PlanAhead ソフトウェア チュートリアル

Tcl および SDC コマンドの使用

UG760 (v 13.1) 2011 年 3 月 1 日

XILINX®

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

改訂履歴

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
2011 年 3 月 1 日	13.1	ISE 13.1 リリース用に改訂

目次

PlanAhead ソフトウェア チュートリアル : Tcl および SDC コマンドの使用

概要

このチュートリアルでは、PlanAhead™ ソフトウェアを使用して Tool Command Language (Tcl) API でスクリプトを記述する方法について説明します。PlanAhead ソフトウェアのグラフィカル ユーザー インターフェイス (GUI) およびプロジェクト フローについては、既に理解していることが前提になっています。

まだ理解していない場合は、付録 A「その他のリソース」に示す PlanAhead チュートリアルの「クイック フロー 概要」を実行し、基本的な機能を試してください。

このチュートリアルでは、ISE® Design Suite ソフトウェアの一部として含まれる PlanAhead ソフトウェアの機能を使用しています。

チュートリアルの目標

このチュートリアルでは、次を実行します。

- サンプル デザインを使用して、Tcl コンソール、GUI と Tcl コマンド間の関係、およびログ ファイル/ジャーナル ファイルなどを確認します。
- Tcl およびバッチ スクリプトを使用して PlanAhead プロジェクトを開きます。
- PlanAhead の GUI、Tcl コンソール、Tcl コマンドのオンライン ヘルプについて理解します。
- PlanAhead のさまざまな実行モードについて学びます。
- 基本的な Tcl ビルトイン コマンドをいくつか実行します。
- バッチ モード プロジェクトを作成し、フローを実行するスクリプトを作成します。
- Tcl オブジェクト、プロパティ、物理制約を確認します。
- UCF タイミング制約を同等の Synopsys デザイン制約 (SDC) に簡単に変換し、インクリメンタルなスタティック タイミング解析レポートを確認します。

はじめに

ソフトウェア要件

PlanAhead ソフトウェアは、ISE Design Suite ソフトウェアをインストールするとインストールされます。チュートリアルを始める前に、PlanAhead が起動できるか、チュートリアル デザイン データがインストールされているかを確認してください。

インストール方法およびその詳細は、付録 A「その他のリソース」に示される『ISE Design Suite : インストールおよびライセンス ガイド』(UG798) を参照してください。

ハードウェア要件

大規模デバイスで PlanAhead ソフトウェアを使用するには、2GB 以上の RAM が推奨されます。このチュートリアルでは、小型の XC6VLX75T デザインを使用し、1 度に開くことができるデザイン数を制限していますので、1GB で十分ですが、パフォーマンスに影響のでることもあります。

チュートリアル デザインの説明

この演習では、Wishbone バス アービタを介して複数のペリフェラル コアに接続される RISC CPU コアを含む小型のサンプル デザインを使用します。このデザインは、XC6VLX75TFF784 デバイスをターゲットにしています。このチュートリアルでは、既に合成済みで使用可能な HDL を含むプロジェクト ファイルを使用します。

チュートリアル デザイン ファイルのディレクトリ

このチュートリアルでは、PlanAhead ソフトウェアのプロジェクト例に含まれるデザイン データを使用します。このデータは、次からも入手できます。

1. 次のいずれかから、PlanAhead_Tutorial.zip ファイルをダウンロードします。
   - PlanAhead ソフトウェア インストールのプロジェクト例のディレクトリ : `<ISE_install_area>/PlanAhead/testcases/`
   - ザイリンクスのウェブサイト : http://japan.xilinx.com/support/documentation/dt_planahead_planahead13-1_tutorials.htm
2. 書き込み権のあるディレクトリに ZIP ファイルを抽出します。

解凍された PlanAhead_Tutorial データ ディレクトリは、このチュートリアルでは <Extract_Dir> と記述します。

チュートリアルのサンプル データは、チュートリアルを実行中に変更されます。各チュートリアルを実行する前に、まず元の PlanAhead_Tutorial データのコピーを取っておいてください。

Tcl 構文

このチュートリアルには、Tcl コンソールで PlanAhead へ入力するコードが含まれます。Tcl 構文は、太字の > の右側に表示される部分になりますので、そのコマンドを Tcl コンソール ウィンドウに入力します。たとえば、次のような構文があるとします。

```
>get_cells cpuEngine
cpuEngine
```

> 文字は、タイプするテキストを示し、その下にそのコマンドに対するレスポンスが表示されます。

チュートリアルの手順

このチュートリアルは、次の手順で構成されています。

手順 1 : プロジェクトを開く

PlanAhead では、使用されるデザイン フローの段階によってさまざまなタイプのプロジェクトを作成できます。RTL (レジスタ転送レベル) ソースは、開発、解析、合成、インプリメンテーション、BIT ファイル生成などのプロジェクトを作成するために使用できます。このチュートリアルでは、まだインプリメントされていない合成済みネットリスト プロジェクトを使用します。

PlanAhead ソフトウェアの起動

PlanAhead ソフトウェアを起動するには、次の手順にしたがってください。

- Windows の場合、Xilinx PlanAhead 13.1 のデスクトップ アイコンをダブルクリックするか、[スタート] → [プログラム] → [Xilinx ISE Design Suite 13.1] → [PlanAhead] → [PlanAhead] をクリックします。
- Linux の場合は、`<EXTRACT_DIR>/PlanAhead_Tutorial/Tutorial_Created_Data` ディレクトリに移動し、planAhead と入力します。

PlanAhead の Getting Started ページが開きます。

プロジェクト例を開く

サンプル デザイン プロジェクトのネットリストは、チュートリアルを実行中に変更されます。元のチュートリアル デザイン プロジェクトを後で再利用できるように、プロジェクトを別名で保存します。

1. Getting Started ページで [Open Example Project] → [CPU (Synthesized)] をクリックします。
2. [File] → [Save Project As] をクリックし、プロジェクトを別名で保存します。

   [Save Project As] ダイアログ ボックスが開きます。

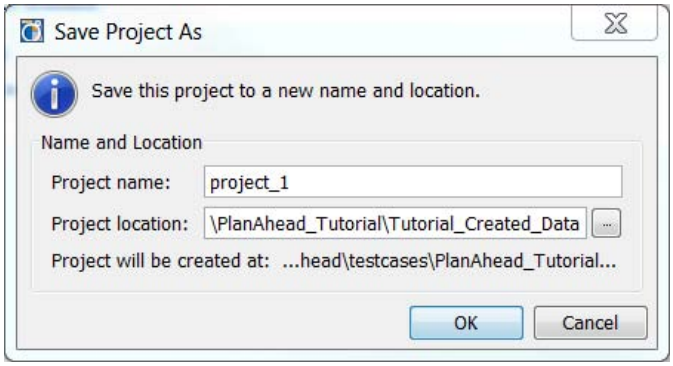

図 1-1 : プロジェクトの保存

3. [Project name] テキスト ボックスにプロジェクト名を入力します (例 : project_tcl)。
4. プロジェクト ディレクトリに次を指定します。

   `<EXTRACT_DIR>/PlanAhead_Tutorial/Tutorial_Created_Data/`
5. [OK] をクリックします。

   Project Manager が開き、[Sources] ビューにデザイン ソースが表示されます。
6. PlanAhead 環境の左側の Flow Navigator で [Netlist Design] をクリックします。

[Netlist Design] ビューが開きます。

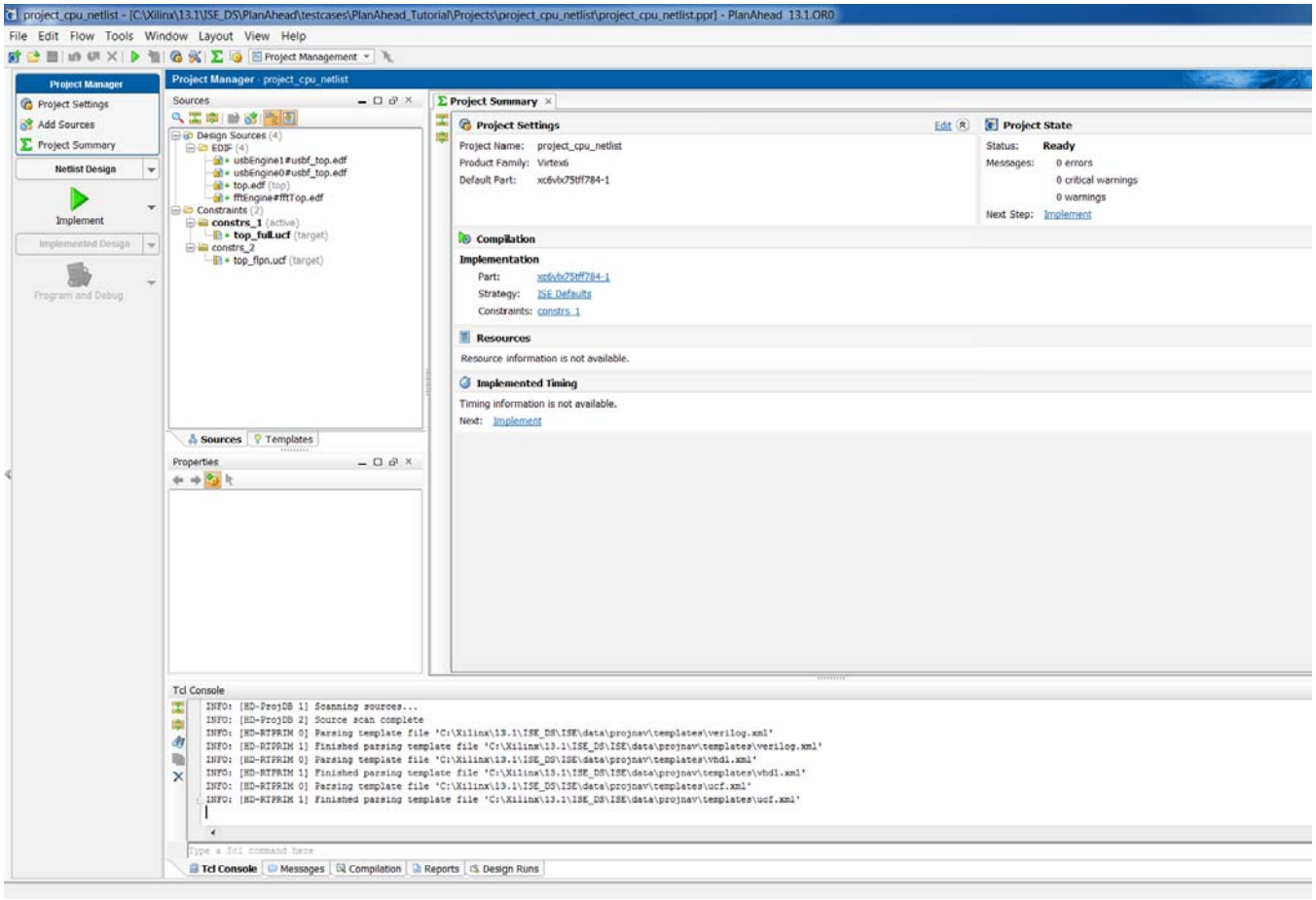

図 1-2 : [Netlist Design] 環境のプロジェクト

手順 2 : Tcl コンソールおよびオンライン ヘルプの確認

PlanAhead をデフォルトの GUI モードで起動すると、PlanAhead 環境の一番下の [Tcl Console] ビューに GUI で実行された操作に関するメッセージが表示されます。[Tcl Console] ビューには、Tcl コマンドを直接入力できるテキスト ボックスとスクロール可能な履歴が表示されます (図 1-3)。

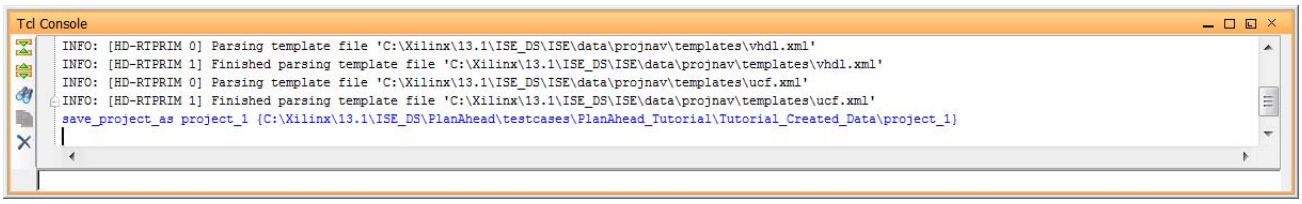

図 1-3 : PlanAhead の [Tcl Console] ビュー

[Tcl Console] ビューおよびメッセージを確認

1. [Tcl Console] ビューのテキスト ボックスに次の Tcl コマンドを入力します。

   >puts “Hello!”

   次の出力がコンソールの履歴に表示されます。

   ```
   Hello!
   ```

   Tcl ビルトイン コマンドの **puts** を使用すると、文字列メッセージがコンソールとログ ファイルの両方に出力されます。

オンライン ヘルプを確認

1. [Tcl Console] ビューのテキスト ボックスに次の Tcl コマンドを入力します。

   >help

   コマンド カテゴリのリストが表示されます。

その他の **help** コマンド構文は、次のとおりです。

- 指定したカテゴリのコマンドをリストするには、次を入力します。
  help -category <category name>
- PlanAhead ソフトウェアの Tcl コマンドの全リストを簡単な説明と共に表示するには、次を入力します。**>help ***

特定のコマンドに関するヘルプを表示

1. [Tcl Console] ビューのテキスト ボックスに次のコマンドを入力します。

   >create_project –help

   メモ : help create_project と入力しても同じです。

   create_project コマンドのヘルプ構文がすべて表示されます。

   ```
   Description:
   Create a new project
   Syntax:
   create_project [-part <arg>] [-force] [-quiet] <name> <dir>
   Returns:
   ```

```
new project object
Usage:
  Name    Optional  Default  Description
  -------------------------------------
  -part   yes                Set the default Xilinx part for a project
  -force  yes                Overwrite existing project directory
  -quiet  yes                Ignore command errors
  <name>  no                 Project name
  <dir>   no                 Directory where the project file is saved
```

2. [Tcl Console] ビューに次のような無効なコマンドを入力してみます。

   ```
   >junk
   ERROR: invalid command name "junk"
   ```

   Tcl インタープリタにより、エラー メッセージが表示されます。

   メモ : [Tcl Console] ビューのスクロールバーの横に小さな赤いバーが表示されます。これは、その行でエラーがあったことを示します。この方法を使用すると、コマンド履歴をスクロールして、警告やエラー メッセージを素早く確認できます。警告は黄色、エラーは赤色で表示されます。警告は黄色、エラーは赤色で表示されます。

   コマンドを Tcl コンソールに入力すると、Tcl インタープリターが既知のコマンドと定義済みプロシージャを検索します。そのコマンドが見つからなかった場合は、OS シェルにそのコマンドを送信します。既知のコマンドが検出されなかった場合は、エラー メッセージが表示されます。

OS シェル コマンドを入力

1. [Tcl Console] ビューに次を入力します。
   - **`>dir`** (Windows)
   - **`>ls`** (Linux)

現在作業中のディレクトリにあるファイルすべてが表示されます。この機能は、OS 別コマンドに Tcl インタープリター内から直接アクセスするために使用できます。Tcl ビルトイン コマンドの **`exec`** コマンドを使用することもできます。[Tcl Console] ビューに **`help exec`** と入力すると、このコマンドの詳細を確認できます。

ログ ファイルとジャーナル ファイルを確認

PlanAhead を起動するたびに、Tcl スクリプトの作成を理解するのに便利な次の 2 つのファイルが作成されます。

- ジャーナル ファイル (planAhead.jou) - ジャーナル ファイルには、[Tcl Console] ビューに入力されたコマンド、スクリプトからのコマンド、Tcl コマンドと同等の GUI で実行されたコマンドなど、セッション中に実行されたすべてのコマンドの履歴が含まれます。
- ログ ファイル (planAhead.log) - ログ ファイルにはすべてのジャーナル コマンドが含まれるほか、コマンドを実行することで作成された情報、警告、エラー メッセージも含まれます。このファイルには、実行されたコマンドの内容が含まれます。

ジャーナル ファイルは、該当するコマンドの Tcl 構文を理解するのに便利です。たとえば、GUI コマンドを使用してからこのファイルを開くと、実行された操作の Tcl 構文を確認できます。

メモ : ジャーナル ファイルとログ ファイルは、Linux の場合 PlanAhead 実行ファイルを起動する現在作業中のディレクトリにあります。Windows の場合、このディレクトリは %APPDATA% 環境変数で定義されています。この変数は、通常次のディレクトリにマップされます。

`C:\Documents and Settings\<user_name>\Application Data\Xilinx\PlanAhead` (Window XP の場合)
`C:\Users\<user_name>\AppData\Roaming\Xilinx\PlanAhead` (Windows 7 の場合)

1. `planAhead.log` および `planAhead.jou` ファイルを確認します。
2. 上記で実行したコマンドと、情報、警告、エラー メッセージが表示されているはずです。

メモ : ジャーナル ファイルとログ ファイルは PlanAhead を起動するたびに上書きされますので、これらのファイルが今後必要な場合は、保存しておいてください。PlanAhead を起動すると、最新のバックアップ コピーが `.jou_backup` および `.log_backup` に保存されます。

手順 3 : Tcl スクリプトの実行

ここまでは、デフォルトの GUI モードで PlanAhead を使用してきました。PlanAhead には、GUI、インタラクティブ シェル、バッチ モードの 3 つのモードがあります。

このセクションでは、それぞれのモードで Tcl コマンドおよびスクリプトを実行してみます。

GUI モードでの Tcl スクリプトの実行

PlanAhead のデフォルト モードは GUI です。Tcl コマンドは [Tcl Console] ビューに直接入力できるほか、[Tools] メニューから実行することもできます。

1. メモ帳、EMACS、VI のようなテキスト エディタで `step3.tcl` というテキスト ファイルを作成し、次のコマンドを記述します。

   ```
   puts “Hello World!”
   ```

2. GUI で [Tools] → [Run Tcl Script] をクリックし、[Run Script] ダイアログ ボックスを開きます (図 1-4)。

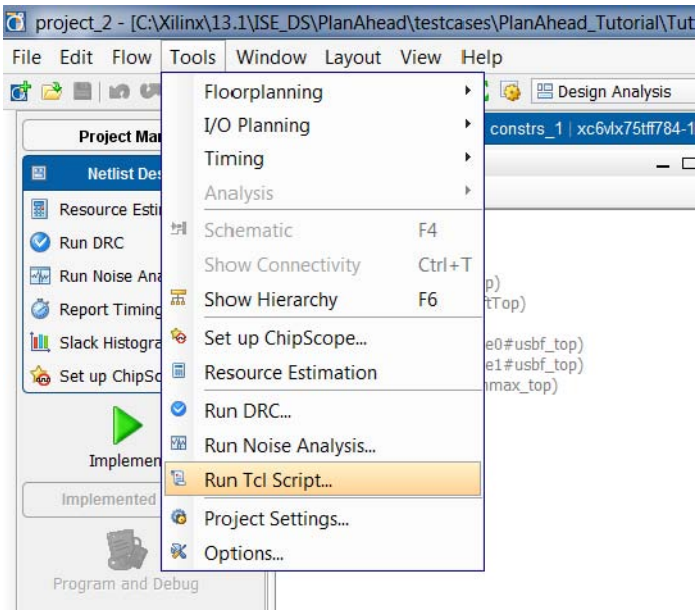

図 1-4 : [Run Tcl Script] コマンドの実行

3. `step3.tcl` ファイルを作成したディレクトリを指定します。
4. ファイルを選択します。
5. [OK] をクリックします。

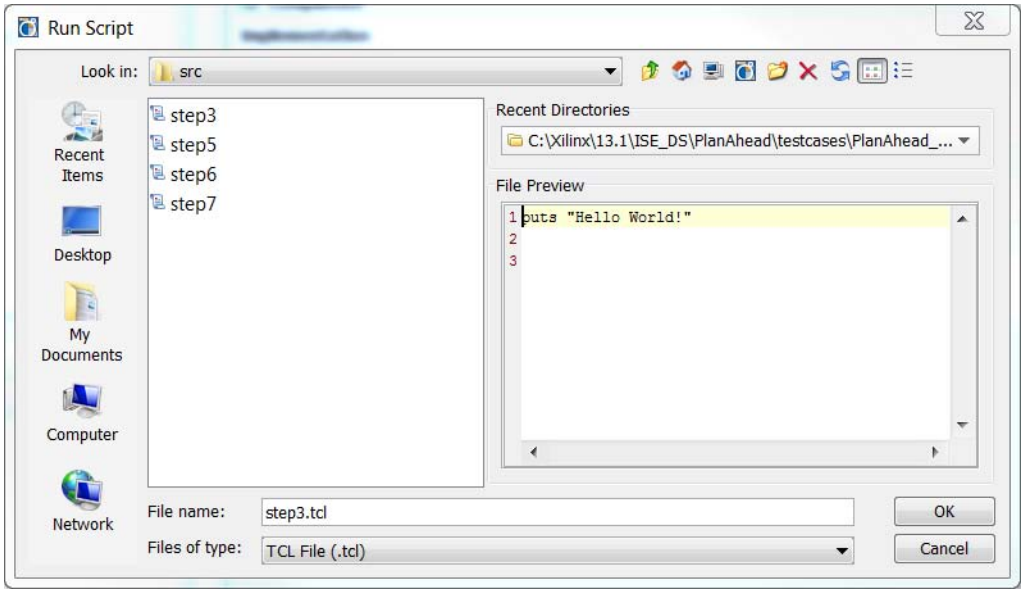

図 1-5 : Tcl スクリプト ファイルのプレビューと選択

[Tcl Console] ビューに“Hello World!” と表示されます。

Tcl スクリプトは [Tools] メニューからだけでなく、[Tcl Console] ビューで次を入力しても起動できます。

```
>source <path_to_file>/step3.tcl
```

6. [File] → [Exit] をクリックするか、[Tcl Console] ビューで exit と入力して PlanAhead を終了します。

コマンド ラインからのスクリプトの読み込み

PlanAhead を起動する際に Tcl コマンドを実行できます。ファイルを保存したディレクトリから DOS ターミナル (cmd) または Linux シェル プロンプトに **-source** コマンド ライン オプションを入力します。

1. PlanAhead を起動し、作成した `step3.tcl` を実行します。

   ```
   >planAhead –source step3.tcl
   ```

   GUI モードで PlanAhead が起動され、指定した Tcl スクリプトが読み込まれます。実行が終了したら、GUI でコマンドを実行したり、[Tcl Console] ビューに Tcl コマンドを入力できるようになります。

   メモ：これは PlanAhead の実行ファイルがインストールされ、コマンド検索パス (Linux では $PATH、Windows では %Path%) に存在していると仮定した場合です。

   PlanAhead の GUI が使用できない場合は、次のいずれかのオプションを使用できます。

   - ISE のインストールに含まれる `settings32.bat` または `settings64.bat` (Linux の場合は `.sh`) 環境設定スクリプトを source コマンドで読み込みます。
   - PlanAhead コマンドの前に PlanAhead のインストール ディレクトリへのパスを付けます。

- `<PlanAhead_install_dir>/PlanAhead/bin` が実行検索パスに含まれるように PATH 環境変数を変更します。

メモ：`"Hello World"` メッセージは、プロジェクトを開くか作成するまで PlanAhead の [Tcl Console] ビューには表示されません。プロジェクトを開くか作成したら、メイン ウィンドウの左下のボタンを使用して [Tcl Console] ビューを展開表示する必要のある可能性があります。

2. [File] → [Exit] をクリックするか [Tcl Console] ビューに **`exit`** と入力して PlanAhead を終了します。

インタラクティブ シェル モードの使用

PlanAhead には、GUI 以外のインタラクティブ セッション用にインタラクティブな Tcl シェル モードが含まれます。

1. インタラクティブ シェル モードを実行するには、次のように **`-mode`** オプションを使用します。

   ```
   >planAhead –mode tcl
   ```

   GUI なしに直接コマンドを入力できる PlanAhead シェル プロンプト (PlanAhead%) が表示されます。すべての Tcl コマンドが使用でき、コマンドをインタラクティブに実行したり、GUI モードと同じようにプロジェクトやデザインを呼び出したりできます。

2. PlanAhead ソフトウェアを終了します。

   ```
   >exit
   ```

   インタラクティブ シェル モードには、Tcl モードで PlanAhead を実行して、開始スクリプトを -source で指定するオプションもあります。

3. Tcl モードでソフトウェアを実行し、スクリプトを source オプションで呼び出します。

   ```
   >planAhead –mode tcl –source step3.tcl
   ```

4. help コマンドを入力し、インタラクティブ シェルでコマンド ヘルプを表示します。

   ```
   >help
   ```

5. PlanAhead ソフトウェアを終了します。

   ```
   >exit
   ```

バッチ モード コマンドの使用

バッチ モードは GUI が起動されない点でインタラクティブ モードと類似していますが、有効な Tcl スクリプトを -source オプションで呼び出す必要がある点が異なります。バッチ モードでは Tcl スクリプトを実行したあと、自動的に PlanAhead を終了します。

1. バッチ モードで -source オプションを付けて `step3.tcl` ファイルを実行します。

   ```
   >planAhead –mode batch –source step3.tcl
   ```

これにより、PlanAhead が起動されます。スクリプトが起動され、実行されます。この後、コマンドを入力するインタラクティブ シェル プロンプトを表示することなく、PlanAhead が閉じます。

手順 4 : 基本的な Tcl ビルトイン コマンドおよび構文の確認

Tcl の機能すべてに関する詳細な説明は、このチュートリアルではカバーし切れません。Tcl の基礎に関する詳細は、次の Web サイトを参照してください。
http://www.tcl.tk/doc/

- Tcl 参照情報
- Tcl 8.5 の入門用チュートリアル
- PlanAhead 専用でないビルトイン コマンドに関する文書

メモ : PlanAhead には、最新の Tcl リリースのバージョン 8.5 が含まれています。

Tcl インタープリターと基本構文の理解

Tcl は変換済みスクリプト言語です。Tcl コードをマシン コードやアセンブリ コードにコンパイルする必要はありません。Tcl には、Tcl コマンドを入力として使用し、言語の意味によってそれらを評価するインタープリターがあります。コマンドで直接デザインに関する質問を入力して、ツールがそれに答えるといったようなことができるので、インタープリタは EDA ツールへの優れたインターフェイスとなっています。

GUI の [Tcl Console] ビューと インタラクティブ シェル モードの `planAhead%` プロンプトがこのインタープリターへのインターフェイスです。これらのプロンプトに直接コマンドをタイプすると、これらのコマンドがインタープリターに渡され評価されます。このインタープリタは、前の手順のセクションで既に使っています。

Tcl コマンドの基本的な構文は、次のとおりです。

```
<command> <options>
```

スクリプトに複数のコマンドが含まれる場合は、それが順番に評価されます。コマンドは左から右に向かって評価されます。

変数と置換の理解

Tcl の変数は set コマンドを使用すると作成できます。Tcl 言語は大まかに入力できるので、どのデータ型を変数に含めればいいか考慮しなくても、データを維持する変数を作成できます。

Tcl プログラムを実行してみます。

1. Tcl モード (`>planAhead -mode tcl`) で PlanAhead を起動するか、インストールしている場合は Tcl シェル (tclsh) を起動します。
2. 次を入力して簡単な文字列変数を作成します。

   ```
   >set var1 “Hello World!”
   Hello World!
   ```

   コンソールの履歴に「Hello World!」という文字列が出力されます。
3. 次のように `set` コマンドを使用して、変数を文字列、整数、浮動小数点に設定します。

   ```
   >set var2 1
   1
   ```

   var2 の値が 1 に設定されます。

   ```
   >set var3 3.14
   3.14
   ```

var3 の値が 3.14 に設定されます。

4. 次のように、新しい変数を置換のコンセプトを使用してほかの変数に設定してみます。

```
>set var4 $var1
Hello World!
```

この例では、$ マークは、var4 へ代入を行う前に、インタープリターに指定した名前の付いた変数を検索して、その値を置換するように伝えています。$ マークは Tcl の特殊文字で、覚えておくと便利です。その他の Tcl の特殊文字には次のようなものがあります。

表 1-1 : Tcl の特殊文字

文字	名前	ビヘイビア
[]	各かっこ	1 つのコマンドを別のコマンドへネスト
{}	波かっこ	文字列リテラル
“”	二重引用符	変数およびコマンド置換の可能なストリング
\	バックスラッシュまたはエスケープ文字	インタープリターが読み込まないように特殊文字の前に使用
;	セミコロン	コマンドの終了
#	シャープ文字	コメント
$	ドル マーク	前に定義した変数の使用

置換されないようにするには、次のいずれかを使用します。

- 文字列リテラルを { } で囲む
- 特殊文字の前にバックスラッシュを記述

5. 変数を $ を含む文字に設定する際にインタープリターで変数が置換されないようにするには、次のいずれかのように入力します。

```
>set var5 {$var4}
```

または

```
>set var5 \$var4
$var4
```

条件文の入力

Tcl では if 文による条件文がサポートされます。次はその例です。

1. 次のコマンドを [Tcl Console] ビューに入力します。

```
>if {$var2 == 1} {puts “Yay!”}
Yay!
```

この例では、{ } は他の文の本体を示します。インタープリターは var2 という変数が 1 と同じかどうかチェックし、同じの場合は 2 つ目の { } の中のコマンドを実行します。

if に elseif コマンドを追加

elseif キーワードを使用すると、if/else 条件文に複数の else 条件を使用することができます。複数の elseif 文をまとめて使用することで、カスケードされた条件を作成できます。次の例では、最初の行で 3 つの条件がインプリメントされ、var2 が 2 の場合は Yay! を、var2 が 3.14 の場合は Nay! を、どちらの条件も満たさない場合は Maybe! が出力されます。

1. 次のコマンドを [Tcl Console] ビューに入力します。

```
>if {$var2 == 2} {puts “Yay!”} elseif {$var3 == 3.14} {puts “Nay!”} else {puts “Maybe!”}
Nay!
```

リストおよびループ

Tcl には、複数の値を維持できるコンテナ変数があります。リストはスペースで値を分けた文字列になります。

表 1-2 : Tcl 変数の例

コマンド	結果	説明
>set colors “red blue green”	red blue green	
>set colors2 [list brown black white]	brown black white	リストを構築するには、明示的なリスト コマンドを使用します。
>llength $colors	3	$colors リストにアイテム数が表示されます。
>lindex $colors 0	red	$colors リストに最初のアイテムの値が表示されます。
>lindex $colors2 end	white	$colors リストに最後のアイテムの値が表示されます。

リストに含まれるものをすべてリストするには、foreach コマンドを使用します。

1. 表 1-2 の colors リストに含まれる色をリストするには、次を [Tcl Console] ビューに入力します。

```
>foreach c $colors {puts $c}
red
blue
green
```

リストがループ検索され、含まれる色がすべて表示されます。

ネスト コマンド

コマンドのネストとは、特殊文字の [ ] を使用し、ほかのコマンドの中に複数のコマンドを埋め込むことです。ネストされたコマンドは、一番内部から外側に向かってインタープリターにより実行されます。コマンドのネストによく使用されるのは演算コマンドの expr です。

```
>set var7 [expr 1 + 1]
2
>set var8 [expr $var3 / 10]
0.314
>set var9 [expr [expr 2 * 3] + 1]
7
```

コマンドはどれでもすべてネストできます。[ ] を使用することでインタープリターで囲まれたコマンドが評価され、結果の値が次のコマンドへ代入されます。

エラー処理

Tcl スクリプトでエラーがあると、通常はスクリプトの実行が停止されます。Tcl にはエラー トラップを処理するビルトイン機能があるので、実行を続行するかどうかユーザーが指定できます。catch コマンドはどのコマンドでも引数として取り入れ、エラーがある場合は 1 を返します。

1. 次のコマンドを入力します。

   ```
   >catch “junk” result
   1
   >puts $result
   invalid command name “junk”
   ```

2. catch コマンドの結果は if 文と組み合わせると、エラーを処理できます。

   ```
   >if {[catch “junk” result]} {puts “ERROR_IN_MY_SCRIPT!”}
   ERROR_IN_MY_SCRIPT!
   ```

   この場合、catch コマンドは未知の junk コマンドから発生したエラーを取り込んで、put コマンドでコード ブロックのコマンドを実行します。

3. PlanAhead ソフトウェアまたは Tcl シェルを終了します。

   ```
   >exit
   ```

手順 5 : 新規プロジェクトの作成、合成およびインプリメンテーション

create_project コマンドで基本的なプロジェクト作成を開始します。プロジェクト作成には、次を指定するオプションが必要です。

- プロジェクト名
- プロジェクトを作成するディレクトリ
- ターゲットとするデフォルト パーツ

この手順では、小型プロジェクトを作成するスクリプトを作成し、バッチ モードで RTL 合成を実行します。

プロジェクトの作成

1. step5.tcl という新しい名前のファイルを作成します。

   次に、このファイルにコマンドを追加します。

2. step5.tcl ファイルに次の基本的な変数定義を入力またはコピーしておきます。<EXTRACT_DIR> は正しいパスに変更してください。

   ```
   set projDir [file dirname [info script]]
   set srcDir  <EXTRACT_DIR>/PlanAhead_Tutorial/Sources/hdl/
   set projName usbf
   set topName usbf_top
   set device xc6vlx75tff484-1
   ```

   これらのコマンドでは、次が実行されます。

   - [info script] コマンドでは、Tcl インタープリターで実行されたスクリプトの完全なファイル名が返されます。
   - ファイル名が file dirname コマンドに渡され、スクリプトのあるディレクトリが返されます。スクリプトがあるのと同じディレクトリにプロジェクトを作成します。

3. ディレクトリが存在しているかどうか確認し、存在しているときは次を入力して削除しします。

   ```
   if {[file exists $projDir/$projName]} {
      # if the project directory exists, delete it and create a new one
      file delete -force $projDir/$projName
   }
   ```

4. 次を入力して、手順 2 で作成したプロジェクト名、ディレクトリ、ターゲット パーツの変数を使用し、create_project コマンドで新規プロジェクトを作成します。

   ```
   create_project $projName $projDir/$projName -part $device
   ```

5. ソース セットの design_mode プロパティを設定します。

   ```
   set_property design_mode RTL [get_filesets sources_1]
   ```

   これによりプロジェクトがすべての RTL ソース ファイルのコンテナ オブジェクトである RTL プロジェクトに設定されます。合成後のネットリストに基づいたネットリスト ベースのプロジェクトを作成する場合、このプロパティは RTL ではなく Netlist にします。

6. 次を入力し、RTL ソースを定義してプロジェクトに追加します。

   ```
   set verilogSources [glob $srcDir/FifoBuffer.v $srcDir/async_fifo.v
   $srcDir/rtlRam.v $srcDir/$projName/*.v]
   ```

   Tcl には、ワイルドカード検索のように、すべてのファイルをクエリー検索する **glob** というビルトイン コマンドが含まれます。[Tcl Console] ビューに **help glob** と入力すると、このコマンドの詳細を確認できます。

7. 次のように import_files コマンドを入力し、ソース ファイルをプロジェクトに追加してローカルにコピーします。

   ```
   import_files -fileset [get_filesets sources_1] -force -norecurse
   $verilogSources
   ```

   このコマンド文字列では、各ファイルがプロジェクトにインポートされ、fileset というコンテナ オブジェクトに入れられます。デフォルトの fileset は sources_1 という名前になり、これらのファイルを保存します。

   このコマンドには、次のオプションを使用できます。

   - -force オプションを使用すると、前のソースがすべて同じ名前で上書きされます。
   - -norecurse オプションを使用すると、PlanAhead で各サブディレクトリを繰り返し検索して検出されたファイルが追加されないようになります。

8. 最上位レベルのモジュールまたはエンティティ名を最上位レベルを合成するように設定します。

   ```
   set_property top $topName [get_property srcset [current_run]]
   ```

9. 作成した step5.tcl ファイルを保存し、-source オプションで呼び出します。

   ```
   >planAhead –mode batch –source step5.tcl
   ```

   PlanAhead が起動され、スクリプトに基づいてプロジェクトが作成され、スクリプトが終了すると PlanAhead が閉じられます。

プロジェクトを合成するコマンドを追加

前の手順では、スクリプトを記述して RTL プロジェクトを作成しました。この手順では、デフォルトの合成ストラテジを使用して合成を実行するコマンドを追加します。

1. step5.tcl をテキスト エディターで開き、次のコマンドをスクリプトの終わりの set_property top コマンドの後に追加します。

   ```
   launch_runs -runs synth_1
   ```

   PlanAhead の GUI で合成またはインプリメンテーションを実行すると、PlanAhead はプロセスを別のスレッドで起動するので、GUI を続けて使用してデザインを解析できます。

   Tcl も同じで、何か特別なことをしなくても合成が実行されます。この場合、バッチ モードで実行しているので、合成の実行が終了するまでに実行をブロックするコマンドを入力する必要があります。こうすることで、インプリメンテーションを実行する次の手順が問題なく実行できます。

2. 実行をブロックするコマンドは、次のように入力します。

   ```
   wait_on_run synth_1
   ```

3. step5.tcl ファイルを保存します。

4. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

   >planAhead –mode batch –source step5.tcl

   PlanAhead は前のプロジェクトを削除し、作成し直してから、デフォルトの合成ストラテジを使用して合成を実行します。PlanAhead には多くの合成ストラテジがビルトインされており、別のストラテジを選択するのは、合成のストラテジ プロパティの設定およびリコンパイルと同じように簡単です。

5. `step5.tcl` の `launch_runs synth_1` コマンド前に次のコマンドを追加します。

   set_property strategy PowerOptimization [get_runs synth_1]

   これにより、電力最適化ストラテジが選択されます。

6. `step5.tcl` ファイルを保存します。
7. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

   >planAhead –mode batch –source step5.tcl

   PlanAhead は前のプロジェクトを削除し、新しいプロジェクトを作成し、電力最適化ストラテジを使用して合成を実行します。

インプリメンテーションの起動

前の手順では、スクリプトを作成し、XST を使用してそのプロジェクトを合成しました。

今度は、インプリメンテーションを実行するコマンドを追加して、スクリプトを実行します。

1. `step5.tcl` をテキスト エディターで開き、次のコマンドをスクリプトの終わりの `wait_on_run synth_1` コマンド後に追加します。

   launch_runs -runs impl_1

   wait_on_run impl_1

   合成 run の場合と同様、これらの 2 つのコマンドはデフォルトのストラテジを使用してインプリメンテーションを起動し、終了するまでブロックします。

2. `step5.tcl` ファイルを保存します。
3. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

   >planAhead –mode batch –source step5.tcl

   このスクリプトは、前のプロジェクトを削除し、作成し直してから、デフォルト ストラテジを使用してインプリメンテーションを実行します。

4. タイミング ドリブン マップ フローのような別のインプリメンテーション ストラテジを使用する場合は、`launch_runs –runs impl_1` コマンドの前に次のコマンドを追加します。

   set_property strategy MapTiming [get_runs impl_1]

5. `step5.tcl` ファイルを保存します。
6. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

   >planAhead –mode batch –source step5.tcl

   このスクリプトは、前のプロジェクトを削除し、作成し直してから、`map –timing` フローを使用してインプリメンテーションを実行します。

インプリメンテーション結果を開く

合成またはインプリメンテーション run が終了した後、[Netlist Design] または [Implemented Design] を開き、メモリに読み込んで、PlanAhead ソフトウェアでアクティブにします。この後、このアクティブ デザインで Tcl コマンドや操作をさらに実行できます。

1. インプリメンテーション後のネットリストを開くには、`step5.tcl` の最後に次を追加します。

   ```
   open_impl_design
   ```

2. `step5.tcl` ファイルを保存します。
3. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

   ```
   >planAhead –mode tcl –source step5.tcl
   ```

   PlanAhead% プロンプトが表示されたら、次を入力して GUI を起動します。

   ```
   PlanAhead% start_gui
   ```

   このスクリプトは、前のプロジェクトを削除し、作成し直してから、インプリメンテーション フローを実行し、今後の解析および操作用にインプリメンテーション結果を開きます。

   メモ : プロセスが終了した後に常に GUI を起動するように設定するには、`start_gui` コマンドを `step5.tcl` スクリプトの最後に追加しておきます。

4. PlanAhead を終了します。

   ```
   >exit
   ```

手順 6 : ネットリスト オブジェクト、プロパティ、物理制約の確認

前の手順では、プロジェクトを作成し、バッチ モードで Tcl を使用して単純なフローを実行しました。この手順では、SDC コマンドを使用してネットリスト アクセス コマンドをさらに確認します。

プロジェクトを開く

1. テキスト エディターで、`step6.tcl` というファイルを前と同じディレクトリに作成し、次のコマンドを入力またはコピーします。`<EXTRACT_DIR>` には、正しいパスを使用してください。

```
set projDir [file dirname [info script]]
set srcDir <EXTRACT_DIR>/PlanAhead_Tutorial/Projects/
set projName project_cpu_netlist
set topName top
set device xc6vlx75tff484-1
# open existing project
open_project $srcDir/$projName/$projName.ppr
# now open the post-synthesis netlist design
open_netlist_design
```

2. `step6.tcl` ファイルを保存します。
3. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

```
>planAhead –source step6.tcl
```

PlanAhead が起動し、プロジェクトが開き、ネットリスト デザインが読み込まれ、さらにコマンドを実行できる状態になります。

オブジェクトとプロパティの確認

まず、PlanAhead でよく使用されるオブジェクト タイプを確認します。run オブジェクトのプロパティを確認しておくと、フロー制御しやすくなります。これは、プロジェクトを検索して現在のプロジェクト ステータスを決定する際に便利です。[Tcl Console] ビューを最大化して結果を見やすくしておきます。

1. [Tcl Console] ビューに次のコマンドを入力し、run オブジェクトのプロパティを確認します。

```
>set runList [get_runs]
>report_property [lindex $runList 0]
>get_property status [lindex $runList 0]
```

これらのコマンドは、Tcl を使用して検索可能な run プロパティを示します。インプリメンテーションはまだ実行していないので、インプリメンテーション run の `impl_1` のステータスは、「Not Started」になっているはずです。

最もよく使用される SDC コマンドは `get_cells` コマンドで、ネットリスト デザインのインスタンスを名前で検索できます。`–hierarchical` オプションを使用すると、PlanAhead にデザインの各階層レベルで提供されるパターンを適用するように命令されます。次のコマンドは、デザインに含まれるすべてのセルを返します。

2. [Tcl Console] ビューに次のコマンドを入力し、セル オブジェクトのプロパティを確認します。

```
>set cellList [get_cells -hierarchical *]
>report_property [lindex $cellList end]
>get_property lib_cell [lindex $cellList end]
```

これらのコマンドは、デザインのすべてのセルを検索し、リストの最後のセルのプロパティ レポートを表示します。最後のコマンドでは、プリミティブ ライブラリのセル プロパティの値が返されます。

`get_nets` コマンドもよく使用されるオブジェクト検索 コマンドで、階層検索パターンが指定できます。

3. [Tcl Console] ビューに次のコマンドを入力し、ネット オブジェクトのプロパティを確認します。

```
>set netList [get_nets *]
>report_property [lindex $netList 0]
>get_property type [lindex $netList 0]
```

これらのコマンドは、階層の最上位レベルにあるネット オブジェクトをすべて検索し、リストの最初のもののプロパティ値のレポートを表示します。最後のコマンドでは、type プロパティが検索され、信号の型が返されます。

ポート オブジェクトは `get_ports` コマンドで検索されます。

4. 次のコマンドを入力し、ポート オブジェクトとそれらのプロパティを確認します。

```
>set portList [get_ports *]
>report_property [lindex $portList 0]
>get_property iostandard [lindex $portList 0]
```

これらのコマンドは、最上位レベルのポートすべてを返し、スクリプトに使用可能なプロパティのリストを表示します。最後に、ポートに適用された I/O 規格が `get_property` コマンドで検索され、LVCMOS25 が表示されます。

ピン オブジェクトは、`get_pins` コマンドを使用して検索できます。

5. [Tcl Console] ビューに次のコマンドを入力し、ピン オブジェクトのプロパティを確認します。

```
>set pin [get_pins OpMode_pad_0_o_0/D]
>report_property $pin
>get_property setup_slack $pin
```

これらのコマンドは特定ピン、フリップフロップへの D 入力を検索し、ピン オブジェクトに使用可能なプロパティすべてをレポートします。ピン オブジェクトはスタティック タイミング解析エンジンと関連しており、ピンはセットアップまたはホールド スラック プロパティに基づいて検索できます。タイミング解析関連のプロパティを検索すると、タイミング グラフを作成するタイミング解析エンジンが起動されます。これは、デバッグおよび解析に便利な機能です。

プロパティを使用してオブジェクト検索をフィルター

`-filter` オプションを `get_` コマンドに組み合わせて使用すると、特定の要件に基づいて返されたオブジェクトのリストをフィルタできます。次に、オブジェクト アクセス コマンドにフィルタを付けた例を示します。

1. 次のコマンドを [Tcl Console] ビューに入力します。

```
>get_nets -filter {type == "Global Clock"}
```

```
>set cellList [get_cells * -hierarchical -filter "lib_cell =~ FD*"]
```

最初のコマンドは、グローバル クロック ネットを検索し、2 つ目のコマンドはプリミティブ タイプが FD で始まる文字列パターンと一致するセルのリストを返します。これにより、FDR プリミティブのような、デザインに含まれるフリップフロップがすべて返されます。`lib_cell` オブジェクト プロパティは Unisim プリミティブに直接マップされます。

オブジェクト同士は、ネットリスト接続により互いに関連しています (ポートがネットに、ネットがピンに、ピンがセルに接続)。

オブジェクト アクセス コマンドに `-of` オプションを使用すると、ネットリスト オブジェクトを指定して検索ができます。

2. これらのコマンドを確認するには、次のコマンドを [Tcl Console] ビューに入力します。

```
>set cell [get_cells fftEngine/control_reg_1]
>set pin [lindex [get_pins -of $cell -filter "direction == IN"] 1]
>set net [get_nets -of $pin]
>set driver [get_pins -of $net -filter “direction == OUT”]
>set driverCell [get_cells -of $driver]
```

これらのコマンドを使用すると、接続関係が検索され、さまざまなプロパティおよび接続情報に基づいてネットリストが指定できるので便利です。

物理制約の確認

物理制約を確認しておくと便利ですが、Tcl で物理制約の値を UCF と同じ方法で設定できるようにしておく必要があります。次は UCF ではなく、Tcl を使用して物理制約を設定する例です。

1. [Tcl Console] ビューに次のコマンドを入力し、適用される物理制約を確認します。

```
>set_property IOSTANDARD SSTL15 [get_ports cpuClk]
>set_property site IOB_X1Y72 [get_ports cpuClk]
>set_property is_fixed true [get_ports cpuClk]
>set_property loc SLICE_X0Y73 [get_cells fftEngine/control_reg_1]
>set_property bel AFF [get_cells fftEngine/control_reg_1]
>set_property is_fixed true [get_cells fftEngine/control_reg_1]
```

これらのコマンドは IOSTANDARDS、LOC、および BEL 制約のような物理制約を設定します。HDL で適用できる属性のほとんどは EDIF ネットリストの属性に伝搬され、`get_property` で検索でき、`set_property` で設定できます。オブジェクトのプロパティすべてを確認する場合は、`report_property` コマンドを使用します。

2. PlanAhead を閉じます。
   - [Tcl Console] ビューに **`exit`** と入力します。
   - [File] → [Exit] をクリックします。
   - メイン ウィンドウ右上の X マークをクリックします。

手順 7 : Tcl および SDC を使用したスタティック タイミング解析

PlanAhead には、TRACE (ISE Design Suite の STA エンジン) とは異なるスタティック タイミング解析 (STA) エンジンが含まれます。PlanAhead の STA エンジンは、SDC (Synopsys Design Constraints) 制約と互換性があり、インクリメンタル タイミング解析をサポートします。

この手順では、単純な UCF タイミング制約ファイルを SDC に変換します。SDC は現在のところ PlanAhead でのみサポートされているので、タイミング制約は ISE インプリメンテーションおよびタイミング解析ツールにはそのまま使用できませんが、ネットリストの解析およびデバッグに優れています。SDC は主にタイミング制約に使用されるので、このセクションでは、特にクロック、I/O 制約、例外といった基本的な SDC について説明します。

UCF 変換の設定

この手順では、サンプル プロジェクトの 1 つを開き、別の名前で保存して、SDC 制約が適用されるよう、UCF ベースのタイミングが伝播されないように設定します。

1. テキスト エディターで、`step7.tcl` というファイルを前と同じディレクトリに作成し、次のコマンドを入力またはコピーします。`$srcDir` 変数の `<EXTRACT_DIR>` は正しいパスに変更してください。

```
set projDir [file dirname [info script]]
set srcDir <EXTRACT_DIR>PlanAhead_Tutorial/Projects/
set projName project_cpu_netlist
set topName top
set device xc6vlx75tff484-1

# open project
open_project $srcDir/$projName/$projName.ppr

# rename the project
set projName ${projName}SDC

# save it to a new name and location
if {[file exists $projDir/$projName]} {
   # if the project dir exists, delete it and create a new clean one
   file delete -force $projDir/$projName}
save_project_as $projName $projDir/$projName

# now disable all ucf files - in preparation for SDC
set_property is_enabled false [get_files *.ucf]

# now open the implementation results
open_netlist_design
```

2. `step7.tcl` を保存します。

3. PlanAhead を実行し、作成したスクリプトを -source で呼び出します。

   >planAhead –source step7.tcl

   このコマンドは、デザインを開き、新しい名前でそれを保存し、デザインを開いた状態で GUI を起動します。プロジェクトは GUI から手動で開いて、名前を変更することもできます。

クロックの管理

この手順では、ファイルを作成してクロックを追加します。後の手順で、制約を追加し、このファイルを繰り返し -source で読み込んで、インクリメンタル スタティック タイミング解析を使用してタイミング制約をデバッグします。

ここでは、次の UCF TIMESPEC 制約を同等の create_clock という SDC 制約に置き換えます。

```
# Timing Constraints:
TIMESPEC TS_cpuClk = PERIOD "cpuClk" 13 ns;
NET "cpuClk" TNM_NET = "cpuClk";

TIMESPEC TS_wbClk = PERIOD "wbClk" 9 ns;
NET "wbClk" TNM_NET = "wbClk";

TIMESPEC TS_usbClk = PERIOD "usbClk" 5.25 ns;
NET "usbClk" TNM_NET = "usbClk";

TIMESPEC TS_phy_clk_pad_0_i = PERIOD "phy_clk_pad_0_i" 11 ns;
NET "phy_clk_pad_0_i" TNM_NET = "phy_clk_pad_0_i";

TIMESPEC TS_phy_clk_pad_1_i = PERIOD "phy_clk_pad_1_i" 11 ns;
NET "phy_clk_pad_1_i" TNM_NET = "phy_clk_pad_1_i";

TIMESPEC TS_fftClk = PERIOD "fftClk" 7 ns;
NET "fftClk" TNM_NET = "fftClk";
```

メモ : UCF からの TIMESPEC PERIOD 制約は SDC の `create_clock` コマンドと同じです。

1. テキスト エディターを使用し、`top.sdc` という新しいファイルを作成します。
2. `top.sdc` ファイルに次を入力します。

   create_clock -name cpuClk -period 13 [get_ports cpuClk]

   このコマンドは、周期 13ns の cpuClk というクロックを作成し、cpuClk という最上位レベル ポートから 6.5ns で立ち上がるように設定しています。

3. `top.sdc` ファイルに次を入力して、ほかのクロックを追加します。

   ```
   create_clock -name wbClk -period 9 [get_ports wbClk]
   create_clock -name usbClk -period 5.25 [get_ports usbClk]
   create_clock -name phy_clk_0 -period 11 [get_ports phy_clk_pad_0_i]
   create_clock -name phy_clk_1 -period 11 [get_ports phy_clk_pad_1_i]
   create_clock -name fftClk -period 7 [get_ports fftClk]
   ```

4. 現在の作業ディレクトリに `top.sdc` を保存します。

5. PlanAhead の [Tcl Console] ビューから top.sdc を -source で読み込んでクロックを作成します。

   >source top.sdc

   メモ：作業ディレクトリが top.sdc ファイルを保存したディレクトリと異なる場合は、代わりに次を入力します。

   >source $projDir/top.sdc

   [Tcl Console] ビューには、create_clock コマンドがコメントとして表示され、最後のコマンドで作成されたクロック名が fftClk と表示されます。

6. 新規作成したクロックを含めたタイミング結果を確認するには、report_timing コマンドを実行します。

   >report_timing

   タイミング エンジンが起動され、ワーストケースのタイミング パスがデフォルトで表示されます。

入力制約の追加

次に、UCF の OFFSET IN 制約と同等の入力タイミング関係を設定します。次は、その例です。

```
NET "DataIn_pad_0_i[0]" OFFSET = IN 3 ns VALID 7 ns BEFORE "TS_usbClk"
RISING;
```

1. top.sdc の最後に次を追加します。

   set_input_delay -clock phy_clk_0 -max 2.25 [get_ports {DataIn_pad_0_i[0]}]

   set_input_delay -add_delay -clock phy_clk_0 -min 4 [get_ports {DataIn_pad_0_i[0]}]

   最初の制約は、DataIn_pad_0_i[0] 信号が phy_clk_0 クロック ドメインの立ち上がりエッジの 2.25n s 後に到着することを示しています。2 つ目の制約は、OFFSET IN の VALID 範囲と同等の最小遅延 (ホールド解析) を提供します。2 つ目の制約の –add_delay は、最小遅延値を追加する際に前に定義された最大遅延を保持するようにタイミング エンジンに伝えるためのオプションです。

2. 現在の作業ディレクトリに top.sdc を保存します。

   PlanAhead には、同じ SDC ファイル何度も読み込むことのできるように、すべてのタイミング制約を削除するコマンドが含まれます。

3. [Tcl Console] ビューのテキスト ボックスに次を入力します。

   >reset_timing

   >source top.sdc

4. クロックを含めたタイミング結果を確認するには、report_timing コマンドを実行します。

   >report_timing -from [get_ports {DataIn_pad_0_i[0]}]

   タイミング パスを確認すると、スラックが 6.488ns であることがわかります。データ パス遅延は 2.262ns です。phy_clk_0 の create_clock でクロック周期は 11ns と定義し、DataIn_pad_0_i[0] の入力遅延を 2.25ns に設定したので、レポートされるのが 11 – 2.262 – 2.25 = 6.488 になります。

出力制約の追加

次に、OFFSET OUT 制約の指定に従って出力信号要件に制約を付ける必要があります。SDC の同等のコマンドは、set_output_delay です。次は、このプロジェクトの UCF の例です。

```
NET "DataOut_pad_0_o[0]" OFFSET = OUT 4 ns AFTER "TS_usbClk" RISING;
```

1. top.sdc の終わりに次の制約を追加します。

   ```
   set_output_delay -clock wbClk 1.25 [get_ports {DataOut_pad_0_o[0]}]
   ```

2. 現在の作業ディレクトリに top.sdc を保存します。
3. PlanAhead の [Tcl Console] ビューの Tcl コマンドテキスト ボックスに次を入力して、タイミングをリセットし、-source で top.sdc で制約を作成し直します。

   ```
   >reset_timing; source top.sdc
   ```

   メモ：コマンドの後のセミコロン (;) は、複数のコマンドを同じ行に含めるときに使用します。Tcl インタープリターはセミコロンの前の reset_timing コマンドをまず実行してから、source top.sdc を実行します。

4. 新しい設定のタイミング結果を確認するには、report_timing コマンドを実行します。

   ```
   >report_timing -to [get_ports {DataOut_pad_0_o[0]}]
   ```

   タイミング レポートを確認すると、スラックが 1.939ns であることがわかります。データ パス遅延は 5.811ns です。wbClk の周期は 9ns に定義し、出力遅延は 1.25ns に設定したので、スラックを計算すると、レポートと同じ 9 – 5.811 – 1.25 = 1.939 になります。

   この段階までで、top.sdc を修正するたびにタイミング グラフを reset_timing で削除してからタイミング解析グラフを作成しなおしてきましたが、これは必ずしも必要ではありません。制約および例外を [Tcl Console] ビューに直接入力することで、インクリメンタル STA の機能を使用できます。

マルチサイクル パスの使用

次にマルチサイクル パスを UCF から SDC 用に変換します。マルチサイクル パスは、TIMESPEC 乗算器の付いた UCF の FROM/TO 制約と同じです。次は、このプロジェクトの例です。

```
# Multi-cycle paths for ALU:
NET "cpuEngine/or1200_cpu/or1200_alu/*" TPTHRU = "GRP_ALU_DATAOUT";
TIMESPEC TS_ALU_MCP = FROM "cpuClk" THRU "GRP_ALU_DATAOUT" TO "cpuClk"
TS_cpuClk * 2;
```

フリップフロップへのタイミングを緩めて、セットアップを満たすまで 2 クロック サイクル指定するために、SDC コマンドの set_multicycle_path を使用します。

1. 次のコマンドを PlanAhead の [Tcl Console] ビューに入力します。

   ```
   >report_timing -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*]
   >set_multicycle_path -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*] 2
   >report_timing -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*]
   ```

   最初のコマンドは、インクリメンタル STA アップデートを実行し、デザインのピン数からタイミング レポートを生成します。ワースト ケースのタイミング パスには、3.835ns のスラックがあります。

   2 つ目のコマンドは、パスに新しい制約を設定し、タイミングを満たすために 2 クロック サイクルを使用できるように指定しています。

   3 つ目のコマンドは、新しい制約の影響を受けるタイミング グラフの部分だけをアップデートし、別のインクリメンタル STA アップデートを実行します。この結果、このパスに必要な時間

が 13ns から 26ns に変更され、タイミングを満たすために 2 クロック周期が含まれるようになり、スラックは 16.835ns になりました。

False パスの使用

False パスでもマルチサイクル パスと同様に、UCF に TIG 制約と同等の SDC コマンドをインクリメンタルに追加できます。この場合、実行するコマンドは `set_false_path` です。解析をしやすくするために、出力ピンへのパスはすべて無視することをお勧めします。クロック ドメイン間のタイミングは、どれも UCF 制約を使用した TRACE で実行されるタイミングに対するデフォルトになりません。SDC ベースのタイミングの場合、デフォルトの推測はすべてクロックに関連します。SDC を使用して同じビヘイビアを取得するには、関連のないクロック ドメイン間に False パスを明確に設定する必要があります。

1. 次のコマンドを [Tcl Console] ビューに入力します。

   ```
   >report_timing -from [get_clocks cpuClk]
   >set_false_path -from [get_clocks cpuClk] -to [get_clocks wbClk]
   >set_false_path -from [get_clocks wbClk] -to [get_clocks cpuClk]
   >report_timing -from [get_clocks cpuClk]
   ```

   最初の `report_timing` コマンドはエラー パスを表示しますが、`cpuClk` と `wbClk` 間のパスなので、有効なパスではありません。これらのクロック ドメイン間に 2 つの False パス コマンドを設定し、タイミングを再実行しました。この結果、クロック ドメイン間にはタイミング計算が実行されなくなっています。

2. PlanAhead ソフトウェアを終了します。

   ```
   >exit
   ```

   メモ：タイミング例外すべてと同様に、False パスとマルチサイクル パス制約も処理する必要があります。デフォルトの解析を上書きするので、これらのコマンドが正しいかどうか必ず確認してください。追加したタイミング例外のために、有効なパスの違反が非表示になることもあります。

まとめ

このチュートリアルでは、次を実行しました。

- サンプル デザインを使用して、Tcl コンソール、GUI と Tcl コマンド間の関係、およびジャーナル ファイルなどを確認しました。
- Tcl を使用して PlanAhead プロジェクトを開く方法を学びました。
- Tcl コンソール、Tcl コマンドのオンライン ヘルプの使用方法を学びました。
- PlanAhead のさまざまな実行モードについて学びました。
- 基本的な Tcl ビルトイン コマンドをいくつか確認しました。
- プロジェクトを作成してフローを実行するバッチ モードのスクリプトを作成しました
- Tcl オブジェクト、プロパティ、物理制約を確認しました。
- UCF タイミング制約を同等の SDC に変換し、インクリメンタルなスタティック タイミング解析レポートを確認しました。

XILINX®

付録 A

その他のリソース

ザイリンクス リソース

- 『ISE Design Suite : インストールおよびライセンス ガイド』(UG798) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/iil.pdf
- 『ISE Design Suite 13 : リリース ノート ガイド』(UG631) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/irn.pdf
- ザイリンクス資料 : http://japan.xilinx.com/support/documentation.htm
- ザイリンクス用語集 : http://japan.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- ザイリンクス サポート : http://japan.xilinx.com/support.htm
- ビデオ デモ : http://japan.xilinx.com/products/design_resources/design_tool/resources/index.htm

PlanAhead 資料

- 『PlanAhead ユーザー ガイド』(UG632) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_UserGuide.pdf
- PlanAhead 手法ガイド : http://japan.xilinx.com/support/documentation/dt_planahead_planahead13-1_userguides.htm
- PlanAhead チュートリアル : http://japan.xilinx.com/support/documentation/dt_planahead_planahead13-1_tutorials.htm
  - 『PlanAhead ソフトウェア チュートリアル : クイック フロー概要』(UG673) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_Tutorial_Quick_Front-to-Back_Overview.pdf

その他

- Tcl の文書 : http://www.tcl.tk/doc/