Leonardo Spectrumソフトウェアよる FLEX 10Kデバイスの性能改善方法

1999年 3月 ver. 1 Application Note 102

イントロダクション

プログラマブル・ロジック・デバイス（PLD）の集積度と複雑さが増加すると共に、PLDベンダとEDAサプライヤがデザイン•タスクの効率化を実現するソフトウェアと設計手法を提供することが非常に重要となっています。性能の改善要求が高まると共に、ロジックの最適化が最高レベルで実現されるようにデザインを構築することが必要になります。

VHDLおよびVerilog HDLのコーディング・テクニック、Exemplar Logic社のLeonardo Spectrumソフトウェアのコンストレイント（制約条件）、およびMAX+PLUS® IIソフトウェアのオプションを活用することによって、FLEX® 10Kデバイスにさらに高い性能を実現することができます。これらのツールの活用は、デザイン・フローの簡素化、アルテラのFLEX 10Kデバイスを使用したデザインの最適化、そして既存の高性能プログラマブル・ロジック・ファミリのスピードをさらに向上させるときに役立ちます。このアプリケーション・ノートは、アルテラとExemplar Logic社のツールを使用したデザイン・フローにおいて、FLEX 10Kデバイスの性能をさらに改善させるための手法およびテクニックについて解説したものです。

この資料を適切に理解するためには、FLEX 10Kファミリのアーキテクチャ、アルテラのMAX+PLUS II開発システム、Exemplar Logic社のLeonardo Spectrum合成ツールに対する基本的な理解が必要です。

Leonardo SpectrumソフトウェアをMAX+PLUS IIソフトウェアと併用する方法についての詳細は、「MAX+PLUS II Altera Commitment to Cooperative Engineering Solutions（ACCESS℠）Key Guidelines」を参照してください。この資料は、アルテラのwebサイトhttp://www.altera.comのAltera Technical Support（Atlas℠）のセクション、または*MAX+PLUS IIプログラマブル・ロジック開発ソフトウェアCD-ROM*（version 8.2以降）に収録されています。

目次

このアプリケーション・ノートは、下記の項目について解説します。

A-AN-102-01/J

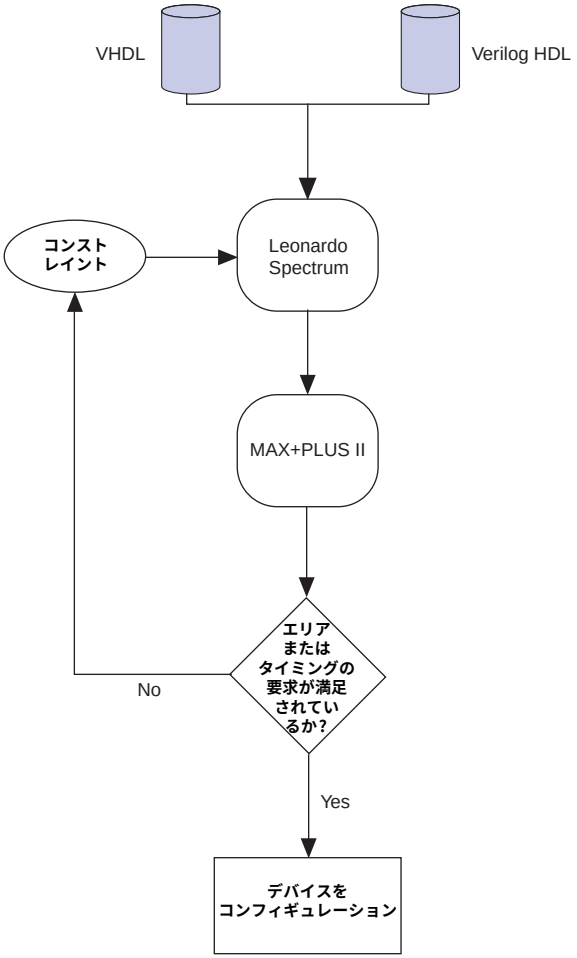
デザイン・フロー

Leonardo SpectrumソフトウェアはVerilog HDLやVHDLで階層構造のデザインを行う場合に広く採用されており、これらのデザインをアルテラのFLEXアーキテクチャにマッピングすることができます。さらに、他社のPLD/FPGAからアルテラのFLEX 10Kに変換する場合にも、Leonardo Spectrumソフトウェアを使用することができます。

デザイン・フローでは、まず最初にLeonardo Spectrumソフトウェアで論理合成を行うためにハードウェア記述言語（HDL）によるデザインの作成を行います。Leonardo Spectrumソフトウェアは、入力されたデザインを中間のデータ構造に変換し、FLEXのアーキテクチャに応じた最適化を実行します。また、この¢*fanin-limited optimization*£と呼ばれる最適化のプロセスでは、組み合わせ回路への入力数が制限されます。次に、Leonardo Spectrumはデザインをルックアップ・テーブル（LUT）にマッピングし、各LUTにlut_functionの論理式を適用します。論理合成後、Leonardo Spectrumが生成したEDIFファイルをMAX+PLUS IIソフトウェアに取り込み、配置・配線します。

図１は、Leonardo SpectrumソフトウェアとMAX+PLUS IIソフトウェアを使用したときに推奨されるデザイン・フローを示したものです。

図1　推奨されるデザイン・フロー



効率的なHDLのデザイン・テクニック

効率的なHDLデザイン・テクニックを活用することで、デザインの簡素化、ロジックの最適化、そして、ロジック遅延の低減が実現され、デザイン全体の性能が改善されます。以下のセクションでは、こうした結果を得るための方法を解説します。

次のセクションで解説されているデザインの手順には、Leonardo Spectrum仕様のシェル・コマンドが使用されています。これらのコマンドは、Leonardo Spectrumソフトウェアのツールバー、またはプルダウン・メニューから使用できます。

HDLの設計手法

適切な設計手法を選択することによって、大幅な性能の向上とデザイン・サイクルの短縮を実現することができます。一般的に、デザインは構造化記述またはビヘイビア記述で作成されます。また、ほとんどのHDLベースのデザインには、トップダウン、またはボトムアップのいずれかの設計手法が採用されています。トップダウンのデザインでは、単独の最適化がトップ・レベルに適用されます。これに対して、ボトム・アップのデザインでは、最適化が各々のサブ・ブロックで実現され、その後にデザインが１つに組み合わされます。このセクションでは双方の手法の利点に関して解説し、論理合成結果の性能を改善するガイドラインについて説明します。

トップダウンの設計手法

トップダウンの設計手法では、最初に最上位（トップ・レベル・ブロック）の機能をデザインします。その後、最下位（ボトム）に達するまで、デザインが複数の下位階層のプリミティブ、またはブロックに分割されます。分割方法やデザインの効率化については、８ページの「デザインの分割方法」を参考にして下さい。

次のセクションでは、Leonardo Spectrumの論理合成ツールを使用した場合のトップダウン•デザインの最適化方法について説明します。

エリア・クリティカル・ブロックの最適化

Leonardo Spectrumソフトウェアの論理合成ツールを使用してトップダウンのデザインを作成する場合、各階層ブロックは50,000ゲート以下にする必要があります。ブロック・サイズを制限して、サブブロックをフラット化することにより、効果的な結果を得ることができます。小さなサブブロックを階層化することで最適化が高速で行われますが、ほとんどの場合はサブブロックをフラット化することにより、さらに性能を向上させ、エリアをさらに縮小することができます。次に、Leonardo Spectrumソフトウェアを使用して、トップダウン・デザインの手法を採用してエリア・クリティカル・ブロックを最適化する手順について説明します。

1. すべてのデザインを読み込み、次のコマンドを使ってエリアの最適化を実行します。

   ```
   optimize -target flex10 -area -chip -effort↵
   ```

2. エリアとタイミングのレポートを生成します。双方のレポートが条件を満足している場合は、デザインが完了します。デザインをさらに最適化する必要がある場合は、ステップ３、４を実行します。

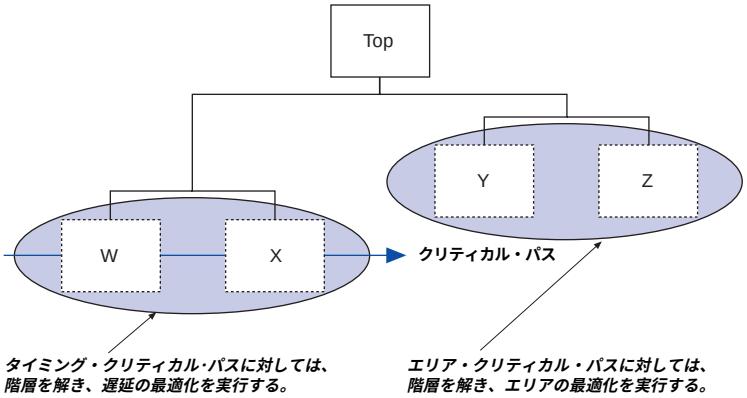
3. 50,000ゲート以下のブロックで、ungroup -all -hierコマンドを使用して、デザインをフラットにします。

4. optimizeコマンドを使用してデザインを再び最適化し、エリアをさらに縮小します。通常、再度最適化されたデザインは最高の結果をもたらしますが、大規模なデザインの場合はこの最適化に非常に長い時間がかかることがあります。

タイミング・クリティカル・ブロックの最適化

タイミングがクリティカルなデザインの場合は、可能な限り小規模なエリアでデザインを実現しながら、要求されるタイミングを達成することが目標となります。この目標を達成するためには、遅延やタイミングの最適化をクリティカル・ブロックだけに制限することが必要です。図２を参照してください。

図２　遅延とタイミングの最適化をクリティカル・ブロックに制限



デザイン・ファイルの例を使用して、タイミング・クリティカル・ブロックに対してのみ遅延とタイミングの最適化を図る方法を以下に説明します。

1. 階層の最適化した後、タイミング・レポートからクリティカル・パスを特定します。

2. groupコマンドを使用して、すべてのタイミング・クリティカル・ブロックを１つの階層ブロックにまとめ、すべての非タイミング・クリティカル・ブロックをべつの階層ブロックにまとめます。

```
LEONARDO{}group w x -inst_name timing_critical↵
LEONARDO{}group y z -inst_name area_critical↵
```

3. ungroup -all -hierコマンドを使用して、timing_criticalブロックの下にあるすべての階層をフラットにします。このコマンドは、トップレベルの階層を維持しながら、下位レベルの階層をフラットにします。

```
LEONARDO{}present_design work.timing_critical↵
LEONARDO{}ungroup -all -hier↵
```

4. timing_criticalのサブブロックに遅延の最適化を実行します。macroスイッチを使用して、各サブブロックのポートにI/Oバッファが挿入されないようにします。

```
LEONARDO{}optimize -target flex10 -delay -effort
  standard -macro↵
```

5. area_criticalのブロックに対してはエリアの最適化を実行せず、ステップ３、４を繰り返します。

```
LEONARDO{}set present_design work.area_critical↵
LEONARDO{}ungroup -all -hier↵
LEONARDO{}optimize -hier -target flex10 -area -
effort
  standard -macro↵
LEONARDO{}set present_design work.top↵
```

6. ネットリスト・ファイルを保存します。

```
LEONARDO{}auto_write -format edif top.edf↵
```

ボトムアップの設計手法

通常、ボトムアップの設計手法は、複数の設計者によってデザインを作成する場合や非常に大規模なデザインを作成するときに採用されます。ボトムアップ・デザインの設計手法を使用する場合は、最初にデザインの構築方法を検討し、次にどのプリミティブが分割ノードとして適用できるかをベースにしてデザインを分割します。最適化は個々のサブブロックに対して行われ、その後にデザインが１つにまとめられます。効率的なデザインの分割方法については、８ページの「デザインの分割方法」を参考にして下さい。

次のセクションでは、Leonardo Spectrum論理合成ツールを使用した場合のボトムアップ・デザインによる最適化方法について解説します。

ブロック内部でのレジスタの配置

Leonardo Spectrumソフトウェアで論理合成されるボトムアップのデザインを作成する場合は、階層の前後のいずれかにレジスタを配置して下さい。階層の前後双方にレジスタを配置すると、レジスタと階層のバウンダリによって最適化が制約されてしまいます。いずれかにレジスタを配置すると、デザインの階層が維持され、最適化の結果に影響を及ぼすこともなく、論理合成に要する時間も大幅に短縮されます。

サブブロックのタイミング・コンストレイント

階層のバウンダリにはレジスタだけが配置されるのが理想ですが、ランダム・ロジックも階層のバウンダリに配置されることが多くあるため、ロジックに適切なコンストレイントが適用される必要があります。サブブロックのタイミングについて詳細な情報がない限り、階層のバウンダリにはクロック周期の半分に等しいコンストレイントを適用してください。バウンダリの両側でタイミング条件が満足されている場合は、接続されたこれらの回路ブロックがタイミング条件を満足していることになります。

FLEXデバイスのサブブロック・ピンに、負荷やドライブのコンストレイントを指定する必要はありません。

中間ネットリスト・ファイルの保存

Leonardo Spectrumソフトウェアでは、次の 2 種類のコマンドでネットリスト・ファイルを保存することができます。

- `auto_write -format edif` *<ファイル名>*`.edif`
- `write -format xdb` *<ファイル名>*`.xdb`

auto_write -format edifのコマンドは、MAX+PLUS IIソフトウェアとのシームレスな統合化機能を活用して、デザインの変更を実行する特定アーキテクチャ用のネットリスト・ポスト・プロセッサを起動します。したがって、中間ネットリスト・ファイルを保存する場合は、auto_write -format edifコマンドではなく、write -formatコマンドを使用してしてください。MAX+PLUS IIの配置配線環境で使用するネットリスト・ファイルを生成するときは、auto_write -format edifのコマンドを使用します。Exemplar Logic社は、中間結果をバイナリ形式のXDBファイルで保存することを推奨しています。これにより、ネットリストとタイミング・データの双方を保存することができます。

デザインの接続

デザインの接続（Stitching）とは、個々のサブブロックに対してボトムアップの最適化を実行した後で、全体のデザインを構築する過程を指します。Leonardo Spectrumソフトウェアは、インスタンス名、ポート名、ビュー名が一致する限り、サブブロックをトップレベルのネットリストに自動的に接続することができます。

また、各デザインはボトムアップで接続される必要があります（最初に最適化されたすべての下位層ブロックがLeonardo Spectrumソフトウェアに読み込まれるようにする）。次に、下記のコマンドを使用して、サブブロック間の接続を行い、トップレベル構成のVHDL、またはVerilog HDLファイルを論理合成します。

```
LEONARDO{}read -format xdb A.xdb B.xdb C.xdb↵
LEONARDO{}read -format vhdl top.vhdl↵
```

各デザインの接続を適切にするには、サブ・ブロックのビュー名とトップレベルにインスタンスされているブロックのビュー名が完全に一致している必要があります。一般的には、問題が生じた場合でも、EDIF、VHDL、またはVerilog HDLのソース・コードを修正することで、問題を簡単に解決することができます。また、add_rename_ruleコマンドを使用して、ビュー名を変更することもできます。

最終的な最適化

デザインが1つにまとめられた時点で、最終的なエリアとタイミングのレポートを生成します。デザインがタイミング仕様を満足していれば、最終的な最適化を実行してチップにI/Oバッファを追加します。最終的な最適化を実行するときは、次のコマンドを使用します。

```
LEONARDO{}optimize -target flex10 -chip -area
  -no_hierarchy↵
```

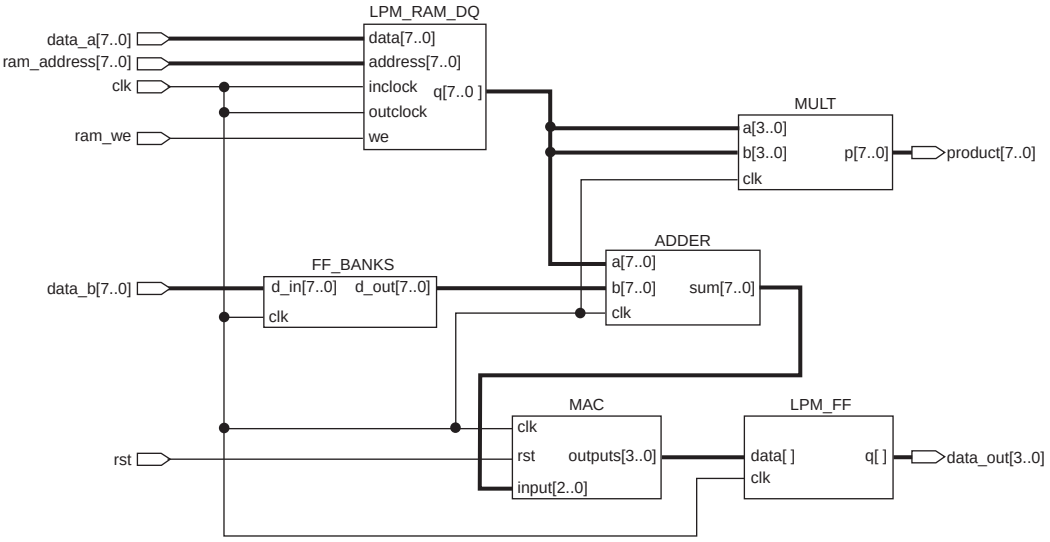
デザインの分割方法

多くのデザインは大規模となっているため、すべての回路機能を１つのデザイン・ファイルで作成することが困難となります。Leonardo Spectrumソフトウェアを使用することで、デザインを複数のファイルで構成し、これらのファイルをひとつの階層でリンクさせることができます。このデザインの構成方法は、デザイン全体の最適化ではなく、各サブデザインごとの最適化やシミュレーションが実行できるようにします。デザインを分割するときは、以下のガイドラインにしたがって行ってください。

- デザインを機能ごとに分割します。この場合、図３に示すようなブロック図や高機能な回路図を使用することで、回路を適切に分割することができます。例えば、ステート・マシン、データ・パス、デコーダ・ロジック、メモリ・エレメント、メガファンクションなどが最適なバウンダリで分割できます。
- 階層化されたブロック間に「グルー・ロジック」は使用しないでください。階層のバウンダリを維持した場合は、グルー・ロジックが階層化されたブロックに接続されません。Leonardo Spectrumソフトウェアがグルー・ロジックを個別に最適化した場合、論理合成の結果が低下する可能性があります。
- ファンクショナル・シミュレーションにおけるデザインのデバッグを容易にするため、サブブロックの規模を10,000から50,000ゲート以内に制限します。
- クロックは、各ブロックごとに１本に制限します。Leonardo Spectrumソフトウェアは、ブロックごとに複数本ある非同期クロックをサポートしていません。
- ステート・マシンを個別の階層ブロックに配置して、最適化に要する時間を短縮し、エンコーディングを効率的にコントロールできるようにします。

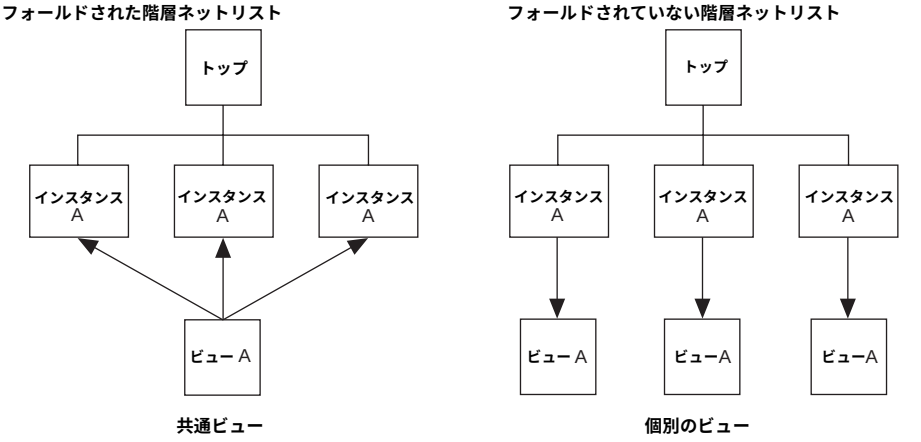
- タイミングがクリティカルなブロックとタイミングがクリティカルでないブロックとを分離します。Leonardo Spectrumソフトウェアは、タイミングとエリアの最適化を個別に実行します。特にエリアや遅延の最適化が行われると思われるブロックには、十分に注意してください。
- クリティカル・パスを１つの階層ブロックに制限します。複数のブロックをグループ化して、クリティカル・パスが単独のブロック内に存在するようにすることができます。
- 論理合成のプロセスを単純化するため、ブロック内のすべての入出力にはレジスタを挿入します。出力にレジスタを挿入することによって、出力に要求される時間を指定する必要がなくなります。さらに、すべてのロジックが同期化されるため、グリッジが発生しません。

図３　階層化されたデザイン



デフォルトの設定では、Leonardo Spectrumソフトウェアがデザインの階層を維持します。実行時間を可能な限り短くするため、ネットリスト・ファイルは「フォールド(fold)」されます。つまり、すべての共通サブブロックは１つのネットリストを参照し、すべてのブロックがワーストケースの条件に合わせて同じ方法で最適化されるようになります。Leonardo Spectrumソフトウェアは、一回しかネットリストを最適化しません。共通になっているサブブロックの２つのインスタンスを別々に最適化したい場合は、ネットリストが「フォールド」されないようにする必要があります。
例えば、１つのブロックをエリアに対して最適化し、２つ目のブロックを遅延に対して最適化することができます。図４は、「フォールド」された階層ネットリストと、「フォールドされていない」階層ネットリストの構成を示したものです。

図４　フォールドされた階層ネットリストとフォールドされていない階層ネットリスト



組み合わせロジック

ある時間での出力がその時間の入力の状態のみで規定されるロジックは、回路の前の状態とは関係なく、組み合わせ回路として記述されます。組み合わせ回路の例としては、デコーダ、マルチプレクサ、アダーなどがあります。

デザインにレジスタを使用することで、論理合成で得られた性能を最適化することができます。FLEXデバイスのシリコン上にはラッチではなく、レジスタが組み込まれています。このため、ラッチを使用したデザインでは、レジスタを使用した場合よりも多くのロジックが生成され、性能も低下します。例えば、MAX+PLUS IIソフトウェアは、１個のラッチの構成にFLEXデバイス内の２個のロジック・エレメント（LE）を使用します。

組み合わせ回路のロジックをデザインするときは、HDLのデザイン記述形式によって意図しないラッチが生成されないように注意する必要があります。例えば、CaseやIfを使用したステートメントが入力の全ての条件をカバーしていない場合は、組み合わせ回路のフィードバックによってラッチが形成されてしまう可能性があります。

シーケンシャル・ロジック

ある時間の出力が、その時間の入力とそれ以前のあらゆる時間の入力とで決定される場合、このロジックはシーケンシャル・ロジック（順序回路）となります。すべてのシーケンシャル回路には１個または複数のレジスタ（フリップフロップ）が含まれている必要があります。FLEXデバイスの各LEには、１個のDタイプのフリップフロップが内蔵されており、回路をパイプライン化した場合でも追加のリソースは必要となりません。デザイン内のロジックを分割したり、デザイン内に特定の値をストアしたい場合でも、追加のLEや配線リソースが使用されることはありません。

意図しないフィードバック・マルチプレサが生成されないように注意する必要があります。If文を使用したときに、可能性のあるすべての入力条件が指定されていないと、フィードバック・マルチプレクサが生成されてしまいます。例えば、図５の青でハイライトされている記述のように、最後にElse節が省略されてしまった場合は、フィードバック・マルチプレクサが生成され、４個のLEが必要になります。最後にElse節を付加した場合は、フィードバック・マルチプレクサが削除され、この機能は３個のLEだけで実現されます。

図５　フィードバック・マルチプレクサを生成するVHDLの記述例

```
LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;

ENTITY seq2 IS
  PORT ( a,b,c,d,clk,rst : IN STD_LOGIC;
         sel             : STD_LOGIC_VECTOR(3 DOWNTO 0);
         oput            : OUT STD_LOGIC);
END seq2;

ARCHITECTURE behave OF seq2 IS
BEGIN
    PROCESS(clk, rst)
    BEGIN
         IF rst = '1' THEN
              oput <= '1';
         ELSIF clk='1' AND clk'EVENT THEN
              IF sel(0) = '1' THEN
                   oput <= a AND b;
              ELSIF sel(1) = '1' THEN
                   oput <= b;
              ELSIF sel(2) = '1' THEN
                   oput <= c;
              ELSIF sel(3) = '1' THEN
                   oput <= d;
              ELSE
                   oput<= 'x'; -- removes feedback
                               -- multiplexer
              END IF;
         END IF;
    END PROCESS;
END behave;
```

内部生成されたゲート付きクロック

内部生成されたクロックの使用は、可能な限り避けることが必要です。内部生成されたクロックがロジックの遅延とクロックのスキューを生成し、FLEXデバイス内の追加配線リソースが必要になることがあります。内部生成されたクロックはグリッジを発生させたり、機能動作の問題を起こす原因になることがあります。さらに、もうひとつのポイントとして、ロジック・アレイ・ブロック（LAB）で使用できるクロックの本数が限定されています。

デザイン内に内部生成されたクロックを使用する場合は、GLOBALのプリミティブを使用して、ゲート付きクロックが多いファン・アウトの内部のグローバル信号のいずれか１本に配置されるようにします。図６で青くハイライトされた箇所は、VHDLのデザイン内にGLOBALのプリミティブを使用して内部生成されたクロックを実現した例を示したものです。

図６　VHDLで内部生成されたゲート付きクロックを実現する方法（1/2）

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY gate IS
  PORT ( a,b       : IN STD_LOGIC;
         c,d       : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         oput      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END gate;

ARCHITECTURE behave OF gate IS
SIGNAL clock       : STD_LOGIC;
SIGNAL gclk        : STD_LOGIC;
SIGNAL count       : STD_LOGIC_VECTOR(3 DOWNTO 0);
ATTRIBUTE black_box : BOOLEAN;

COMPONENT GLOBAL
  PORT ( a_in  : IN STD_LOGIC;
         a_out : OUT STD_LOGIC);
END COMPONENT;

ATTRIBUTE black_box OF GLOBAL: COMPONENT IS TRUE;

BEGIN
  clock <= a AND b;
  clk_buf: GLOBAL PORT MAP (clock, gclk);

  PROCESS(gclk)
  BEGIN
```

図 6　VHDLで内部生成されたゲート付きクロックを実現する方法（2/2）

```
IF gclk='1' AND gclk'EVENT THEN
  count <= c + d;
  END IF;
  END PROCESS;
  oput <= count;
END behave;
```

ステート・マシンの合成

Leonardo Spectrumソフトウェアは、合成のプロセスでステート・マシンをエンコードします。ステート・マシンがいったんエンコードされると、デザインが最適化のプロセスで再度エンコードされることはありません。Leonardo Spectrumソフトウェアでステート・マシンを認識させるためには規定されたVHDLやVerilog HDLのコーディング・スタイルを使用する必要があります。

アルテラは、できるだけステート・マシンを別の階層ブロック内に記述して、他のブロックと分離することを推奨します。この分離によって性能が最適化され、ステート・マシンのエンコーディングの変更が容易になります。

Leonardo Spectrumソフトウェアは、以下のステート・マシンのエンコーディング・スタイルをサポートしています。

- ***バイナリ*** — 必要最小限のフリップフロップでステート・マシンを生成します。バイナリ・ステート・マシンは、タイミング条件が厳しくなく、エリアが重要となるデザインに有効です。
- ***グレイ*** — 状態遷移するときに１つのフリップフロップだけが変化するステート・マシンを生成します。グレイ・エンコーデッド・ステート・マシンは、グリッジを発生させない傾向があります。
- ***ランダム*** — ランダム・ステート・マシン・エンコーディングによるステート・マシンを生成します。ランダム・ステート・マシン・エンコーディングは、他のすべての実現方法では要求された結果が達成されないときにだけ使用されます。
- ***ワン･ホット*** — 各ステートに１個のフリップフロップが割り当てられるステート・マシンを生成します。ワンホット・ステート・マシンは、最高の性能と最短のClock-to-Output遅延を実現します。ただし、ワンホットを使用した場合は、Binaryの場合よりもデザインの規模が大きくなります。

Leonardo Spectrumソフトウェアでは、VHDLのattributeの指定、Verilog HDLのPragmaの使用、または set encoding 変数により、特定のステート・マシン・エンコーディング・スタイルを指示することができます。

VHDLのAttributeの設定

VHDLでステート・マシンのエンコーディング・スタイルを設定するときは、コード内に以下の記述を加えます。

```
--Declare the type_encoding_style attributes
type_encoding_style is (BINARY, ONEHOT, GRAY, RANDOM);
ATTRIBUTE TYPE_ENCODING_STYLE: ONEHOT;

--Declare your state machine enumeration type
type_my_state_type is (s0,s1,s2,s3,s4);

--Set the type_encoding_style of the state type
ATTRIBUTE type_encoding_style of my_state_type is ONEHOT;
```

Verilog HDL Pragmasの使用

Verilog HDLでステート・マシン・エンコーディング・スタイルを設定するときは、Verilog HDLモデル内で、ステート・マシン・モデルの上部に以下のコメント文を付加します。

```
parameter[3:0]//pragma enum state_parameters onehot
idle=4'b0001,
halt=4'b0010
run=4'b0100
stop=4'b1000;
reg[3:0]/*pragma enum state_parameters*/state;
```

上記の例では、最初の行でワンホット・ステート・マシン・エンコーディングを指定しています。ただし、この部分をバイナリ、グレイ、またはランダムに指定することもできます。エンコーディングのデフォルトはワンホットですが、set encoding変数によりデフォルトを変更することができます。

set encoding変数の使用方法

VHDLやVerilog HDLのコードを読み込む前に、set encoding変数を使ってステート・マシンのエンコーディング・スタイルを設定することができます。変数が設定されると、以降のステート・マシンには別のset encoding変数が検出されるまで指定されたエンコーディング・スタイルが適用されます。表 1 にset encoding変数の引数を示します。

表1　set encoding 変数の引数	
引数	内容
binary	ステート・マシンのエンコーディングをバイナリに設定
onehot	ステート・マシンのエンコーディングをワンホットに設定
gray	ステート・マシンのエンコーディングをグレイに設定
random	ステート・マシンのエンコーディングをランダムに設定

VHDL:

```
LEONARDO{}set encoding onehot↵
LEONARDO{}read uart_control_sm.vhdl↵
```

Verilog HDL:

```
LEONARDO{}set encoding binary↵
LEONARDO{}read -format verilog control.v↵
```

☞ VHDLのattributeやVerilog HDLのpragmaは、set encoding 変数よりも優先されます。

モジュールの生成

これまで、データ・パス•ロジックとして一般的に使用される演算回路や不等号回路を論理合成ツールで合成することは困難となっていました。現在では、Exemplar Logic社のmodgenユーティリティとLPM（Libray of Parameterized Module）を使用することにより、特定のアーキテクチャに最適化されたデータ・パスの合成を簡単に行えるようになっています。

modgen vs. アルテラ固有の演算子

ファンクションの生成には、LPMやLeonardo Spectrumのmodgen機能を利用することができます。最も簡単な方法は、VHDLやVerilog HDLの論理演算子シンボルや算術演算子シンボルを用いてLeonardo Spectrumソフトウェア内でこれらの演算子を参照させることです。例えば、sum <= a + bというコードに対しては、modgenが+シンボルを認識し、回路の最適化を行います。別の方法としては、HDLコード内で直接モジュール・セルをインスタンス化する方法があります。セルはEDIFネットリスト・ファイルに引き渡されて、MAX+PLUS IIソフトウェアに取り込まれます。

modgen

Leonardo Spectrumソフトウェアは、VHDLやVerilog HDLで使われる算術演算子や不等号演算子を、多様なデバイス固有のアーキテクチャに実現することができます。この実現方法では、ファンクションがターゲットとなるアーキテクチャに最適化されるため、一般的に合成結果が小規模で高速となり、コンパイル時間も短縮されます。表2に、アルテラFLEXアーキテクチャに対応した演算子を示します。

表2　対応する演算子

演算子タイプ	シンボル	定　義
不等号	=	等しい
	/=	等しくない
	<	より小さい
	<=	以下
	>	より大きい
	>=	以上または等しい
算術演算	+	加算
	−	減算
	*	乗算
その他の機能	N/A	カウンタ（アップ / ダウン、ローダブル） RAM インクリメンタ ディクリメンタ 絶対値 単項マイナス

それぞれのアーキテクチャに対応したモジュール・ジェネレータは、可能な限りキャリー・チェインやカスケード・チェインなどの専用のハードウェア・リソースを使用します。このため、modgenで加算、減算、カウンタ、不等号記号のような演算子を使用した場合は、一般的にエリアが小さくなり、遅延も小さくなります。

以下に、modgenが各演算子をFLEX 10Kデバイスにどのように実現するかを示します。

- アダーは、FLEXデバイスの専用のキャリー・チェインを使って実現されます。これにより、非常に高速なキャリーの伝搬が実現され、タイミング性能が大幅に向上します。
- カウンタは、FLEX 10Kデバイスのカウンタ・モードを使用して実現されます。これにより、デザインが小さくなり、高速となります。
- RAMは、FLEX 10Kデバイスの専用RAMブロックに実現されます。

ユーザ定義スイッチを使用することによって、modgenに演算子を遅延またはエリアのどちらに最適化させるかをコントロールすることができます。このスイッチは、Leonardo SpectrumのOptimizeダイアログ・ボックスで設定します。Optimizeダイアログ・ボックスで選択できる項目は、使用するアーキテクチャやフォーマットに依存します。

Optimizeダイアログ・ボックスで*Use Technology Specific Module Generation Library* オプションをオンにすると、表３に示すオプションにより、最適化の方法をコントロールすることができます。このオプションがオフの場合、modgenはデフォルトのモジュール生成ライブラリを使用します。

表３　Leonardo Spectrumオプション

オプション	説明	設定（オン/オフ）	インタラクティブ・シェル・オプション・レベル３	バッチ・モード
Auto	エリアの最適化を実行するときは最小のエリアになるようにし、遅延の最適化を実行するときは、遅延が最小になるように実現する。	オン *(1)*	`set modgen_select auto`	`-select_modgen=auto`
Smallest	実現可能なもっともコンパクトなサイズで実現する。	オフ	`set modgen_select smallest`	`-select_modgen=smallest`
Small	コンパクトなサイズで実現する。	オフ	`set modgen_select small`	`-select_modgen=small`
Fast	高速になるように実現する。	オフ	`set modgen_select fast`	`-select_modgen=fast`
Fastest	最も高速になるように実現する。	オフ	`set modgen_select fastest`	`-select_modgen=fastest`

注：
(1)　グラフィカル・ユーザ・インタフェース（GUI）でのデフォルト・オプションです。インタラクティブ・シェルやバッチ・モード・オプションも入力できます。

LPMファンクション

アルテラは、LPMを使用して演算回路やRAM、カウンタ・ロジックなどを生成することができるモジュール・ゼネレータを提供しています。これらのモジュール・ゼネレータは、アルテラFLEXアーキテクチャに最適化されています。

Leonardo Spectrumによる論理合成では、LPMファンクションがブラック・ボックスとしてコンパイルされます。LPMファンクションのパラメータ値は、Verilog HDLのメタ・コメント、またはVHDLの属性として、EDIFファイルやテキスト・デザイン・ファイル（.tdf）に受け渡されます。

図 7 は、lpm_counterファンクションを使用して、カウンタを作成したVHDLのデザイン例を示したものです。

図 7　VHDLでLPMファンクションをインスタンス化した記述例

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY count4 IS
PORT (d       : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      clk, clr: IN STD_LOGIC;
      result  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END count4;

ARCHITECTURE lpm OF count4 IS

COMPONENT lpm_counter
   GENERIC (lpm_width  : POSITIVE;
            lpm_type   : STRING := "lpm_counter");
   PORT    (data       : IN STD_LOGIC_VECTOR(lpm_width-1 DOWNTO 0);
            clock      : IN STD_LOGIC;
            aclr       : IN STD_LOGIC;
            q          : OUT STD_LOGIC_VECTOR(lpm_width-1 DOWNTO 0));
END COMPONENT;

BEGIN
  u1: lpm_counter
       GENERIC MAP (lpm_width=>4);
       PORT MAP (data=>d, aclr=>clr, clock=>clk, q=>result);
END lpm;
```

図 8 はlpm_ram_dqファンクションを使用して、RAMブロックを作成したVerilog HDLのデザイン例を示したものです。

図 8　Verilog HDLでLPMファンクションをインスタンス化した記述例

```
module test_lpm_ram (q, data, inclock, outclock, we, address);

  parameter width = 8;
  parameter widthad = 2;
  parameter numwords = 4;

  input [width-1:0] data;
  input [widthad-1:0] address;
  input inclock, outclock, we;
  output [width-1:0] q;

  lpm_ram_dq instance_r (q, data, we, inclock, outclock, address);
  defparam instance_r.lpm_width = width;
  defparam instance_r.lpm_numwords = numwords;
  defparam instance_r.lpm_widthad = widthad;

endmodule
```

LPMファンクションに対するMAX+PLUS IIのサポート状況は、MAX+PLUS IIのヘルプ機能を使用して確認することができます。

メモリ・ブロックの推論

Leonardo Spectrumソフトウェアは、レジスタ・トランスファ・レベル（RTL）コードから、RAMブロックを推論することができます。RAMブロックが2次元のアレイとしてモデル化されている場合は、Leonardo Spectrumソフトウェアがこのファンクションを認識し、ネットリストにブラック・ボックスを挿入し、プロパティ情報を添付します。アルテラのMAX+PLUS IIソフトウェアは、このプロパティ情報を認識してデザインに適切なRAMブロックを挿入します。

予備ライブラリを生成したり、RAMのセル・タイミングを定義することにより、タイミング・アークを作成することができます。タイミング・アークの生成方法に関する詳細は、Exemplar Logic社のwebサイトhttp://www.exmplar.com/supportを参照してください。

非同期のフィードバック・ループ

タイミング・ループが発生すると、Leonardo Spectrumソフトウェアはループを抜けて次に移行する前に各ループに5,000回の評価を行うようになっているため、タイミング解析の速度が極端に遅くなる可能性があります。このため、アルテラは各セッションの先頭で、delay_break_loops変数をTRUEに再定義することを推奨しています。この定義を行うことにより、自動的にループが絶たれ、タイミング・レポートにワーニング・メッセージが記述されます。

Leonardo Spectrumで**コンストレイントを設定する方法**

Leonardo Spectrumソフトウェアでは簡単にコンストレイントを設定できます。簡単にデザインの動作周波数をコンストレイントに指定できたり、またパワフルにフリップフロップ間のマルチサイクル・パスをコンストレイントに設定できます。

コンストレイントは、Leonardo Spectrumソフトウェアにデザインが読み込まれた後で、最適化が実施される前に適用される必要があります。Leonardo Spectrumソフトウェアは、直感的に得られるデフォルト条件を想定します。ただし、少なくとも、クロックの定義、入力ポートへの到達時間、出力ポートに要求される時間を定義しておく必要があります。

☞ デザインに過度なコンストレイントを与えないように注意してください。過度なコンストレイントを与えると、デザイン・サイズの増加や、最適化に要する時間の増大のような予期しない影響が現れることがあります。

次のセクションでは、Leonardo Spectrumソフトウェアで性能を最適化するときに使用されるコンストレイントとその他のオプションについて解説します。

☞ 次のセクションでは、デザインに対するコンストレイントの設定に、Leonardo Spectrumソフトウェア固有のシェル・コマンドが使用されています。Leonardo Spectrumソフトウェアのコンストレイント・エディタを使用することで、これらのすべてのコマンドが使用可能となっています。

デザインにコンストレイントを与える最も簡単な方法は、グローバルなタイミング・コンストレイントを指定することです。例えば、デザインの最高動作周波数を20MHzに設定するときは、以下のコマンドを使用してクロック周期を50nsに設定します。

```
LEONARDO{} set register2register 50↵
```

同様に、以下のコマンドを使用して、デザイン内の他の最大遅延値を指定することができます。

```
LEONARDO{}↵
  set input2register 50↵
  set input2output 50↵
  set register2output 50↵
```

クロックのコンストレイント

クロックはレジスタ間のタイミングを定義します（図９参照）。クロックを定義しないと、すべてのレジスタにコンストレイントが与えられないため、レジスタ間のすべての組み合わせ回路がタイミングの最適化時に無視されます。クロックを定義すると、レジスタ間の組み合わせ回路に対して１クロック周期のコンストレイントが効率的に適用されることになります。

図９　クロックのコンストレイントを示すデザイン例

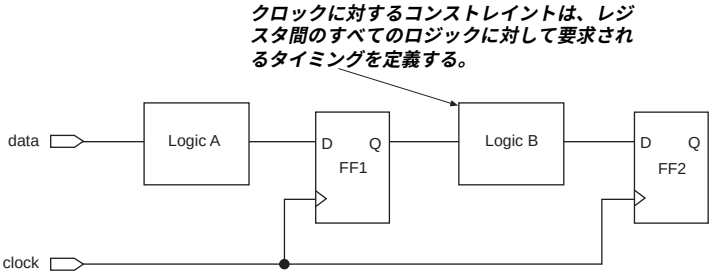


図９では、ロジックBが１クロック周期になるようなコンストレイントが適用されています。このため、クロック周期のコンストレイントが50nsになっている場合は、ロジックBがFF2に対して約50nsのセットアップを確保してタイミングを満足させる必要があります。

Leonardo Spectrumソフトウェアでは、次の３つの基本的なコマンドを使用してクロックのコンストレイントを定義します。

```
clock_cycle <clock period> <primary input port> ↵
pulse_width <clock pulse width> <primary input port> ↵
clock_offset <clock offset> <primary input port> ↵
```

デフォルトでは、クロックのネットワークが理想的（遅延がない）になっている状態が想定されます。したがって、クロックはすべてのフリップフロップに同時に到達します。クロック・ネットワークが伝搬遅延を持つように変更するときは、propagate_clock_delayの変数をTRUEに設定します。図10にサンプル・デザインのタイミング波形を示します。

図10　タイミング波形の例

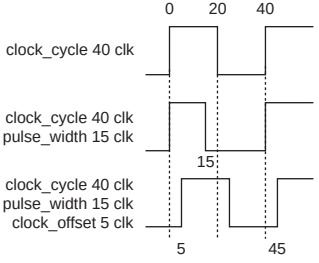


図10では、１番上の波形が40nsのクロック周期で、clkポートに入力されています。デフォルトのデューティ・サイクルが50%になっているため、この波形のクロック・パルス幅は20nsになります。２番目の波形のパルス幅は15nsになっています。３番目の波形はクロック・オフセットを示しており、相対的なクロック・スキューを指定するときに有効です。

入力到達時間

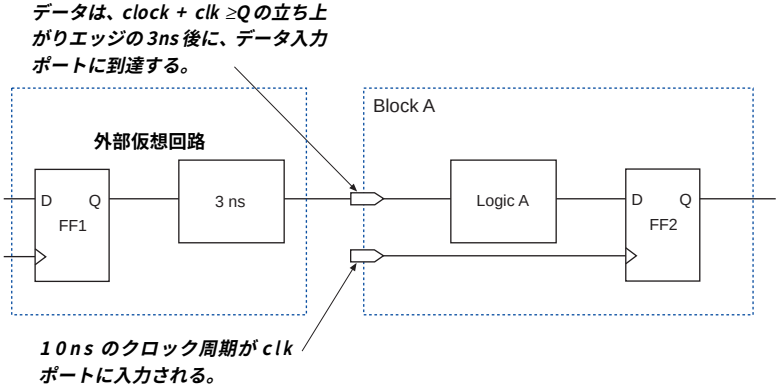
入力到達時間（Arrival Time）は、外部ロジックから合成されたデザインの入力ポートに到達するまでの最大遅延時間を規定したものです。入力到達時間は、arrival_timeコマンドを使用して指定することができます。図11のブロックAで3nsの入力到達時間を設定するときは、以下のコマンドを使用します。

```
LEONARDO{}arrival_time 3 data_min↵
```

図11のクロック周期が10nsになっているとき、このタイミング要求を満足させるためには、FF2のセットアップ•タイムとして与えられるロジックAの組み合わせ回路の遅延時間が7nsになる必要があります。

すべての入力到達時間はゼロ時間から始まり、特定のクロック・エッジに関連した指定を行うことはできません。特定のクロック・エッジに対する入力到達時間を調節するときは、クロックのオフセットを到達時間に加算します。

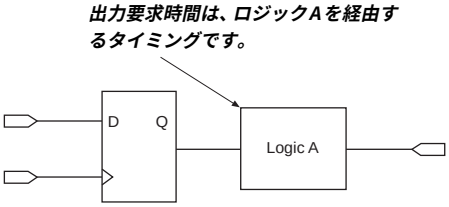
図11　入力到達時間のコンストレイントを持つデザイン例



出力要求時間

出力要求時間（Output Required Time）は、データが出力ポートに出力されるまでに必要な時間として指定されます（図12参照）。また、この時間は常にタイム・ゼロに対して指定されます（出力要求時間を特定のクロック・エッジを基準に指定することはできません）。

図12　出力要求時間コンストレイントを示すデザイン例



出力要求時間は、以下のコマンドを使用して指定することができます。

```
LEONARDO{}required_time <integer> portname↵
```

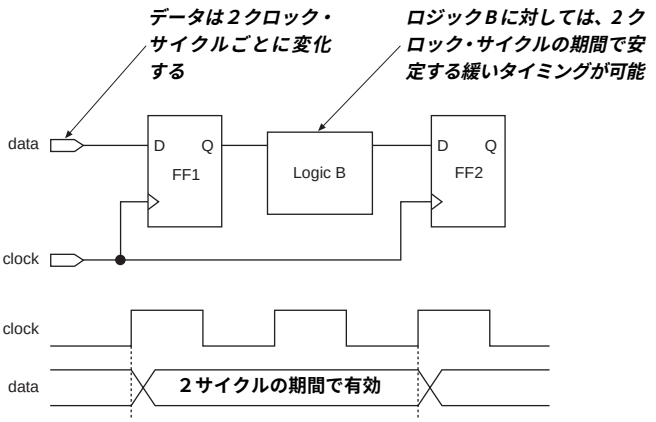
マルチ・サイクル・パスのコンストレイント

Leonardo Spectrum ソフトウェアのバージョン4.2以上では、マルチ・サイクルのコマンドを使用して個別のパスに１サイクル以上のコンストレイントを適用することが可能です。例えば、図13のロジックBに２クロック・サイクルのコンストレイントを与えるときは、以下のコマンドを使用します。

```
LEONARDO{}set_multicycle_path -from{FF1} -to{FF2}
  -value 2↵
```

マルチ・サイクルのコンストレイントを指定すると、タイミング解析に要する時間が長くなるため、その適用には十分注意する必要があります。少ない箇所へのマルチ・サイクル・コンストレイントの適用であれば大きな影響はありませんが、多くなるほどタイミングの最適化に要する時間が大幅に増加します。

図13　マルチ・サイクル・コンストレイントを示すデザイン例



Falseパスのコンストレイント

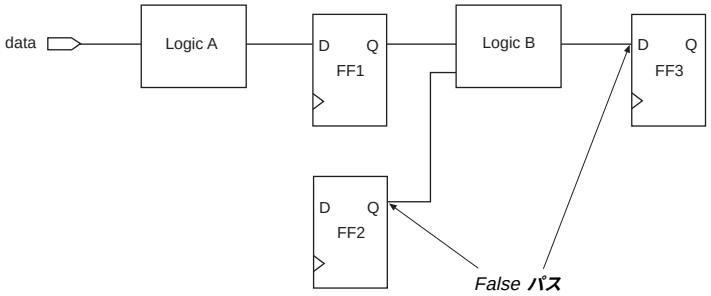
Falseパスは、Leonardo Spectrumソフトウェアのタイミング最適化で無視されるパスです。マルチ・サイクル・コマンドを使用して、パスをfalseに指定することができます。例えば、以下のコマンドを使用して図14のFF2からFF3までのパスをfalseに指定することができます。

```
LEONARDO{}set_multicycle_path -value 1000 -from{FF2}
  -to {FF3}↵
```

このコマンドは、FF2とFF3の間のパスに1,000クロック・サイクルのコンストレイントを適用しています。ロジックBは1,000サイクル以上の遅延を必要としないと考えられるため、このパスはタイミングの最適化とタイミング解析から除外されます。

☞ 多数のfalseパスを指定しないでください。これによってタイミング解析の実行時間が増加する可能性があります。

図14　Falseパスを示すデザイン例

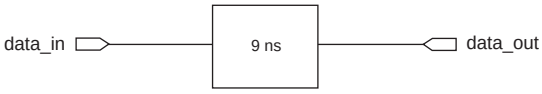


純粋な組み合わせ回路のデザイン

純粋な組み合わせ回路のデザインには、クロックが使用されません。このようなデザインには、単純に最大の遅延時間をコンストレイントとして与えます。例えば、以下のコマンドを使用して図15の最大伝搬遅延を9nsに指定することができます。

```
LEONARDO{}set input2output 9↵
```

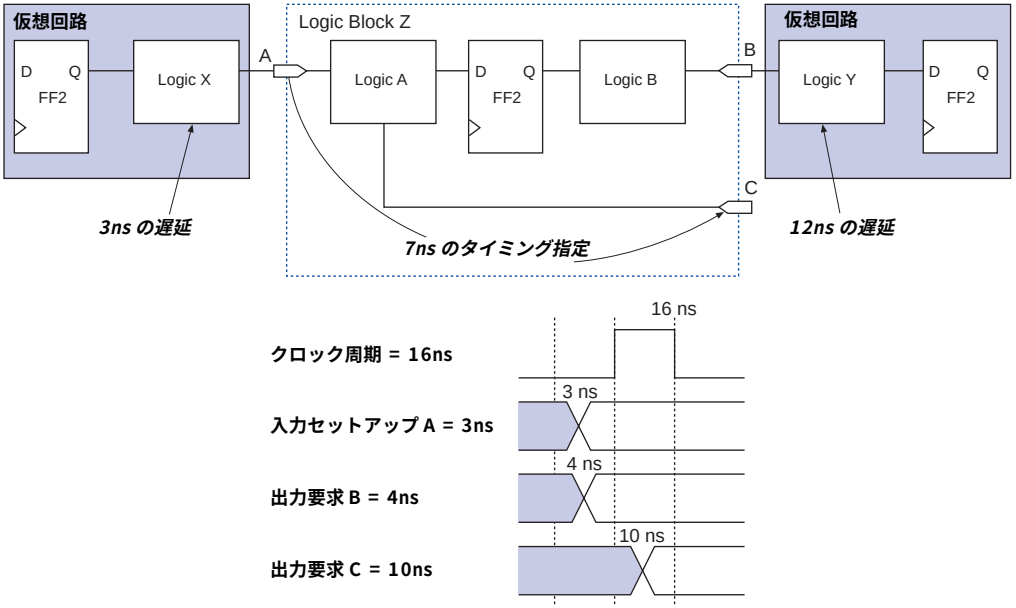
図15　9nsのコンストレイントを示すデザイン



ミックスド・デザイン

ミーリ型のステート・マシンのようなブロックには、回路内に同期化したパスと純粋な組み合わせ回路のパスの双方が含まれます。このような回路にコンストレイントを適用する場合は、同期パスのポートには同期コンストレイントを適用し、非同期パスのポートには非同期コンストレイントを適用します。図16に、同期コンストレイントと非同期コンストレイントが混在したデザインの例を示します。

図16　ミックスド・デザインに対するコンストレイントの適用例



次に、図16に示したデザイン例のミックスド・デザインにおいて、コンストレイントを指定する手順について説明します。

1. クロックに対するコンストレイントを設定します。

```
LEONARDO{}clock_cycle 16 clk↵
```

2. デザインが完全にシーケンシャルであること仮定して、入力到達時間のコンストレイントを設定します。

```
LEONARDO{}arrival_time 3 A↵
```

3. シーケンシャルな出力ポートだけに出力要求時間を設定します。とりあえず、組み合わせ回路のパスは無視して下さい。

```
LEONARDO{}required_time 4 B↵
```

4. 組み合わせ回路の出力パスに出力到達時間を設定します。これらのパスに適用される最大遅延コンストレイントは、入力到達時間と出力到達時間の差になります。ここで示した例では、入力到達時間が３に設定されています。また、組み合わせ回路のパスの最大遅延を7nsにするためには、出力要求時間を10nsに設定する必要があります（10ns－3ns = 7ns）。

```
LEONARDO{}required_time 10 C↵
```

最適化の方法

Leonardo Spectrumソフトウェアは、エリアと遅延の２種類に対してデザインの最適化を実行することができます。PLDでは配線に制限が生じることがあるため、小規模なデザインほど高速になることが多くなります。このため、アルテラは最初にエリアの最適化を行うことを推奨しています（MAX+PLUS IIを使用して配置配線を行い、次にタイミングがクリティカルとなる回路に着目する）。

Leonardo Spectrumソフトウェアでは、optimizeコマンドを使用して最適化の種類を指定することができます。表４にoptimizeコマンドで使用される引数を示します。

表４　最適化コマンドの引数

引数	概要
-target	デザインのターゲット・アーキテクチャを指定
-single_level	階層の最上位だけに最適化を実施
-effort	最適化の種類：remap、quick、standard
-nopass < *リスト* >	最適化を行わないパスを指定
-chip | -macro	-chip はトップレベルのブロックに I/O バッファを挿入 -macro はサブブロックに I/O バッファを挿入しない
-pass < *リスト* >	最適化するパスを指定
-area | -delay	-area はエリアが最小になるように最適化する（デフォルト）-delay は遅延が最小になるように最適化する
-flatten	すべての階層を展開してフラットにする

次のセクションでは、Leonardo Spectrumソフトウェアを使用して、エリアおよびタイミングの最適化方法と、アルテラ固有の最適化方法について解説します。

このセクションで述べる最適化方法では、Leonardo Spectrumのシェル・コマンドが使用されています。Leonardo Spectrumソフトウェアのツールバーやプルダウン・メニューから、これらすべてのコマンドが使用可能です。

エリアを最適化する方法

デザインがタイミング条件を満足していて、回路のサイズを可能な限りさらに小さくしたい場合は、以下の手順で最適化を行います。

1. present_designとungroup -all -hier コマンドを使用して、50,000ゲート以上になっている階層ブロックをフラットにします。

2. area_weight変数を1に設定し、delay_weight変数を0に設定します。これらの設定により、遅延よりもエリアを重視した最適化のコスト・ファンクションが再定義されます。

```
LEONARDO{}set area_weight 1↵
LEONARDO{}set delay_weight 0↵
```

3. 標準のeffortで、エリアの最適化を実行します。

```
LEONARDO{}optimize -target flex10 -area -effort
  standard↵
```

タイミングを最適化する方法

一般的に、デザインはタイミングがクリティカルとなるブロックとそうでないブロックが組合わせられたものになります。タイミングの最適化を行うことは、タイミング条件を満足させながら、可能な限り小規模なサイズでデザインを作成することになります。Leonardo Spectrumソフトウェアには、タイミングの最適化を行う２種類のコマンドが用意されています。optimize -delayのコマンドは、マッピングのプロセスで高速な構造になるように回路を生成し、ロジック・レベルを圧縮するアルゴリズムで動作します。また、optimize_timingのコマンドは、コンストレイントを基準にしたタイミングの最適化を実行します。表５に、optimize_timingコマンドで使用される引数を示します

表５　optimize_timingコマンドの引数

引数	概　要
`-through <リスト>`	最適化を行うエンド・ポイントの明示リストを指定
`-single_level`	最上位階層だけに最適化を行う
`-force`	タイミング違反を修正してから最適化を行う

以下にタイミングの最適化を行う手順を説明します。

1. 階層を維持したまま、quick effortの条件でエリアの最適化を実行します。この最適化結果から、タイミングがクリティカルとなっているブロックを特定します。

```
LEONARDO{}optimize -target flex10 -area -effort
  quick↵
```

2. タイミングがクリティカルとなっているブロックを特定するため、タイミング・レポートを生成します。

3. groupコマンドを使用して、クリティカルとなっているすべてのブロックを１つのブロックにまとめます。次に、present_designコマンドとungroup -all -hierコマンドを使用して、そのブロック内の階層をフラットにします。

```
LEONARDO{}group a b -inst_name ab_instance↵
LEONARDO{}present_design work.ab_instance↵
LEONARDO{}ungroup -all -hier↵
```

4. タイミング・クリティカルなサブブロックで、標準のeffortの条件で遅延の最適化を実行し、タイミング・レポートを生成します。

5. タイミング条件が満足されていない場合は、再度、標準のeffortの条件で遅延の最適化を実行します。この結果が変わらなくなるまで、この手順を繰り返します。

6. 依然としてタイミング条件が満足されない場合は、タイミング・コンストレイントを設定してoptimize_timingコマンドを実行します。このコマンドによりコンストレイントを基準にしたタイミングの最適化が実行されます。

7. タイミングのクリティカルなサブブロックでタイミング条件が満足されたときは、present_designに最上位階層（トップ・レベル）を設定します。そして、トップレベルのデザインがタイミング条件を満足しているかどうかを検証します。条件が満足されていない場合は、タイミングがクリティカルな他のブロックに対してステップ４から６を繰り返し実行します。すべてのブロックのタイミング条件が満足されているときは、ステップ８に移行します。

8. クリティカルではないブロックのすべてを１階層のグループにまとめます。次に、27ページに示した「エリアを最適化する方法」にしたがった処理を行います。

アルテラ固有の最適化方法

このセクションでは、アルテラのFLEX 10Kアーキテクチャに対応したLeonardo Spectrumソフトウェアのコマンドを使用して、デザインを最適化する方法を解説します。

ピン・アサインメントの方法

Leonardo Spectrumソフトウェアのインタラクティブ・シェルのコマンドを使用するか、またはVHDLデザインのソースでI/O信号ピンを定義することができます。

インタラクティブ・シェル：

```
pin_number 10 <パス名>
```

VHDL：

```
ATTRIBUTE PIN_NUMBER OF <信号名 >: SIGNAL is <値 >
```

図17は、VHDLファイルでピン・アサインメントを行う方法を示したものです。

図17　VHDLでピン・アサインメントを行う方法

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
LIBRARY exemplar; USE work.exemplar_1164.all;

ENTITY example IS
    PORT(
       clk : bit;
       din : in std_logic_vector (4 DOWNTO 0)
      q    : out std_logic_vector (4 DOWNTO 0)
     );
    ATTRIBUTE PIN_NUMBER OF clk : SIGNAL IS "1"
    ATTRIBUTE ARRAY_PIN_NUMBER OF din : SIGNAL IS ("2", "3",
      "4", "5", "6");
END example;

ARCHITECTURE exemplar OF example IS
BEGIN
    PROCESS (clk)
    BEGIN
      IF (clk = '1' and clk'event) then
       q <= din;
      END IF
    END PROCESS
END exemplar
```

ルックアップ・テーブルのマッピング

ルックアップ・テーブル（LUT）のマッピングでは、使用される４入力LUTの総数と遅延が最小になるように、ロジックが４入力LUTまたはカスケードされたゲートに配置されます。出力されたEDIFネットリスト・ファイルでは、LUTのバウンダリがLCELLバッファで表示されます。

図18は、４対１のマルチプレクサを実現するVelirog HDLの記述例です。

図18　Verilog HDLコードによる４対１のマルチプレクサ

```
module mux4 (out, in, sel);
  output out;
  input[3..0] in;
  input[1..0] sel;
  assign out = in[sel];
end module
```

LUTのマッピングでは、図18のマルチプレクサが２個の４入力LUTと１個のカスケードされたゲートにマッピングされます。Leonardo Spectrumソフトウェアは、LUTをAND-ORゲートに分解してMAX+PLUS IIソフトウェアに出力します。

エンベデッド・アレイ・ブロックへの実現

Leonardo Spectrumソフトウェアは、マルチプレクサや演算論理ユニット（ALU）などの機能をFLEX 10Kのエンベデッド・アレイ•ブロック（EAB）に実現することができます。必要なインスタンスに対して、Leonardo Spectrumソフトウェアのimplement_in_eab属性をオンに設定することにより、EABの使用が可能になります。また、VHDLデザインやVerilog HDLデザインの中でも、この属性を設定することができます。図19は、サンプルのファンクションを１個EABに実現するVHDLの記述例を示しています。

図19　ファンクションをEABに実現するVHDLコード記述例

```
ENTITY mult IS
    PORT (A,B: INTEGER RANGE 0 TO 255);
      Q: OUT INTEGER RANGE 0 TO 255);
END mult;

ARCHITECTURE BEHAVIOR OF mult IS
BEGIN
      Q <= A * B
END BEHAVIOR

ENTITY eab_test IS
    PORT (CLK,MAC,RST:bit; A,B: INTEGER RANGE 0 TO 15;
      Q: BUFFER INTEGER RANGE 0 TO 255);
END eab_test;

ARCHITECTURE BEHAVIOR OF eab_test IS
    SIGNAL P: integer range 0 to 255;

COMPONENT mult
    PORT (A,B: IN INTEGER RANGE 0 TO 15;
      Q: OUT INTEGER RANGE 0 TO 255);
END COMPONENT;

ATTRIBUTE logic_option:STRING;
ATTRIBUTE noopt:BOOLEAN;
ATTRIBUTE logic_option OF u1:LABEL IS
    "implement_in_eab=on";
ATTRIBUTE NOOPT of u1:LABEL IS TRUE;

BEGIN
    u1:mult PORT MAP (A,B,P); --Product of A and B
    PROCESS (RST,CLK)
      BEGIN
      IF (RST='1') THEN --Reset
        Q <= 0;
      ELSE
      IF (CLK='1' and CLK'event) THEN  --Clock (edge
                                       --triggered)
        IF(MAC='1') THEN
          Q <= P + Q;
        ELSE
          Q <= P;
        END IF;
      END IF;
  END PROCESS;
END BEHAVIOR;
```

ACFの生成

Leonardo Spectrumソフトウェアは、MAX+PLUS IIソフトウェアに対するアサインメント・アンド・コンフィギュレーション・ファイル（.acf）を自動的に生成します。このファイルには、タイミング・コンストレイントやデバイス・タイプ、ピン・ロケーション、ライブラリ・マッピング・ファイル（LMF）ロケーション、グローバル・ロジック・シンセシス・セッティングなどの配置配線を行う上で必要な情報がすべて含まれています。ACFを生成するときに正確な結果を得るため、Leonardo Spectrumソフトウェアでデザインに対して適切なコンストレイントが適用されている必要があります。

ACFを自動生成するときは、place_and_routeコマンドを実行します。以下に、place_and_routeコマンドの使用例を示します。

```
place_and_route design.edf -target flex10 max_acf_only↵
```

表 6 には、Leonardo Spectrumソフトウェアのplace_and_routeコマンドで使用できる引数が示されています。

表 6　place_and_routeコマンドの引数

引　数	説　明
-target	ターゲット・デバイス・ファミリ
-part	ターゲット・デバイス
-speed_grade	ターゲットのスピード・グレード
-max_acf_only | -gui	-max_acf_only は ACF だけを生成 -gui は GUI モードで MAX+PLUS II コンパイラを起動
-max_no_acf	ACF の生成を禁止
-max_ta_setup	タイミング解析を実行し、セットアップ及びホールド・タイムをレポートさせる。
-max_ta_reg	タイミング解析を実行し、レジスタ動作の最大周波数をレポートさせる
-max_area | -max_delay	-max_areaはエリアが最小になるように配置配線を最適化 -max_delay は遅延が最小になるように配置配線を最適化
-max_auto_fast_io	MAX+PLUS II で *Fast I/O* のオプションを ON に設定
-max_auto_implement_packing	MAX+PLUS II ソフトウェアの *Register Packing* オプションを ON に設定
-exe_path < *パス名* >	MAX+PLUS II のパスを明示

バック・アノテーションのフロー

MAX+PLUS IIソフトウェアで配置配線を実行した後で、Standard Delay Formatファイル（.sdf）またはネットリスト・ファイルをLeonardo Spectrumソフトウェアに再度取り込むことができます。これによって、Leonardo Spectrumソフトウェアを使用してさらにデザインのスタティック•タイミング解析を実行することができます。

高性能を実現するためのMAX+PLUS IIのオプション

Leonardo Spectrumソフトウェアによるデザインの論理合成が完了すると、生成されたEDIFネットリスト・ファイルをMAX+PLUS IIに取り込むことができます。以下のセクションでは、最適化された性能を得るために推奨されるMAX+PLUS IIの合成スタイルとその他のオプションについて解説します。

合成スタイル

Leonardo Spectrumソフトウェアで「*Map Logic to LCELLs*」のオプションがONに設定されていると、MAX+PLUS IIソフトウェアがデザインを自動的にFLEX 10Kのアーキテクチャに最適化します。このため、アルテラはMAX+PLUS IIソフトウェアで*WYSIWYG*の論理合成スタイルを使用することを推奨しています（*NOT-Gate Push-Back*のオプションはOFFに設定）。この合成スタイルで要求される性能が得られなかった場合は、*FAST*の合成スタイルを選択してみてください。一般的には、*WYSIWYG*の論理合成スタイルで最高の結果が得られますが、場合によっては*FAST*の論理合成スタイルでより良い結果が得られることもあります。

以下の手順でMAX+PLUS IIのコンパイレーション・オプションの設定を行うことができます。

1．MAX+PLUS IIのソフトウェアを起動します。

2．Project Name（Fileメニュー）を選択します。Project Nameのダイアログ・ボックスで、Leonardo Spectrumソフトウェアが生成したEDIFファイル、＜ワーキング・ディレクトリ＞/＜プロジェクト名＞.edfを選択し、OKをクリックします。

アルテラはLeonardo Spectrumソフトウェアが生成したEDIFファイルをHDLのソース・ファイルと別のディレクトリにストアすることを推奨します。

3．Compiler（MAX+PLUS IIメニュー）を選択します。

4．EDIF Netlist Reader Settingsのダイアログ・ボックス（Interfacesメニュー）で、vendorとして*Leonardo Spectrum*を選択します。

5．Deviceのダイアログ・ボックス（Assignメニュー）で適切なデバイスを選択し、OKをクリックします。

6．適切なネットリスト・ライタのコマンドをONに設定します（Interfacesメニュー）。例えば、VHDL出力ファイル（.vho）を作成したい場合は、VHDL Netlist Writerのコマンドを ONに設定します。

7．適切なネットリスト・ライタに対する設定を選択します（Interfacesメニュー）。例えば、VHDL出力ファイルを作成したい場合は、VHDL Netlist Writer Settingsを選択します。次に現れるダイアログ・ボックスで、ネットリスト・ライタに対する設定をONにします。VHDL出力ファイルを作成したい場合は、*VHDL Output File* [*.vho*]のオプションをONに設定します。OKをクリックします。

8．Global Project Logic Synthesis（Assignメニュー）を選択します。Global Project Logic Synthesisのダイアログ・ボックスで、Leonardo SpectrumのEDIFファイルでLEにマッピングされているFLEXのデザインに対して*WYSIWYG*の合成スタイルを設定します。Define Synthesis Styleをクリックします。Define Synthesis Styleのダイアログ・ボックスで、Advanced Optionsをクリックします。Advanced Optionsのダイアログ・ボックスで、「*NOT-Gate Push-Back*」のオプションをOFFに設定します。OKを３回クリックして、ダイアログ・ボックスをクローズします。

9．Startボタンをクリックして、MAX+PLUS IIのコンパイラを起動します。

デザインの作成方法、MAX+PLUS IIによるコンパイル方法の詳細については、MAX+PLUS II ACCESS Key GuidelinesにあるLeonardo Spectrum softwareのセクションを参照してください。

Fast I/Oロジック・オプションの使用

FLEX 10KデバイスにはI/Oピンの近傍にレジスタが配置されています。これらのIOE（Input/Output Element）レジスタは、デバイス内部に埋め込まれているレジスタよりも高速の「Clock-to-Output」遅延（t_{co}）を実現します。

*Fast I/O*のロジック・オプションの設定により、これらのレジスタの使用が可能になります。このオプションの設定は、LEのレジスタを使用するか、入力またはI/Oピンとのダイレクトな高速接続を提供するIOEのレジスタを使用するかをMAX+PLUS IIのコンパイラに指示します。*Fast I/O*のオプションをオンに設定することによって、高速のセットアップ・タイムまたは高速のClock-to-Output時間が実現されるため、タイミング性能を最大にすることが可能になります。

*Fast I/O*のロジック・オプションは、下記のように各ピンにも適用することができます。

- 入力ピンに対して、MAX+PLUS IIのコンパイラが入力と接続されるレジスタをIOセルまたはロジック・セルに移動させます。

- 出力ピンに対しては、MAX+PLUS IIのコンパイラが出力と接続されるレジスタをI/Oセルまたはロジック・セルに移動させます。

下記の手順の設定を行うことによって、MAX+PLUS IIのコンパイラはプロジェクト全体に対して*Fast I/O*を指定するようになります。

1. Global Project Logic Synthesis（Assignメニュー）を選択します。
2. Global Project Logic Synthesisのダイアログ・ボックス「*Automatic Fast I/O*」のオプションをONに設定し、OKをクリックします。

各レジスタごとに*Fast I/O*の指定を行う場合は、下記の手順で行います。

1. Logic Options（Assignメニュー）を選択します。Logic Optionsのダイアログ・ボックスで、レジスタのノード名を入力します。
2. Individual Logic Optionsをクリックします。Individual Logic Optionsのダイアログ・ボックスで、*Fast I/O*のオプションをONに設定します。該当するダイアログ・ボックスでOKを２回クリックして、変更した内容をセーブします。
3. Startボタンをクリックして、MAX+PLUS IIのコンパイラにコンパイルを開始させます。

タイミング・ドリブン・コンパイレーション

タイミング・ドリブン・コンパイレーションを使用することによって、デバイス・リソースの使用率に応じて、デバイス性能が平均で15%から30%まで改善されます。

一般的に、より高いリソース使用率となっているデバイスには、コンパイルに長い時間がかかります。リソースがフルに使用されている場合でも、MAX+PLUS IIのソフトウェアは、すべてのタイミング要求を満足させながら、デザインをフィッティングさせようとします。このようなリソースをフルに使用したデザインでは、コンパイル時間が10倍まで増加することがあります。

タイミング・ドリブン・コンパイレーションを実行するときには、表7に示す4種類のタイミング・コンストレイントを指定することができます。

表7　タイミング・ドリブン・コンパイレーションに使用されるコンストレイント

パラメータ	定 義	MAX+PLUS II ソフトウェアによる実現
t_{PD}	t_{PD}（入力からレジスタなし出力までの遅延）は、信号が入力ピンから組み合わせ回路を通って外部の出力ピンに現れるまでに必要な時間。	要求される t_{PD} をプロジェクト全体、または任意の入力ピン、出力ピン、出力ピンと接続される TRI バッファに対して指定することができる。
t_{CO}	t_{CO}（「clock-to-output」遅延）は、許容される「clock-to-output」の最大遅延時間を規定する。レジスタのクロックとなる入力ピンからレジスタと接続された出力ピンまでの遅延（クロック信号の遷移後の遅延）。この時間は常に外部ピン間遅延として現れる。	要求される t_{CO} をプロジェクト全体、または任意の入力ピン、出力ピン、出力ピンと接続される TRI バッファに指定することができる。
t_{SU}	t_{SU}（クロックのセットアップ・タイム）は、クロック・ピンにレジスタのクロック信号がアサートされるまで、レジスタに接続される入力データまたはイネーブル入力が入力ピンに保持される必要のある時間の長さ。	要求される t_{SU} をプロジェクト全体、または任意の入力ピン、双方向ピンに指定することができる。
f_{MAX}	f_{MAX}（最高クロック周波数）は、内部のセットアップ・タイムおよびホールド・タイムの規定に違反することなく達成できるクロックの最高周波数。	要求される f_{MAX} をプロジェクト全体、または任意の入力ピン、双方向ピン、レジスタに指定することができる。

☞ タイミングの要求をグローバルに指定するときは、Global Project Timing Requirements（Assignメニュー）を選択してください。個別のクリティカル・パスに対するタイミング要求を指定する場合は、Timing Requirements（Assignメニュー）を選択してください。

ピン・ロッキング

MAX+PLUS IIソフトウェアでは、Assignメニューのコマンドを選択して現在のプロジェクトに対するピン・アサインメントを入力したり、フロアプラン・エディタの特定のノードやピンを移動させたり、ACFをマニュアルで修正することができるようになっています。テキスト・エディタのウィンドウでACFの内容をマニュアルで修正した場合は、Assignメニューのコマンドを選択したり、フロアプラン・エディタに切り換える前に、修正したACFをセーブする必要があります。マニュアルによるACFの変更ではエラーが発生する可能性が高いため、アルテラはAssignメニューのコマンドもしくはフロアプラン・エディタを使用したアサインメントの入力を行うことを推奨します。

また、アルテラは、まず最初にピン・アサインメントなしでプロジェクトをコンパイルすることを推奨します。最初のコンパイル前に特定のピン・アサインメントを行いたい場合は、下記の手順で行ってください。

1．Pin/Location/Chip（Assignメニュー）を選択します。Pin/Location/Chipのダイアログ・ボックスで、アサインメントを入力したいノードまたはピンの名前を入力します。

2．*Chip Resources*からピン、ロジック・セル、または他のリソースを選択します。

3．Addをクリックして、次にOKのボタンをクリックします。

4．MAX+PLUS IIのCompilerのウィンドウでStartボタンをクリックして、コンパイルを開始させます。

Leonardo Spectrumのレベル

Leonardo Spectrumソフトウェアには、レベル１、レベル２、レベル３の３段階のツール・レベルが提供されています。

Leonardo Spectrum レベル１

Leonardo Spectrumのレベル１は、高性能な論理合成エンジンを使用した操作が容易な合成ツールです。高品質のネットリスト・ファイルを生成する場合も、設計者は入力デザインと対象デバイスを選択し、Runボタンをクリックするだけです。

Leonardo Spectrum レベル２

Leonardo Spectrumのレベル２は、操作が容易な合成、タイミング解析、バック・アノテーションが行えるツールです。高品質のネットリスト・ファイルを生成するときは、入力デザインと対象デバイスを選択し、Runボタンをクリックします。レベル２には以下の独自の機能があります。

- すべてのアーキテクチャに対応した仕様
- 特定アーキテクチャ向け演算子の生成
- 特定アーキテクチャへの最適化
- 特定アーキテクチャに対する高精度のタイミング解析
- PLDデザイン・フローとして認定済み
- RTLおよび合成後のゲート・レベルでの検証
- タイミングのバック・アノテーション
- すべてのプラットフォームに対応

Leonardo Spectrumのレベル２は、GUIおよびコマンドライン・モードで使用できます。

Leonardo Spectrum レベル３

Leonardo Spectrumのレベル３は、PLDに対してアーキテクチャに依存しないデザイン手法が使用できるように開発された多機能でインタラクティブな論理合成ツールであり、最適化、分析が行えるツールです。多様なデザインを効率的に無駄なく単独のデザインに統合化し、デザインの階層化を維持しながら処理することが可能です。また、Leonardo Spectrumのレベル３は以下の独自の機能を提供しています。

- PLDデザイン・フローとして認定済み：Leonardo Spectrumレベル３が生成したネットリスト・ファイルや各種設定情報はMAX+PLUS IIソフトウェアに正しく受け渡すことができます。また、タイミング検証や論理検証を行うために、配置配線後のタイミング情報を読み取り、バック・アノテーションすることが可能です。
- 特定アーキテクチャ向けモジュール生成
- 特定アーキテクチャへの最適化
- 特定アーキテクチャに対する高精度なタイミング解析
- RTLおよび合成後のゲート・レベルでの検証
- タイミングのバック・アノテーション

まとめ

プログラマブル・ロジックの業界では、設計時間の短縮と性能の向上が重要な要素となっています。このアプリケーション・ノートには、デザインの効率化により、目標とする性能の達成や設計時間の短縮に役立つ多様なテクニックが示されています。VHDLやVerilog HDLのコーディング・テクニック、Leonardo Spectrumソフトウェアのコンストレイント、MAX+PLUS IIソフトウェアのオプションを使用することで、FLEX 10Kデバイスに実現されるデザイン性能を改善し、最終的にはデザイン全体を改善することができます。

日本アルテラ株式会社

〒163-0436
東京都新宿区西新宿2-1-1
新宿三井ビル私書箱261号
TEL. 03-3340-9480　FAX. 03-3340-9487
http://www.altera.com/japan/
E-mail: japan@altera.com

本社 Altera Corporation

101 Innovation Drive,
San Jose, CA 95134
TEL : (408) 544-7000
http://www.altera.com