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

UG 760 (v 12.2) 2010 年 7 月 23 日

XILINX®

Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the
Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications” Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2010 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Demo Design License

© 2010 Xilinx, Inc.

This Design is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Library General Public License along with this design file; if not, see:
http://www.gnu.org/licenses/

PlanAhead™ ソース コードには、次のプログラムのソース コードが使用されています。

Centerpoint XML

- The initial developer of the original code is CenterPoint – Connective Software
- Software Engineering GmbH. portions created by CenterPoint – Connective Software
- Software Engineering GmbH. are Copyright© 1998-2000 CenterPoint - Connective Software Engineering GmbH. All Rights Reserved. Source code for CenterPoint is available at http://www.cpointc.com/XML/

NLView Schematic Engine

- Copyright© Concept Engineering.

Static Timing Engine by Parallax Software Inc.

- Copyright© Parallax Software Inc.

Java Two Standard Edition

- Includes portions of software from RSA Security, Inc. and some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/
- Powered By JIDE – http://www.jidesoft.com

The BSD License for the JGoodies Looks

Copyright© 2001-2010 JGoodies Karsten Lentzsch. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of JGoodies Karsten Lentzsch nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

Free IP Core License

This is the Entire License for all of our Free IP Cores.

Copyright (C) 2000-2003, ASICs World Services, LTD. AUTHORS

All rights reserved.

Redistribution and use in source, netlist, binary and silicon forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of ASICS World Services, the Authors and/or the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

目次

Tcl および SDC コマンドの使用 7
はじめに 7
サンプル デザイン データ 7
ザイリンクス ISE および PlanAhead ソフトウェア 8
ハードウェア要件 8
PlanAhead のマニュアルと情報 8
Tcl 構文 8
チュートリアルの説明 9
チュートリアルの目標 9
チュートリアルの手順 9
プロジェクトを開く 手順 1 10
GUI、Tcl コンソール、およびオンライン ヘルプ 手順 2 12
Tcl スクリプトの実行 手順 3 15
基本的な Tcl ビルトイン コマンドおよび構文の確認 手順 4 18
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の実行 手順 5 22
ネットリスト オブジェクト、プロパティ、物理制約の確認 手順 6 26
Tcl および SDC を使用したスタティック タイミング解析 手順 7 29
まとめ 32

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

はじめに

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、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。

http://japan.xilinx.com/support/documentation/dt_planahead_planahead12-2_tutorials.htm

PlanAhead の機能の詳細は、ほかのチュートリアルで紹介していますが、すべてのコマンド オプションについて説明されているわけではりあせん。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。チュートリアルに関する質問および問題は、ザイリンクス テクニカル サポート (ホットライン) までご連絡ください。

サンプル デザイン データ

このチュートリアルでは、PlanAhead をインストールすると次のディレクトリに含まれるサンプル デザイン データと、この文書と共にリリースされるリファレンス デザインの ZIP ファイルを使用します。PlanAhead のインストール時に含まれるサンプル デザイン データは、次のディレクトリにあります。

<*ISE_install_Dir*>/PlanAhead/testcases**/PlanAhead_Tutorial.zip**

書き込み権のあるディレクトリに ZIP ファイルを保存し、抽出します。チュートリアルでは、解凍ファイルのディレクトリを <*Install_Dir*> と記述しています。

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。サンプル デザインの詳細は、「チュートリアルの説明」セクションを参照してください。

ザイリンクス ISE および PlanAhead ソフトウェア

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、サンプル 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。ソフトウェアのインストール方法および詳細は、次のザイリンクス サイトから『ISE Design Suite 12 : インストール、ライセンス、リリース ノート』を参照してください。

http://japan.xilinx.com/support/documentation/sw_manuals/xilinx12_2/irn.pdf

ハードウェア要件

ターゲット デバイスが大規模の場合、2 GB 以上の RAM 容量が必要です。このチュートリアルでは、小型のデザインを使用し、1 度に開くことができるデザインの数を制限していますので、1 GB 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。

PlanAhead のマニュアルと情報

PlanAhead ソフトウェアの詳細については、次のマニュアルを参照してください。

- 『PlanAhead ユーザー ガイド』(UG632) – PlanAhead ソフトウェアに関する詳細情報
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx12_2/PlanAhead_UserGuide.pdf
- 『フロアプラン手法ガイド』(UG633) – フロアプランのヒント情報
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx12_2/Floorplanning_Methodology_Guide.pdf
- 『階層デザイン手法ガイド』(UG748) – PlanAhead の階層デザインの概要
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx12_2/Hierarchical_Design_Methodology_Guide.pdf
- ビデオ デモなど、PlanAhead のその他の情報については、http://japan.xilinx.com/planahead を参照してください。

Tcl 構文

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。

チュートリアルに関する質問および問題は、ザイリンクス テクニカル サポート (ホットライン) までご連絡ください。

チュートリアルの目標

- 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確認します。
- 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。

チュートリアルの手順

チュートリアルは各手順に分けられ、それぞれで大まかな手順が説明された後、細かい手順が説明されていますので、スキル レベルに合った方の手順を参照してください。このチュートリアルには、PlanAhead の Tcl コンソールに直接入力できるコード部分が含まれます。大まかな手順でわからない場合はその後の詳細な手順を参照してください。既に手順を理解している場合は、その部分は飛ばして次の手順に進んでください。

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

手順 2 : GUI、Tcl コンソール、およびオンライン ヘルプ

手順 3 : GUI、バッチ、およびインタラクティブ シェル モードでのスクリプトの実行

手順 4 : Tcl ビルトイン コマンドおよび基本コマンドの使用

手順 5 : 新規プロジェクトとフロー制御の作成

手順 6 : ネットリスト オブジェクト、プロパティ、物理制約の使用

手順 7 : 基本的なスタティック タイミング解析と SDC の使用

プロジェクトを開く　　手順 1

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 ソースまたは合成済みネットリストは、開発、解析、またはインプリメンテーション ～ ビット ファイル生成まで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。

1-1. ソフトウェアを起動します。

- Windows の場合、Xilinx PlanAhead 12.2 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 12.2] → [PlanAhead] → [PlanAhead] をクリックします。
- Linux の場合は、<*Install_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。

1-2. project_cpu_netlist プロジェクト例を開きます。

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。

1-2-1. Getting Started ページで [Open Example Project] → [CPU (Synthesized)] をクリックします。

1-2-2. **[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。

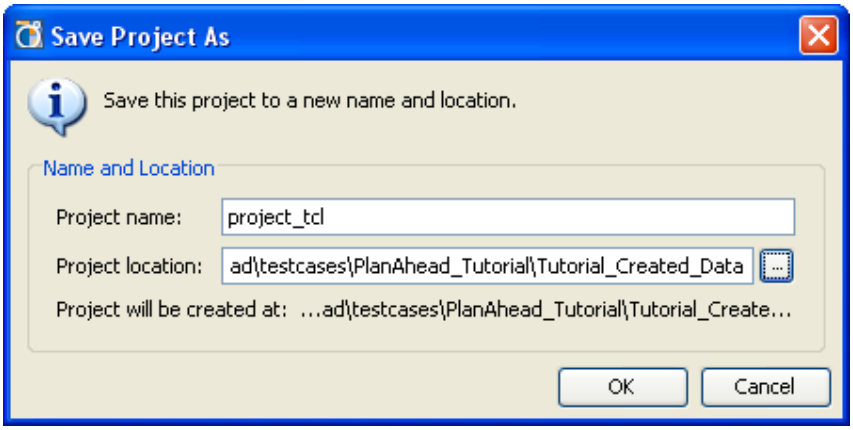

図 1 : プロジェクトの保存

1-2-3. [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**)。

1-2-4. プロジェクト ディレクトリに次を指定します。

<*Install_Dir*>/PlanAhead_Tutorial/Tutorial_Created_Data/

1-2-5. **[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。

1-2-6. PlanAhead 環境の左側の Flow Navigator で **[Netlist Design]** ボタンをクリックします。

[Netlist Design] ビューが図 2 のように開きます。

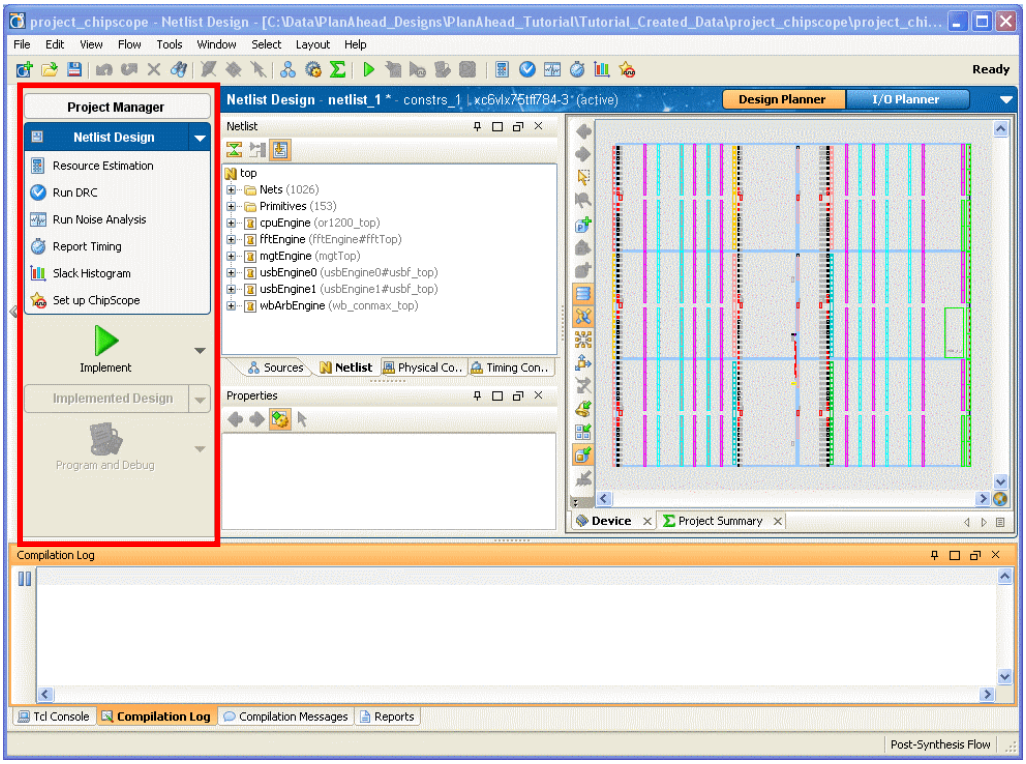

図 2 : [Netlist Design] 環境のプロジェクト

GUI、Tcl コンソール、およびオンライン ヘルプ 手順 2

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 3 : PlanAhead の [Tcl Console] ビュー

2-1. [Tcl Console] ビューおよびメッセージを確認します。

2-1-1. [Tcl Console] ビューに Tcl コマンドを入力します。

コンソールに表示された履歴を確認してください。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。次に例を示します。

```
>puts “Hello!”
“Hello!”
```

2-2. オンライン ヘルプを確認します。

2-2-1. [Tcl Console] ビューのテキスト ボックスをクリックし、次のコマンドを入力します。

```
>help
```

すべての Tcl コマンドが簡単な説明と共に画面にリストされます。

2-3. 特定のコマンドに関するヘルプを表示します。

2-3-1. [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
```

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
```

2-3-2. [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。

コマンドをコンソールに入力すると、Tcl インタープリタは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。

2-4. OS シェル コマンドを入力します。

2-4-1. [Tcl Console] ビューに次を入力します。

- **dir** コマンド (Windows).
- **ls** コマンド (Linux)

```
>dir
```

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 インタープリタ内から直接アクセスしたり、Tcl ビルトイン コマンドの exec の代わりに使用したりできます。

2-5. PlanAhead のログ ファイルとジャーナル ファイルを確認します。

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ジャーナル ファイル (.jou) とログ ファイル (.log) の 2 つが作成されます。

- ジャーナル ファイルには、GUI でインタラクティブに入力されたコマンドや Tcl に同等のものがある GUI 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 ファイルにはすべてのジャーナル コマンドが含まれるほか、各コマンドの内容と共に情報、警告、エラー メッセージも含まれます。

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。GUI を使用してこのファイルを開くと、実行するほとんどの操作に対する 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 HDI ディレクトリ (通常は、C:\Documents and Settings\<*user_name*>\Application Data\HDI) の下の %APPDATA% 環境変数で定義されています。

2-5-1. planAhead.log および planAhead.jou ファイルを確認します。

Windows の場合は、Windows エクスプローラを開いて C:\Documents and Settings\<user_name>\Application Data\HDI ディレクトリを表示してください。

2-5-2. planAhead.log および planAhead.jou ファイルを確認します。

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。

Tcl スクリプトの実行 手順 3

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、次の 3 つのモードがあります。

- GUI
- インタラクティブ シェル
- バッチ モード

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。

3-1. GUI モードで Tcl スクリプトを実行します。

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。

3-1-1. 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、保存します。

```
> puts “Hello World!”
Hello World!
```

3-1-2. 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。

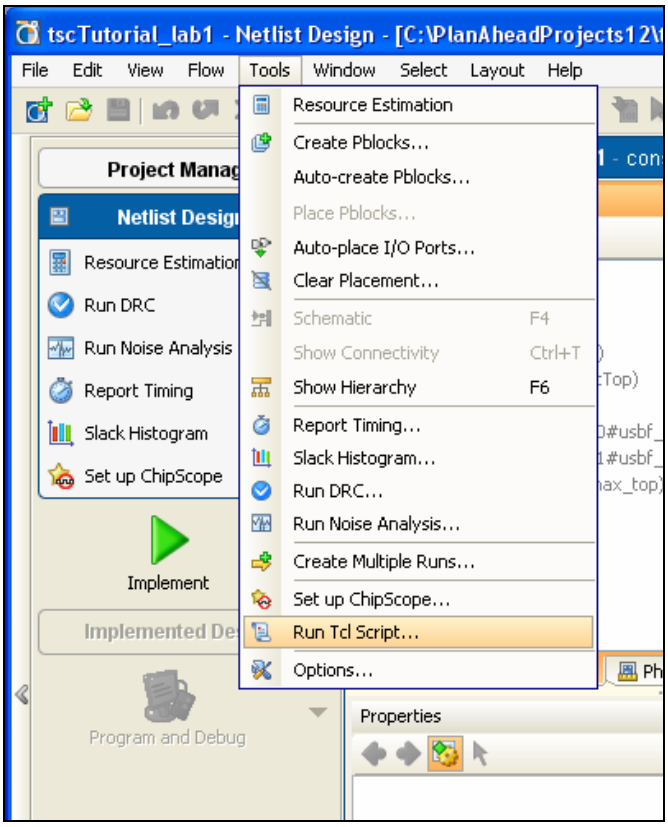

図 4 : [Run Tcl Script] コマンドの実行

3-1-3. *<INSTALL_DIR>*/src/step3.tcl を選択して、**[開く]** をクリックします。

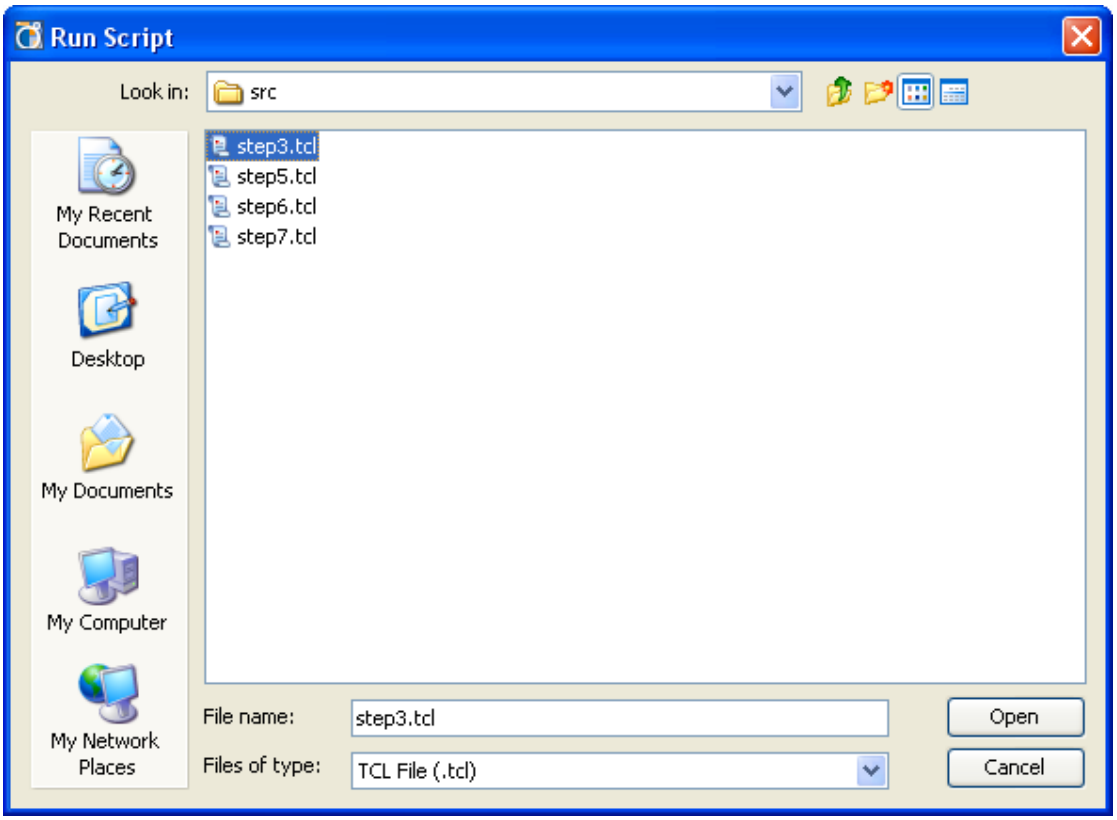

図 5 : [Run Script] ダイアログ ボックス

[Tcl Console] ビューに「Hello World!」と出力されます。

Tcl スクリプトは PlanAhead を GUI モードで起動している場合でもコマンド ラインから起動できます。

3-2. コマンド ラインからスクリプトを読み込んで実行します。

3-2-1. PlanAhead コマンドに -source コマンド ライン オプションを付け、DOS ターミナル (cmd) または Linux シェル プロンプトに入力します。

```
>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 制御がユーザーに戻ります。

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。

3-3. インタラクティブ シェル モードを使用します。

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。

3-3-1. インタラクティブ シェル モードで PlanAhead を起動するには、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。

3-3-2. 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
```

3-3-3. インタラクティブ シェル モードから PlanAhead を終了するには、exit コマンドを入力します。

```
>exit
```

3-4. バッチ モード コマンドを使用します。

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 Tcl スクリプトを実行すると、スクリプトの最後の行で exit コマンドが実行されたように、PlanAhead が自動的に終了します。

3-4-1. 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
```

PlanAhead が起動され、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。

基本的な Tcl ビルトイン コマンドおよび構文の確認 手順 4

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。ここでは、残りのチュートリアルを理解するのに役立つ基本的なものだけを説明します。Tcl の基礎に関するすべてのリファレンスおよびチュートリアルについては、次の Web サイトを参照してください。

http://www.tcl.tk/doc/

入門用チュートリアルは、次から入手できます。

http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html

特別なビルトイン (PlanAhead 以外の) コマンドに関する文書は、次から入手できます。

http://www.tcl.tk/man/tcl8.5/TclCmd/contents.htm

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。

4-1. Tcl インタープリタと基本構文を理解します。

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。

4-2. 変数と置換について理解します。

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。

4-2-1. 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。

4-2-2. 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
```

var2 の値が 1 に設定され、Tcl コンソールに表示されます。

```
>set var3 3.14
```

var3 の値が 3.14 に設定され、Tcl コンソールに表示されます。

4-2-3. 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。

- 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

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

変数を $ を含む文字に設定する際にインタープリタに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}
$var4
```

または

```
>set var6 “\$var4”
$var4
```

4-3. 条件文を入力します。

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。

4-3-1. 次のコマンドを Tcl コンソールに入力します。

```
>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。

4-4. if に elseif コマンドを追加します。

4-4-1. 次のコマンドを Tcl コンソールに入力します。

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

4-4-2. 次を入力して最後の else の値を提供します。

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

4-5. リストおよびループ文を入力します。

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。次に例を示します。

```
>set colors “red blue green”
red blue green
```

次のように list コマンドでリストを構築することもできます。

```
>set colors2 [list brown black white]
brown black white
```

Tcl には、次のようなリストに関する便利な情報を返すビルトイン コマンドが多く含まれます。

```
>llength $colors
3
```

$colors リストに含まれるアイテムの数を返します。

```
>lindex $colors 0
red
```

$colors リストに含まれる 1 つ目のアイテムの名前を返します。

```
>lindex $colors2 end
white
```

$colors リストに含まれる最後のアイテムの名前を返します。

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。

4-5-1. colors リストに含まれる色をリストするには、次を Tcl コマンドに入力します。

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

リストが検索され、含まれる色がすべて表示されます。

4-6. コマンドをネストします。

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がインタープリタ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。

4-7. エラー処理について理解します。

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。

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

`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。

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の実行 手順 5

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

これを Tcl インフラストラクチャを使用して実行するには、Tcl スクリプトを作成して、超小型のプロジェクトのバッチ モード作成を実行し、RTL 合成およびインプリメンテーションを実行します。

チュートリアルをダウンロードした Web サイトから小型の labData.zip ファイルをダウンロードします。このファイルには、小型デザインのソース コードとこのチュートリアルでリファレンスとして使用するスクリプトが複数含まれています。

5-1. プロジェクトを作成します。

5-1-1. labData.zip を <INSTALL_DIR> という空のディレクトリに抽出します。

5-1-2. ディレクトリをチュートリアル デザイン データのディレクトリに変更するには、次を入力します。

cd INSTALL_DIR

5-1-3. step5.tcl という新しい名前のファイルを作成します。

5-1-4. step5.tcl ファイルに次の基本的な変数定義を記述します。

```
set projDir [file dirname [info script]]
set srcDir $projDir/src
set projName oneFlopHier
set topName 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。

5-1-5. 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
clean one file delete -force $projDir/$projName
}
```

5-1-6. 次を入力して、手順 5-1-5 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
```

5-1-7. 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。

5-1-8. 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/*.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。

5-1-9. 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 なので、元ファイルがプロジェクトに対してローカルになります。

–force オプションは前のソースをすべて同じ名前で上書きし、–norecurse オプションは PlanAhead に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を追加するように伝えます。

5-1-10. ライブラリ名を設定します。work ライブラリはデフォルトなので、このコマンドはオプションです。

```
set_property library work [get_filesets sources_1]
```

5-1-11. 次を入力して UCF 制約ファイルを設定します。

```
set ucfSources [glob $srcDir/*.ucf]
```

```
import_files -fileset [get_filesets constrs_1] -force -norecurse
$ucfSources
```

5-1-12. 最上位モジュールまたはエンティティの名前を設定し、合成エンジンに合成する最上位レベルがどれか伝えます。

```
set_property top $topName [get]
```

5-1-13. 作成した step5.tcl ファイルを保存し、PlanAhead コマンドの 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ます。

メモ : スクリプトが実行されると、PlanAhead が終了します。この後、PlanAhead を通常どおりに開いて、[File] → [Open Project] をクリックすると、作成した .ppr プロジェクト ファイルを GUI で使用できます。

5-2. 合成を実行します。

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、合成を実行するコマンドを追加します。

5-3. デフォルトの合成ストラテジを使用してプロジェクトを合成するコマンドを追加します。

5-3-1. step5.tcl をテキスト エディタで開き、次のコマンドをスクリプトの終わりの set_property top 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。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。 これは、次の手順のインプリメンテーションの起動を設定するためです。

5-3-2. 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
```

5-3-3. step5.tcl ファイルを保存します。

5-3-4. PlanAhead コマンドで作成したスクリプトを呼び出します。

```
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と、合成のストラテジ プロパティが設定され、リコンパイルされます。

5-3-5. 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。

5-3-6. 作成した step5.tcl ファイルを保存し、PlanAhead コマンドの source オプションで呼び出します。

```
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。

5-4. インプリメンテーションを実行します。

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、このスクリプトにインプリメンテーション run の起動を追加します。

5-4-1. step5.tcl をテキスト エディタで開き、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ます。

5-4-2. 作成した step5.tcl ファイルを保存し、PlanAhead コマンドの source オプションで呼び出します。

```
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。

5-4-3. タイミング ドリブンのマップ 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]
```

5-4-4. 作成した step5.tcl ファイルを保存し、PlanAhead コマンドの source オプションで呼び出します。

```
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。

5-5. インプリメンテーション結果を開きます。

run が終了した後デザインを開くと、デザインのネットリストがメモリにアクティブ デザインとして読み込まれ、Tcl コマンドや操作をさらに実行できます。

5-5-1. 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
```

作成した step5.tcl ファイルを保存し、PlanAhead コマンドの source オプションで呼び出します。

```
planAhead -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、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。つまり、フローが実行された後、インプリメント済みデザインが開かれ、別のコマンドや解析ができる状態の GUI が起動されます。

ネットリスト オブジェクト、プロパティ、物理制約の確認 手順 6

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。

6-1. プロジェクトを開きます。

6-1-1. テキスト エディタで、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。

```
set projDir [file dirname [info script]]
set srcDir $projDir/src
set projName oneFlopHier
set topName top
set device xc6vlx75tff484-1
# open new project
open_project $projDir/$projName/$projName.ppr
# now open the post-synthesis netlist design
open_netlist_design
```

6-1-2. step6.tcl を保存し、デフォルトの GUI モードで PlanAhead でプロジェクトを実行します。

```
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ます。

6-2. オブジェクトとプロパティを確認します。

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。

6-2-1. PlanAhead GUI の [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。合成 run の synth_1 のステータスは、「**XST Complete!**」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。below 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。

6-2-2. Tcl コンソール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 0]
>get_property site [lindex $cellList 0]
```

これらのコマンドは、デザインのすべてのセルを検索し、リストの最初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、ロケーション制約がセルに適用されている場合、その値が返されます。

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。

6-2-3. Tcl コンソールに次のコマンドを入力し、ネット オブジェクト 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、グローバル クロック ネットが返されます。

ポート オブジェクトは get_ports コマンドで検索されます。

6-2-4. 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。

6-2-5. Tcl コンソール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 oneFlopInst/dataIn/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、ピン オブジェクトを検索し、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。これは、デバッグおよび解析に便利な機能です。

6-3. プロパティを使用してオブジェクト検索にフィルタをかけます。

-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。

6-3-1. 次のコマンドを Tcl コンソールに入力します。

```
>get_nets -filter {type == "Global Clock"}
>set cellList [get_cells * -hierarchical -filter "lib_cell =~ FD*"]
```

これらのコマンドは、グローバル クロック ネットを検索し、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。

6-3-2. これらのコマンドを確認するには、次のコマンドを Tcl コンソールに入力します。

```
>set cell [get_cells oneFlopInst/dataIn]
>set pin [lindex [get_pins -of $cell -filter "direction == IN"] 0]
>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確認されます。

6-4. 物理制約を確認します。

物理制約の値は、UCF に合わせた方法で設定する必要があります。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。

6-4-1. Tcl コンソールに次のコマンドを入力し、適用されている物理制約を確認します。

```
>set_property IOSTANDARD SSTL15 [get_ports in]
>set_property site IOB_X0Y72 [get_ports in]
>set_property is_fixed true  [get_ports in]
>set_property site OLOGIC_X0Y73 [get_cells oneFlopInst/dataIn]
>set_property bel OUTFF [get_cells oneFlopInst/dataIn]
>set_property is_fixed true [get_cells oneFlopInst/dataIn]
```

これらのコマンドは IOSTANDARDS、LOC、BEL および BEL 制約のような物理制約を設定します。HDL で適用できたり、EDIF ネットリストの属性に伝搬されるような属性のほとんどは、get_property で検索でき、set_property で設定できます。

6-4-2. Tcl コンソールに exit と入力するか、**[File]** → **[Exit]** をクリックするか、メイン ウィンドウ右上の X マークをクリックして PlanAhead を閉じます。

Tcl および SDC を使用したスタティック タイミング解析 手順 7

PlanAhead には、TRCE (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 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。

このチュートリアルでは、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スタティック タイミング解析ツールにはそのまま使用できません。SDC は、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。

7-1. UCF 変換を設定します。

このチュートリアルでは、手順 5 および 6 で作成したプロジェクトを使用し、別の名前で保存して、2 つのプロジェクトを比較できるようにします。チュートリアル ファイルに含まれる /src ディレクトリに、この変換を実行するスクリプトが含まれています。

./src ディレクトリの step7.tcl を手順 5 で使用したディレクトリにコピーします。

7-1-1. 手順 5 の作業ディレクトリで次を入力します。

```
copy .\src\step7.tcl .
```

メモ：Windows の場合はバックスラッシュ (\ ) ですが、Linux の場合はスラッシュ (/) にしないとコピーできません。

7-1-2. 手順 5 および 6 と同じディレクトリで次のスクリプトを実行します。

```
planAhead -source step7.tcl
```

このコマンドは、手順 5 のディレクトリからデザインを開き、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。

次は、手順 5 からの単純なデザインに制約を付ける UCF コマンドです。これらのコマンドを SDC の同等コマンドに変換します。

```
TIMESPEC TS_clk = PERIOD "clk" 10 ns;
NET "clk" TNM_NET = "clk";


NET "in" OFFSET = IN 3 ns VALID 7 ns BEFORE "clk" RISING;
NET "rst" OFFSET = IN 3 ns VALID 7 ns BEFORE "clk" RISING;


NET "out" OFFSET = OUT 8 ns AFTER "clk" RISING;
```

7-2. クロックを管理します。

7-2-1. テキスト エディタで top.sdc というファイルを新しく作成します。

このファイルを繰り返し読み込んで、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追加およびデバッグします。

UCF の TIMESPEC PERIOD 制約と同等のコマンドは、create_clock です。

7-2-2. top.sdc ファイルに次を入力します。

```
create_clock -name TS_clk -period 10 -waveform {0 5} [get_ports
clk]
```

このコマンドは、周期 10ns のクロックを clk という名前の最上位ポートをルートに 5ns で立ち上がるように作成します。

7-2-3. 現在の作業ディレクトリに top.sdc を保存します。

7-2-4. PlanAhead の GUI の [Tcl Console] ビューで source top.sdc を実行してクロックを作成します。

```
>source top.sdc
```

7-2-5. report_timing コマンドを実行して、クロックを含むタイミング結果を確認します。

```
>report_timing -from [get_clocks TS_clk]
```

[Tcl Console] ビューに TS_clk クロック ドメインのワースト クリティカル パスと同等のパス トレースが表示されます。

7-3. 制約を入力します。

OFFSET IN 制約と同等の入力タイミング関係を設定します。

7-3-1. top.sdc の最後に次を追加します。

```
set_input_delay -max -clock [get_clocks TS_clk] 7 [get_ports in]

set_input_delay -add_delay -min -clock [get_clocks TS_clk] 4
[get_ports in]
```

この制約は、in 信号が TS_clk クロック ドメインの立ち上がりエッジの 3 ns 前に到着することを示しています。2 つ目の制約は、ポート/ピン リストの前の制約を保持するようにツールに命令をする –add_delay オプションを含み、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。

次は、リセット信号 rst にも同じ入力関係を設定します。

7-3-2. top.sdc の終わりに次の制約を追加します。

```
set_input_delay -max -clock [get_clocks TS_clk] 7 [get_ports rst]

set_input_delay -add_delay -min -clock [get_clocks TS_clk] 4
[get_ports rst]
```

7-3-3. 現在の作業ディレクトリに top.sdc を保存します。

7-3-4. PlanAhead の GUI の [Tcl Console] ビューで source top.sdc を実行して制約を作成し直します。

SDC を source で実行しただけの場合、TS_clk という別のクロックの作成が試みられますが、そのクロックは既に存在しているのでエラーになります。

PlanAhead には、同じ SDC ファイル何度も読み込むことのないように、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。

7-3-5. Tcl コマンド テキスト ボックスに次を入力します。

```
>reset_timing

>source top.sdc
```

7-3-6. クロックを含むタイミング結果をすぐに確認するには、report_timing コマンドを実行します。

```
>report_timing -from [all_inputs]
```

このコマンドは、制約を付けた入力ピンのいずれかからのワースト クリティカル パスをレポートします。

7-4. 制約を出力します。

最後に、OFFSET OUT 制約の指定に従って出力信号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。

7-4-1. top.sdc の終わりに次の制約を追加します。

```
>set_output_delay -clock [get_clocks TS_clk] 2 [get_ports out]
```

7-4-2. 現在の作業ディレクトリに top.sdc を保存します。

7-4-3. PlanAhead の GUI 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、タイミングをリセットし、source 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。

7-4-4. クロックを含むタイミング結果をすぐに確認するには、report_timing コマンドを実行します。

```
>report_timing -to [all_outputs]
```

パストレースは、出力ピンへのワースト クリティカル パスを表示します。

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、これは必要ではありません。 制約および例外を Tcl コンソール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。

7-5. マルチサイクル パスを使用します。

次に入力ポート in から 1 つの FlopInst/dataIn フリップフロップへのマルチサイクル パスを指定します。これは TIMESPEC 乗算器の付いた UCF の FROM/TO 制約と同等です。

この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 サイクル指定する場合は、SDC コマンドの set_multicycle_path を使用します。

7-5-1. 次のコマンドを PlanAhead の GUI の [Tcl Console] ビューに入力します。

```
> set_multicycle_path -to [get_cells oneFlopInst/dataIn] 2
> report_timing -to [get_cells oneFlopInst/dataIn]
```

これによりインクリメンタル STA アップデートが実行され、新しい制約の影響を受けるタイミング グラフの部分だけがアップデートされます。このパスに必要な時間が 7ns から 17ns に変更され、このパスに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ました。

7-6. False パスを使用します。

False パスでも同様に、UCF の TIG 制約と同等の SDC コマンドをその部分のみ追加できます。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。

7-6-1. 次のコマンドを PlanAhead の GUI の [Tcl Console] ビューに入力します。

```
> set_false_path -to [all_outputs]
> report_timing -to [all_outputs]
```

出力ピンにはスラック計算がないことがわかります。データ パス遅延がレポートされますが、このデザインのクリティカル パスは出力パスではなくなっています。

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。ユーザーの追加したタイミング例外のために、有効なパスの違反を非表示にすることともできます。

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理解しました。
- 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。
- 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 Synopsys デザイン制約 (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。