Quartusソフトウェアでの Tclスクリプト使用方法

1999年 11月 ver. 2.0 Application Note 118

イントロダクション

Quartus™ソフトウェアでは、Tcl（Tool command language）を使用したスクリプトを作成し、これを動作させることによって、デザインに対するコンパイルの実行のような単純な処理から、複数の一般的なタスクを自動化して実行させる手順の書き込みなどのような複雑な処理までを幅広く実行させることができます。このアプリケーション・ノートは、Quartusソフトウェアで使用されるTclの作成方法を解説したものです。

☞ Quartusソフトウェアは、Scriptics Corporation（http://www.scriptics.com）から供給されているTclのバージョン8.03をサポートしています。

Tclとは？

Tclは、多くのシェル・スクリプトやハイ・レベルなプログラミング言語に類似したポピュラなスクリプト言語です。Tclは制御文、変数、ネットワーク・ソケット・アクセス、他のソフトウェアと統合するためのAPI（Application Programming Interfaces）などをサポートしています。

Tclは理解しやすく、使いやすい変換可能な言語です。 Tclを利用してカスタム化されたコマンドや処理手順を作成することができます。また、TclはUNIXやWindows NTのような、ほとんどの開発環境のプラットフォーム上でシームレスに動作するため、Tclをマルチ・プラットフォームのプログラムとして使用することができます。Tclの詳細については、16ページに示されている参考資料を参照してください。

Tclの使用方法

QuartusのAPIは、Tclが使用されるときにコール可能になっている複数のインタフェース・ファンクションによって構成されています。Tclをある程度理解しているユーザであれば、APIを使用してQuartusソフトウェア内で複数のタスクを自動的に実行させるTclスクリプトを作成することができます。そして、APIをTclのコマンドのように実行させ、単独のスクリプトでデザイン・プロジェクトのコントロールやコンパイルのスタートとストップの指定、アサインメントの指定、シミュレーションの実行などを行うことができます。

Tclコマンドの基本的なシンタックスは、下記のようになります。

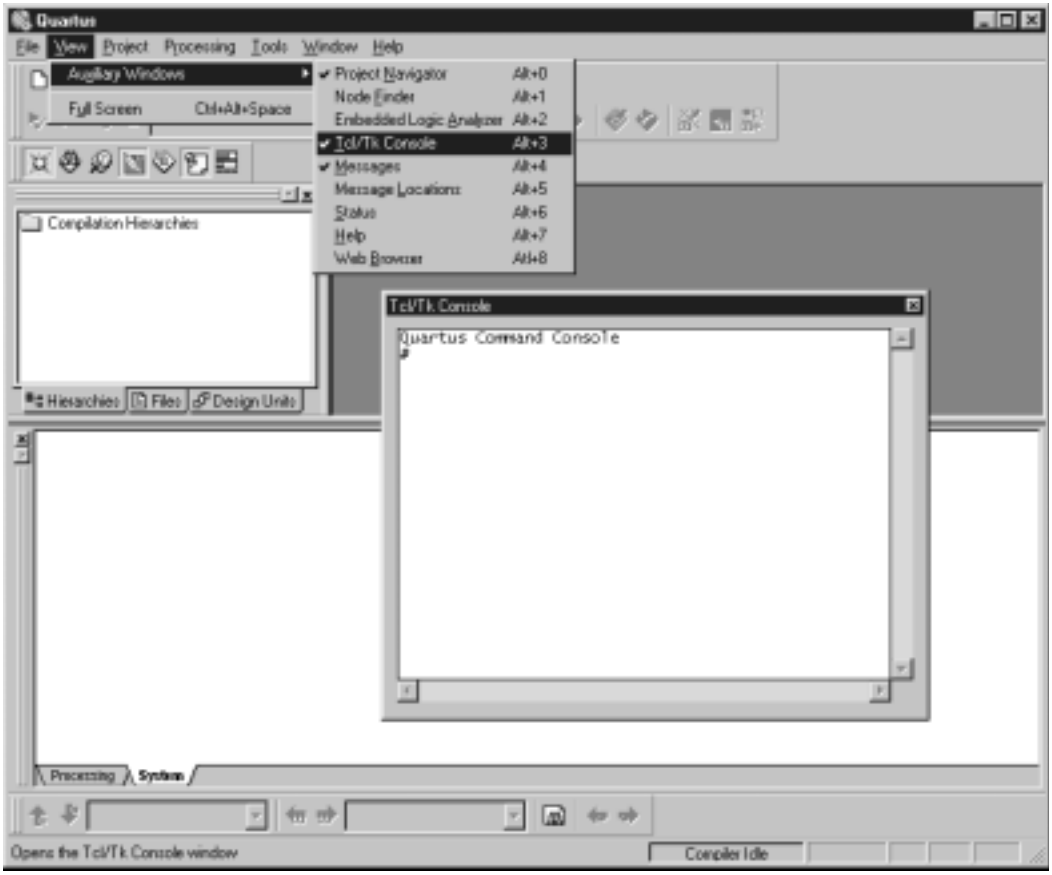
＜コマンド＞[＜*引数1*＞＜*引数2*＞＜*引数3*＞...]

ここで、コマンドのシンタックスは、組み込まれているコマンド名か手順のいずれか、または複数のコマンドのセットになります。各コマンドと引数はスペースで分離され、改行またはセミコロンによってコマンドが終了します。コマンドに対する引数はストリングとして受け渡されます。

Tclスクリプトをインタラクティブに動作させる方法

Tclのコマンドは、QuartusのTcl/Tk Consoleのウィンドウからダイレクトに実行することができます。Tcl/Tk Consoleのウィンドウを開くときは、図1に示すように、Auxiliary Windows（Viewメニュー）の中からTcl/Tk Consoleを選択します。

図1 Tcl/Tk Consoleの画面



Tcl/Tk Consoleのウィンドウは履歴をサポートしますが、コマンドを複数のラインに拡張させることはできません。Tclのメッセージは、MessageウィンドウのSystemタブに現れます。

Tclをバッチ・モードで実行する方法

Tclスクリプト・ファイル（.tcl）の作成が完了したら、Tcl/Tk Consoleのウィンドウに下記のコマンドを入力して、このTclを実行させます。

```
source <スクリプト・ファイル名>↵
```

また、Run Script（Toolsメニュー）を選択して、実行させることもできます。

DOSまたはUNIXからTclスクリプトを実行する方法

Quartusソフトウェアは -f <スクリプト・ファイル名> の形式によるコマンド・ライン上での引数の指定もサポートしています。このコマンドは、Run Script（Toolsメニュー）を選択したときと同じ動作を実行します。DOSまたはUNIXのプロンプトからこのコマンドを実行させるときは、下記のシンタックスを使用します。

```
quartus_cmd -f <スクリプト・ファイル名>↵
```

基本的なTclコマンド

QuartusソフトウェアのTclコマンドは、プロジェクトのコントロール、コンパイラやシミュレータによる処理などの基本的なタスクを実行させることができます。表1は、アサインメントの内容に応じて各種の設定条件がセーブされるTclのインタフェース・ファイルの種類を示しています。

表1　Tcl 設定ファイルの種類

ファイル・タイプ	説明
Quartus File (**.quartus**)	**Quartusファイルには、プロジェクト全体に対する設定条件が含まれている。**
Project Settings File (**.psf**)	**PSF には、プロジェクト全体に対する設定条件が含まれる。**
Compiler Settings File (**.csf**)	**CSF には、コンパイラに対する設定条件が含まれる。**
Entity Settings File (**.esf**)	**ESF には、各エンティティとノードに対する設定条件が含まれる。*(1)***
Simulator Settings File (**.ssf**)	**SSF には、シミュレータに対する設定条件が含まれる。**

注：
(1)　特定の条件が指定されたエンティティごとに個別のESFが存在します。

☞ 表2から表10までに示されているすべてのコマンドは、Quartusソフトウェアに含まれているFIR（Finite Impulse Response）フィルタのチュートリアルに適用されます。サポートされているTclコマンドの詳細については、Quartusのヘルプ機能で「Application Programming Interface Functions for Tcl」の項目をサーチして確認してください。また、サンプル・ファイルの中で使用されているTclの基本コマンドについては、10ページを参照してください。

Quartusプロジェクトの作成方法

Quartusソフトウェアでプロジェクトを作成するときに使用されるAPIの機能が表2に示されています。各タスクを実行する前に、新しいプロジェクトを作成するか、または既存のプロジェクトをオープンしておく必要があります。

表2　Quartusのプロジェクト・コマンド

コマンド	説明
`project exists <プロジェクト名>`	project existsコマンドにより、プロジェクトの作成時に、<プロジェクト名>という名前のプロジェクトがすでに存在するかを検証して、エラーの発生を防ぐ。
`project create <プロジェクト名>`	project create コマンドにより、<プロジェクト名>という名前のプロジェクトと<プロジェクト名>.quartusを含む関連する付属ファイルを作成する。
`project open <プロジェクト名>`	project open のコマンドを使用して <プロジェクト名> という名前のすでに作成済みの Quartus プロジェクトをオープンする。
`project close`	project close のコマンドを使用して、現在オープンされているプロジェクトをクローズする。

プロジェクトに対するアサインメントの指定方法

プロジェクトの作成が完了したら、Tclコマンドを使用してデザイン・ファイルの追加、アルテラ・デバイスの指定、プロジェクト全体または特定のエンティティに対するアサインメントの指定を行うことができます。表3に、プロジェクトに対するアサインメントの追加、または削除を行うときに使用されるコマンドが示されています。

表3　アサインメントの追加／削除コマンド

コマンド	説明
`project add_assignment <エンティティ><セクション識別子><ソース><ターゲット><変数><値>`	project add_assignment コマンドでプロジェクトに対するアサインメントを追加する。プロジェクト全体に対するアサインメントは、PSF または Quartus ファイルに書き込まれる。特定のエンティティに対するアサインメントは、ESF に書き込まれる。
`project remove_assignment <エンティティ><セクション識別子><ソース><ターゲット><変数><値>`	project remove_assignmentコマンドにより、プロジェクトから既存のアサインメントを取り除く。

表４は、表３に示されているコマンドの引数を定義したものです。

表４　Quartusプロジェクトの引数

引数	説明
<エンティティ>	**<エンティティ>の引数により、アサインメントが適用されているエンティティを定義する。**
<セクション識別子>	**<セクション識別子>の引数でセッティング・ファイル内のセクション名を定義する。**
<ソース>	**<ソース>の引数により、ターゲット範囲に対する開始インスタンス名を定義する。**
<ターゲット>	**<ターゲット>の引数により、<ソース>で開始されたターゲット範囲に対する終了名を定義する。**
<変数>	**<変数>の引数により、追加、変更または削除する変数を定義する。**
<値>	**<値>の引数により、変数に対して適用する値を定義する。**

適用しない変数の値は、（" "）の記号を使用してエンプティ・ストリングとして受け渡される必要があります。例えば、プロジェクトに対するソース・ファイルを定義する場合はプロジェクト全体に対するアサインメントとなるため、特定のエンティティに対する引数の指定は不要です。

アサインメントの方法に不安がある場合は、GUIをオープンしてアサインメントを行い、Quartusソフトウェアがどのファイルに書き込みを行ったかを確認してください。それがPSFまたはQuartusファイルになっている場合は、アサインメントがプロジェクト全体に対して適用されるため、特定のエンティティに対する引数の指定は不要です。また、それがESFになっている場合は、特定のエンティティに対する引数の指定が含まれている必要があります。この方法は、シンタックスのチェックにも有効です。

アサインメントを行うときのシンタックスとその使用方法などの詳細については、Quartusのヘルプ機能を利用して確認してください。

コンパイラおよびシミュレータ用設定条件の作成方法

デザインのコンパイルまたはシミュレーション行う前に、コンパイラまたはシミュレータの設定とアサインメントを行っておく必要があります。コンパイラの設定条件はCSFにセーブされ、シミュレータの設定条件はSSFにセーブされます。表５は、これらのファイルに使用されるコマンドを示したものです。

表5　コンパイラおよびシミュレータの設定条件を規定するコマンド

コマンド	説明
`project cmp_exists <設定条件名>`	<設定条件名>.csf という名前の CSF がすでに存在するかどうかをチェックする。
`project create_cmp <設定条件名>`	<設定条件名>.csf という名前の CSF を作成する。この機能は、プロジェクトに対するコンパイラの現在の設定条件を適用するときにも使用される。
`project set_active_cmp <設定条件名>`	Quartus ソフトウェアでコンパイルを実行するときに適用される条件を<設定条件名>で定義する。
`project sim_exists <設定条件名>`	<設定条件名>.ssf という名前の SSF がすでに存在するかどうかをチェックする。
`project create_sim <設定条件名>`	<設定条件名>.ssf という名前の SSF を作成する。
`project set_active_sim <設定条件名>`	Quartus ソフトウェアでシミュレーションを実行するときに適用される条件を<設定条件名>.ssf で定義する。

表6はCSFまたはSSF内のアサインメントを変更するときに使用できるコマンドを示したものです。

表6　CSF/SSF内でアサインメントを変更するときのコマンド

コマンド	説明
`cmp add_assignment <セクション識別子> <ソース><ターゲット><変数><値>`	現在のコンパイラ設定条件にアサインメントを追加する。
`cmp remove_assignment <セクション識別子> <ソース><ターゲット><変数><値>`	現在のコンパイラ設定条件からアサインメントを削除する。
`sim add_assignment <セクション識別子> <ソース><ターゲット><変数><値>`	現在のシミュレータ設定条件にアサインメントを追加する。
`sim remove_assignment <セクション識別子> <ソース><ターゲット><変数><値>`	現在のシミュレータ設定条件からアサインメントを削除する。

表7は、表6に示されたコマンドの引数を定義したものです。

表7　コンパイラおよびシミュレータの設定条件を規定する引数

引数	説明
<セクション識別子>	*<セクション識別子>*の引数により、アサインメントを行うCSFまたはSSFのセクション名を定義する。
<ソース>	*<ソース>*の引数により、ターゲット範囲に対する開始インスタンス名を定義する。
<ターゲット>	*<ソース>*で開始されたターゲット範囲に対する終了名を定義する。
<変数>	ここでCSFまたはSSFに追加、変更またはこれらから削除する変数を定義する。
<値>	ここでCSFまたはSSF内の変数に対して適用する値またはこれらから削除する値を定義する。

コンパイラおよびシミュレータに対するアサインメントを行うときのシンタックスとその使用方法などの詳細については、Quartusのヘルプ機能を使用して確認してください。

コンパイラのコントロール方法

Quartusのコンパイラに対する設定条件を指定した後で、このコンパイラをコントロールするときに使用できるコマンドが表8に示されています。

表8　コンパイラのコマンド

コマンド	説明
`cmp start`	コンパイラの設定条件をアクティブにしてコンパイラを起動する。
`cmp stop`	コンパイラを終了させる。
`cmp is_running`	コンパイラのステータスをチェックする。コンパイラが動作中であれば1の値が、停止していれば0の値が返される。

複数のコマンドを結合させて、条件つきでタスクを順番に実行することもできます。例えば、下記の図2に示すコードにより、コンパイルの状態をチェックし、コンパイラがまだ動作中であれば、その動作を終了させることができます。

図2　Tclコマンドを結合させた例

```
# check if the Compiler is running, and if so, stop it
if {[cmp is_running] == 1} {
  cmp stop
}
```

図3に示す例のように、whileのループを使用することによって、コンパイル中にスクリプトが実行されることを防ぐことができます。コンパイルが完了すると、Tclのインタプリタがこのループから抜けます。whileのループの中にあるafter以下のステートメントにより、インタプリタに対してコンパイラが動作中かどうかをチェックし、メッセージ（FlushEventQueueのステートメント）を流す前に、何ミリ秒の期間待つか、または停止するかが指示されます。リアル・タイムでのメッセージを必要とする場合はafterにLowの値を設定し、大規模なデザインをコンパイルするためにリアル・タイムのメッセージが重要でないときは、 Highの値を設定します。

図3　Whileループの例

```
while {[cmp is_running]} {
  after 1000
  FlushEventQueue
}
```

Quartusのメッセージをstdout、またはそれを取り扱うファイルに受け渡すときは、図4に示されているライブラリまたはコーリング・スクリプトの手順を使用します。

図4　Quartusのメッセージをパイプするときのコード記述例

```
proc postMessage {report msg} {
  # this function overrides the postMessage procedure in
  # quartus/bin/ccl_msg.tcl
  split $msg " "
  set line " Q> [lindex $msg 0] : [lindex $msg 3] "
  puts stdout $line
} ; #__________postMessage________#

proc InternalError {report text} {
  # this function overrides the InternalError procedure in
  # quartus/bin/ccl_msg.tcl
  puts stdout "Q>report (IE = $report"
  puts stdout "Q>msg (OE) = $text"
} ; #__________InternalError______"
```

シミュレータのコントロール方法

表9は、Quartusのシミュレータに対する設定条件を指定した後で、シミュレータをコントロールするときに使用できるコマンドを示したものです。

表9　シミュレータのコマンド

コマンド	説明
sim initialize	シミュレータをイニシャライズしてすべてのネットリストを読み込み、シミュレーション時間をゼロにセットする。
sim run *<時間> (1)*	シミュレータに対する設定条件をアクティブにし、シミュレータを起動する。*(2)*
sim stop	シミュレータを終了する。
sim is_initialized	シミュレータがイニシャライズされているかをチェックする。
sim is_running	シミュレータが動作中であるかどうかをチェックする。

注：
(1)　このコマンドの詳細については、QuartusのHelp機能で「sim run」の項目をサーチして確認してください。
(2)　このコマンドの詳細については、QuartusのHelp機能で「sim start」の項目をサーチして確認してください。

シミュレーションのインタラクティブな実行方法

Tclコマンドを使用することによって、Quartusのシミュレータにインタラクティブなシミュレーションを実行させることができます。表10には、デバッグのときに使用できる便利なコマンドが示されています。

表10　インタラクティブ・シミュレーションに使用するコマンド

コマンド	説明
`sim force_value <信号名><値>`	<信号名>で指定された信号の値を強制的に1か0にする（1はHighに、0はLowにする）。
`sim release_value <信号名>`	<信号名>で指定された信号に対する設定値を解除し、元の値に戻す。これによって、シミュレータはシミュレーションを実行しながら現在の値をオーバライトすることができる。
`sim get_value <信号名>`	<信号名>で指定された信号の値をチェックする。
`sim run <時間>` *(1)*	sim runのコマンドの後にこの値を追加して、シミュレータの実行時間の長さを規定する。*(2)*
`sim run end` *(1)*	シミュレーションが完了するまで、シミュレータを動作させる。
`sim get_time`	実行中のシミュレーションで現在の動作時間をチェックする。

注：
(1)　このコマンドの詳細については、QuartusのHelp機能で「sim run」の項目をサーチして確認してください。
(2)　このコマンドの詳細については、QuartusのHelp機能で「sim start」の項目をサーチして確認してください。

インタラクティブなシミュレーションを実行させるときの、シンタックスや使用方法などの詳細については、QuartusのHelp機能を利用して確認してください。

Tclスクリプトの例

図5は、Tclを使用してプロジェクトの作成、アサインメントの指定、および単純なコンパイルを実行するサンプル・ファイルを示したものです。

図5　Tclで単純なコンパイルを実行させる例

```
# Change to the working directory
cd D:/qdesigns/tutorial

# check the existence of a project, and if it exists, delete the files
if [project exists filtref] {
  file delete -force filtref.quartus
  file delete -force filtref.psf
  file delete -force filtref.esf
  file delete -force filtref.csf
  file delete -force filtref.ssf
  file delete -force db
}

# create project
project create filtref

# open project
project open filtref

# add source files to current project
project add_assignment "" "" "" "" SOURCE_FILE filtref.bdf
project add_assignment "" "" "" "" SOURCE_FILE acc.v
project add_assignment "" "" "" "" SOURCE_FILE accum.v
project add_assignment "" "" "" "" SOURCE_FILE hvalues.v
project add_assignment "" "" "" "" SOURCE_FILE mult.v
project add_assignment "" "" "" "" SOURCE_FILE state_m.v
project add_assignment "" "" "" "" SOURCE_FILE taps.v

# assign signal clk as a global signal
project add_assignment filref "" "" "" "" |clk" GLOBAL_SIGNAL ON

# create Compiler settings for filtref
project create_cmp filtref

# set the current Compiler settings to filtref
project set_active_cmp filtref

# assign device family
cmp add_assignment "" "" "" "" FAMILY APEX 20K

# assign device
cmp add_assignment filtref "" "" "" DEVICE EP20K100TC144-1

# Start compilation
cmp start
```

よく使用されるコマンド

Tclを使用すると、コマンドと実行手順をカスタマイズしてデザインの機能に適合させることができ、プロジェクトのコントロールと拡張が容易になります。このセクションでは、よく使用されるTclコマンドの例を示して、その使用方法を解説します。

マルチサイクル・パス

マルチサイクル・パスとは、意図的に2クロック・サイクル以上で安定するように設計されたパスを意味します。マルチサイクル・パスを宣言することによって、タイミング・アナライザが与えられたパスでセットアップ・タイム違反をリポートしないようにし、タイミング値の測定方法を変更して*x*クロック・サイクル（*x*は入力されるクロック・サイクル数）での測定を行うことができます。

図6に示されているマルチサイクル・パス・コマンドの使用例では、project_nameというプロジェクトで40MHzのf_{MAX}を必要とするbase_clockというクロック条件が作成されています。次に、base_clockの設定を使用する信号にclockが指定されています。そして、レジスタstartから、レジスタdata_outA_outまでのパスが3サイクルのマルチサイクル・パスに指定されています。

図6　マルチサイクル・パスの例

```
project add_assignment "" "base_clock" "" ""
  FMAX_REQUIREMENT 40MHZ
project add_assignment project_name "" "" "|clock"
  USE_CLOCK_SETTINGS base_clock
project add_assignment entity_name "" "|start"
  "|data_outA_out" MULTICYCLE 3
```

マルチクロック・ドメイン

マルチクロック・ドメインは、１個のデバイス内に２本以上のクロックが存在するクロックの構成を意味します。Tclを使用してマルチクロック・ドメインを設定することができます。

図７に示されているマルチクロック・ドメインの例では、project_name内にマルチクロック・ドメインが指定されています。ここには、clock1とclock2の２本のクロックが存在し、それぞれclock_40MHzとclock_32MHzの条件が指定されています。ここでは、clock_40MHzが絶対クロックとなっており、clock_32MHzはclock_40MHzを基準に生成されるクロックに指定されています。

図７　マルチクロック・ドメインの例

```
project add_assignment "" "clock_40MHz" "" ""
  FMAX_REQUIREMENT 40MHz
project add_assignment "" "clock_32MHz" "" ""
  BASED_ON_CLOCK_SETTINGS clock_40MHz
project add_assignment "" "clock_32MHz" "" ""
  MULTIPLY_BASE_CLOCK_PERIOD_BY 4
project add_assignment "" "clock_32MHz" "" ""
  DIVIDE_BASE_CLOCK_PERIOD_BY 5
project add_assignment project_name "" "" "|clock2"
  USE_CLOCK_SETTINGS clock_40MHz
project add_assignment project_name "" "" "|clock1"
  USE_CLOCK_SETTINGS clock_32MHz
```

t_{PD}アサインメントの追加

伝播遅延時間(t_{PD})は、入力ピンからの信号が組み合わせ回路を通って出力ピンに現れるまでに必要な時間です。このt_{PD}の条件をプロジェクト全体、および任意の入力、出力、またはバッファ・ピンに指定することができます。

デザイン全体にt_{PD}を指定するときは、下記のTclコマンドを使用します。この例では、デザイン全体に11nsのt_{PD}が指定されています。

```
project add_assignment "" "" "" "" TPD_REQUIREMENT 11ns
```

ロジック・セル、1cellから出力ピンまでのt_{PD}を指定するときは、下記のTclコマンドを使用します。この例では、1cellから出力ピンまでのt_{PD}が4nsに設定されています。

```
project add_assignment entity_name "" "|1cell"
  "|output_pin" TPD_REQUIREMENT 4ns
```

デバイスの追加

プロジェクトにデバイスを追加するときは、cmp add_assignmentのコマンドを使用します。下記の例は、Tclスクリプトを使用してAPEX 20KまたはAPEX 20KEファミリのデバイスを追加する方法を示しています。

```
cmp add_assignment "" "" "" FAMILY <family name>
cmp add_assignment entity_name "" "" DEVICE <device name>
```

ここで
<*family name*> = APEX20K, APEX20KE
<*device_name*> = 指定されたデバイス・ファミリ内の有効な任意のデバイス

ピン配置の追加

ピン・アサインメントを追加するときは、下記のTclコマンドを使用します。

```
cmp add_assignment <chip_name> "" "<signal_name>" LOCATION
Pin_<pin>
```

ここで
<*chip_name*> = チップ名、通常はプロジェクト名
<*signal_name*> = I/O信号名
<*pin*> = 追加されるI/Oピンの有効なピン番号

Verilog HDLおよびVHDLシミュレーション・ファイルの生成

標準的なVerilog HDLまたはVHDLシミュレータ用のVerilog HDLまたはVHDLシミュレーション・ファイルを生成するときは、下記のTclコマンドを使用します。

```
project add_assignment "" project_name "" ""
  EDA_SIMULATION_TOOL <simulation tool>
```

ここで
<*simulation tool*>= Modelsim, SpeedWave, VCS, Verilog-XL, VSS, Custom Verilog HDL, or Custom

EDA合成ツールの使用

シノプシス社のDesign CompilerやFPGA Express、メンター・グラフィック社（エグゼンプラ）のLeonardoSpectrum、シンプリシティ社のSynplify、イノヴェータ社（ビューロジック）のViewDrawのような業界標準のEDAツールとのインタフェースを行うときは、下記のTclコマンドを使用します。

```
project add_assignment "" project_name "" ""
  EDA_DESIGN_ENTRY_SYNTHESIS_TOOL <EDA tool>
```

ここで
<EDA tool> = `Design Architect, Design Compiler, FPGA Compiler II, FPGA Express, Leonardo Spectrum, Synplify, ViewDraw, Custom`

テクノロジ・マッパの設定

テクノロジ・マッパの設定により、コンパイラに対してデザインの階層をROM、プロダクト・ターム、ルック・アップ・テーブル（LUT）またはAUTO（どのロジック構成にするか自動的に決定される）で実現するかを指示することができます。テクノロジ・マッパの指定を行うときは、下記のTclコマンドを使用します。

```
project add_assignment entity_name "" "" ""
  TECHNOLOGY_MAPPER <technology>
```

ここで、
<technology> = `ROM, product_term, LUT, AUTO`

最適化テクニックの設定

プロジェクトをエリアまたはタイミングに対して最適化するときは、下記のようなTclコマンドにより最適化テクニックを指定することができます。

```
project add_assignment "" "" "" ""
  OPTIMIZATION_TECHNIQUE <technique>
```

ここで
<technique> = `speed, area`

アルテラは、Quartusソフトウェアによるデザインをサポートするために広範囲な資料を提供しています。技術的なサポートがご必要な場合は、日本アルテラまたは販売代理店へご連絡ください。また、日本アルテラでは、電子メールによるご質問も受け付けています。japan@altera.comへ日本語のメールを送付頂くか、QuartusソフトウェアからQuartus Web Supportにダイレクトに接続することもできます。

参考資料

Tclの詳細については、下記の参考資料を参照してください。

- *Practical Programming in Tcl and TK*, Brent B. Welch.
- *Tcl and TK Toolkit*, John Ousterhout.
- *Effective Tcl/TK Programming*, Michael McLennan and Mark Harrison.
- http://www.scriptics.com

日本アルテラ株式会社

〒163-1332
東京都新宿区西新宿6-5-1
新宿アイランドタワー32F 私書箱1594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