

はじめに

ツールコマンド言語 (Tcl) スクリプトを開発および実行してアルテラの Quartus® II ソフトウェアを制御することによって、デザインをコンパイルしたり、共通タスクを自動化するプロシーダを記述したりするなど、さまざまな機能を実行することが可能になります。

Tcl スクリプトは、Quartus II プロジェクトの管理、アサインメントの作成、デザイン制約の定義、デバイス・アサインメントの作成、コンパイルの実行、タイミング解析の実行、LogicLock™ 領域アサインメントのインポート、Quartus II Chip Editor の使用、およびレポートへのアクセスに使用することができます。Quartus II アサインメントは Tcl スクリプトを使用して自動化することができるため、個別に作成する必要はありません。また Tcl スクリプトによってプロジェクトまたはアサインメントの移行が容易になります。例えば、同じプロトタイプまたは開発ボードを異なるプロジェクトに対して使用するときには、新しい各プロジェクトにおけるピン配置の再アサインメントを自動化することができます。Quartus II ソフトウェアはまた、プロジェクトにおけるすべての現在のアサインメントに基づいて Tcl スクリプトを生成し、アサインメントを別のプロジェクトに簡単に切り替えることができますようにします。

Quartus II ソフトウェアの Tcl コマンドは、EDA 業界の Tcl アプリケーション・プログラミング・インタフェース (API) に基づいて、コマンドライン・オプションを使用して引数を指定します。そのため、Tcl コマンドは簡単に習得し使用できます。コマンド引数を使用してエラーが発生した場合、Tcl インタプリタは正しい使用法を示すヘルプ情報を提示します。

この章では、Quartus II ソフトウェアを自動化するための Tcl スクリプトのサンプルを示します。これらのサンプル・スクリプトを変更して独自のデザインで使用することができます。その他の Tcl スクリプトは、アルテラ・ウェブサイトのサポートの設計例セクションで入手可能です。

Tcl とは何ですか？

Tcl (ティクルと読みます) は、多くのシェル・スクリプト言語および高水準プログラミング言語に似た定評のあるスクリプト言語です。制御構造、変数、ネットワーク・ソケット・アクセス、および API がサポートされています。Tcl は、Synopsys、Mentor Graphics®、Synplicity、および Altera ソフトウェアで採用されている、EDA 業界標準のスクリプト言語です。これによって、カスタム・コマンドを作成し、大部分の開発プラットフォームの間でシームレスに作業することが可能になります。Tcl に関する推奨文献のリストについては、[3-52 ページの「参考文献」](#)を参照してください。

基本的な Tcl コマンド、ユーザ定義のプロシージャ、および Quartus II API 関数を含むスクリプトを記述することによって、独自のプロシージャを作成することができます。そして、デザイン・フローを自動化したり、バッチ・モードで Quartus II ソフトウェアを実行したり、Quartus II Tcl のインタラクティブ・シェルで個々の Tcl コマンドを対話的に実行したりすることができます。

Tcl スクリプティングについて熟知していない場合、または Tcl の初心者である場合は、Tcl スクリプティングの概要については、[3-44 ページの「Tcl スクリプティングの基礎知識」](#)を参照してください。

Quartus II ソフトウェアは、バージョン 4.1 以降で、tcl.activestate.com の Tcl DeveloperXchange から提供される Tcl/Tk バージョン 8.4 をサポートしています。

Quartus II Tcl パッケージ

Quartus II Tcl コマンドはパッケージ内で関数ごとにグループ化されています。表 3-1 で各 Tcl パッケージについて説明します。

| パッケージ名 | パッケージの説明 |
|---------------------------------|---|
| advanced_timing | タイミング・ネットリストを横断し、タイミング・ノードについての情報を取得します。 |
| backannotate | バック・アノテート・アサインメント |
| chip_editor | リソースの使用量および Chip Editor との配線を識別および変更します。 |
| database_manager | バージョン・コンパチブル・データベース・ファイルを管理します。 |
| device | デバイス・データベースからデバイスおよびファミリの情報を取得します。 |
| flow | プロジェクトをコンパイルし、実行コマンドおよびその他の共通フローを実行します。 |
| insystem_memory_edit | アルテラ・デバイス内のメモリ内容を読み取り、編集します。 |
| jtag | JTAG チェーンを制御します。 |
| logic_analyzer_interface | ロジック・アナライザのインタフェース出力ピン状態を調べ、変更します。 |
| LogicLock | LogicLock 領域を作成し管理します。 |
| misc | その他のタスクを実行します。 |
| project | プロジェクトおよびリビジョンを作成および管理し、タイミング・アサインメントを含むプロジェクト・アサインメントを作成します。 |
| report | レポートの表から情報を取得し、カスタム・レポートを作成します。 |
| sdq | TimeQuest アナライザに対して制約および例外を指定します。 |
| simulator | シミュレーションをコンフィギュレーションし実行します。 |
| sta | TimeQuest タイミング・アナライザから高度な情報を取得する Tcl ファンクションのセットを含みます。 |
| stp | SignalTap® II ロジック・アナライザを実行します。 |
| timing | タイミング・ネットリストを遅延情報とともにアノテートし、タイミング・パスを計算しレポートします。 |

| パッケージ名 | パッケージの説明 |
|-------------------|---|
| timing_assignment | クロック・アサインメントを含めて、プロジェクト全体のタイミング・アサインメントを行う Tcl ファンクションのセットを含みます。クラシック・タイミング・アナライザのアサインメントを処理するよう設計されたすべてのTclコマンドは、このパッケージに移動されています。 |
| timing_report | タイミング・パスをリストします。 |

デフォルトでは、最少数のパッケージのみが各 Quartus II 実行コマンドで自動的にロードされます。これによって、各実行コマンドのメモリ要件が可能な限り軽減されます。最少数のパッケージが自動的にロードされるため、その他のパッケージのコマンドを実行するには、それらのパッケージをユーザによってロードする必要があります。

表 3-2 に、Quartus II 実行コマンドで利用可能な Quartus II Tcl パッケージをリストし、パッケージがデフォルトでロードされる (●) か、または必要に応じてロードすることが可能である (◐) かを示します。白い円 (○) は、パッケージがその実行コマンドで利用できないことを意味します。

| パッケージ | Quartus II 実行コマンド | | | | | | |
|--------------------------|-------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | Quartus_sh | Quartus_tan | Quartus_cdb | Quartus_sim | Quartus_stp | Quartus_sta | Tcl Console |
| advanced_timing | ○ | ◐ | ○ | ○ | ○ | ○ | ○ |
| backannotate | ○ | ○ | ◐ | ○ | ○ | ○ | ◐ |
| chip_editor | ○ | ○ | ◐ | ○ | ○ | ○ | ○ |
| device | ● | ◐ | ● | ● | ○ | ● | ◐ |
| flow | ◐ | ◐ | ◐ | ◐ | ○ | ◐ | ◐ |
| insystem_memory_edit | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| jtag | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| logic_analyzer_interface | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| logiclock | ○ | ◐ | ◐ | ○ | ○ | ○ | ◐ |

表 3-2. Quartus II 実行コマンドで利用可能な Tcl パッケージ (2 / 2)

| パッケージ | Quartus II 実行コマンド | | | | | | |
|-------------------|-------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | Quartus_sh | Quartus_tan | Quartus_cdb | Quartus_sim | Quartus_stp | Quartus_sta | Tcl Console |
| misc | ● | ● | ● | ● | ● | ● | ● |
| old_api | ○ | ○ | ○ | ○ | ○ | ○ | ● |
| project | ● | ● | ● | ● | ● | ● | ● |
| report | ◐ | ◐ | ◐ | ● | ○ | ● | ◐ |
| sdc | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| simulator | ○ | ○ | ○ | ● | ○ | ○ | ○ |
| sta | ○ | ○ | ○ | ○ | ○ | ● | ○ |
| stp | ○ | ○ | ○ | ○ | ● | ○ | ○ |
| timing | ○ | ● | ○ | ○ | ○ | ○ | ○ |
| timing_assignment | ● | ● | ● | ● | ● | ○ | ○ |
| timing_report | ○ | ◐ | ○ | ○ | ● | ○ | ● |

表 3-2 の注：

- (1) 黒い円 (●) は、パッケージが自動的にロードされることを意味します。
- (2) 半円 (◐) は、パッケージが利用できるものの自動的にロードされないことを意味します。
- (3) 白い円 (○) は、パッケージがその実行コマンドで利用できないことを意味します。

利用できるパッケージは実行コマンドごとに異なるため、スクリプトで使用するパッケージを含む実行コマンドでスクリプトを実行する必要があります。例えば、**timing** パッケージのコマンドを使用する場合、**timing** パッケージをサポートするのは **quartus_tan** 実行コマンドのみであるため、**quartus_tan** 実行コマンドを使用してスクリプトを実行する必要があります。

パッケージのロード

Quartus II Tcl パッケージをロードするには、**load_package** コマンドを次のように使用します。

```
load_package [-version <バージョン番号>] <パッケージ名>
```

このコマンドは Tcl コマンドの **package require** と似ていますが (3-8 ページの表 3-3 で説明)、**load_package** コマンドを使用して異なるバージョンの Quartus II Tcl パッケージの間で容易に切り替えることができます。



これらのコマンドおよびその他の Quartus II 実行コマンドについて詳しくは、「Quartus II ハンドブック Volume 2」の「Command-Line Scripting」の章を参照してください。

Quartus II Tcl API ヘルプ

システム・コマンド・プロンプトで次のコマンドを入力して、Quartus II Tcl API ヘルプ・リファレンスにアクセスします。

```
quartus_sh --qhelp ←
```

このコマンドによって Quartus II コマンドラインおよび Tcl API ヘルプ・ブラウザが起動し、このブラウザに Quartus II Tcl API のコマンドおよびオプションのすべてが示されます。各コマンドについての詳細な説明と例が含まれています。

さらに、Tcl API ヘルプの情報は「Quartus II Scripting Reference Manual」でも参照でき、これはアルテラ・ウェブサイトの Quartus II 資料のページから PDF 形式で入手することができます。

Quartus II Tcl ヘルプでは、Quartus II Tcl コマンドについての情報に簡単にアクセスすることができます。ヘルプ情報にアクセスするには、例 3-1 に示すように Tcl プロンプトで help と入力します。

例 3-1. ヘルプの出力

```
tcl> help
```

```
-----
-----
利用可能な QuartusII Tcl パッケージ
-----
-----
```

| Loaded | Not Loaded |
|------------------------------|-------------------------|
| ::quartus::misc | ::quartus::device |
| ::quartus::old_api | ::quartus::backannotate |
| ::quartus::project | ::quartus::flow |
| ::quartus::timing_assignment | ::quartus::logicclock |
| ::quartus::timing_report | ::quartus::report |

```
* Type "help -tcl"
  to get an overview on Quartus II Tcl usages.
```

help で -tcl オプションを使用すると、Tcl コマンド（ショート・ヘルプおよびロング・ヘルプ）と Tcl パッケージについてのヘルプ情報を取得する方法を中心とした Quartus II Tcl API の概要が表示されます。



また、Tcl API ヘルプは Quartus II オンライン・ヘルプでも参照することができます。コマンドまたはパッケージについての詳細については、そのコマンドまたはパッケージの名前を検索します。

表 3-3 に、Tcl 環境で利用可能なヘルプ・オプションを示します。

| 表 3-3. Quartus II Tcl 環境で利用可能なヘルプ・オプション (1 / 2) | |
|---|--|
| ヘルプ・コマンド | 説明 |
| help | ロードされた、またはロードされていない利用可能な Quartus II Tcl パッケージのリストを表示します。 |
| help -tcl | Tcl パッケージをロードし、コマンドライン・ヘルプにアクセスするためのコマンドのリストを表示します。 |
| help -pkg <パッケージ名> [-version <バージョン番号>] | <p>利用可能な Tcl コマンドのリストを含む、指定した Quartus II パッケージのヘルプを表示します。簡単にするために、パッケージ・プレフィックスの ::quartus:: を省略し、help -pkg <パッケージ名> ↵ と入力することもできます。</p> <p>-version オプションを指定しない場合、現在ロードされているパッケージのヘルプがデフォルトで表示されます。ヘルプを表示させるパッケージがロードされていない場合、最新バージョンのパッケージのヘルプがデフォルトで表示されます。</p> <p>例： help -pkg ::quartus::p ↵ help -pkg ::quartus::project ↵ help -pkg project rhelp -pkg project -version 1.0 ↵</p> |
| <コマンド名> -h または <コマンド名>-help | <p>パッケージがロードされている Quartus II Tcl コマンドのショート・ヘルプを表示します。</p> <p>例： project_open -h ↵ project_open -help ↵</p> |
| package require ::quartus::<パッケージ名> [<バージョン>] | <p>バージョンを指定して Quartus II Tcl パッケージをロードします。<version> が指定されていない場合、最新バージョンのパッケージがデフォルトでロードされます。</p> <p>例： package require ::quartus::project 1.0 ↵</p> <p>このコマンドは load_package コマンドに類似しています。load_package を使用する利点は、バージョンが異なる同一のパッケージ間で自由に切り替えができることです。</p> <p>バージョンを指定して Quartus II Tcl パッケージをロードするには、<パッケージ名> [-version <バージョン番号>] ↵ と入力します。-version オプションが指定されていない場合、最新バージョンのパッケージがデフォルトでロードされます。</p> <p>例： load_package ::quartus::project -version 1.0 ↵</p> |

表 3-3. Quartus II Tcl 環境で利用可能なヘルプ・オプション (2 / 2)

| ヘルプ・コマンド | 説明 |
|--|--|
| <pre>help -cmd < コマンド名 > [-version < バージョン番号 >] または < コマンド名 > -long_help</pre> | <p>Quartus II Tcl コマンドのロング・ヘルプを表示します。 < コマンド名 > -long_help の場合のみ、関連する Tcl パッケージがロードされていることが必要です。 -version オプションを指定しない場合、現在ロードされているパッケージのヘルプがデフォルトで表示されます。 ヘルプを表示させるパッケージがロードされていない場合、最新バージョンのパッケージのヘルプがデフォルトで表示されます。</p> <p>例： <pre>project_open -long_help ↵ help -cmd project_open ↵ help -cmd project_open -version 1.0 ↵</pre></p> |
| <pre>help -examples</pre> | <p>Quartus II Tcl パッケージの使用例を表示します。</p> |
| <pre>help -quartus</pre> | <p>現在実行中の Quartus II 実行コマンドについての情報を表示するためにアクセスすることができる定義済みのグローバル Tcl 配列についてのヘルプを表示します。</p> |
| <pre>quartus_sh --qhelp</pre> | <p>Quartus II コマンドライン・ヘルプの Tk ビューアを起動し、実行コマンドおよび Tcl API パッケージのヘルプを表示します。</p> <p>このユーティリティについて詳しくは、「Quartus II ハンドブック Volume 2」の「Command-Line Scripting」の章を参照してください。</p> |

Tcl をサポート する実行 コマンド

一部の Quartus II 実行コマンドは Tcl スクリプティングをサポートしています (表 3-4 を参照)。各実行コマンドは、Tcl パッケージの異なるセットをサポートしています。スクリプトの実行に適切な実行コマンドを決定するには、表 3-4 を参照してください。

| コマンド名 | 実行コマンドの説明 |
|--------------------|--|
| quartus_sh | Quartus II シェルは、アサインメント、一般的なレポート、およびコンパイルに便利なシンプルな Tcl スクリプティング・シェルです。 |
| quartus_tan | Quartus II クラシック・タイミング・アナライザを使用して、シンプルなタイミング・レポートおよび高度なタイミング解析を実行します。 |
| quartus_cdb | Quartus II コンパイル・データベースは、バック・アノテーション、LogicLock 領域の操作、および Chip Editor ファクションをサポートしています。 |
| quartus_sim | Quartus II シミュレータは、デザイン・シミュレーションの自動化をサポートしています。 |
| quartus_sta | TimeQuest タイミング・アナライザは、制約入力およびレポートの SDC ターミノロジをサポートしています。 |
| quartus_stp | Quartus II SignalTap II 実行コマンドはイン・システム・デバッグ・ツールをサポートしています。 |

quartus_tan および **quartus_cdb** 実行コマンドは、**quartus_sh** 実行コマンドでサポートされるパッケージのスーパーセットをサポートしています。プロジェクト管理コマンドおよびアサインメント・コマンドでのみ Tcl スクリプトを実行する場合、またはメモリ・フットプリントの小さな Quartus II 実行コマンドが必要な場合は、**quartus_sh** 実行コマンドを使用します。



これらの実行コマンドについて詳しくは、「Quartus II ハンドブック Volume 2」の「Command-Line Scripting」の章を参照してください。

コマンドライン・オプション `-t`、`-s` および `--tcl_eval`

表 3-5 に、Tcl をサポートする実行コマンドで使用することができる 3 つのコマンドライン・オプションを示します。

| コマンドライン・オプション | 説明 |
|---|---|
| <code>-t <スクリプト・ファイル></code> [<スクリプト引数>] | 指定した Tcl スクリプトをオプションの引数付きで実行します。 |
| <code>-s</code> | インタラクティブ Tcl シェル・モードで実行コマンドを開きます。 |
| <code>--tcl_eval <Tcl コマンド></code> | 残りのコマンドライン引数を Tcl コマンドとして評価します。例えば、次のコマンドを実行すると、プロジェクト・パッケージのヘルプが表示されます。 <code>quartus_sh --tcl_eval help -pkg project</code> |

Tcl スクリプトの実行


`-t` オプションを指定して実行コマンドを実行すると、指定した Tcl スクリプトが起動します。またスクリプトに引数を指定することもできます。`argv` 変数を利用して引数にアクセスするか、または次の形式の引数をサポートする **cmdline** などのパッケージを使用します。

`-<引数名> <引数値>`

cmdline パッケージは、<Quartus II ディレクトリの>/common/tcl/packages ディレクトリに格納されています。

例えば、**myscript.tcl** という名前のスクリプトを、1 つの引数 `Stratix` を指定して実行するには、システム・コマンド・プロンプトで次のコマンドを入力します。

```
quartus_sh -t myscript.tcl Stratix ↵
```

 バージョン 4.1 以降では、Quartus II ソフトウェアは `argv` 変数をサポートしています。以前のソフトウェア・バージョンでは、スクリプト引数は `quartus (args)` グローバル変数でアクセスされていました。

詳しくは、3-37 ページの「コマンドライン引数へのアクセス」を参照してください。

インタラクティブ・シェル・モード

-s オプションを指定して実行コマンドを実行すると、インタラクティブ Tcl シェルが起動し、tcl> プロンプトが表示されます。例えば、クラシック・タイミング・アナライザの実行コマンドをインタラクティブ・シェル・モードで開くには、システム・コマンド・プロンプトで `quartus_tan -s` と入力します。Tcl シェルに入力したコマンドは、**Enter** を押したときに解釈されます。次のコマンドを使用して Tcl スクリプトをインタラクティブ・シェルで実行することができます。

```
source <スクリプト名> ↵
```

コマンドは、シェルで認識されない場合、外部コマンドと解釈され、**exec** コマンドで実行されます。

Tcl としての評価

--tcl_eval オプションを指定して実行コマンドを実行すると、その実行コマンドは残りのコマンドライン引数を Tcl コマンドとして直ちに評価します。これは、シンプルな Tcl コマンドをその他のスクリプト言語から実行する場合に便利です。

例えば、次のコマンドは、**project** パッケージで利用可能なコマンドを出力する Tcl コマンドを実行します。

```
quartus_sh --tcl_eval help -pkg project ↵
```

Quartus II Tcl Console ウィンドウの使用

Tcl コマンドは、Quartus II Tcl Console ウィンドウで直接実行することができます。View メニューで、**Utility Windows** をクリックします。デフォルトでは、Tcl Console ウィンドウは Quartus II GUI の右下にドッキングされています。Tcl Console に入力した内容はすべて、Quartus II Tcl シェルによって解釈されます。



Quartus II Tcl Console ウィンドウは、古いデザインおよび EDA ツールとの下位互換性を実現するために、Quartus II ソフトウェア・バージョン 3.0 およびそれ以前で使用される Tcl API をサポートしています。

Tcl メッセージは **System** タブ (Message ウィンドウ) に表示されます。また、stdout および stderr に書き込まれるエラーおよびメッセージは、Quartus II Tcl Console ウィンドウにも表示されます。

終端間 デザイン・ フロー

Quartus II GUI の Tcl Console で実行できるタイミング解析には制限があることに注意してください。 **timing_report** パッケージでは、 **list_path** コマンドを使用して、タイミング・レポートにリストされたパスの詳細を取得することができます。ただし、タイミング・レポートにリストされていないタイミング・パスについての情報を取得する場合は、 **quartus_tan** 実行コマンドをシェル・モードで使用するか、または目的のパスをレポートするスクリプトを実行する必要があります。

デザインで TimeQuest タイミング・アナライザを使用する場合、スクリプト化されたタイミング解析を TimeQuest Tcl Console で実行する必要があります。

表 3-2 に示すように、Quartus II GUI の Tcl Console は、すべてのパッケージをサポートしているわけではないため、サポートされていないパッケージのコマンドを使用するスクリプトを実行することはできません。

Tcl スクリプトを使用して、スクリプティング・インタフェースを含む他のソフトウェアの制御を含め、デザイン・フローのすべての側面を制御することができます。

一般に、EDA ツールには、コア言語の機能をツール固有のコマンドで拡張した独自のスクリプト・インタプリタが含まれています。例えば、Quartus II Tcl インタプリタは、すべてのコア Tcl コマンドをサポートするとともに、Quartus II ソフトウェアに固有の多数のコマンドが追加されています。1 つの Tcl スクリプトにコマンドを含めて別のスクリプトを実行することができ、それによってスクリプトを結合または連結して異なるツールを制御することが可能になります。異なるツールのスクリプトは異なる Tcl インタプリタで実行する必要があるため、1 つのスクリプトでファイルに情報を書き込み、他のスクリプトでそれを読み取らなければ、スクリプト間で情報を受け渡すことが困難になります。

Quartus II ソフトウェアでは、単一のスクリプトからデザイン・フロー内の多数のさまざまな操作（合成、フィッティング、タイミング解析など）を実行することができ、グローバル・ステート情報の維持と操作間でのデータの受け渡しが容易になります。ただし、各実行コマンドでさまざまなパッケージがサポートされているため、単一のスクリプトで実行できる操作にはいくつかの制限があります。例えば、 **advanced_timing** パッケージ内のコマンドを使用しながら **simulator** パッケージ内のコマンドでシミュレーションを実行する単一のスクリプトを記述することはできません。これらの 2 つのパッケージは、同じ実行コマンドでは利用することができません。Tcl シミュレーションおよび高度なタイミング解析コマンドを含める場合、2 つのスクリプトを記述する必要があります。


実行コマンドからのフローの実行に対しては、制限事項はありません。フローは、Quartus II GUI の Processing メニューの Start セクションにある操作を含み、`execute_flow` Tcl コマンドでも表示されます。Quartus II ソフトウェアで設定することができ、フローを実行して目的の結果が得られる場合は、同じ設定を行い、同じフローを任意の実行コマンドで実行することができます。

シミュレーションおよびタイミング解析を含む例を再度実行するために、タイミング解析をコンフィギュレーションする設定とともに、シミュレーションをコンフィギュレーションする設定を含む 1 つのスクリプトを記述することができます。その場合、`execute_flow` コマンドでシミュレーションおよびタイミング解析のフローを実行します。

シミュレーションのコンフィギュレーションでは、シミュレーション・ファイルの名前と場所、シミュレーションの継続期間、グリッチ検出を実行するかどうかなどの設定を指定することが必要です。タイミング解析のコンフィギュレーションでは、必要なクロック周波数、レポートするパスの数、どのタイミング・モデルを使用するかなどの設定を指定することが必要です。この設定を行うと、どの Quartus II 実行コマンドでも `execute_flow` コマンドでフローを実行することができます。

プロジェクトの作成およびアサインメントの作成

Tcl スクリプティング API の利点の 1 つは、既存のプロジェクトに対してすべてのアサインメントを作成するスクリプトを容易に作成できることです。スクリプトはいつでも使用でき、プロジェクト設定を既知の状態に復元することができます。Project メニューから、**Generate Tcl File for Project** をクリックして、すべてのアサインメントを含む Tcl ファイルを自動的に生成します。このファイルを基にしてプロジェクトを再作成し、ファイルを編集して、デザインをコンパイルするなどのその他のコマンドを追加することができます。このファイルは、プロジェクト管理コマンドおよびアサインメント・コマンドについて習得する上で、適切な開始点となります。

 スクリプトのソーシングについて詳しくは、[3-12 ページの「インタラクティブ・シェル・モード」](#)を参照してください。すべての Quartus II プロジェクト設定およびアサインメントについてのスクリプト情報は、「[QSF Reference Manual](#)」にあります。

例 3-2 に、プロジェクトの作成方法、アサインメントの作成方法、およびプロジェクトのコンパイル方法を示します。これは、`fir_filter` チュートリアル・デザイン・ファイルを使用しています。

例 3-2. プロジェクトの作成およびコンパイル

```
load_package flow

# プロジェクトを作成し、
# 既存の設定ファイルを上書き
project_new fir_filter -revision filtref -overwrite
# デバイス、トップレベル BDF の名前、
# およびトップレベル・エンティティの名前を設定
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
set_global_assignment -name BDF_FILE filtref.bdf
set_global_assignment -name TOP_LEVEL_ENTITY filtref
# その他のピン・アサインメントをここで追加
set_location_assignment -to clk Pin_G1
# ベース・クロックおよび派生クロックを作成
create_base_clock -fmax "100 MHz" -target clk clocka
create_relative_clock -base_clock clocka -divide 2 \
    -offset "500 ps" -target clkx2 clockb
# デザイン内の 2 つのクロック・ドメイン間で、
# 2 のマルチサイクル・アサインメントを作成
set_multicycle_assignment -from clk -to clkx2 2
execute_flow -compile
project_close
```



プロジェクトを開いている間に作成または修正されたアサインメントは、**export_assignments** または **project_close** (**dont_export_assignments** が指定されていない場合) を明示的に呼び出さなければ、Quartus II 設定ファイルに登録されません。**execute_flow** を実行するときなど、場合によっては、Quartus II ソフトウェアは自動的に変更内容を登録します。

HardCopy デバイスのデザイン



HardCopy II デザインに対するスクリプト化されたデザイン・フローについて詳しくは、「HardCopy シリーズ・ハンドブック」の「Script-Based Design Flow for HardCopy Devices」の章を参照してください。この章には、サンプル・スクリプトと HardCopy II デザインを容易にするための推奨事項が記載されています。

また、「HardCopy シリーズ・ハンドブック」の別の章の「Timing Constraints for HardCopy II」には、HardCopy II デバイスに対するスクリプト・ベースの設計についての情報が、タイミング制約を中心に記載されています。

EDA ツールのアサインメント

Tcl で Tcl コマンドの `set_global_assignemnt` を使用して、EDA ツールの対象を Quartus II ソフトウェアのプロジェクトとすることができます。各 EDA ツールに対してデフォルトのツール設定を使用するには、使用する EDA ツールを指定します。Quartus II ソフトウェアで利用可能な EDA インタフェースは、デザイン・エントリ、シミュレーション、タイミング解析、およびボード・デザイン・ツールに対応しています。フォーマル検証や再合成などのより高度な EDA ツールは、それらの独自のグローバル・アサインメントによってサポートされます。

デフォルトでは、EDA インタフェース・オプションは `<none>` に設定されています。表 3-6 に、Quartus II ソフトウェアで利用可能な EDA インタフェース・オプションのリストを示します。スペースを含むインタフェース・アサインメント・オプションは引用符で囲みます。

表 3-6. Quartus II ソフトウェアでの EDA インタフェース・オプション (1 / 2)

| オプション | 指定可能な値 |
|--|---|
| デザイン・エントリ (EDA_DESIGN_ENTRY_SYNTHESIS_TOOL) | <ul style="list-style-type: none"> ● Design Architect ● Design Compiler ● FPGA Compiler ● FPGA Compiler II ● FPGA Compiler II Altera Edition ● FPGA Express ● LeonardoSpectrum™ ● LeonardoSpectrum-Altera (Level 1) ● Synplify ● Synplify Pro ● ViewDraw ● Precision Synthesis ● Custom |
| Simulation (EDA_SIMULATION_TOOL) | <ul style="list-style-type: none"> ● ModelSim (VHDL output from the Quartus II software) ● ModelSim (Verilog HDL output from the Quartus II software) ● ModelSim-Altera (VHDL output from the Quartus II software) ● ModelSim-Altera (Verilog HDL output from the Quartus II software) ● SpeedWave ● VCS ● Verilog-XL ● VSS ● NC-Verilog (Verilog HDL output from the Quartus II software) ● NC-VHDL (VHDL output from the Quartus II software) ● Scirocco (VHDL output from the Quartus II software) ● Custom Verilog HDL ● Custom VHDL |

表 3-6. Quartus II ソフトウェアでの EDA インタフェース・オプション (2 / 2)

| オプション | 指定可能な値 |
|--|---|
| タイミング解析 (EDA_TIMING_ANALYSIS_TOOL) | <ul style="list-style-type: none"> ● PrimeTime (VHDL output from the Quartus II software) ● PrimeTime (Verilog HDL output from the Quartus II software) ● Stamp (board model) ● Custom Verilog HDL ● Custom VHDL |
| Board level tools (EDA_BOARD_DESIGN_TOOL) | <ul style="list-style-type: none"> ● Signal Integrity (IBIS) ● Symbol Generation (ViewDraw) |
| フォーマル検証 (EDA_FORMAL_VERIFICATION_TOOL) | <ul style="list-style-type: none"> ● Conformal LEC |
| 再合成 (EDA_RESYNTHESIS_TOOL) | <ul style="list-style-type: none"> ● PALACE ● Amplify |

例えば、NC-Sim Verilog シミュレーション出力ファイルを生成するには、[例 3-3](#) に示すように、EDA_SIMULATION_TOOL は目的の出力として NC-Sim Verilog をターゲットとするように設定する必要があります。

例 3-3.

```
set_global_assignment -name eda_simulation_tool \  
"NcSim (Verilog HDL output from Quartus II)"
```



サードパーティのシミュレーション・ツールの使用については、「Quartus II ハンドブック Volume 3」を参照してください。

例 3-4 は、fir_filter デザイン・ファイルをコンパイルして NC-Sim Verilog シミュレーション用の VHDL Output (.vho) ファイル出力を生成する状態を示します。

例 3-4. .vho 出力を使用したシンプルなデザイン

```
# このスクリプトは quartus_sh 実行コマンドで動作します
# プロジェクト名を filtref に設定
set project_name filtref

# プロジェクトをオープン存在しない場合は作成
if [catch {project_open $project_name}] {project_new \ $project_name}

# ファミリを設定
set_global_assignment -name family APEX 20KE

# デバイスを設定
set_global_assignment -name device ep20k100eqc208-1

# スピードに対して最適化
set_global_assignment -name optimization_technique speed

# Fastfit フィルタ・オプションをオンにしてコンパイル時間を短縮
set_global_assignment -name fast_fit_compilation on

# NC-Sim Verilog シミュレーション・ネットリストを生成
set_global_assignment -name eda_simulation_tool "NcSim\
(Verilog HDL output from Quartus II)"

# clk1 という名前の FMAX=50MHz アサインメントをピン clk に作成
create_base_clock -fmax 50MHz -target clk clk1

# ピン clk のピン・アサインメントを作成
set_location -to clk Pin_134

# コンパイル・オプション 1
# システム・コールの前に、必ず制約ファイルに
# アサインメントを書き込む。そうしないと、スタンドアロン・ファイルが
# アサインメントを取得しない
#export_assignments
#qexec quartus_map <プロジェクト名>
#qexec quartus_fit <プロジェクト名>
#qexec quartus_asm <プロジェクト名>
#qexec quartus_tan <プロジェクト名>
#qexec quartus_eda <プロジェクト名>

# コンパイル・オプション 2 (推奨)
# ::quartus::flow パッケージおよび execute_flow コマンドを使用すると、
# 自動的にアサインメントがエクスポートされる
```

```
# これはコンパイル・オプション 1 で記述した手順と同じ
```

```
load_package flow  
execute_flow -compile
```

```
# プロジェクトをクローズ
```

```
project_close
```

カスタム・オプションを利用してその他の EDA ツールをターゲットとすることが可能カスタム EDA コンフィギュレーションを行う場合、付加的なアサインメントを作成することにより個々の EDA インタフェース・オプションを変更することができます。



利用可能な各 EDA 設定行のすべてのリストについては、Quartus II ヘルプを参照してください。

LogicLock 領域の使用

Tcl コマンドを使用して、LogicLock™ 領域を操作することができます。次の例では、LogicLock 領域をエクスポートおよびインポートしてその他のデザインで使用方法を示します。この例では、LogicLock チュートリアル・デザインのファイルを使用しています。



LogicLock 設計手法について詳しくは、「Quartus II ハンドブック Volume 2」の「LogicLock Design Methodology」の章を参照してください。

デザインをコンパイルし LogicLock 領域をエクスポートするには、次の手順に従います。

1. プロジェクトを作成しアサインメントを追加します。
2. 仮想ピンを割り当てます。
3. LogicLock 領域を作成します。
4. デザイン・エントリを領域に割り当てます。
5. プロジェクトをコンパイルします。
6. 領域をバック・アノテートします。
7. 領域をエクスポートします。

例 3-5 に、**lockmult** という名前のプロジェクトを作成し、必要なすべてのアサインメントを実行してプロジェクトをコンパイルするスクリプトを示します。このスクリプトはプロジェクトをコンパイルし、デザインをバック・アノテートし、LogicLock 領域をエクスポートします。このスクリプトでは、**assign_virtual_pins** という名前のプロシージャを使用しています。このプロシージャについては、例の後で説明します。**quartus_cdb** 実行コマンドを使用してこのスクリプトを実行します。

例 3-5. LogicLock エクスポート・スクリプト

```
load_package flow
load_package logiclock
load_package backannotate

project_new lockmult -overwrite
set_global_assignment -name BDF_FILE pipemult.bdf
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
set_global_assignment -name TOP_LEVEL_ENTITY pipemult
```

これらの 2 つのアサインメントによって、Quartus II ソフトウェアは、
LogicLock 領域内のロジックに対する VQM ファイルを
生成します。VQM ファイルはトップレベル・デザインに
インポートされます。

```
set_global_assignment -name \
    LOGICLOCK_INCREMENTAL_COMPILE_FILE pipemult.vqm
set_global_assignment -name \
    LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT ON
```

```
create_base_clock -fmax 200MHz -target clk clk_200
assign_virtual_pins { clk }
# LogicLock 関連のコマンドの前に、
# LogicLock-related データ構造を作成
initialize_logiclock
```

```
# lockmult という名前の領域を作成し、
# それに pipemult を割り当てる
# 領域は自動サイズでフローティング
set_logiclock -region lockmult -auto_size true \
    -floating true
set_logiclock_contents -region lockmult -to pipemult
execute_flow -compile
```

```
# LogicLock 領域をバック・アノテートし、QSF をエクスポート
logiclock_back_annotate -region lockmult -lock
logiclock_export -file_name pipemult.qsf
```

```
uninitialize_logiclock
project_close
```

assign_virtual_pins コマンドは、プロシージャへの引数として指定された信号を除き、すべての下位レベルのデザイン・ピンに仮想ピンのアサインメントを行う手法です。このプロシージャは例 3-6 で定義されています。

例 3-6. assign_virtual_pins Procedure

```
proc assign_virtual_pins { skips } {  
  
    # 解析およびエラポレーションをまず実行してピン名を取得することが必要  
    execute_flow -analysis_and_elaboration  
  
    # すべてのピン名をまとめて取得  
  
    set name_ids [get_names -filter * -node_type pin]  
    foreach_in_collection name_id $name_ids {  
        # ピンの階層パス名を取得  
        set hname [get_name_info -info full_path $name_id]  
        # スキップされる信号のリストにピンがある場合は  
        # 仮想ピン・アサインメントをスキップ  
        if {[lsearch -exact $skips $hname] == -1} {  
            post_message "Setting VIRTUAL_PIN on $hname"  
            set_instance_assignment -to $hname -name VIRTUAL_PIN ON  
        } else {  
            post_message "Skipping VIRTUAL_PIN for $hname"  
        }  
    }  
}
```

スクリプトを実行すると、**pipemult.vqm** という名前のネットリスト・ファイルと、バック・アノテートされたアサインメントを含む **pipemult.qsf** という名前の Quartus II 設定ファイルが生成されます。これで LogicLock 領域を他のデザインにインポートすることが可能になります。この例では、**topmult** ディレクトリ内のトップレベルのデザインを使用しています。

これをトップレベルの LogicLock チュートリアル・デザインに 4 回インポートするには、次の手順に従います。

1. トップレベル・プロジェクトを作成します。
2. アサインメントを追加します。
3. デザインを完成させます。
4. LogicLock 制約をインポートします。
5. プロジェクトをコンパイルします。

例 3-7 に、これまでの手順を表すスクリプトを示します。

例 3-7. LogicLock インポート・スクリプト

```
load_package flow
load_package logiclock

project_new topmult -overwrite
set_global_assignment -name BDF_FILE topmult.bdf
set_global_assignment -name VQM_FILE pipemult.vqm
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
create_base_clock -fmax 200MHz -target clk clk_200

# LogicLock 領域はトップレベル・デザインで
# 4 回使用される。これらのアサインメントでは、
# QSF 内のバック・アノテートされたアサインメントが
# 4 つのエンティティに適用される
# ように指定
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst1
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst2
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst3

execute_flow -analysis_and_elaboration
initialize_logiclock
logiclock_import
uninitialize_logiclock
execute_flow -compile
project_close
```



LogicLock 設計手法について詳しくは、「Quartus II ハンドブック Volume 2」の「LogicLock Design Methodology」の章を参照してください。

デザインの コンパイル

Quartus II 実行コマンドは Tcl スクリプトから実行することができます。組み込まれている **flow** パッケージを使用して、さまざまな Quartus II コンパイル・フローを実行するか、または各実行コマンドを直接実行します。

flow パッケージ

flow パッケージには、Quartus II 実行コマンドを標準的なコンパイル・シーケンスで個別にまたは同時に実行する 2 つのコマンドが含まれています。**execute_module** コマンドを利用すると、個々の Quartus II 実行


コマンドを実行することが可能になります。**execute_flow** コマンドを利用すると、一般的に使用される組み合わせでモジュールの一部またはすべてを実行することが可能になります。

システム・コールを使用してコンパイラ実行コマンドを実行する代わりに、**flow** パッケージを使用することをお勧めします。

Tcl 環境から Quartus II 実行コマンドを実行するもう 1 つの方法は、Tcl **exec** コマンドの Quartus II 実装である **qexec** Tcl コマンドを使用することです。例えば、Quartus II テクノロジ・マップをあるプロジェクトで実行するには次のように入力します。

```
qexec "quartus_map <プロジェクト名>" ←
```

qexec コマンドを使用してデザインをコンパイルすると、Tcl スクリプトで（または Tcl シェルから）作成されたアサインメントは、コンパイルの前にプロジェクト・データベースおよび設定ファイルにエクスポートされません。**export_assignments** コマンドを使用するか、または **flow** パッケージ内のコマンドでプロジェクトをコンパイルして、アサインメントがプロジェクト・データベースおよび設定ファイルにエクスポートされるようにします。

 システム・コールを行うには、**qexec** コマンドを使用する必要があります。

また、システム・コマンド・プロンプトにおけるものと同じ構文を使用して、実行コマンドを Tcl シェルで直接実行することもできます。例えば、Quartus II テクノロジ・マップをあるプロジェクトで実行するには、Tcl シェル・プロンプトで次のように入力します。

```
quartus_map <プロジェクト名> ←
```

レポート

コンパイルが終了すると、レポートから情報を抽出して結果を評価することが必要になる場合があります。例えば、どの程度の数のデバイス・リソースをデザインで使用するか、またはそれが性能要件を満たすかどうかを知ることが必要になる場合があります。Quartus II Tcl API ではレポート・データに簡単にアクセスすることができるため、テキスト・レポート・ファイルを解析するスクリプトを記述する必要はありません。

レポート・データに一度に 1 つのロウまたは一度に 1 つのセルだけアクセスするコマンドを使用します。セルを正確に把握している場合、またはアクセスするセルが既知の場合は、**get_report_panel_data** コマンドを使用し、ロウおよびカラムの名前（または x および y 座標）と該当する

レポート・パネルの名前を指定します。レポート・パネルではデータを検索することがあります。そのためには、**get_report_panel_row** コマンドで一度に1つのロウだけレポートを読み取るループを使用します。

レポート・パネルのカラム見出しはロウ0にあります。一度に1つのロウだけレポートを読み取るループを使用する場合、ロウ1から開始してカラム見出しを含むロウをスキップすることができます。**get_number_of_rows** コマンドは、カラム見出しのロウを含めて、レポート・パネル内のロウの数を返します。ロウの数にはカラム見出しのロウが含まれるため、次の例に示すように、ループ・インデックスがロウの数よりも小さい限り、ループは継続することが必要です。

レポート・パネルは階層的に配置され、階層の各レベルはパネル名の文字列“|”で示されます。例えば、**Fitter Settings** レポート・パネルの名前は、**Fitter Settings** レポート・パネルが **Fitter** フォルダ内にあるため **Fitter|Fitter Settings** となります。階層の最上位レベルにあるパネルには、“|”文字列は使用されません。例えば、**Flow Settings** レポート・パネルは **Flow Settings** という名前になります。

例 3-8 に、プロジェクト内のすべてのレポート・パネル名のリストを出力するコードを示します。

例 3-8. すべてのレポート・パネル名の出力

```
set panel_names [get_report_panel_names]
foreach panel_name $panel_names {
    post_message "$panel_name"
}
```

次の例では、デザイン内の各クロック・ドメインにおける障害が発生したパスの数が出力されます。ここでは、**Timing Analyzer Summary** レポート・パネルの各ロウにアクセスするループを使用しています。クロック・ドメインは、**Clock Setup: '<クロック名>'** の形式で、**Type** という名前のカラムにリストされます。その他の要約情報も、**Type** カラムにリストされます。**Type** カラムがパターン“**Clock Setup***”に一致する場合、このスクリプトは、**Failed Paths** という名前のカラム内にリストされた障害が発生したパスの数を出力します。

例 3-9. 障害が発生したパスの数をクロックごとに出力

```
load_package report
project_open my-project
load_report
set report_panel_name "Timing Analyzer|Timing Analyzer Summary"
set num_rows [get_number_of_rows -name $report_panel_name]

# Type および Failed Paths カラムに対するカラム・インデックスを取得
set type_column [get_report_panel_column_index -name \
    $report_panel_name "Type"]
set failed_paths_column [get_report_panel_column_index -name \
    $report_panel_name "Failed Paths"]

# レポート・パネルの各行を進む
for {set i 1} {$i < $num_rows} {incr i} {

    # データのロウを取得し、ロウ内の要約情報のタイプ、
    # および障害が発生したパスの数を取得
    set report_row [get_report_panel_row -name \
        $report_panel_name -row $i]
    set row_type [lindex $report_row $type_column]
    set failed_paths [lindex $report_row $failed_paths_column]
    if { [string match "Clock Setup*" $row_type] } {
        puts "$row_type has $failed_paths failing paths"
    }
}
unload_report
```

Excel 用の CSV ファイルの作成

タイミング解析結果を表示したり操作したりするために、Microsoft Excel ソフトウェアを使用することがあります。任意の Quartus II レポートのデータで Excel にインポートする CSV ファイルを作成することができます。この例では、レポートのタイミング解析パネルからのデータで CSV ファイルを簡単に作成する方法を示します。使用するプロジェクト、レポート・パネル、および出力ファイルの名前に引き渡すコマンドライン引数を使用するように、スクリプトを修正することもできます。

例 3-10. レポートからの CSV ファイルの作成

```
load_package report
project_open my-project

load_report

# CSV ファイルとして保存するレポート・パネルの名前
set panel_name "Timing Analyzer|Clock Setup:'clk'"
set csv_file "output.csv"

set fh [open $csv_file w]
set num_rows [get_number_of_rows -name $panel_name]

# 見出しを含むロウを含めて、レポート・ファイル内の
# すべてのロウを進み、カンマ区切りデータを書き出す
for { set i 0 } { $i < $num_rows } { incr i } {
    set row_data [get_report_panel_row -name $panel_name \
        -row $i]
    puts $fh [join $row_data ","]
}

close $fh
unload_report
```

短いオプション名

Quartus II ソフトウェアのバージョン 6.0 以降では、コマンドのオプション間で区別ができれば、コマンド・オプションの短いバージョンを使用することができます。例えば、**project_open** コマンドは、`-current_revision` と `-revision` の 2 つのオプションをサポートしています。`-revision` オプションに対しては、短いバージョンの `-r`、`-re`、`-rev`、`-revi`、`-revis`、および `-revisio` のいずれかを使用することができます。オプションは、その他のオプションに同じ文字で始まるものがないため、`-r` まで短縮してとして使用することができます。ただし、**report_timing** コマンドには、`-recovery` と `-removal` が含まれます。これらのオプションは、互いに一意に区別することができなくなるため、いずれも `-r` または `-re` と短縮して使用することはできません。`-rec` または `-rem` を使用することはできません。

**タイミング
解析**

Quartus II ソフトウェアには、クラシック・アナライザと TimeQuest アナライザの両方に対応する広範な Tcl API が含まれています。この項には、クラシック・アナライザに対するシンプルで高度なスクリプト例と TimeQuest Tcl API に関する初歩的なスクリプティング情報が記載されています。

クラシック・タイミング解析

次のスクリプト例では、**quartus_tan** 実行コマンドを使用して **fir_filter** チュートリアル・デザインでタイミング解析を実行します。

fir_filter デザインは、タイミング解析にベース・クロックと相対クロックの関係を必要とする 2 クロック・デザインです。このスクリプトはまず、2 クロックの関係の解析を行い、clk と clkx2 の間の t_{SU} スラックを調べます。タイミング解析の第 1 のパスでは、クロックの 1 つに対して負のスラックを検出します。スクリプトの第 2 の部分では、clk から clkx2 へマルチサイクル・アサインメントを追加し、デザインをマルチクロック、マルチサイクルのデザインとして再解析します。

このスクリプトではデザインを新しいタイミング・アサインメントで再コンパイルせず、このデザインの合成および配置においてタイミング・ドリブン・コンパイルは使用しません。新しいタイミング・アサインメントは、タイミング・アナライザが **create_timing_netlist** および **report_timing** Tcl コマンドでデザインを解析するためにのみ追加されません。



例 3-11 に示すスクリプト例を実行する前に、プロジェクトをコンパイルする必要があります。

例 3-11. クラシック・タイミング解析

```
# この Tcl ファイルは quartus_tan.exe とともに使用
# この Tcl ファイルは、グローバル・タイミング・アサインメントを作成し、
# マルチクロック・アサインメントを作成することによって、
# Quartus II チュートリアルの fir_filter デザイン・タイミング解析部分を実行し、
# マルチクロック、マルチサイクル・デザインに対してタイミング解析を実行し、
# project_name を fir_filter に設定し、
# revision_name を filtref に設定する
set project_name fir_filter
set revision_name filtref

# プロジェクトをオープン
# project_name はプロジェクト名
project_open -revision $revision_name $project_name;

# TAN チュートリアル・ステップを実行して、このセクションは
# マルチクロックかつマルチサイクルの設定でタイミング解析モジュールを再実行する
# ----- タイミング・アサインメントの作成 -----#

# グローバル FMAX 要件の指定 (tan チュートリアル)
set_global_assignment -name FMAX_REQUIREMENT 45.0MHz
set_global_assignment -name CUT_OFF_IO_PIN_FEEDBACK ON

# ベース基準クロック "clocka" を作成し、
# 以下を指定
```

```

#   BASED_ON_CLOCK_SETTINGS = clocka;
#   INCLUDE_EXTERNAL_PIN_DELAYS_IN_FMAX_CALCULATIONS = OFF;
#   FMAX_REQUIREMENT = 50MHZ;
#   DUTY_CYCLE = 50;
#   基準クロック clocka をピン "clk" に割り当てる
create_base_clock -fmax 50MHZ -duty_cycle 50 clocka -target clk

#   次の設定で基準クロック "clocka" に基づいて、
#   相対クロック "clockb" を作成
#   BASED_ON_CLOCK_SETTINGS = clocka;
#   MULTIPLY_BASE_CLOCK_PERIOD_BY = 1;
#   DIVIDE_BASE_CLOCK_PERIOD_BY = 2; クロック周期はベース clk の半分
#   DUTY_CYCLE = 50;
#   OFFSET_FROM_BASE_CLOCK = 500ps; オフセットは .5 ns (または 500 ps)
#   INVERT_BASE_CLOCK = OFF;
#   基準クロックをピン "clkx2" に割り当てる
create_relative_clock -base_clock clocka -duty_cycle 50\
-divide 2 -offset 500ps -target clkx2 clockb

#   新しいタイミング設定に基づいて新しいタイミング・ネットリストを作成
create_timing_netlist

#   clkx2 に対して解析を実行
#   パスのリストを 1 パスに制限
#   clkx2 に対してクロック設定解析を実行
report_timing -npaths 1 -clock_setup -file setup_multiclock.tao

#   マルチクロック・デザインのみを指定した場合、出力ファイルには、
#   clkx2 に対する負のスラックが記載される。負のスラックは、基準クロックから 500 ps のオフセット
#   で作成された

#   report_timing による解析用の新しいタイミング・ネットリストを作成できるように、
#   古いタイミング・ネットリストを削除
delete_timing_netlist

#   マルチクロック設定を追加すると、マルチサイクル・アサインメントが clkx2 に
#   追加されて設定時間が延長されることにより、負のスラックが修正される
set_multicycle_assignment 2 -from clk -to clkx2

#   付加的なタイミング・アサインメントの create_timing_netlist に
#   基づいて、新しいタイミング・ネットリストを作成

#   clkx2 に対してのみ、結果をファイルに出力
report_timing -npaths 1 -clock_setup -clock_filter clkx2 \
-file clkx2_setup_multicycle.tao
#   新しい出力ファイルには、clkx2 に対する正のスラックが記載される
project_close

```

高度なクラシック・タイミング解析

タイミング解析のレポートに利用可能なコマンドでは、必要な特定のデータにアクセスできないことがあります。**advanced_timing** パッケージには、デザインのタイミング・ネットリストを表すデータ構造にアクセスするコマンドが用意されています。それらのコマンドでは、タイミング遅延、ノード・ファン・インおよびファン・アウト、ならびにタイミング・データに関する下位レベルの詳細を取得することができます。タイミング・ネットリストを横断し情報を抽出するプロシージャを記述すると、必要なデータを正確に取得するための最大限の制御が得られません。

タイミング・ネットリストは、ノードとエッジで構成されたグラフで表されます。ノードは、レジスタ、組み合わせノード、ピン、およびクロックなど、デザイン内のエレメントを表します。エッジはノードを接続し、デザイン内のロジック間の接続を表します。エッジおよびノードには、それらをタイミング・ネットリスト内で識別する固有の正整数の ID が与えられています。タイミング・ネットリストに関する情報を取得するためのすべてのコマンドは、テキスト・ベースの名前の代わりに、ノードおよびエッジの ID を使用します。

advanced_timing パッケージの使用例として、出力バスのピンをドライブするすべてのレジスタからのレジスタ・ピン間遅延を表示する方法を例 3-12 に示します。出力バスの名前 (address など) を指定します。スクリプトは、バスのピンをドライブするすべてのレジスタの名前と、レジスタからピンまでの遅延を出力します。

例 3-12. レジスタ・ピン間遅延のレポート

```
load_package advanced_timing
package require cmdline

# このプロシージャは、引き渡されたバス名に一致する名前のピンに対する
# ID のリストを返す
proc find { bus_name } {

    set to_return [list]

    foreach_in_collection node_id [get_timing_nodes -type pin] {
        set node_name [get_timing_node_info -info name $node_id]
        if { [string match $bus_name* $node_name] } {
            lappend to_return $node_id
        }
    }
    return $to_return
}

# スクリプトに必要な引数は、プロジェクトおよびリビジョンの名前、
# ならびに解析するバスの名前
set options { \
    { "project.arg" "" "Project name" } \
```

```

    { "revision.arg" "" "Revision name" } \
    { "bus_name.arg" "" "Name of the bus to get timing data for" } \
}
array set opts [::cmdline::getoptions quartus(args) $options]

project_open $opts(project) -revision $opts(revision)

# これにアクセスする前に、タイミング・ネットリストの作成が必要
create_timing_netlist

# これにより、付加的なタイミング・データでデータ構造を作成
create_p2p_delays

# 指定したバスの各ピンを移動
foreach pin_id [find $opts(bus_name)] {
    set pin_name [get_timing_node_info -info name $pin_id]
    puts "$pin_name source registers and delays"
    # get_delays_from_keepers コマンドは、関連する遅延で
    # 指定したタイミング・ノードにファン・インするデザイン内の
    # すべての非組み合わせノードのリストを返す。
    foreach data [get_delays_from_keepers $pin_id] {
        set source_node [lindex $data 0]
        set max_delay [lindex $data 1]
        set source_node_name \
[get_timing_node_info -info name $source_node]
        puts " $source_node_name $max_delay"
    }
}
project_close

```

システム・コマンド・プロンプトで、例 3-13 に示すコマンドを入力して、このスクリプトを実行します。

例 3-13.

```

quartus_tan -t script.tcl -project fir_filter
            -revision filtref -bus_name yn_out ↵

```

TimeQuest タイミング解析

TimeQuest タイミング・アナライザには、**::quartus::sdc** パッケージ内の SDC コマンドに対するサポートが含まれています。



TimeQuest タイミング・アナライザでタイミング解析を実行する方法について詳しくは、「Quartus II Handbook」の「TimeQuest タイミング解析」の章を参照してください。

TimeQuest スクリプティング

Quartus IIソフトウェアの6.0より前のバージョンでは、`::quartus::project` Tcl パッケージには、タイミング・アサインメントを作成するための SDC に似た次のコマンドが含まれていました。

- `create_base_clock`
- `create_relative_clock`
- `get_clocks`
- `set_clock_latency`
- `set_clock_uncertainty`
- `set_input_delay`
- `set_multicycle_assignment`
- `set_output_delay`
- `set_timing_cut_assignment`

これらのコマンドは SDC 準拠ではありません。バージョン 6.0 以降では、これらのコマンドは `::quartus::timing_assignment` という名前の新しいパッケージにあります。既存の Tcl スクリプトとの下位互換性を確保するために、`::quartus::timing_assignment` パッケージは次の実行コマンドでデフォルトでロードされます。

- `quartus`
- `quartus_sh`
- `quartus_cdb`
- `quartus_sim`
- `quartus_stp`
- `quartus_tan`

`::quartus::timing_assignment` パッケージは、`quartus_sta` 実行コマンドにはデフォルトでロードされません。`::quartus::sdc` Tcl パッケージには、上記のコマンドの SDC 準拠バージョンが含まれています。このパッケージは、`quartus_sta` 実行コマンドでのみ利用でき、デフォルトでロードされません。

スクリプト 実行の自動化

Quartus II ソフトウェアのバージョン 4.0 以降では、コンパイル中のさまざまな時点で自動的に実行するようにスクリプトをコンフィギュレーションすることができます。カスタム・レポートを実行し、特定のアサインメントを作成し、その他の多数のタスクを実行するスクリプトを自動的に実行するには、この機能を使用します。

次の3つのグローバル・アサインメントは、スクリプトが自動的に実行される時期を制御します。

- `PRE_FLOW_SCRIPT_FILE` — フローの開始前
- `POST_MODULE_SCRIPT_FILE` — モジュールの終了後

■ POST_FLOW_SCRIPT_FILE — フローの終了後

POST_FLOW_SCRIPT_FILE および POST_MODULE_SCRIPT_FILE アサインメントはバージョン 4.0 以降でサポートされ、PRE_FLOW_SCRIPT_FILE アサインメントはバージョン 4.1 以降でサポートされています。

1 つモジュールは、フロー内の 1 ステップを実行する Quartus II 実行コマンドです。例えば、2 つのモジュールは解析および合成 (**quartus_map**) とタイミング解析 (**quartus_tan**) です。

フローとは、Quartus II ソフトウェアが定義済みのオプションを指定して実行する一連のモジュールです。例えば、デザインのコンパイルは、一般に次のステップ（指示されたモジュールによって実行）で構成されるフローです。

1. 解析および合成 (**quartus_map**)
2. フィッタ (**quartus_fit**)
3. アセンブラ (**quartus_asm**)
4. タイミング・アナライザ (**quartus_tan**)

その他のフローについては、**execute_flow** Tcl コマンドのヘルプで説明しています。さらに、Quartus II GUI の Processing メニューのコマンドの多くは、このデザイン・フローに対応しています。

アサインメントの作成

自動的にスクリプトを実行するようにアサインメントを作成するには、次の形式でアサインメントをプロジェクトの Quartus II 設定ファイルに追加します。

```
set_global_assignment -name <アサインメント名> \  
    <実行コマンド>:<スクリプト名>
```

アサインメント名は次のうちのいずれかです。

- PRE_FLOW_SCRIPT_FILE
- POST_MODULE_SCRIPT_FILE
- POST_FLOW_SCRIPT_FILE

実行コマンドは、Tcl インタープリタを含む Quartus II 実行コマンドの名前です。

- quartus_cdb
- quartus_sh
- quartus_sim
- quartus_sta
- quartus_stp
- quartus_tan

このスクリプト名は使用している Tcl スクリプトの名前です。

スクリプトの実行

Quartus II ソフトウェアは、例 3-14 に示すようにスクリプトを実行します。

例 3-14.

```
<実行コマンド> -t <スクリプト名><フロー名またはモジュール名> <プロジェクト名>  
<リビジョン名>
```

argv 変数 (または quartus (args) 変数) に引き渡される第 1 引数は、使用するアサインメントに応じて、実行中のフローまたはモジュールの名前になります。第 2 引数はプロジェクトの名前、第 3 引数はリビジョンの名前です。

POST_MODULE_SCRIPT_FILE アサインメントを使用する場合、指定したスクリプトはフロー内の各実行コマンドに続いて自動的に実行されます。モジュール名 (スクリプトに引き渡された第 1 引数) との文字列比較を使用して、スクリプト処理を特定のモジュールに分離することができます。

実行例

完全なフローで自動スクリプト実行がどのように動作するかを例 3-15 に示します。ここでは、現在のリビジョンが **rev_1** という名前で **top** という名前のプロジェクトがあり、プロジェクトに対して次のアサインメントが Quartus II 設定ファイルにあると仮定しています。

例 3-15.

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE quartus_sh:first.tcl  
set_global_assignment -name POST_MODULE_SCRIPT_FILE quartus_sh:next.tcl  
set_global_assignment -name POST_FLOW_SCRIPT_FILE quartus_sh:last.tcl
```

プロジェクトをコンパイルするとき、PRE_FLOW_SCRIPT_FILE アサインメントによって、次のコマンドがコンパイルの開始前に実行されます。

```
quartus_sh -t first.tcl compile top rev_1
```

次に、Quartus II ソフトウェアはコンパイルを開始し、**quartus_map** 実行コマンドによって解析および合成が実行されます。解析および合成が終了すると、POST_MODULE_SCRIPT_FILE アサインメントによって、次のコマンドが実行されます。

```
quartus_sh -t next.tcl quartus_map top rev_1
```

次に、Quartus II ソフトウェアはコンパイルを継続し、**quartus_fit** 実行コマンドによってフィッタが実行されます。フィッタが終了すると、POST_MODULE_SCRIPT_FILE アサインメントによって次のコマンドが実行されます。

```
quartus_sh -t next.tcl quartus_fit top rev_1
```

対応するコマンドは、コンパイルのその他の段階の後に実行されます。最後に、コンパイルが完了すると、POST_FLOW_SCRIPT_FILE アサインメントによって次のコマンドが実行されます。

```
quartus_sh -t last.tcl compile top rev_1
```

処理の制御

POST_MODULE_SCRIPT_FILE アサインメントによって、スクリプトが各モジュールに続いて実行されます。各モジュールに続いて同じスクリプトが実行されるため、ユーザのスクリプトにおける処理を特定のモジュールに制限する条件文を追加することが必要になる場合があります。

例えば、スクリプトをタイミング解析後にのみ実行する場合は、[例 3-16](#)に示すような条件テストを含めることが必要です。これは、第 1 引数としてスクリプトに引き渡されたフロー名またはモジュール名をチェックし、モジュールが **quartus_tan** である場合にコードを実行します。

例 3-16. 単一モジュールへの処理の制限


```
set module [lindex $quartus(args) 0]

if [string match "quartus_tan" $module] {

    # タイミング解析後に実行されるコマンドを
    # ここに追加
    # post_message コマンドを使用してメッセージを
    # 表示
    post_message "Running after timing analysis"
}
```

メッセージの表示

Quartus II ソフトウェアがスクリプトを自動的に実行する方法による都合上、メッセージを表示するには、**puts** コマンドではなく **post_message** コマンドを使用する必要があります。この要件は、3-31 ページの「スクリプト実行の自動化」に示す3つのアサインメントによって実行されるスクリプトにのみ適用されます。

 このコマンドについて詳しくは、3-37 ページの「post_message コマンドの使用」を参照してください。

その他のスクリプティング機能


Quartus II Tcl API には、その他にも、この項で説明する汎用のコマンドおよび機能が含まれています。

ナチュラル・バス・ネーミング

バージョン 4.2 以降では、Quartus II ソフトウェアはナチュラル・バス・ネーミングをサポートしています。ナチュラル・バス・ネーミングとは、ハードウェア記述言語でバス・インデックスを指定するために使用される角括弧を、Tcl がそれらをコマンドとして解釈しないようにエスケープする必要がないことを意味します。例えば、**address** という名前のバス内のある信号は、**address\ [0\]** とせずに **address [0]** として識別することができます。例 3-17 のように、アサインメントを作成するときには、ナチュラル・バス・ネーミングを活用することができます。

例 3-17. ナチュラル・バス・ネーミング

```
set_location_assignment -to address[10] Pin_M20
```

Quartus II ソフトウェアはデフォルトでナチュラル・バス・ネーミングを使用します。ナチュラル・バス・ネーミングは、**disable_natural_bus_naming** コマンドでオフにすることができます。ナチュラル・バス・ネーミングについて詳しくは、Quartus II Tcl プロンプトで **enable_natural_bus_naming -h**  と入力してください。

コレクション・コマンドの使用

一部の Quartus II Tcl ファンクションは、Tcl リストとして非効率的な非常に大きなデータのセットを返します。これらのデータ構造はコレクションと呼ばれ、Quartus II Tcl API はコレクション ID を使用してこのコレクションにアクセスします。コレクションを操作するには、**foreach_in_collection** と **get_collection_size** の2つの Quartus II Tcl コマンドがあります。コレクション ID を変数に割り当てるには、**set** コマンドを使用します。



どの Quartus II Tcl コマンドがコレクション ID を返すかについて詳しくは、Quartus II ヘルプを参照し、**foreach_in_collection** コマンドについて検索してください。

foreach_in_collection コマンド

foreach_in_collection コマンドは、Tcl コマンドの **foreach** に似ています。コレクション内のすべてのエレメントに対して処理を繰り返すには、このコマンドを使用します。例 3-18 では、開いているプロジェクト内のすべてのインスタンス・アサインメントが出力されます。

例 3-18. コレクション・コマンドの使用

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
    # 各アサインメントについての情報はリストで
    # 返されます。リスト要素について詳しくは、
    # get-all-instance_assignments コマンドについての
    # ヘルプを参照してください。
    set to [lindex $asgn 2]
    set name [lindex $asgn 3]
    set value [lindex $asgn 4]
    puts "Assignment to $to:$name = $value"
}
```

get_collection_size コマンド

コレクション内のエレメントの数を取得するには、**get_collection_size** コマンドを使用します。例 3-19 では、開いているプロジェクト内のグローバル・アサインメントの数が出力されます。

例 3-19. get_collection_size コマンド

```
set all_global_assignments [get_all_global_assignments -name *]
set num_global_assignments [get_collection_size $all_global_assignments]
puts "There are $num_global_assignments global assignments in your project"
```

post_message コマンドの使用

Quartus II ソフトウェア・メッセージと同様にフォーマットされたメッセージを出力するには、**post_message** コマンドを使用します。**post_message** コマンドによって出力されたメッセージは、Quartus II GUI で Message ウィンドウの **System** タブに表示され、スクリプトが実行されたときに標準出力に書き込まれます。**post_message** コマンドの引数には、オプションのメッセージ・タイプと必須のメッセージ文字列があります。

メッセージ・タイプは次のうちのいずれかです。

- info (default)
- extra_info
- warning
- critical_warning
- error

タイプを指定しない場合、Quartus II ソフトウェアはデフォルトで **info** を使用します。

Windows で Quartus II ソフトウェアを使用している場合は、システム・コマンド・プロンプトに表示されるコード・メッセージに **post_message** コマンドでカラーを指定することができます。次の行を **quartus2.ini** ファイルに追加します。

```
DISPLAY_COMMAND_LINE_MESSAGES_IN_COLOR = on
```

例 3-20 に、**post_message** コマンドの使用法を示します。

例 3-20. post_message コマンド

```
post_message -type warning "Design has gated clocks"
```

コマンドライン引数へのアクセス

多くの Tcl スクリプトは、プロジェクトやリビジョンの名前など、コマンドライン引数を受け取るように設計されています。グローバル変数 **quartus(args)** は、コマンドラインで Tcl スクリプト名に続けて入力された引数からなるリストです。例 3-21 に、**quartus(args)** 変数内のすべての引数を出力するコードを示します。

例 3-21. コマンドライン引数への簡単なアクセス

```
set i 0
foreach arg $quartus(args) {
    puts "The value at index $i is $arg"
    incr i
}
```

以前の例のスクリプトを **print_args.tcl** という名前のファイルにコピーした場合、例 3-22 に示すコマンドをコマンド・プロンプトで入力したとき、次の出力が表示されます。

例 3-22. スクリプトへのコマンドライン引数の引渡し

```
quartus_sh -t print_args.tcl my_project 100MHz ↵
The value at index 0 is <my_project>
The value at index 1 is 100MHz
```

バージョン 4.1 以降で、Quartus II ソフトウェアは、コマンドライン引数へのアクセス用に、`argv`、`argc`、および `argv0` をサポートしています。表 3-7 に、以前のバージョンのこれに対応する情報を示します。

| バージョン 4.1 以降 | 以前のバージョンにおける同等のサポート |
|--------------------|--------------------------------------|
| <code>argc</code> | <code>llength \$quartus(args)</code> |
| <code>argv</code> | <code>quartus(args)</code> |
| <code>argv0</code> | <code>info nameofexecutable</code> |

cmdline パッケージの使用

より堅固で自己文書化されたコマンドライン引数の引渡しには、Quartus II ソフトウェアに含まれている **cmdline** パッケージを使用することができます。**cmdline** パッケージは、`-<option> <value>` の形式でコマンドライン引数をサポートしています。

例 3-23 では、**cmdline** パッケージを使用しています。

例 3-23. cmdline パッケージ

```

package require cmdline
variable ::argv0 $::quartus(args)
set options { \
  { "project.arg" "" "Project name" } \
  { "frequency.arg" "" "Frequency" } \
}
set usage "You need to specify options and values"

array set optshash [::cmdline::getoptions ::argv $options $usage]
puts "The project name is $optshash(project)"
puts "The frequency is $optshash(frequency)"

```

これらのコマンドを **print_cmd_args.tcl** という名前の Tcl スクリプトに保存した場合、例 3-24 に示すコマンドをコマンド・プロンプトで入力したとき、次の出力が表示されます。

例 3-24. スクリプトへのコマンドライン引数の引渡し

```

quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz ↵
The project name is <my_project>
The frequency is 100MHz

```

実質的にすべての Quartus II Tcl スクリプトは、プロジェクトを開くことを必要とします。例 3-25 では、プロジェクトを開き、リビジョン名をオプションで指定することができます。この例では、指定したプロジェクトが存在するかどうかをチェックします。存在する場合は、現在のリビジョンまたは指定したリビジョンを開きます。

例 3-25. プロジェクトを開くためのフル機能の方法

```

package require cmdline
variable ::argv0 $::quartus(args)
set options { \
  { "project.arg" "" "Project Name" } \
  { "revision.arg" "" "Revision Name" } \
}
array set optshash [::cmdline::getoptions ::argv0 $options]

# Ensure the project exists before trying to open it
if {[project_exists $optshash(project)]} {

  if {[string equal "" $optshash(revision)]} {

    # 指定されたリビジョン名がないため、デフォルトで
    # 現在のリビジョンを使用する
    project_open $optshash(project) -current_revision
  } else {

```

```

# 指定されたりビジョン名があるので、そのリビジョンの
# プロジェクトを開く
project_open $optshash(project) -revision \
    $optshash(revision)
}
} else {
    puts "Project $optshash(project) does not exist"
    exit 1
}
# スクリプトの残りをここに記載

```

このような柔軟性またはエラー・チェックが必要でない場合、例 3-26 に示すように、単純に **project_open** を使用することもできます。

例 3-26. プロジェクトを開くための簡単な方法

```

set proj_name [lindex $argv 0]
project_open $proj_name

```



cmdline パッケージについて詳しくは、<Quartus II インストール・ディレクトリ>/common/tcl/packagesにあるパッケージの文書を参照してください。

インタラクティブ・モードでの Quartus II Tcl シェルの使用

この項では、**quartus_sh** インタラクティブ・シェルを使用して、いくつかのプロジェクト・アサインメントを作成し、FIR (Finite Impulse Response) フィルタ・チュートリアル・プロジェクトをコンパイルする例を示します。この例では、プロジェクト・ディレクトリに FIR フィルタ・チュートリアル/デザイン・ファイルが用意されていることを想定しています。

始めるには、インタラクティブ Tcl シェルを起動します。コマンドおよび初期出力を例 3-27 に示します。

例 3-27. インタラクティブ Tcl シェル

```

tcl> quartus_sh -s
tcl> Info: *****
Info:Running Quartus II Shell
Info: Version 6.0 Internal Build 170 03/29/2006 SJ Full Version
Info: Copyright (C) 1991-2006 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions
Info:and other software and tools, and its AMPP partner logic
Info:functions, and any output files any of the foregoing
Info:(including device programming or simulation files), and any
Info:associated documentation or information are expressly subject
Info:to the terms and conditions of the Altera Program License
Info:Subscription Agreement, Altera MegaCore Function License
Info:Agreement, or other applicable license agreement, including,
Info:without limitation, that your use is for the sole purpose of
Info:programming logic devices manufactured by Altera and sold by

```

```

Info:Altera or its authorized distributors.Please refer to the
Info:applicable agreement for further details.
Info:Processing started:Tue Apr 04 12:24:13 2006
Info: *****
Info:The Quartus II Shell supports all TCL commands in addition
Info:to Quartus II Tcl commands.All unrecognized commands are
Info:assumed to be external and are run using Tcl's "exec"
Info:command.
Info:- Type "exit" to exit.
Info:- Type "help" to view a list of Quartus II Tcl packages.
Info:- Type "help <package name>" to view a list of Tcl commands
Info:available for the specified Quartus II Tcl package.
Info:- Type "help -tcl" to get an overview on Quartus II Tcl usages.
Info: *****
tcl>

```

Tcl プロンプトで次のコマンドを入力して、**filtref** という名前のリビジョンで **fir_filter** という名前の新しいプロジェクトを作成します。

```
project_new -revision filtref fir_filter ↵
```



プロジェクト・ファイルとプロジェクト名が同じである場合、Quartus II ソフトウェアはプロジェクトと同じ名前をリビジョンに付けます。

filtref という名前のリビジョンはトップレベル・ファイルと一致するため、すべてのデザイン・ファイルが階層ツリーから自動的に選択されます。

次に、次のコマンドでデバイスのグローバル・アサインメントを設定します。

```
set_global_assignment -name family Cyclone ↵
```



-name オプションで使用することができるアサイン名について詳しくは、Quartus II ヘルプを参照してください。



アサインメント名にスペースが含まれる場合、その値は引用符で囲む必要があります。

デザインを素早くコンパイルするには、`::quartus::flow` パッケージを使用します。このパッケージは、新しいプロジェクト・アサインメントを適切にエクスポートし、一連の実行コマンドを正しい順序で使用してデザインをコンパイルします。まず、パッケージをロードします。

```
load_package flow ↵
```

次のように返されます。

1.0

::quartus::flow パッケージについて詳しくは、次のように入力して Tcl プロンプトでコマンドライン・ヘルプを参照してください。

```
help -pkg ::quartus::flow ↵
```

例 3-28 に、代替コマンドとその結果として得られる出力を示します。

例 3-28. ヘルプの出力

```
tcl> help -pkg flow
```

```
-----
Tcl Package and Version:
-----
::quartus::flow 1.0
-----
Description:
-----
    This package contains the set of Tcl functions
    for running flows or command-line executables.
-----
Tcl Commands:
-----
    execute_flow
    execute_module
-----
```

このヘルプ表示では、フロー・パッケージとそのパッケージで利用可能なコマンドについての情報が示されます。Tcl コマンドの **execute_flow** に対して利用可能なオプションについては、Tcl プロンプトで次のコマンドを入力してください。

```
execute_flow -h ↵
```

追加情報および使用例を表示するには、Tcl プロンプトで次のコマンドを入力します。

```
execute_flow -long_help ↵
```

または

```
help -cmd execute_flow ↵
```

FIR フィルタ・デザインのフル・コンパイルを実行するには、例 3-29 に示すように、`-compile` オプションを指定して `execute_flow` コマンドを使用します。

例 3-29.

```
tcl> execute_flow -compile ↵
Info:*****
Info:Running Quartus II Analysis & Synthesis
Info:Version 6.0 SJ Full Version
Info:Processing started:Tues Apr 04 09:30:47 2006
Info:Command:quartus_map --import_settings_files=on --
export_settings_files=of fir_filter -c filtref
.
.
.
Info:Writing report file filtref.tan.rpt
tcl>
```

このスクリプトでは、FIR フィルタ・チュートリアル・プロジェクトをコンパイルし、プロジェクト・アサインメントをエクスポートし、**quartus_map**、**quartus_fit**、**quartus_asm**、および **quartus_tan** を実行します。このイベントのシーケンスは、Quartus II GUI で Processing メニューから **Start Compilation** を選択するのと同じです。

プロジェクトが終了すると、例 3-30 に示すように、**project_close** コマンドを使用してプロジェクトを閉じます。


例 3-30.

```
project_close ↵
```

次に、インタラクティブ Tcl シェルを終了するために、Tcl プロンプトで `exit` ↵と入力します。

Quartus II でのレガシー Tcl サポート

Quartus II ソフトウェアのバージョン 3.0 以降では、実行コマンドは、以前のバージョンの Quartus II ソフトウェアで使用されていた Tcl コマンドはサポートしていません。これらのコマンドは、Quartus II Tcl コンソールを使用して GUI で、またはシステム・コマンド・プロンプトで `quartus_cmd` を使用することによってサポートされます。これらのコマンドのいずれかを使用する以前のバージョンの Quartus II ソフトウェア用に作成された Tcl スクリプトのソースを基にする場合、プロジェクト・アサインメントはプロジェクト・データベースおよび設定ファイルに移植されます。その場合、実行コマンドを使用して結果的として得られるプロジェクトを処理することができます。これは、最新バージョンの Quartus II ソフトウェアに対応する Tcl スクリプトを生成しない EDA ツールを使用して Tcl ファイルを作成する場合には、必須となります。

 新しいプロジェクトおよびTclスクリプトはすべて、最新バージョンの Quartus II Tcl API で作成する必要があります。

Tcl スクリプティングの基礎知識

コア Tcl コマンドは、変数、制御構造、およびプロセスをサポートしています。さらに、ファイル・システムおよびネットワーク・ソケットにアクセスしたり、その他のプログラムを実行したりするためのコマンドもあります。Tk ウィジェット・セットを使用すると、プラットフォームに依存しないグラフィカル・インタフェースを作成することができます。

Tcl コマンドは、インタラクティブ Tcl シェルで入力すると直ちに実行されます。また、スクリプト（この章の例を含めて）をファイルとして作成し、Tcl インタープリタで実行することもできます。Tcl スクリプトは特殊なヘッダを必要としません。

インタラクティブ Tcl インタープリタを起動するには、コマンド・プロンプトで `quartus_sh -s` と入力します。入力したコマンドは、インタープリタ・プロンプトで直ちに実行されます。一連の Tcl コマンドをファイルに保存すると、そのファイルは Tcl インタープリタで実行することができます。`myscript.tcl` という名前のスクリプトを実行するには、コマンド・プロンプトで `quartus_sh -t myscript.tcl` と入力します。

Hello World の例

以下に、Tcl での基本的な「Hello world」の例を示します。

```
puts "Hello world"
```

単語の `hello` と `world` を 1 つの引数としてグループ化するには、二重引用符を使用します。二重引用符ではグループ内で置換を行うことができます。置換は、簡単な変数置換とすることも、[3-45 ページの「置換」](#)で説明するように、ネストしたコマンドの実行結果とすることもできます。置換しない場合には、中括弧 `{}` をグループ化に使用します。

変数

変数に値を代入するには、`set` コマンドを使用します。変数は、使用前に宣言する必要はありません。Tcl 変数の名前は、大文字と小文字で区別されます。[例 3-31](#) では、`a` という名前の変数に値 `1` を代入します。

例 3-31. 変数への代入

```
set a 1
```

変数の内容にアクセスするには、変数名の前にドル記号を使用します。[例 3-32](#) では、別の方法で `"Hello world"` が表示されます。

例 3-32. 変数へのアクセス

```
set a Hello
set b world
puts "$a $b"
```

置換

Tcl は次の 3 つのタイプの置換を実行します。

- 変数値の置換
- ネストしたコマンドの置換
- バックスラッシュの置換

変数値の置換

変数値の置換とは、[例 3-32](#) に示すように、変数名の前にドル記号 ("`$`")を使用することにより、変数に格納された値にアクセスすることです。

ネストしたコマンドの置換

ネストしたコマンドの置換とは、Tcl インタープリタが角括弧内の Tcl コードを評価する方法です。Tcl インタープリタは、ネストしたコマンドを最も内側のネストしたコマンド、および同じレベルでネストしたコマ

ンドを左から右に評価します。ネストした各コマンドの結果は、外側のコマンドで置換されます。例 3-33 では、a を文字列 foo の長さに設定します。

例 3-33. コマンドの置換

```
set a [string length foo]
```

バックスラッシュの置換

バックスラッシュの置換を使用すると、ドル記号 (“\$”) や括弧 (“[]”) など、Tcl で予約されている文字を引用することが可能になります。また、タブや復帰改行などのその他の特殊な ASCII 文字をバックスラッシュの置換で指定することもできます。バックスラッシュ文字は、Tcl コマンドが複数行に折り返されるときに使用される、Tcl のライン継続文字です。例 3-34 に、ライン継続にバックスラッシュ文字を使用する方法を示します。

例 3-34. バックスラッシュの置換

```
set this_is_a_long_variable_name [string length "Hello \  
world."]
```

演算

演算を実行するには、**expr** コマンドを使用します。中括弧 (“{ }”) を使用してこのコマンドの引数をグループ化すると、演算がより効率的となると同時に数値精度が維持されます。例 3-35 では、b を変数 a の値と 2 の平方根との合計に設定します。

例 3-35. expr コマンドを使用した演算

```
set a 5  
set b [expr { $a + sqrt(2) }]
```

Tcl はまた、&& (AND)、|| (OR)、! (NOT) などのブール演算子、ならびに < (より小さい)、> (より大きい)、== (等しい) などの比較演算子をサポートしています。

リスト

Tcl リストとは一連の値のことです。サポートされるリスト操作には、リストの作成、リストの追加、リスト・エレメントの抽出、リストの長さの計算、リストのソーティングなどがあります。例 3-36 では、a を 3 つの値を持つリストに設定します。

例 3-36. 簡単なリストの作成

```
set a { 1 2 3 }
```

lindex コマンドを使用すると、リスト内の特定のインデックスにある情報を抽出することができます。インデックスはゼロが基準となります。リスト内の最後のエレメントを指定するにはインデックス `end` を、リストの末尾からカウントするには `end-<n>` を使用することができます。例 3-37 では、a に格納されたリスト内の第 2 エレメント（インデックス 1）を出力します。

例 3-37. リスト・エレメントへのアクセス

```
puts [lindex $a 1]
```

llength コマンドはリストの長さを返します。例 3-38 では、a に格納されたリストの長さを出力します。

例 3-38. リストの長さ

```
puts [llength $a]
```

lappend コマンドはリストにエレメントを追加します。リストがまだ存在しない場合、指定したリストが作成されます。リスト変数名はドル記号では指定されません。例 3-39 では、a に格納されたリストにいくつかのエレメントを追加します。

例 3-39. リストへの追加

```
lappend a 4 5 6
```

配列

配列は、文字列ベースのインデックスを使用することを除いて、リストに類似しています。Tcl 配列はハッシュ・テーブルとして実装されています。配列は、各エレメントを個別に設定するか、または **array set** コマン

ドを使用することによって作成することができます。days という名前の配列内でインデックスが Mon のエレメントを Monday の値に設定するには、次のコマンドを使用します。

```
set days (Mon) Monday
```

array set コマンドにはインデックス / 値のペアからなるリストが必要です。この例では、days という名前の配列を設定します。

```
array set days { Sun Sunday Mon Monday Tue Tuesday \
                Wed Wednesday Thu Thursday Fri Friday Sat Saturday }
```

例 3-40 に、特定のインデックスに対する値にアクセスする方法を示します。

例 3-40. 配列エレメントへのアクセス

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

特定の配列内のすべてのインデックスのリストを取得するには、**array names** コマンドを使用します。インデックス値は、決められた順序で返されるわけではありません。例 3-41 に、配列内のすべての値に対して繰り返し処理をする方法を示します。

例 3-41. 配列に対する繰り返し

```
foreach day [array names days] {
    puts "The abbreviation $day corresponds to the day \
name $days($day) "
}
```

配列は、Tcl スクリプトで情報を極めて柔軟に格納する方法であると同時に、複雑なデータ構造を構築するのに適した方法です。

制御構造

Tcl は、**if-then-else** 条件や **for**、**foreach**、および **while** ループなど、一般的な制御構造をサポートしています。次の例に示すように中括弧を配置すると、制御構造コマンドが効率的かつ正しく実行されます。例 3-42 では、変数 a の値が正、負、またはゼロであるかを出力します。

例 3-42. if-then-else 構造

```
if { $a > 0 } {  
    puts "The value is positive"  
} elseif { $a < 0 } {  
    puts "The value is negative"  
} else {  
    puts "The value is zero"  
}
```

例 3-43 では、**for** ループを使用してリスト内の各エレメントを出力しています。

例 3-43. for ループ

```
set a { 1 2 3 }  
for { set i 0 } { $i < [llength $a] } { incr i } {  
    puts "The list element at index $i is [lindex $a $i]"  
}
```

例 3-44 では、**foreach** ループを使用してリスト内の各エレメントを出力しています。

例 3-44. foreach ループ

```
set a { 1 2 3 }  
foreach element $a {  
    puts "The list element is $element"  
}
```

例 3-45 では、**while** ループを使用してリスト内の各エレメントを出力しています。

例 3-45. while ループ

```
set a { 1 2 3 }  
set i 0  
while { $i < [llength $a] } {  
    puts "The list element at index $i is [lindex $a $i]"  
    incr i  
}
```

制御構造コマンド内の論理式では、**expr** コマンドが自動的に呼び出されるため、**expr** コマンドを使用する必要はありません。

プロシージャ

Tcl プロシージャ(その他のスクリプト言語およびプログラミング言語ではサブルーチンまたは関数として知られています)を定義するには、**proc** コマンドを使用します。プロシージャにおける変数の範囲は、プロシージャに対してローカルです。プロシージャが値を返す場合、**return** コマンドを使用してプロシージャから値を返します。例 3-46 では、2つの値を乗算してその結果を返すプロシージャを定義します。

例 3-46. 簡単なプロシージャ

```
proc multiply { x y } {  
    set product [expr { $x * $y }]  
    return $product  
}
```

例 3-47 に、コード内で **multiply** プロシージャを使用する方法を示します。プロシージャは、以下に示すように、スクリプトで呼び出す前に定義する必要があります。

例 3-47. プロシージャの使用

```
proc multiply { x y } {  
    set product [expr { $x * $y }]  
    return $product  
}  
set a 1  
set b 2  
puts [multiply $a $b]
```

プロシージャはスクリプトの先頭付近で定義する必要があります。プロシージャ内でグローバル変数にアクセスする場合、グローバル変数を使用する各プロシージャで **global** コマンドを使用します。例 3-48 では、値からなるグローバル・リスト内のエレメントが出力され、次にプロシージャが呼び出されます。

例 3-48. グローバル変数へのアクセス

```
proc print_global_list_element { i } {  
    global my_data  
    puts "The list element at index $i is [lindex $my_data $i]"  
}  
set my_data { 1 2 3}  
print_global_list_element 0
```

ファイル I/O

Tcl にはファイルとの間で読み書きするコマンドがあります。ファイルは、開いてから読み書きし、読み書き操作が終了すれば閉じる必要があります。ファイルを開くには **open** コマンドを使用し、ファイルを閉じるには **close** コマンドを使用します。ファイルを開くときには、ファイルの名前とファイルを開くモードを指定します。モードを指定しない場合、Tcl ではデフォルトで読み取りモードになります。ファイルに書き込むには、例 3-49 に示すように、書き込みモードに対して **w** を指定します。

例 3-49. 書き込み用にファイルを開く

```
set output [open myfile.txt w]
```

Tcl はその他にも、既存ファイルへの追加や同じファイルとの間の読み書きなどのモードをサポートしています。

open コマンドは、読み取りまたは書き込みアクセスに使用するファイル・ハンドルを返します。**puts** コマンドを使用すると、例 3-50 に示すように、ファイル・ハンドルを指定することによって、ファイルに書き込むことができます。

例 3-50. ファイルへの書き込み

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

gets コマンドでは一度に 1 行だけファイルを読み込むことができます。例 3-51 では、**gets** コマンドを使用して、ファイルの各行をその行番号とともに出力します。

例 3-51. ファイルからの読み込み

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # ここでテキストの行を処理
    puts "$line_num:$line"
    incr line_num
}
close $input
```

構文およびコメント

Tcl コマンドへの引数は空白で区切られ、Tcl コマンドは復帰改行文字またはセミコロンによって終了します。3-45 ページの「置換」に示すように、Tcl コマンドが複数行にわたるときには、バックスラッシュを使用する必要があります。

Tcl ではハッシュまたはポンド文字 (#) を使用してコメントを開始します。コマンドは # 文字で始まる必要があります。コマンドと同じ行にコメントを含める場合、コマンドは # 文字の前でセミコロンで終了することが必要です。例 3-52 は、**set** コマンドとコメントを含む有効なコード行です。

例 3-52. コメント

```
set a 1;# a を初期化
```

セミコロンがなければ、**set** コマンドはコメント後の復帰改行まで終了されないため、これは無効なコマンドとなります。

Tcl インタープリタはコメント内の中括弧をカウントし、これによって解明が困難なエラーが発生することがあります。例 3-53 では、中括弧が閉じられていないためエラーが発生します。

例 3-53. コメント内の閉じられていない括弧

```
# if { $x > 0 } {  
if { $y > 0 } {  
    # ここにコードを記述  
}
```

参考文献



Tcl の使用方法について詳しくは、次の資料を参照してください。

- *Practical Programming in Tcl and Tk*, Brent B. Welch
- *Tcl and the TK Toolkit*, John Ousterhout
- *Effective Tcl/TK Programming*, Michael McLennan and Mark Harrison
- www.altera.com/support/examples/tcl/tcl.html の Quartus II Tcl のスクリプト例
- tcl.activestate.com の Tcl Developer Xchange

改訂履歴

表 3-8 に、本資料の改訂履歴を示します。

| 表 3-8. 改訂履歴 | | |
|--------------------|---|----|
| 日付 & ドキュメント・バージョン | 変更内容 | 概要 |
| 2006 年 11 月 v6.1.0 | 改訂履歴を追加。 | |
| 2006 年 5 月 v6.0.0 | Quartus II ソフトウェア・バージョン 6.0.0 向けに更新。 <ul style="list-style-type: none"> ● 内容を再編。 ● TimeQuest タイミング・アナライザ機能を追加。 | |
| 2005 年 10 月 v5.1.0 | Quartus II ソフトウェア・バージョン 5.1.0 向けに更新。 | |
| 2005 年 8 月 v5.0.1 | 本文のマイナー・チェンジ。 | |
| 2005 年 5 月 v5.0.0 | Quartus II ソフトウェア・バージョン 5.0.0 向けに更新。 | |
| 2004 年 12 月 v2.1 | <ul style="list-style-type: none"> ● 表および図を更新。 ● Quaruts II ソフトウェア・バージョン 4.2 の新機能。 | |
| 2004 年 8 月 v2.1 | <ul style="list-style-type: none"> ● 誤記修正。 ● スクリプト例の追加 / 更新。 | |
| 2004 年 6 月 v2.0 | <ul style="list-style-type: none"> ● 表および図を更新。 ● Quaruts II ソフトウェア・バージョン 4.1 の新機能。 | |
| 2004 年 2 月 v1.0 | 初版 | |

