまでロジックを分離できます。

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。これは、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 (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にパーシャル 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が使用不可能) のパフォーマンスに対する影響を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でも動作しているので、新規ロジックの使用が開始したときの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 ロジックだけでなく、I/O ロジックでも同じです。

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」で説明したように、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 BIT ファイルの一部であるアドレス指定により決定)、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ラブル 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、デザイン、リビジョン、およびモジュールを示す固有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 : パーシャル リコンフィギュレーションのデータ インテグリティおよびセキュリティ コントローラー』(XAPP887) で説明されるように、暗号化および CRC チェックと同様に、または連動させて使用できます。

BitGen には、デザインのリビジョンを指定する機能があります。-g USR_ACCESS オプションを使用すると、ビットストリームに直接リビジョン ID を入力できます。この ID は USR_ACCESS レジスタに配置され、FPGA ファブリックから同じ名前のライブラリ プリミティブを使用してアクセスできます。パーシャル リコンフィギュレーション デザインでこの値を読み出し、パーシャル ビットストリームの情報と比較して、リビジョンが一致しているかどうかを確認できます。この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「BitGen」の章を参照してください。

リコンフィギャラブル パーティション境界の定義

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 BIT ファイルが作成される際、特定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 ソフトウェアで消費されるリコンフィギャラブル領域の数と対応するビットストリームのサイズの概算がレポートされます。PlanAhead の概算は、2 ～ 3 % の誤差内で算出されます。

パーティションの境界をリコンフィギャラブル フレームの境界と揃える必要はありませんが、揃えると効率的な配置配線結果が得られます。スタティック ロジックが次の条件を両方満たす場合、リコンフィギュレーションされるフレーム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 制約で強制的に内部に配置されていない場合)
- ブロック RAM、分散 (LUT) RAM、SRL などのダイナミック エレメントを含まない

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に配置されると、そのスタティック ロジックの機能そのまま記述し直され、グリッは発生しません。

T 型や L 型など、長方形以外のパーティションも使用できますが、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。パーティションの境界が接触していてもかまいませんが、お勧めしません。分離することで配線の制限に関する問題を回避できることがあります。リコンフィギャラブル パーティションをネストさせたり (パーティション内のパーティション)、重複させたりすることはできません。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 BIT 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 BIT ファイルを生成し、複雑な構造やエラーを回避するには、リコンフィギャラブル パーティションのリコンフィギャラブル 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] ビューの [General] タブで不必要なエレメント タイプをオフにします (69 ページの図 4-17)。

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

リコンフィギャラブル 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ラブル パーティションからのロジックを含めることはできません。複数のリコンフィギャラブル 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ラブル 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、競合が発生することがあります。ソフトウェアは、このよう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 ロジックとリコンフィギャラブル ロジック間のインターフェイスとして定義されます。この定義に特別なロジックやタグは必要ありません。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ラブル 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ラブル 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 で制約を設定できます。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すべてのプロキシ ロジック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。このセクションでは、個々のパーティション ピンおよびパーティション ピンのグループにタイミング制約を設定するための情報も示します。

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。パーティションの境界がこの接続にまたがる場合に、プロキシ ロジックをそのパスに挿入す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 ツールでプロキシ ロジックが挿入されないように設定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、未接続のワイヤが作成されないように、各リコンフィギャラブル モジュール内に配置する必要があります。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ラブル 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ラブル 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 BIT 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 ファイルを指定せずにリコンフィギャラブル 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 ボックスには論理記述で制約が設定されたユーザー ロジックはありませんが、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、リコンフィギャラブル パーティションへのインターフェイスとして各パーティション ピンに LUT1 が挿入されます。これらのプロキシ LUT はリコンフィギャラ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**)、BIT 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、BIT 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 BIT ファイル (フルおよびパーシャル) が圧縮された BIT 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 リコンフィギャラブル 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益です。

モジュール レベルの制約ファイル

デザイン全体を適切に制約するには、デザインのスタティック部分とリコンフィギャラ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、PlanAhead ソフトウェアまたは Tcl スクリプトで使用される最上位ネットリストと メインの UCF の制約で制御されます。I/O ロケーション制約など、リコンフィギャラブル パーティションのすべてのパターンで共有される制約は、最上位 UCF に含める必要があります。

特定のリコンフィギャラブル モジュールにのみ適用される制約は、次の 3 つの方法のいずれかで指定できます。

- 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埋め込むことができるので、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。

- リコンフィギャラブル モジュール ネットリストの UCF 内で指定

  ネットリストをリコンフィギャラブル 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、リコンフィギャラブル　モジュール内のインスタンスへの参照にはインスタンスへのフル階層パスは使用できません。

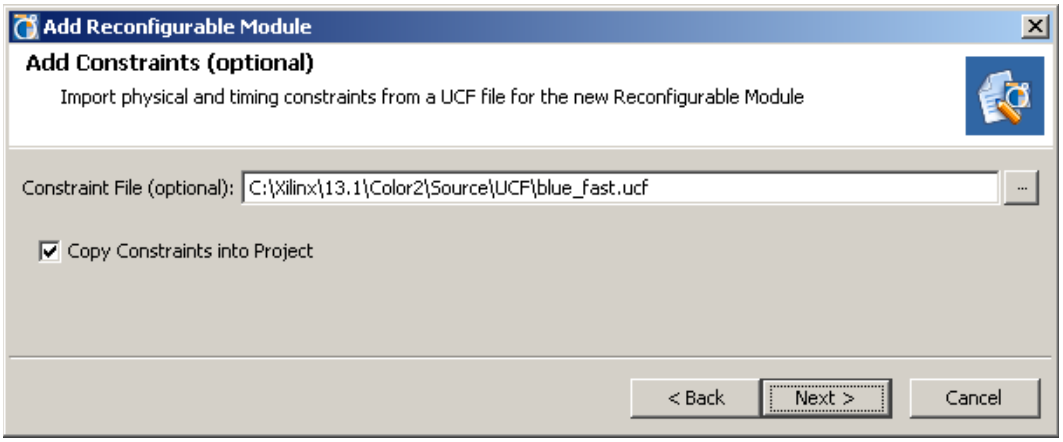

図 7-2 : リコンフィギャラブル モジュール ネットリストと共に指定する UCF ファイル

- `ngcbuild` を使用してリコンフィギャラブル 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、107 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、リコンフィギャラブル モジュール内のインスタンスへの参照にはインスタンスへのフル階層パスは使用できません。

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の障害となあり、リコンフィギャラブル フレームには特定のレイアウト制約が必要です。これらは、リコンフィギャラブル デザインを構築する際に追加される障害です。タイミングおよびエリアのニーズによる追加のオーバーヘッドは、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。

リコンフィギャラブル デザインのコンフィギュレーションを構築する際は、最も困難なコンフィギュレーションを最初にインプリメントしてください。各リコンフィギャラブル パーティションの各リコンフィギャラブル モジュールに十分なリソース (特にブロック RAM、DSP48、I/O などのリソース) が含まれるよう物理領域を選択し、各リコンフィギャラブル パーティションでタイミングまたはエリアにおいて最も困難なリコンフィギャラブル モジュールを選択します。ほかのコンフィギュレーションのリコンフィギャラブル モジュールがそれよりも小型か遅い場合、それらの要件を満たしやすくなります。すべてのリコンフィギャラブル 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で発生した配置配線の問題を解決する方法は、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、検証方法はすべてパーシャル 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。

高速トランシーバーの使用

ザイリンクスの高速トランシーバー (GT11、GTP、GTX) の多くのピンには、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 を汎用 I/O とは別の方法で処理する必要があります。ツールで直接接続を認識させるには、トランシーバー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配置され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、バッファー、トランシーバー 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」
- 「System Generator for DSP および CORE Generator との連動」

ChipScope Pro との連動

ChipScope™ Pro Aalyzer は、ロジック アナライザー、バス アナライザー、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。デザインの計測には、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 Pro 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。リコンフィギャラブル パーティションの境界が変更されなければ、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 Pro 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 ロジックに配置する必要があります。

ILA または VLO コアをリコンフィギャラブル パーティションにインスタンシエートする場合は、別の手法が必要です。フロアプランで指定する領域には、ChipScope Pro コアをインプリメントするのに必要なエレメント (特に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) をすべて含める必要があります。この要件のサイズおよび物理ロケーションによっては、リコンフィギャラブル 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。

ICON コアと ILA または VIO コアを接続する CONTROL バスは、HDL インスタンシエーションを簡潔にするため、双方向として定義します。このバスは、ICON から ILA に送信される 35 個の信号と、ILA から ICON に送信される 1 つの信号から構成されます。リコンフィギャラブル モジュールでは、プロキシ ロジック挿入のため双方向信号は使用できないので、各 ChipScope Pro コアにラッパーを作成してこれらの入出力ポートを入力と出力に変換する必要があります。ChipScope Pro コアをリコンフィギャラブル モジュールに挿入する方法の詳細と、ICON、ILA、および VIO コアの HDL ラッパーの例は、アンサー 42899 を参照してください。

複数領域 (スタティックおよびリコンフィギャラブル) 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、トリガー、または制御バス) をリコンフィギャラブル パーティションから最上位に送信する必要があります。これには、各リコンフィギャラブル モジュールごとにパーティション インターフェイスを変更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 Pro 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変更されます。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。リコンフィギャラブル 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でうまく動作します。リコンフィギャラブル領域の定義 (適切なエレメントが含まれるようにする) とリコンフィギャラブル 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変更ロジック (BUFG、DCM、PLL など) 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ラ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 コアと同様、そのブロックをスタティック ロジックに配置する必要があります。

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ウェブ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`) の最上位ネットリストを指定します。

  リコンフィギャラブル 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を選択できます。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ラブル 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 BIT ファイルを生成する場合は、BIT 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で、EDK で生成した ELF ファイルを指定できます。

  このオプションは、BitGen コマンド ラインからも使用できます。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では、次の項目を考慮してください。

- グローバル クロック バッファーまたはクロック変更ブロック (DCM、MMCM、PLL) を使用していますか。
  - グローバル クロック バッファーおよびクロック変更ブロックは、スタティック ロジックに配置する必要があります。
    - 詳細は、この章の「リコンフィギャラブル 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 バッファーを使用していますか。
  - リージョナル クロック バッファーの要件は、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 ロジックに配置することをお勧めします。
    - 詳細は、この章の「リコンフィギャラブル 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。
- 一緒にパックする必要のあるロジックが同じリコンフィギャラブル　パーティションに配置されていますか。
  - 一緒にパックする必要のあるロジックは、すべて同じリコンフィギャラブル パーティション/リコンフィギャラブル モジュールに配置する必要があります。
    - 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。
  - リコンフィギャラブル 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。
- リコンフィギャラブル モジュールに I/O が含まれますか。
  - リコンフィギャラブル 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「リコンフィギャラブル モジュールの I/O」を参照してください。
- リコンフィギャラブル 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ラブル　パーティションの出力は不定のステートになるので、データが破損しないようデカップリング 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ーが含まれますか。
  - 高速トランシーバーには、特定の要件があります。
    - この要件については、この章の「高速トランシーバーの使用」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、特定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、特定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ウェブ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 BIT 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 BIT 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 BIT 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 : パーシャル リコンフィギュレーションのデータ インテグリティおよびセキュリティ コントローラー』(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。
- リコンフィギャラブル 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。
  - リコンフィギャラブル 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。
- リコンフィギャラブル 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「リコンフィギャラブル パーティション境界の定義」を参照してください。
- すべてのコンフィギュレーションが一貫していることを確認しましたか。
  - pr_verify を使用して、すべてのコンフィギュレーションにインポートされるリソースが一致していることを確認できます。
    - 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。付録 C「その他のリソース」にコンフィギュレーション ユーザー ガイドへのリンクがあります。

XILINX®

付録 A

既知の問題と制限

この付録では、13.3 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をリストします。

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は、アンサー 35019 を参照してください。

既知の問題は、次のとおりです。

- 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、リコンフィギャラブル モジュール、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ないので、合成およびインプリメンテーションで正しいファイル名および設定が適用されていることを、レポート ファイルで確認してください。

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。
- Kintex™-7 および Virtex®-7 デバイスのビットストリームの生成はハードウェア検証中であり、現在のところディスエーブルになっています。
- PlanAhead では、リコンフィギャラブル 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。
- PlanAhead のビットストリーム サイズの予測は、スタックド シリコン インターコネクト (SSI) デバイスでは使用できません。これらのデバイスのパーシャル ビットストリームのサイズは、RBT ファイルの最終情報で確認してください。詳細は、第 6 章の「コンフィギュレーション時間」を参照してください。
- 暗号化されたパーシャル BIT 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 BIT ファイルを生成する方法は、ザイリンクス アプリケーション ノート 『PRC/EPRC : パーシャル リコンフィギュレーションのデータ インテグリティおよびセキュリティ コントローラー』 (XAPP887) を参照してください。

暗号化されたパーシャル BIT ファイルは、7 シリーズおよび 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 Virtex-6 のパーシャル BIT ファイルのパーシャル リコンフィギュレーションには、8 ビット バスの場合にのみ、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することはできません。

- 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 ロジックとリコンフィギャラブル ロジック間のインターフェイスには、単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) ソリュ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 デザインにアップグレードする方法を説明します。

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、第 4 章「PlanAhead サポート」の手順に従ってください。

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ます。バス マクロ機能の一部は、この自動化で置き換えられています。早期アクセスでは同期および非同期のバス マクロの両方が使用できました。最適な階層デザイン手法に従って境界にレジスタを付け、リコンフィギャラブル ロジックをデカップリングするには、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置き換えることができます。

パーティションの境界にはレジスタを付け、これらのレジスタ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領域の動作は不明なので、リコンフィギュレーション中のロジックの出力が使用されると、デザインが破損する可能性があります。このため、境界にイネーブル付きレジスタを付け、リコンフィギュレーション中はリコンフィギャラ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

早期アクセスではさまざまな環境変数を設定する必要がありましたが、ISE 13 ではその必要はありません。早期アクセス用に設定した環境変数は、すべて解除してください。

MODE 制約の廃止

早期アクセスでは、どのエリア グループがリコンフィギャラブルかをツールで指定する必要がありました。これには、UCF に専用の制約 (MODE=RECONFIG) が追加されていましたが、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] オプションで置き換えられており、`xpartition.pxml` に Reconfigurable=TRUE という情報が追加されるようになっています。

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 で指定す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、この後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。リコンフィギャラブル 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 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

早期アクセスでは、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 ISE 13 でもコマンド ラインから実行できますが、ISE 13 ツールでデザインがパーシャル リコンフィギュレーション デザインとして処理されるためには、PMXL ファイルが必要です。このため、Tcl に PMXL ファイルを生成するフローを記述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、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生成しておくことができます。リコンフィギャラブル 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

早期アクセスでは、変換 (Translate) 後の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)

早期アクセスでは、BUFR には複数の制限がありましたが、ネットワークにはバス マクロは必要ありませんでした。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 13.3 PlanAhead に直接アップグレードしないでください。

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
  - 欠点 : 時間がかかり、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
  - 欠点 : 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 マクロを削除せずにバス 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 マクロ出力を置換できます。逆に、複数のバス マクロに関連する制御ロジックがあり、制御ロジックが 2 つの信号のインターフェイスとなるため、これらのバス マクロ タイプを保持するには入力信号と出力信号の両方が必要です。

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 : 変更前の 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 マクロを置換する VHDL コード (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;
```

これは単純な非同期バス 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な制御ロジック (イネーブル、クロック イネーブルなど) に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 : 変更前の 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 マクロを置換する VHDL コード (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、バス マクロと同じ名前で新規に作成したネットリストに置換できます。同期バス マクロはロジックのデカップリングに直接使用できるので、この方法は同期バス マクロに推奨されます。ロジック デザインは同じなので、パーシャル リコンフィギュレーションを再検証するタスクは大幅に簡略化されます。

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、必要な内部代入を定義します。合成中は、I/O バッファーの挿入をディスエーブルにしてください。XST では、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ラブル パーティションの入力であっても出力であっても、スタティック ロジック内に配置されます。

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、作業量が多いように見えますが、デザイン全体でバス マクロが数百個も使用さ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、変換が簡単に短時間で実行できます。これらのバス マクロを再定義すると、モジュールの問題を修正でき、変更がそのタイプのバス マクロすべてで一貫したものになります。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ません。VHDL のフローに従い、Verilog 構文を使用してください。

13.3 での PlanAhead プロジェクトの作成

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。短時間で、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ります。

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 ウェブ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 ウェブ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。

- 『ISE Design Suite を使用したザイリンクス FPGA のパーシャル リコンフィギュレーション』(WP374) : http://japan.xilinx.com/support/documentation/white_papers.htm
- 『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_2/PlanAhead_Tutorial_Partial_Reconfiguration.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_2/PlanAhead_Tutorial_Reconfigurable_Processor.pdf
- 『パーシャル リコンフィギュレーションを使用した PCI Express テクノロジの高速コンフィギュレーション』(XAPP883) : http://japan.xilinx.com/support/documentation/application_notes/xapp883_Fast_Config_PCIe.pdf
- 『PRC/EPRC : パーシャル リコンフィギュレーションのデータ インテグリティおよびセキュリティ コントローラー』(XAPP887) : http://japan.xilinx.com/support/documentation/application_notes/xapp887_PRC_EPRC.pdf
- 『差分ベースのパーシャル リコンフィギュレーション』(XAPP290) : http://japan.xilinx.com/support/documentation/application_notes/xapp290.pdf
- 『階層デザイン手法ガイド』(UG748) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_2/Hierarchical_Design_Methodolgy_Guide.pdf
- 『再現可能な結果を活用したデザインの保持』(WP362) : http://japan.xilinx.com/support/documentation/white_papers/wp362.pdf
- 『PlanAhead ユーザー ガイド』(UG632) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_2/PlanAhead_UserGuide.pdf

- 『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_2/devref.pdf
- 『制約ガイド』(UG625) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_2/cgd.pdf
- 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』(UG470) :
  http://japan.xilinx.com/support/documentation/7_series_user_guides.htm
- 『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/virtex-6_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/virtex-5_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/virtex-4_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_2/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_2/xst_v6s6.pdf