8. コンポーネント・インタフェース Tcl リファレンス

この資料は英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。こちらの日本語版は参考用としてご利用ください。設計の際には、最新の英語版で内容をご確認ください。

QII51023-11.0.0

Qsys コンポーネントの定義は、Component Editor でプロパティと動作を宣言するか、あるいは直接 **_hw.tcl**（ハードウェエア Tcl ファイル）で行われます。各 **_hw.tcl** ファイルはそれぞれ 1 つのコンポーネントを表しており、Qsys システムに追加できます。また、他の設計者とコンポーネントを共有することもできます。コンポーネントの柔軟性を最大限にするには、ほかのユーザーがデフォルトのパラメータを変更して各自のデザイン要件を満たすことができるよう、パラメータ化可能な動作を考慮する必要があります。

Qsys コンポーネントは通常、以下の 4 種類のファイルから構成されています。

- **_hw.tcl** ファイル — インタフェース動作など、Qsys 関連の特性を説明します。このファイルは必須です。
- HDL ファイル — ハードウェア・ファイル、シミュレーション・ファイル、および制約ファイルとしてコンポーネントの機能を定義します。これらのファイルはオプションです。
- **_sw.tcl** ファイル — ソフトウェア・ビルド・ツールがこのファイルを使用して、コンポーネントのドライバ・コードをコンパイルします。このファイルはオプションです。
- コンポーネント・ドライバ・ファイル — ソフトウェアがコンポーネントを制御できるように、コンポーネント・レジスタ・マップおよびドライバ・ソフトウェアを定義します。これらのファイルはオプションです。

この章では、以下の情報について説明します。

Tcl 構文については、Tcl Developer Xchange ウェブサイトを参照してください。

ハードウェア Tcl ファイルの情報

通常の **_hw.tcl** ファイルには、以下の情報が含まれています。

- コンポーネントの基本情報 — コンポーネントの名前、バージョン、説明、ドキュメントへのリンク、合成およびシミュレーション用 HDL 実装ファイルのポインタなどの情報が含まれます。
- パラメータの宣言 — パラメータとは、メモリのサイズなどコンポーネントの実装方法に影響する設定可能な値です。各パラメータ・プロパティには、パラメータの名前、表示可能かどうか、表示可能な場合は表示するテキストなどがあります。Qsysy システムの生成時に、パラメータは通常、Verilog HDL パラメータまたは VHDL ジェネリックとしてコンポーネントに適用されます。
- インタフェースの信号およびプロパティ — コンポーネントのインタフェースは、コンポーネントをシステムに接続する方法を定義し、システム内の他のコンポーネントがこのコンポーネントと交信する方法を決定します。コンポーネントのインタフェースを定義する場合、各インタフェースを構成する信号を宣言します。また、Avalon Memory-Mapped（Avalon-MM）インタフェースのウェイト・ステートなど、インタフェース・プロパティも定義します。

コンポーネントの定義

Qsys システム統合ツールを使用して、下記の 2 種類のコンポーネントを実装できます。

- HDL コンポーネント —HDL ファイルによって機能を定義し、**_hw.tcl** ファイルによって Qsys や他のツールに識別されるコンポーネントです。
- 合成コンポーネント — 合成コンポーネントとは、Qsys の階層デザイン機能を活用して、他の複数コンポーネントを組み合わせて構築されるコンポーネントです。合成コンポーネントは、他の使用可能なコンポーネントのインスタンスを接続させる **_hw.tcl** ファイルに含まれる Tcl コマンドによって定義されます。合成コンポーネントには独自の HDL ファイルがありません。合成コンポーネントの機能は、**_hw.tcl** ファイルおよびインスタンス化されたコンポーネントの HDL ファイルによって定義されます。

次に、2 種類の Qsys コンポーネントの開発について解説します。

HDL コンポーネントの実装

HDL による Qsys コンポーネントの開発は、4 つのフェーズに分かれています。

- Main Program—Qsys は最初にコンポーネントを検出し、それをコンポーネント・ライブラリに追加します。**_hw.tcl** が実行され、Tcl ステートメントは Qsys に対してインスタンス非固有情報を提供します。このフェーズでは、一部のコンポーネントの定義が不完全な場合、またはポート幅が 0 または -1 となる（ポート幅が変数であることを示す）場合があります。
- Editor— このフェーズでは、コンポーネントのインスタンスが Qsys システムに追加された後に、パラメータ・エディタでパラメータを編集できます。

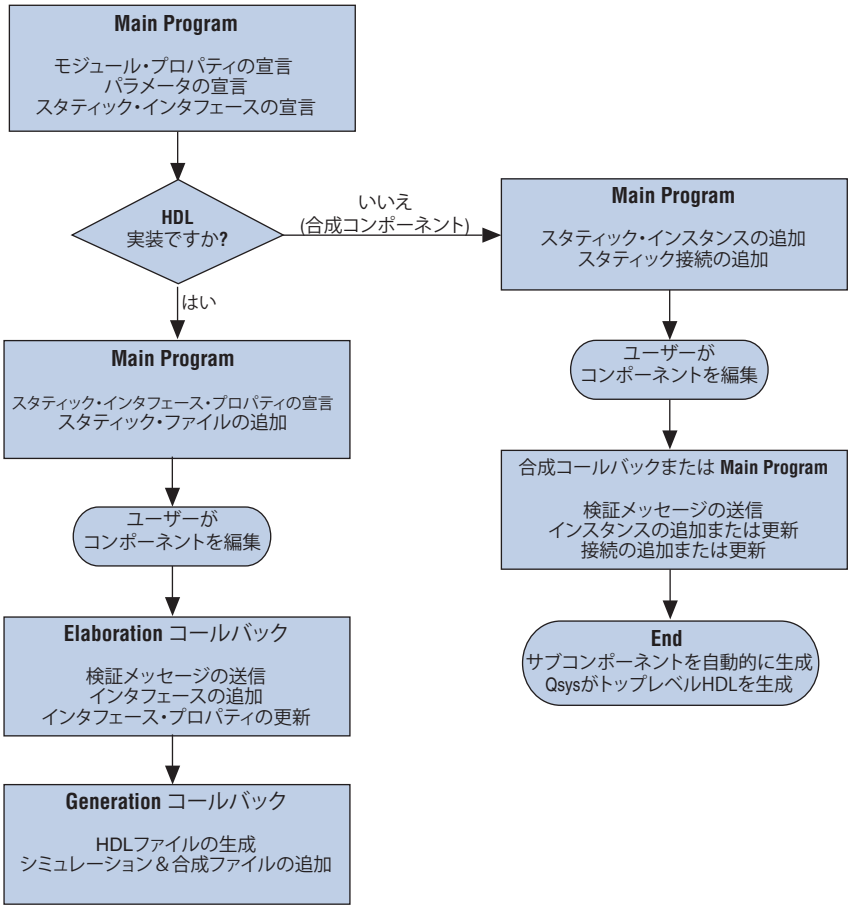
- Elaboration および Validitation— Elaboration は、Qsys がコンポーネントにインタフェース情報を照会するときに実行されます。Main Program フェーズで定義されたインタフェースは、Elaboration フェーズでイネーブルまたはディセーブルできます。Elabration フェーズは、コンポーネントのインスタンス作成時、パラメータ変更時、まははシステムの他のプロパティの変更時に発生します。Validation は、コンポーネントによるエラー、警告、または情報メッセージの生成を可能にします。Elaboration と Validation は常に Generation フェーズの前に行われます。Elaboration 後、コンポーネントを完全に定義する必要があります。例えば、すべてのポート幅は正の値でなければなりません。
- Generation—Generation フェーズでは、Quartus® II ソフトウェアまはた HDL シミュレータに必要なすべての情報が生成されます。通常、必須ファイルには VHDL または Verilog HDL ファイル、シミュレーション・モデル、およびタイミング制約が含まれます。

合成コンポーネントの実装

合成コンポーネントはほかのコンポーネントを組み合わせて実装されるため、HDL デザイン・フローに必要な Elaboration と Validation または Generation のフェーズが不要です。合成コンポーネントは、Main Program フェーズで Tcl コマンドを使用して定義するか、または個別の合成コールバックで定義することができます。

図 8–1 に、HDL コンポーネントおよび合成コンポーネントの作成手順を示します。

図 8-1. HDL コンポーネントおよび合成コンポーネントの実装



ハードウェア Tcl ファイルの記述

この項では、**_hw.tcl** ファイルに関する詳細な情報を提供し、5 つのフェーズのすべてのコンポーネントのデフォルト動作を説明します。以下の例では、簡単なパラメータ設定によるシンプルな UART を使用しています。

基本情報の提供

通常の **_hw.tcl** ファイルでは、最初に名前、位置および含まれるファイルなどの基本情報が宣言されます。**_hw.tcl** ファイルにある最初のコマンドは、次のコマンドに示すように、使用される **_hw.tcl** API のバージョンを指定する必要があります。

```
package require -exact sopc <version>
```

<version> は Quartus II のリリース・バージョンです（例えば、11.0）。Qsys では、特定の sopc package を要求する有効な **_hw.tcl** ファイルが次期バージョンのツールでも同じように動作することが保証されます。Quartus II ソフトウェアの各バージョンが多少異なるため、1 つの sopc パッケージで正常に機能する HDL ファイルが別のバージョンのパッケージでも正常に機能するとは限りません。

☞ この章で説明したコンポーネント動作は、sopc `11.0` パッケージを要求しています。

Tcl 構文については、Tcl Developer Xchange ウェブサイトを参照してください。

例 8-1. _hw.tcl ファイルの基本情報

```
# package コマンドは、ファイル内の最初のコマンドでなければならない
package require -exact sopc 11.0

# コンポーネントの名前とバージョン
set_module_property NAME example_uart
set_module_property VERSION 1.0

# コンポーネントのライブラリで表示される名前を指定
set_module_property DISPLAY_NAME "Example Component"

# コンポーネントの説明
set_module_property DESCRIPTION "An Example Component"

# コンポーネントの属するライブラリ・グループ
set_module_property GROUP Examples
```

パラメータの宣言

_hw.tcl ファイルにコンフィギュレーション・パラメータを含めることで、ユーザーがそれらのパラメータを調整できます。パラメータの表示と使用をコントロールするために、各パラメータには、名前、タイプ、表示名、およびデフォルト値などのプロパティを持っています。例 8–2 に、コンポーネントのユーザーが設定できるパラメータの使用を示します。

例 8-2. パラメータの宣言

```
# Baud Rate パラメータを整数として宣言し、デフォルト値が 9600 となる
add_parameter BAUD_RATE int 9600

# このパラメータがパラメータ・エディタで「Baud Rate」と表示するように設定
set_parameter_property BAUD_RATE DISPLAY_NAME "Baud Rate (bps)"

# 3 つのボーレートのみがサポートされる
set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}
```

パラメータは、ユーザー・パラメータ、システム情報パラメータ、および派生パラメータの 3 種類に分かれています。次に、これらのパラメータ・タイプについて説明します。

ユーザー・パラメータ

ユーザー・パラメータは、ユーザーが制御できるパラメータです。ユーザー・パラメータはコンポーネントのパラメータ・エディタで表示されます。

派生パラメータ

派生パラメータは、コンポーネント自身がユーザー・パラメータや他の派生パラメータから推測したパラメータです。例えば、クロック周期パラメータがデータ・レート・パラメータから推論できます。派生パラメータによって、HDL で実行できない操作を実行することができます。例えば、コンポーネントが要求するアドレス・ビットの数を対数関数で確認することは、Tcl では簡単にできますが、HDL では不可能です。

SYSTEM_INFO パラメータ

SYSTEM_INFO パラメータを使用して、特定のパラメータ値がシステムに関する情報を取り付けるように設定することができます。例えば、クロック入力に接続したクロックの周波数を知りたい場合を考えてみましょう。SYSTEM_INFO プロパティを宣言するとき、<info-type> および詳細な引数を提供する必要があります。<info-type> は、望ましい情報の種類です（例：clock_rate）。追加の引数は、必要なクロック入力インタフェースなどを指定するのに使用されます。例 8–3 に、SYSTEM_INFO パラメータの使用を示します。SYSTEM_INFO パラメータ・プロパティについては、28 ページの表 8–5 を参照してください。

例 8-3. SYSTEM_INFO パラメータを使用する Tcl コマンドの構文

```
set_parameter_property my_parameter SYSTEM_INFO {<info-type> [<arg>]}
```

インタフェースの宣言

インタフェースを宣言するには、add_interface コマンドを使用します。その後、set_interface_property および add_interface_port コマンドを使用して、インタフェースのプロパティを設定し、そのインタフェースに属する信号を指定します。インタフェースの宣言には、インタフェースの名前、インタフェースの方向、および関連するクロックおよびリセット・インタフェースが含まれています。クロックに関連していないインタフェース（例えば、クロック・インタフェース自体）の場合、関連するクロック・インタフェースを省略するか、または *asynchronous* ワードを使用します。例 8–4 に、インタフェースの宣言を示します。

例 8-4. インタフェースの宣言

```
# clock sink インタフェースの宣言。インタフェース名＝ "clock_sink"、種類 =clock、方向 =sink
# HDL および複合デザイン・フローの両方を使用するコンポーネントの場合、クロックおよびリセットが追加される
add_interface clock_sink clock sink

# reset sink インタフェースの宣言。インタフェース名＝ "reset_sink"、種類 =reset、方向 =sink、
# 関連するクロック =clock_sink
add_interface reset_set reset sink clock_sink

# Avalon slave インタフェースの宣言 。インタフェース名 =avalon_slave_0、種類 =avalon, 方向 =end
# HDL および合成コンポーネントの両方がトップ・レベル・インタフェースを宣言する
add_interface avalon_slave_0 avalon end
set_interface_property avalon_slave_0 export_of nios2.slave

# 以下のコマンドは、HDL デザイン・フローによるコンポーネントにのみ適用される。合成コンポーネント・デザイン・フローによるコンポーネントには適用されない。

# クロック・インタフェースは 1 つの信号を持っている。名前 ="clk" 、種類 ="clk"
add_interface_port clock_sink clk clk input 1
set_interface_property reset_sync associatedClock clock_sink

# リセット・インタフェースは 1 つの信号を持っている。名前 ="reset_n"、種類 ="reset_n"
add_interface_port reset_sink reset_n reset_n input 1

# Avalon Slave インタフェースに関するプロパティを設定する
set_interface_property avalon_slave_0 writeWaitTime 0
set_interface_property avalon_slave_0 addressAlignment DYNAMIC
set_interface_property avalon_slave_0 readWaitTime 1
set_interface_property avalon_slave_0 readLatency 0

# Avalon slave にクロック・インタフェースおよびリセット・インタフェースを関連付ける
set_interface_property avalon_slave_0 associatedClock clock_sink
set_interface_property avalon_slave_0 associatedReset reset_sink

# my Avalon Slave インタフェースに属するすべての信号を宣言する
add_interface_port avalon_slave_0 my_readdata readdata output 8
add_interface_port avalon_slave_0 my_read read input 1
add_interface_port avalon_slave_0 my_write write input 1
add_interface_port avalon_slave_0 my_waitrequest waitrequest output 1
add_interface_port avalon_slave_0 my_address address input 24
add_interface_port avalon_slave_0 my_writedata writedata input 8
```

ファイルの追加と生成のガイド

コンポーネント記述ファイルは通常、生成およびダ下流ツールに必要なすべての情報を提供し、コンポーネントに使用されるファイル（HDL ファイルなど）を識別します。また、どの追加ファイルがトップレベルの HDL ファイルかを識別し、ファイル内のどの Verilog モジュールまたは VHDL エンティティがコンポーネントのトップレベルのモジュールかを識別します。例 8–5 に、一般に生成および下流ツールに必要とされるファイルを示します。合成コンポーネントは既にコンポーネント記述ファイルを提供しているインスタンスをインスタンス化したため、シミュレーション・ファイルおよび合成ファイルが不要です。

例 8-5. ファイルの追加

```
# 合成およびシミュレーション用の HDL ファイルをコンポーネントに追加する
add_file simple_uart.v {SYNTHESIS SIMULATION}

# Quartus タイミング制約を含む Timequest ファイルを追加する
add_file simple_uart.sdc SYNTHESIS

# コンポーネントを記述するトップレベル・モジュール / エンティティを有する追加の HDL ファイルを指定
# トップレベル・モジュール / エンティティの名前を指定
set_module_property TOP_LEVEL_HDL_FILE simple_uart.v
set_module_property TOP_LEVEL_HDL_MODULE simple_uart
```

基本動作

前項で説明した **_hw.tcl** ファイルは、Elaboration フェーズと Generation フェーズでは基本動作をしています。これらの基本動作は、コンポーネントのインスタンスに適用されます。この項では、各フェースの基本動作について説明します。これらのデフォルト動作を変更するには、8-11 ページの「HDL で実装するコンポーネントにおけるデフォルト動作の変更」を参照してください。

Elaboration フェーズと Validation フェーズでの動作

デフォルトでは、Qsys の Generation フェーズと Validitation フェーズは各パラメータ値を ALLOWED_RANGES プロパティと比較してチェックを行います。指定した値が許容範囲外の場合、エラー・メッセージが表示されます。

各パラメータの ALLOWED_RANGES プロパティは、パラメータが取り得る値の範囲のリストであり、単一値またはコロンで区切られている開始値と終了値で表示されます。表 8-1 に、ALLOWED_RANGES プロパティが取り得る値の例を示します。

表 8-1. ALLOWED_RANGES プロパティ

ALLOWED_RANGES	意味
{a b c}	a または b または c
{1 2 4 8 16}	1、2、4、8、または 16
1:3	1 ～ 3（1 と 3 を含む）
{1 2 3 7:10}	1、2、3 または 7 ～ 10（7 と 10 を含む）

Main Program が明示的にすべてのポートを定数値または式に定義していなければ、デフォルトでは、Qsys の Elaboration プロセスは quartus_map 呼び出して、正しいポート幅を決定します。Main Program ですべてのポート幅を定義している場合、quartus_map は呼び出されません。

自動ポート幅

ポート幅が指定されていない場合、またはポート幅が「-1」の場合、quartus_map がポート幅をパラメータ・セットの関数として設定します。このプロセスにより、コンポーネントのオーサリングが容易になりますが、コンポーネントの生成が遅くなります。自動ポート幅を使用するとき、パラメータの affects_elaboration プロパティを false に設定すると、そのパラメータがポート幅またはインタフェースに影響を与えないように指定できます。これで、そのパラメータの値が変更されていても、quartus_map は呼び出されません。ただし、そのパラメータが実際に Elaboration に影響を与える場合、そのパラメータが Elaboration に影響を与えないように指定すると、デバッグの難しい問題を引き起こす可能性があります。

自動ポート幅の代替手段として、width_expr プロパティを使用してポート幅をシンプルな HDL 式に設定することができます。width_expr は、ポート幅を記述する式を持つ文字列です。width_expr プロパティにより、ポート幅を、HDL ファイルの検証や Elaboration コールバックでポート幅を設定する必要のない式として定義することができます。幅の式の構文は、使用される HDL 言語と同じです。ただし、加算、減算、乗算、および除算のみが使用できます。更に複雑なポート幅を使用する場合、ポートの幅はコンポーネントのパラメータの任意関数として設定できます。幅の式は、add_interface_port コマンドの最後の引数となります。例 8–6 に、算数演算子と width_expr プロパティの使用を示します。

例 8-6. 簡単な算数演算子によるポート幅の定義

```
add_interface_port din din_data data input {WIDTH * SYMBOLS}
set_port_property din_data width_expr WIDTH
```

パラメータ化されたパラメータ幅

VHDL を使用する場合、Qsys では std_logic_vector パラメータの幅が他のパラメータによって定義できます。std_logic_vector タイプのパラメータを追加するとき、そのパラメータの幅をパラメータ・プロパティとして指定することができます。幅は、定数または他のパラメータの名前となります。例 8–7 のコマンドでは、myParameter という std_logic_vector パラメータが追加されています。myParameter の幅は他のパラメータ、即ち dataWidth によって設定されます。

例 8-7. パラメータの追加

```
add_parameter myParameter STD_LOGIC_VECTOR
set_parameter_property myParameter WIDTH dataWidth
```

Generation フェーズの動作

Generation フェーズの時、Qsys は Verilog HDL または VHDL ラッパー・モジュールを作成して、トップレベル・モジュールをインスタンス化し、システム設計者が選択したパラメータを適用します。基礎となる HDL ファイルで宣言されていないパラメータは、ラッパーに適用しません。

Edit フェーズの動作

Qsys のデフォルトの Edit 動作では、すべてのパラメータ定義を使用して、パラメータ・エディタを作成します。25 ページの表 8–4 に示す各パラメータ・プロパティは、Qsys がデフォルト GUI を構築するときのガイダンスになります。

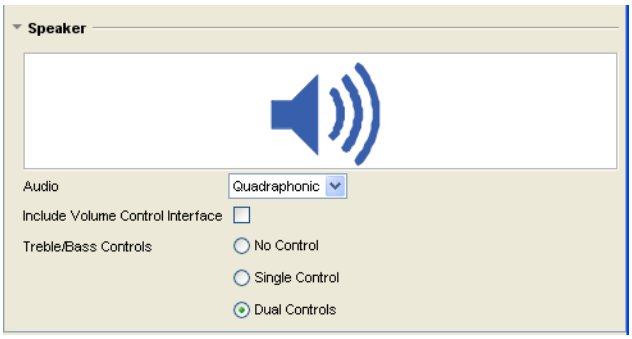
パラメータを論理グループに配置したり、画像とテキストを提供して、コンポーネントにカスタムのパラメータ・エディタを作成することができます。例 8–8 に、4 つのパラメータを定義し、add_display_item コマンド、DISPLAY_HINT パラメータと ALLOWED_RANGES パラメータの使用を示します。

例 8-8. パラメータ・エディタの定義とカスタマイズ

```
# アイコンを挿入し、パラメータを定義
add_display_item icon Speaker speaker-image speaker.png
add_parameter sound string 0 0
add_parameter volume_control boolean 0 0
add_parameter separate_control string 0 0

# パラメータの表示名を設定
set_parameter_property sound DISPLAY_NAME Audio
set_parameter_property volume_control DISPLAY_NAME "Include Volume Control Interface"
set_parameter_property separate_control DISPLAY_NAME "Treble/Bass Controls"

# Speaker グループですべてのパラメータを表示させる
add_display_item Speaker sound parameter
add_display_item Speaker volume_control parameter
add_display_item Speaker separate_control parameter

# sound パラメータに 4 つの選択肢を設定
# スペースのある文字列にはダブル・クォーテーション・マーク (") が必要
set_parameter_property sound allowed_ranges {"0:No Audio" 1:Monophonic 2:Stereo
4:Quadraphonic}
set_parameter_property separate_control allowed_ranges {"No Control" "Single Control"
"Dual Controls"}

# パラメータを表示する方法を指定
set_parameter_property volume_control DISPLAY_HINT boolean
set_parameter_property separate_control DISPLAY_HINT radio
```

図 8–2 に、例 8–8 の Tcl コマンドで生成されたパラメータ・エディタを示します。

図 8-2. 音声コンポーネントのパラメータ・エディタ



HDL で実装するコンポーネントにおけるデフォルト動作の変更

コールバックを使用して、デフォルトの動作を変更することができます。この項では、コンポーネント開発の各フェーズでのコールバック・プロシージャについて説明します。

Elaboration コールバック

Elaboration コールバックを使用して、デフォルトの範囲チェックを超える Elaboration および Validation を実行することができます。Elaboration コールバックは、例 8–9 に示すように、ELABORATION_CALLBACK モジュール・プロパティを Elaboration コールバック・プロシージャの名前と同じように設定することによって定義されます。38400 のボーレートおよび奇数のパリティを設定すると、この Elaboration プロシージャはエラーを表示します。

また、Elaboration コールバックを使用して合成パラメータの値を設定することもできます。合成パラメータは、他の複数のパラメータから合成されたパラメータです。合成パラメータは変更不可で、Qsys システム・ファイル (**.qsys**) にも保存されません。合成パラメータを指定するには、そのパラメータの DERIVED プロパティを true に設定します。例 8–9 の BAUDRATE_PRESCALE は合成パラメータであり、BAUDRATE パラメータの 1/16 の値を持っています。

また、Elaboration コールバックを使用して、インタフェース・プロパティを変更するか、または新規のインタフェースをパラメータ値の関数として追加することもできます。コンポーネントの一部のパラメータ設定だけが特定のインタフェースを必要とする場合、Elaboration コールバックでそのインタフェースをイネーブル / ディセーブルすることができます。例 8–9 に、USE_STATUS_INTERFACE パラメータを使用して Avalon-MM スレーブ・インタフェースをコンポーネントのインスタンスに入れる方法を示します

例 8-9. Elaboration コールバック

```
# コールバックを宣言
set_module_property ELABORATION_CALLBACK my_elaboration_callback

# BAUDRATE_PRESCALE パラメータを追加し、このパラメータが合成パラメータであることを指定
add_parameter BAUDRATE_PRESCALE int 600
set_parameter_property BAUDRATE_PRESCALE DERIVED true

# PARITY パラメータを追加
add_parameter PARITY string ODD
set_parameter_property PARITY ALLOWED_RANGES {EVEN ODD}

# USE_STATUS_INTERFACE パラメータを追加
add_parameter USE_STATUS_INTERFACE boolean

# ステータス・スレーブ・インタフェースを宣言
add_interface status_slave avalon end
set_interface_property status_slave associatedClock clock_sink
set_interface_property status_slave associatedReset clock_sink
set_interface_property status_slave enabled false

# 信号を宣言
add_interface_port status_slave st_readdata readdata output 16
add_interface_port status_slave st_read read input 1
add_interface_port status_slave st_write write input 1
add_interface_port status_slave st_waitrequest waitrequest output 1
add_interface_port status_slave st_address address input 24
add_interface_port status_slave st_writedata writedata input 16

The elaboration callback
proc my_elaboration_callback {} {
# 関心のあるパラメータの値を取得
    set br [get_parameter_value BAUD_RATE]
    set p [get_parameter_value PARITY]
    set use_status [get_parameter_value USE_STATUS_INTERFACE]

     # 無効な組合せに対してエラー・メッセージを表示
     if {($br==38400) && ($p=="ODD")} {
          send_message warning "Odd parity at 38400 bps is not supported."
     }
     # 合成パラメータの値を設定
     set bp [expr $br / 16]
     set_parameter_value BAUDRATE_PRESCALE $bp


# オプションでステータス・インタフェースを追加
if { $use_status }  {
  set_interface_property status_slave ENABLED true
} else {
  set_interface_property status_slave ENABLED false
}
```

☞ AFFECTS_ELABORATION=false のあるパラメータが変更された場合、Elaboration コールバックは呼び出されません。

Generation コールバック

Generation コールバックが定義されている場合、Qsys はコンポーネントにパラメータ値を適用するための HDL ラッパー・ファイルを生成しません。代わりに、Generation 時に提供された Generation コールバックを呼び出して、コンポーネントからのプログラムによる HDL の生成を可能にします。例 8–10 に示すように、Generation コールバックは、GENERATION_CALLBACK モジュール・プロパティを Generation コールバック関数名に設定することによって定義されます。

Generation コールバックは通常、コンポーネントのパラメータの現在の値、および Generation プロセスをガイドする Generation プロパティを取得し、続いて Tcl で、または外部プログラムを呼び出して、HDL ファイルとサポート・ファイルを生成します。また、コールバックは add_files コマンドで Qsys に必須ファイルを報告します。Generation コールバックで追加されたファイルは、**_hw.tcl** ファイルの本体に追加されるファイルに加えられます。

Generation コールバックは、Verilog、SystemVerilog、および VHDL に対して、それぞれ <*output_name*>**.v**、<*output_name*>**.sv**、または <*output_name*>**.vhd** を生成します。出力ファイルは、特定の <*output_directory*> に書き込まれます。このファイルは、コンポーネントのパラメータ化されたインスタンスです。メモリ初期化用の **.hex** ファイルなど他のサポート・ファイルは、<*output_directory*> に書き込まれることがあります。これらのファイル名は、<*output_name*> で始まる必要があります。コンポーネントのすべてのパラメータ設定のサポート・ファイルが同じな場合、Generation コールバックではなく、Main Program でこれらのファイルを追加します。システムがコンポーネントの複数のインスタンスを有し、それらのインスタンスのパラメータ設定がそれぞれ異なる場合、エラーを防止するために、Main Program フェーズでサポート・ファイルを追加する必要があります。特定のスタティック・サポート・ファイルがコンポーネントの一部のパラメータ設定にのみ必要な場合、Main Program フェーズでこのファイルを追加し、そして Elaboration コールバックで SYNTHESIS および SIMULATION プロパティを使用して適切に ON と OFF の切り替えをする必要があります。

例 8-10. Generation コールバックの例

```
set_module_property GENERATION_CALLBACK my_generate

# My generation 手法

proc my_generate {} {
  send_message info "Starting Generation"

  # generation の設定を取得

  set language [get_generation_property HDL_LANGUAGE]
  set outdir [get_generation_property OUTPUT_DIRECTORY ]
  set outputname [get_generation_property OUTPUT_NAME ]

  # parameter の値を取得

  set p1 [get_parameter_value PARAMETER_ONE]
  set csr [get_parameter_value CSR_ENABLED]

# コールバックは、exec を使用して外部プログラムを呼び出し、$outdir$outputname.v をここに
# 書き込む必要がある

  # add_file は、hw.tcl ディレクトリに関連するファイルを作成する
  # そのため、合成ファイルとシミュレーション・ファイルの $outdir を指定する

  exec perl my_generate.pl lang=$language dir=$outdir name=$outputname p1=$p1 csr=$csr
  add_file ${outdir}${outputname}.v SYNTHESIS
  add_file ${outdir}${outputname}_sim.v SIMULATION
}
```

合成コンポーネントの実装

合成コールバックを使用して、他のコンポーネントを組み合わせて構築されるコンポーネントを定義することができます。合成コマンドは、Main Program フェーズまたは独立した合成コールバックで使用できます。

- Main Program — Main Program では、add_instance, set_instance_parameter_value、および add_connection などの合成コマンドを使用して、サブコンポーネントのインスタンスを作成とパラメータ化することができます。
- 合成コールバック —Main Program フェーズで基本的なコンポーネント・テンプレートを設定した後、合成コールバックを使用して、コンポーネントのパラメータ値の関数としてサブコンポーネントをインスタンス化とパラメータ化することができます。合成コールバックを定義するには、COMPOSE_CALLBACK モジュール・プロパティを合成コールバック関数の名前に設定します。

使用されると、合成コールバックが Elaboration および Generation に取って代わります。コンポーネント・インタフェースの情報は、エクスポートされたサブコンポーネント上のインタフェースを解析することによって収集されます。すべてのサブコンポーネントおよびそれらを統合するトップレベルを生成することによって、HDL が生成されます。4 ページの図 8-1 に、合成コンポーネントを定義する手順を示します。

インタフェースのエクスポートというのは、インタフェースを内部で接続する代わりに、コンポーネントの外部から見えるようにすることです。外部から見えるインタフェースの EXPORT_OF プロパティを設定して、そのコンポーネントがサブモジュールのインタフェースのエクスポート・ビューであることを指定します。EXPORT_OF プロパティのフォーマットについては、8-37 ページの「get_interface_properties」を参照してください。これは、Main Program または合成コールバックで設定できます。

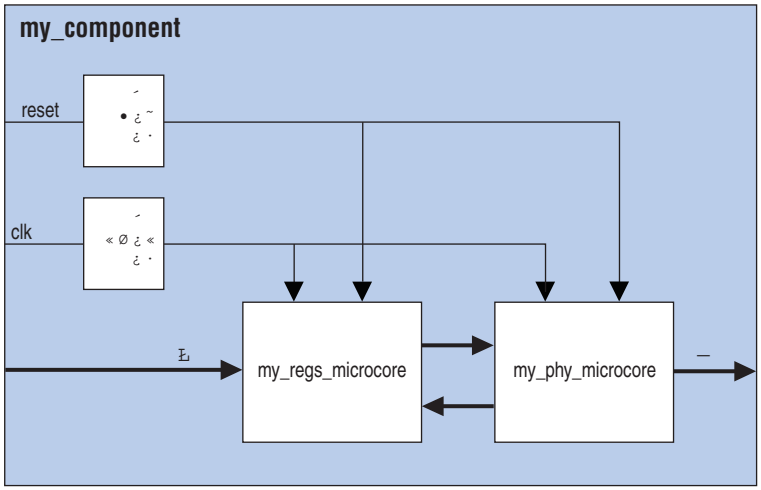
インタフェースのエクスポートと、2 つのインタフェース間の接続とは異なります。エクスポートされたインタフェースは、サブコンポーネントの内部インタフェースのコピーです。例えば、内部がバーストなしの 32 ビット Avalon-MM マスタである場合、エクスポートされたインタフェースも同様です。

☞ エクスポートされたインタフェースが内部インタフェースのコピーであるため、変更はできません。

エクスポートされたインタフェースを作成する際、エクスポート・インタフェースのプロパティは変更することなく、サブコンポーネントのインタフェースからコピーされます。サブコンポーネントのインタフェースからコピーされたポートは、独自性を保つために、合成コンポーネント上のエクスポートされたポートの名前だけが変更されています。

図 8-3 は、例 8-11 に示す合成コンポーネントのブロック図です。

図 8-3. 合成コンポーネントのトップレベル



例 8–11 に、2 つのサブコンポーネントをインスタンス化する合成 **_hw.tcl** ファイルの例を示します。このファイルは、2 つのサブコンポーネントを接続し、クロックおよびリセットを接続します。2 つのサブコンポーネントが共通のクロックとリセット入力に接続できるようにするためには、クロック・ブリッジとリセット・ブリッジが必要です。

例 8-11. 2 つのサブコンポーネントを持つ合成 _hw.tcl ファイル

```
package require -exact sopc 11.0
set_module_property name my_component
...
add_interface clk clock end
set_interface_property clk EXPORT_OF clk.in_clk

add_interface reset reset end
set_interface_property reset EXPORT_OF reset.in_reset

add_interface pins conduit end
set_interface_property pins EXPORT_OF phy.pins

add_interface slave avalon slave
set_interface_property slave EXPORT_OF regs.slave

add_instance clk altera_clock_bridge
add_instance reset altera_reset_bridge
set_instance_property_value reset synchronous_edges deassert
add_connection clk.out_clk reset.clk

add_instance phy my_phy_microcore
add_connection clk.out_clk phy.clk
add_connection reset.out_reset phy.clk_reset

add_instance regs my_regs_microcore
add_connection clk.out_clk regs.clk
add_connection reset.out_reset regs.reset

add_connection phy.output regs.input
add_connection regs.output phy.input
```

ハードウェア Tcl コマンド・リファレンス

この項では、すべてのハードウェア Tcl コマンドのリファレンスを以下のとおり示します。

各コマンドの説明では、利用可能なフェーズ、プログラムの本体（Main）、または Elaboration、Composition および Generation コールバック時、あるいはこれらの組み合わせを示します。表 8-2 に、それらのコマンドおよび詳細な説明へのリンクを示します。

表 8-2. コマンド一覧 *（注 1）* **(1 / 3)**

コマンド	詳細な説明
モジュール定義	
package <require> -exact sopc <*version*>	18 ページ
get_module_properties	19 ページ
get_module_property <*propertyName*>	20 ページ
set_module_property <*propertyName*> <*propertyValue*>	20 ページ
get_module_ports	21 ページ
get_module_assignments	21 ページ
get_module_assignment <*moduleName*>	22 ページ
set_module_assignment <*moduleName*> *[value]*	22 ページ
add_documentation_link <*title*> <*fileOrUrl*>	22 ページ
send_message <*messageLevel*> <*messageText*>	23 ページ
パラメータ	
add_parameter <*parameterName*> <*parameterType*> *[*<*defaultValue*> <*description*>*]*	24 ページ
get_parameters	24 ページ
get_parameter_properties	24 ページ
get_parameter_property <*parameterName*> <*propertyName*>	29 ページ
set_parameter_property <*parameterName*> <*propertyName*> <*value*>	30 ページ
get_parameter_value <*parameterName*>	30 ページ
set_parameter_value <*parameterName*> <*value*>	30 ページ
decode_address_map <*address_map_XML_string*>	31 ページ
アイテムの表示	
add_display_item <groupName> <id> <type> [<additionalInfo>]	32 ページ
get_display_items	34 ページ
get_display_item_properties	34 ページ
get_display_item_property <*itemName*> <*propertyName*>	34 ページ

表 8-2. コマンド一覧 *（注 1）*（2 / 3）

コマンド	詳細な説明
set_display_item_property *<itemName> <propertyName> <value>*	34 ページ
インタフェースおよびポート	
add_interface *<interfaceName> <interfaceType> <direction> [<associatedClock>]*	36 ページ
get_interfaces *<interfaceName>*	36 ページ
get_interface_property *<interfaceName> <propertyName>*	37 ページ
set_interface_property *<interfaceName> <propertyName> <value>*	38 ページ
add_interface_port *<interfaceName> <portName> <portRole> [<direction> <width_expr>]*	38 ページ
get_interface_ports *[<interfaceName>]*	39 ページ
get_port_properties	39 ページ
get_port_property *<portName> <propertyName>*	40 ページ
set_port_property *<portName> <propertyName> [<value>]*	41 ページ
get_interface_assignments	41 ページ
get_interface_assignment *<interfaceName> <name>*	41 ページ
set_interface_assignmet *<interfaceName> <name> [<value>]*	42 ページ
合成	
add_instance *<instanceName> <instanceType> <version>*	42 ページ
get_instances	43 ページ
get_instance_parameters <instanceName>	43 ページ
set_instance_parameter <instanceName> *<parameterName> <parameterValue>*	43 ページ
get_instance_parameter_value <instanceName> *<parameterName>*	43 ページ
get_instance_parameter_property *<instanceName> <parameterName> <propertyName>*	44 ページ
get_instance_interfaces *<instanceName>*	45 ページ
get_instance_interface_properties *<instanceName> <interfaceName>*	45 ページ
get_instance_interface_property *<instanceName> <interfaceName> <propertyName>*	45 ページ
get_instance_interface_ports *<instanceName> <portName>*	46 ページ
get_instance_port_property *<instanceName> <interfaceName> <propertyName>*	46 ページ
add_connection [<instanceName>] <startInterface> <endInterface>	46 ページ
get_connections	47 ページ
get_connection_parameters *<instanceName>*	48 ページ
get_connection_parameter *<connectionName> <parameterName>*	48 ページ
set_connection_parameter_value *<connectionName> <parameterName> <parameterValue>*	49 ページ
ファイル・セットおよび生成	
get_files	49 ページ
add_file filename *[<fileProperties> . . .]*	50 ページ
add_fileset <filesetName> <filesetKind> <callbackProcName> [<displayName>]	50 ページ
add_fileset_file <fileDestination> <fileKind> <fileSource> <contentsOrPath> [<attributes>]	51 ページ
get_file_properties	52 ページ

表 8-2. コマンド一覧 *（注 1）*（ 3 / 3 ）

コマンド	詳細な説明
get_file_property <*filename*> <*propertyName*>	53 ページ
set_file_property <*filename*> <*propertyName*> <*propertyValue*>	53 ページ
create_temp_file <fileName>	53 ページ
get_generation_properties	54 ページ
get_generation_property <*propertyName*>	55 ページ

表 8–2 の注：

(1) 角括弧（ [ ]）で囲まれた引数はオプションです。

モジュール定義

この項では、モジュールの定義およびクエリーに使用されるコマンドについて説明します。

package

package コマンドによって Qsys ソフトウェアのバージョンを指定し、ソフトウェア互換性の問題を回避することができます。package コマンドを **_hw.tcl** ファイルの最初に使用する必要があります。使用される場合、コンポーネント・ファイルが指定バージョンの Qsys ソフトウェアによって解釈されるように動作します。package コマンドが使用されていない場合、インストールされる Qsys ソフトウェアのバージョンが仮定されます。9.0 以前に設計されたコンポーネントの場合、要求パッケージを 9.0 に設定することができます。本資料では、コンポーネントの動作の記述が、package require -exact sopc 10.1 で開始します。それ以前のリリースについては、該当するリリースの資料を参照してください。

package は標準の Tcl コマンドです。このコマンドについて詳しくは、Tcl Developer Xchange ウェブサイトの Package ページを参照してください。

package		
コールバック可能なフェーズ	Main (ファイルの最初)	
使用方法	package require -exact sopc <*version*>	
戻り値	なし	
引数	version	要求される Qsys バージョンを 10 進数で指定します。
例	package require -exact sopc 10.0	

get_module_properties

このコマンドは、使用可能なすべてのモジュール・プロパティの名前を文字列のリストとして返します。get_module_property コマンドはプロパティ値を返すことができ、set_module_property コマンドはプロパティの値を設定することができます。特定バージョンの Qsys では、このコマンドによって返した値は常に同じです。

get_module_properties	
コールバック可能なフェーズ	Main、Elaboration、Generation、Composition
使用方法	get_module_properties
戻り値	List of strings
引数	なし
例	get_module_properties

表 8–3 に、使用可能なモジュール・プロパティとその使用法、および設定のタイミングを示します。

表 8-3. モジュール・プロパティ (1 / 2)

プロパティ名	プロパティ・タイプ	設定時	説明
ANALYZE_HDL	Boolean	Main program	false に設定されると、Quartus II Mapper がポート幅および方向を検証しなくなり、一部の検証チェックを犠牲にして生成時間を短縮します。このプロパティが false に設定された場合、無効なポート幅および方向は Quartus II コンパイルで検出されます。
AUTHOR	String	Main program	モジュールの作成者です。
DESCRIPTION	String	Main program	モジュールの説明です。（例：”Example Qsys Module”）
DISPLAY_NAME	String	Main program	モジュール参照時に表示される名前です。（例：“My SOPC Component”）
EDITABLE	Boolean	Main program	コンポーネントがコンポーネント・エディタで編集可能かどうかを示します。
ELABORATION_CALLBACK	String	Main program	Elaboration コールバックの名前。スタティック・コンポーネントまたは生成されたコンポーネントの場合、このプロパティが設定されていなければ、デフォルトの Elaboration が使用されます。
GENERATION_CALLBACK	String	Main program	Generation コールバックの名前です。
GROUP	String	Main program	モジュールが所属するコンポーネント・グループ（例：“Example Components”）です。
ICON_PATH	String	Main program	モジュール・パラメータのエディタに表示されるアイコンのパスです。
MODULE_TCL_FILE	String	リードのみ、設定不可	**_hw.tcl** ファイルへのパスです。
NAME	String	Main program	モジュール名です。（例：my_sopc_component）
TOP_LEVEL_HDL_FILE	String	Main program	add_file コマンドで追加されるファイルのうち、どのファイルにモジュールのトップレベル HDL が含まれるかを示します。

表 8-3. モジュール・プロパティ （2 / 2）

プロパティ名	プロパティ・タイプ	設定時	説明
TOP_LEVEL_HDL_MODULE	String	Main program	モジュールのトップレベル HDL ファイルで定義しなければならないトップレベルのモジュールの名前を示します。
VERSION	String	Main program	モジュールのバージョンです。（例：10.0）
COMPOSE_CALLBACK	String	Main Program	Compose コールバックの名前です。Compose コールバックを定義する場合、Generation コールバックまたは Elaboration コールバックを定義してはいけません。

TOP_LEVEL_HDL_MODULE および GENERATION_CALLBACK コマンドは、コンポーネントに使用される Generation のタイプの選択に使用されます。これらのコマンドは、ファイルの Main Program で同時に設定できません。

get_module_property

このコマンドは、モジュール・プロパティの値を返します。

get_module_property		
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition	
使用方法	get_module_property <propertyName>	
戻り値	String、boolean、または file	
引数	propertyName	19 ページの表 8–3 に示すプロパティのいずれかです。
例	set my_name [get_module_property NAME]	

set_module_property

このコマンドにより、モジュール・プロパティの値を設定することができます。

set_module_property		
コールバック可能なフェーズ	Main program	
使用方法	set_module_property <propertyName> <propertyValue>	
戻り値	なし	
引数	propertyName	19 ページの表 8–3 に示すプロパティのいずれかです。
	propertyValue	プロパティの新しい値です。
例	set_module_property VERSION 10.0	

get_module_ports

このコマンドは、現在定義されているすべてのポートの名前のリストを返します。

get_module_ports		
コールバック可能なフェーズ	Main、Elaboration、および Generation	
使用方法	`get_module_ports`	
戻り値	`String`	
引数	なし	
例	`get_module_ports`	

get_module_assignments

このコマンドは、モジュールのアサインメント変数を返します。

get_module_assignments	
コールバック可能なフェーズ	Main、Elaboration、および Composition
使用方法	`get_module_assignments`
戻り値	`String`
引数	なし
例	`get_module_assignments`

get_module_assignment

このコマンドは、特定の引数の値を返します。`get_module_assignment` と `set_module_assignment` 、および `get_interface_assignment` と `set_interface_assignment` コマンドを使用して、ハードウェア・コンポーネントに関する情報をエンベデッド・ソフトウェア・ツールおよびアプリケーションに転送することができます。

get_module_assignment		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	`get_module_assignment <name>`	
戻り値	`String`	
引数	`name`	値が読み出される名前です。
例	`get_module_assignment embeddedsw.CMacro.colorSpace`	

ソフトウェア・ツールでの情報の指定について詳しくは、「*Nios II Software Developer's Handbook*」の「*Publishing Component Information to Embedded Software*」を参照してください。

set_module_assignment

このコマンドは、特定の引数の値を設定します。

set_module_assignment		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	`set_module_assignment <name> [<value>]`	
戻り値	なし	
引数	name	値が設定される名前です。
	value	*<name>* 異数の値です。
例	`set_module_assignment embeddedsw.CMacro.colorSpace CMYK`	

add_documentation_link

このコマンドでは、コンポーネントに複数のリンクを追加することができます。

add_documentation_link		
コールバック可能なフェーズ	Main	
使用方法	`add_documentation_link filename <title> <fileOrUrl>`	
戻り値	なし	
引数	title	メニューおよびボタンに使用されるドキュメント・タイトルです。
	fileOrUrl	コンポーネント資料へのパス。相対パスではなく、URL 全体を表示する構文を使用してください。例えば、**http://www.mydomain.com/my_memory_controller.html** 、または **file:///datasheet.txt**
例	`add_documentation_link "Avalon Verification IP Suite User Guide" http://www.altera.com/literature/ug/ug_avalon_verification_ip.pdf`	

send_message

このコマンドは、メッセージをコンポーネントのユーザーに送信します。メッセージ・テキストは通常、HTML として解釈されます。*<b>* エレメントは、テキストを強調するのに使用できます。メッセージ・テキストを HTML として解釈したくない場合、{ `info text` } などのリストをメッセージ・レベルとして渡します。

send_message		
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition	
使用方法	`send_message <messageLevel> <messageText>`	
戻り値	なし	
引数	`messageLevel`	以下の 6 つのメッセージ・レベルがサポートされます。 ■ `Error`— エラー・メッセージを送ります。エラー・メッセージがある間は Qsys システムを生成できません。 ■ `ToDoError`— システム生成前に処置が必要な問題があることを知らせます。例えば、Nios II コンポーネントを生成する前には、Nios II リセット・ベクタを割り当てる必要があります。 ■ `Warning`— 警告メッセージを送ります。 ■ `Info`— 情報メッセージを送ります。 ■ `Progress`— 生成の進行状況をレポートします。 ■ `Debug`— デバッグ・モードが有効のときにメッセージを送ります。
	`messageText`	メッセージのテキストです。
例	`send_message Error "<b>param1</b> must be greater than param2."`	

パラメータ

パラメータにより、ユーザーが Verilog HDL パラメータまたは VHDL ジェネリックと同様に、コンポーネントの動作を調整することができます。

add_parameter

このコマンドは、コンポーネントにパラメータを追加します。ほとんどのパラメータ・タイプは C プログラミング言語または HDL で使用されるものなので、説明する必要はありません。ただし、GUI で表を作成するのに使用される `string_list` および `integer_list` パラメータには説明が必要です。

- `add_parameter` コマンドを `string_list` または `integer_list` パラメータ・タイプと一緒に使用する場合、定義されたパラメータは **add** と **remove** ボタンを持つ可変サイズの表で表示されます。
- `string_list` または `integer_list` タイプのパラメータを複数定義する場合、`add_display_item` コマンドを使用して、パラメータがそれぞれ表のカラムとして表示されるように指定することができます。`string_list` または `integer_list` タイプの各パラメータは表のカラムになります。例 8–12 に、`integer_list` パラメータ・タイプによる複数列の表の作成を示します。

例 8-12. string_list および integer_list パラメータ・タイプによる表の作成

```
add_parameter bitsWide INTEGER
add_parameter divider INTEGER
add_parameter coefficients INTEGER_LIST
add_parameter positions INTEGER_LIST
add_display_item myTable coefficients TABLE
add_display_item myTable positions TABLE
```

add_parameter		
コールバック可能なフェーズ	Main program	
使用方法	add_parameter <parameterName> <parameterType> [<defaultValue> <description>]	
戻り値	String	
引数	parameterName	コンポーネント作成者が決めるパラメータの名前です
	parameterType	Integer、Natural、Positive、Boolean、Std_logic、Std_logic_vector、String, String_list、および Integer_list がサポートされます。
	defaultValue	パラメータのデフォルト値です。
	description	パラメータの使用を説明します。
例	add_parameter seed integer 17 "The seed to use for data generation."	

get_parameters

このコマンドは、add_parameter によって定義されているすべてのパラメータの名前をスペース区切りのリストとして返します。

get_parameters	
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition
使用方法	get_parameters
戻り値	List of strings
引数	なし
例	set parameter_summary [get_parameters]

get_parameter_properties

このコマンドは、使用可能なすべてのパラメータ・プロパティを文字列リストとして返します。get_parameter_property および set_parameter_property コマンドはそれぞれ、プロパティ値の取得および設定に使用されます。

get_parameter_properties	
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition
使用方法	get_parameter_properties
戻り値	List of strings
引数	なし
例	set property_summary [get_parameter_properties]

使用可能なパラメータ・プロパティとその使用法、および設定のタイミングについては、表 8-4 で説明しています。

表 8-4. パラメータ・プロパティ （1 / 3）

プロパティ名	プロパティ・タイプ / デフォルト値	設定時	説明
AFFECTS_ELABORATION	Boolean、true	Main Program	モジュールの外部インタフェースに影響のないパラメータに対しては、AFFECTS_ELABORATION を false に設定します。isNonVolatileStorage は、外部インタフェースに影響のないパラメータの一例です 。また、width パラメータも外部インタフェースに影響を与えません。パラメータに変更が発生した時、パラメータが AFFECTS_ELABORATION=false を設定したら、Elaboration（コールバックおよびハードウェア解析）フェーズは繰り返されることなく、パフォーマンスが向上します。AFFECTS_ELABORATION のデフォルト値が true であるため、パラメータが変更される毎に、提供された HDL ファイルは通常新しいポート幅とコンフィギュレーションを決定するために再解析されます。
AFFECTS_GENERATION	Boolean、説明を参照	Main Program	トップレベル HDL モジュールを提供する場合、AFFECTS_GENERATION のデフォルト値は false になり、カスタム Generation コールバックを提供する場合、AFFECTS_GENERATION のデフォルト値は true になります。パラメータの値がシステム生成の結果に影響を及ばない場合、AFFECTS_GENERATION を false に設定します。
ALLOWED_RANGES	String、""	Main Program	パラメータ値の許容範囲を示します。整数の場合、ALLOWED_RANGES は使用可能なパラメータのリストとなり、その範囲は単一値またはコロンに区切られている開始値と終了値によって定義される範囲（例：「11:15」）になります。また、このプロパティは正当値を指定するか、整数に対して文字列を表示することができます。例えば、{0:None 1:Monophonic 2:Stereo 4:Quadrophonic} は、0、1、2、4 が正当値であることを意味します。また、文字列変数に対して、長い文字列がパラメータ・エディタで表示されるように割り当てることができます。（例：ALLOWED_RANGES {"dev1:Cyclone IV GX" "dev2:Stratix V GT"}）。このプロパティの使用例についての詳細は、10 ページの例 8–8 および 10 ページの図 8–2 を参照してください。
DEFAULT_VALUE	String または Boolean	Main Program	デフォルトの値です。
DERIVED	Boolean、false	Elaboration コールバック	true の場合、このパラメータを格納する必要がないことを示します。これは通常、パラメータが Elaboration コールバックから設定されるためです。デフォルト値は false です。
DESCRIPTION	String、""	Main Program	ユーザーから見える、パラメータの説明です。

表 8-4. パラメータ・プロパティ （2 / 3）

プロパティ名	プロパティ・タイプ / デフォルト値	設定時	説明
DISPLAY_HINT	String、""	Main Program	プロパティの表示方法に関するヒントを提供します。次の値が使用できます。 ■ boolean—値が0または1である整数のパラメータに使用されます。パラメータはオン / オフの切り替えを選択できるように表示されます。 ■ radio— ドロップダウン・リストの代わりに、複数の値を持つパラメータをラジオ・ボタンとして表示します。 ■ hexadecimal— 整数パラメータの場合、値を 16 進数として表示および解釈します。例えば 16 ではなく、0x00000010 が表示されます。 ■ fixed_size— string_list および integer_list パラメータの場合、fixed_size DISPLAY_HINT は表から **add** と **remove** ボタンを削除します。 このプロパティの使用例については、10 ページの例 8–8 および 10 ページの図 8–2 を参照してください。
DISPLAY_NAME	String、""	Main Program	パラメータの左側に表示される GUI ラベルです。
DISPLAY_UNITS	String、""	Main Program	パラメータの右側に表示される GUI ラベルです。
ENABLED	Boolean、true	Main Program および Elaboration コールバック	false の場合、パラメータは無効になります。パラメータは表示されますが、グレー表示となり、パラメータ・エディタでは編集できなくなります。
GROUP	String、""	Main	GUI におけるパラメータのレイアウトを制御します。使用例については、例 8–8 を参照してください。
HDL_PARAMETER	Boolean、false	Main Program	true の場合、このパラメータは HDL コンポーネント記述に渡す必要があります。
NEW_INSTANCE_VALUE	String、""	Main Program	このプロパティにより、このパラメータに対して defaultValue 引数でデフォルト値を割り当てた古いコンポーネントに影響することなく、パラメータのデフォルト値を変更することができます。実際の結果として、古いコンポーネントはこのパラメータに対して defaultValue による値を使用し、新しいコンポーネントは NEW_INSTANCE_VALUE による値を使用することができます。

表 8-4. パラメータ・プロパティ （3 / 3）

プロパティ名	プロパティ・タイプ / デフォルト値	設定時	説明
SYSTEM_INFO	String、""	Main Program	定義されているパラメータに、インスタンス化されているシステムに関する情報を割り当てることを可能にします。SYSTEM_INFO は、要求された情報の種類を指定するキーワード引数、*<info-type>*、を必要とします。*<info-type>* は 1 つの引数を取ることができます。この Tcl コマンドの構文は以下の通りです。 set_parameter_property my_parameter SYSTEM_INFO *<info-type> [<arg>]* 下記の *<info-type>* の値は定義済みです。 ADDRESS_MAP、ADDRESS_WIDTH、CLOCK_DOMAIN、CLOCK_RATE、CLOCK_RESET_INFO、CUSTOM_INSTRUCTION_SLAVES、DEVICE、DEVICE_FAMILY、DEVICE_FEATURES、INTERRUPTS_USED、MAX_SLAVE_DATA_WIDTH、RESET_DOMAIN、および TRISTATE_ONDUIT_MASTERS *<info_type>* 引数の説明については、表 8–5 を参照してください。
SYSTEM_INFO_TYPE	各種	Main Program	28 ページの表 8–5 に示す情報種類のいずれかを指定します。
SYSTEM_INFO_ARG	String、""	Main Program	特定の SYSTEM_INFO 関数に渡す引数を定義します。
TYPE	String、""	Main Program	次のタイプのいずれかを指定します。 INTEGER、NATURAL、POSITIVE、BOOLEAN、STD_LOGIC、STD_LOGIC_VECTOR、STRING、STRING_LIST、INTEGER_LIST、LONG、または FLOAT
UNITS	String、""	Main Program	パラメータの単位を設定します。使用可能な値は以下の通りです。 None、Picoseconds、Nanoseconds、Microseconds、Milliseconds、Seconds、Hertz、Kilohertz、Megahertz、Gigahertz、Address、Bits、Bytes、Kilobytes、Megabytes、Gigabytes、BitsPerSecond、KiloBitsPersecond、MegaBitsPerSecond、BigaBitsPerSecond、Percent、および Cycles 例：set_parameter_property frequency UNITS gigahertz
VISIBLE	Boolean、true	Main Program、Elaboration コールバック	パラメータ設定 GUI にパラメータを表示するかどうかを指示します。
WIDTH	String、""	Main Program	

- 表 8-5 に、system_info パラメータと一緒に使用可能なプロパティを説明します。system_info パラメータ・プロパティの使用方法について詳しくは、8-6 ページの「SYSTEM_INFO パラメータ」を参照してください。

表 8-5. SYSTEM_INFO プロパティ （1 / 2）

プロパティ名	プロパティ・タイプ	説明
ADDRESS_MAP	String	指定されたパラメータのアドレス・マップを記述する XML フォーマットの文字列を割り当てます。 set_parameter_property <my_parameter> SYSTEM_INFO {ADDRESS_MAP <my_avalon-mm_master>}
ADDRESS_WIDTH	Integer	指定されたパラメータに整数値を割り当てます。この整数値は、Avalon-MM マスタがバイト・アドレスを用いてすべてのスレーブのアドレスを指定するのに必要なビット数です。 set_parameter_property <*my_parameter*> SYSTEM_INFO {ADDRESS_WIDTH <*my_avalon-mm_master*>}
CLOCK_DOMAIN	Integer	指定されたパラメータに、クロック・ドメインを表す整数を割り当てます。このコマンドにより、モジュール内の複数のインタフェースが同じクロック・ドメインにあるかどうかを確認できます。この整数の絶対値は任意ですが、2 つのインタフェースが同じクロック・ドメインにある場合、CLOCK_DOMAIN 値は同じ、そして 1 以上でなければなりません。 set_parameter_property <*my_parameter*> SYSTEM_INFO {CLOCK_DOMAIN <*my_clk*>}
CLOCK_RATE	Integer または String	指定されたクロック入力インタフェースに、クロック周波数（Hz 単位）を表す整数を割り当てます。クロック・レートが不明な場合、0 を割り当てます。 set_parameter_property <*my_parameter*> SYSTEM_INFO {CLOCK_RATE <*my_clk*>}
CLOCK_RESET_INFO	String	モジュールのクロックまたはリセット・シンク・インタフェースの名前を指定します。（クロック・シンク・インタフェースはグローバル・リセットを使用するデザインで指定されます）
CUSTOM_INSTRUCTION_SLAVES	String	名前、ベース・アドレス、アドレス・スパン、およびクロック・サイクル・タイプを含むカスタム・インストラクション・スレーブ情報を提供します。
DEVICE	String	アルテラの部品番号（例：EP2S15F484C3）を指定します。
DEVICE_FAMILY	String	指定されたパラメータに、現在選択しているデバイスのファミリ名（特定デバイスの部品番号ではなく）を割り当てます。 set_parameter_property <my_parameter> SYSTEM_INFO {DEVICE_FAMILY}
DEVICE_FEATURES	String	スペースで区切られているキー / 値のペアのリストを作成し、現在選択されているデバイス・ファミリにサポートされるデバイス機能を示します。リストのフォーマットは、array set Tcl コマンドに渡すのに適しています。このリストは、指定されたパラメータに割り当てられます。サポートされる機能は次の通りです。 M512_MEMORY、M4K_MEMORY、M9K_MEMORY、M144K_MEMORY、MRAM_MEMORY、MLAB_MEMORY、ESB、DSP、および EMUL set_parameter_property <my_parameter> SYSTEM_INFO {DEVICE_FEATURES}
INTERRUPTS_USED	Integer または String	割り込みレシーバ・ベクタのどちらのビットが割り込みセンダに接続されるかを示すマスクを作成します。このマスクは指定されたパラメータに割り当てます。割り込みマスクを使用して、割り込みを扱うロジックを最適化することができます。 set_parameter_property <my_parameter> SYSTEM_INFO {INTERRUPTS_USED <my_interrupt_receiver>}

表 8-5. SYSTEM_INFO プロパティ （2 / 2）

プロパティ名	プロパティ・タイプ	説明
MAX_SLAVE_DATA_WIDTH	Integer	指定されたパラメータに整数を割り当てます。この整数は、特定の Avalon-MM マスタに接続された最も幅の広いスレーブのデータ幅です。 set_parameter_property <my_parameter> SYSTEM_INFO {MAX_SLAVE_DATA_WIDTH <*my_avalon_mm_master*>}
RESET_DOMAIN	Integer	指定されたパラメータのリセット・ドメインを表す整数を割り当てます。このコマンドで、モジュール内の複数のインタフェースが同じリセット・ドメインにあるかどうかを確認できます。この整数の絶対値は任意ですが、2 つのインタフェースが同じリセット・ドメインにある場合、CLOCK_DOMAIN 値は同じ、そして 1 以上でなければなりません。 set_parameter_property <*my_parameter*> SYSTEM_INFO {RESET_DOMAIN <*my_reset*>}
TRISTATECONDUIT_MASTERS	String	トライ・ステート・コンジット・インタフェースであるモジュール・インタフェースの名前を指定します。
TRISTATECONDUIT_INFO	String	特定のコンポーネント上の特定の Avalon-TC スレーブ・インタフェースに接続された Avalon-TC マスタに関する情報を含む XLM フォーマットの文字列を返します。返された文字列は、次のすべての情報を含むことがあります。 ■ Avalon-TC スレーブ・インタフェースの名前 l ■ Avalon-TC マスタ・モジュールおよびインタフェースの名前 ■ Avalon-TC 信号の名前、方向および幅 SYSTEM_INFO_ARG への引数は、特定のインタフェースを指定する正規表現です。次の例は、CFI_FLASH.uas という TC スレーブ・インタフェースに、TC_slave_info という XML 文字列を返します。 add_parameter TC_slave_info string “” set_parameter_property TC_slave_info SYSTEM_INFO_TYP TRISTATECONDUIT_INFO set_parameter_property TC_slave_info SYSTEM_INFO_ARG “uas” すべてのスレーブ・インタフェースに関する情報を取得するには、次の例に示すように、インタフェース名を “*” に切り替えます。 set_parameter_property TC_slave_info SYSTEM_INFO_ARG “*”

get_parameter_property

このコマンドは 1 つのパラメータのプロパティを返します。

get_parameter_property		
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition	
使用方法	get_parameter_property <parameterName> <propertyName>	
戻り値	プロパティに応じて、string、boolean、または units になります。25 ページの表 8–4 を参照してください。	
引数	parameterName	プロパティ値が取得されるパラメータの名前です。
	propertyName	25 ページの表 8–4 に示すプロパティのいずれかです。
例	get_parameter_property parameter1 GROUP	

set_parameter_property

このコマンドは、1 つのパラメータ・プロパティを設定します。

set_parameter_property		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	set_parameter_property <parameterName> <propertyName> <value>	
戻り値	プロパティに応じて、prstring、boolean、または units になります。	
引数	parameterName	設定されるパラメータを指定します。
	propertyName	parameterName の設定されるプロパティを指定します。プロパティの一覧については、25 ページの表 8-4 を参照してください。
	value	値を提供します。
例	set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}	

get_parameter_value

このコマンドは、以前に add_parameter によって定義されているパラメータの現在の値を返します。

get_parameter_value		
コールバック可能なフェーズ	Elaboration *(1)*、Generation、および Composition	
使用方法	get_parameter_value <parameterName>	
戻り値	String	
引数	parameterName	取得されるパラメータを指定します。
例	set fifo_width [get_parameter_value fifo_width]	

注：

(1) AFFECTS_ELABORATION=false の場合、そのパラメータは Elaboration コールバック時に使用できません。affects_generation=false の場合、そのパラメータは Generation コールバック時に使用できません。

set_parameter_value

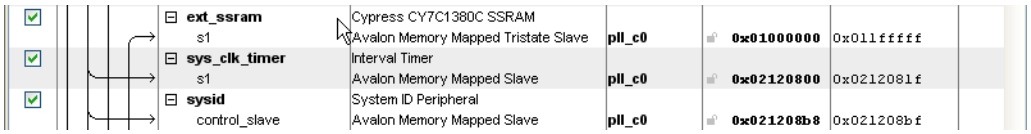
このコマンドは、パラメータの値を設定します。合成パラメータの値は、Elaboration コールバックから設定できます。

set_parameter_value		
コールバック可能なフェーズ	Elaboration および Composition	
使用方法	set_parameter_value <parameterName> <value>	
戻り値	なし	
引数	parameterName	設定されるパラメータを指定します。
	value	parameterName の値を指定します。
例	set_parameter_value BAUD_RATE 19200	

decode_address_map

これは、XML フォーマットのアドレス・マップを Tcl リストのリストに変換するユーティリティ関数です。各内部リストは、アレイに変換するための正しいフォーマットとなります。XML コードは、各スレーブの名前、開始アドレス、および終了アドレス +1 などを記述します。図 8–4 に、3 つの Avalon-MM スレーブ・デバイスを持つ Qsys システムの一部を示します。

図 8-4. 3 つの Avalon-MM スレーブを持つ Qsys システム



例 8–13 に、これらのスレーブにアクセスする Avalon-MM マスタのアドレス・マップを記述する XML を示します。提供された XML 文字列は、ここに示したものと異なる場合があります。エレメント間のスペースが異なるか、ほかの属性やエレメントが追加される可能性があります。decode_address_map コマンドによって Avalon-MM マスタのアドレス・マップを表すのは比較的容易であり、コードが XML 次期バージョンの XML アドレス・マップでも使用できるようになります。

☞ アルテラは、自分のパーサーではなく、例 8–13 のコードを使用してアドレス・マップ内のコンポーネントを列挙することを推奨しています。

例 8-13. Avalon-MM マスタのアドレス・マップ

```
<address-map>
  <slave name='ext_ssram' start='0x01000000' end='0x01200000' />
  <slave name='sys_clk_timer' start='0x02120800' end='0x02120820' />
  <slave name='sysid' start='0x021208B8' end='0x021208C0' />
</address-map>
```

decode_address_map		
コールバック可能なフェーズ	Elaboration、Generation および Composition	
使用方法	decode_address_map <address_map_XML_string>	
戻り値	Tcl リストのリスト。各リストはアレイ・セットに渡すことができます。	
引数	address_map_XML_string	Avalon-MM マスタのアドレス・マップを記述する XML 文字列です。

例	set address_map_xml [get_parameter_value my_map_param]
	set address_map_dec [decode_address_map $address_map_xml]
	foreach i $address_map_dec {
	  array set info $i
	  send_message info "Connected to slave $info(name)"
	}	

アイテムの表示

Display コマンドでコンポーネントの GUI を設定します。

add_display_item

このコマンドを使用して、下記のようにコンポーネント表示を指定することができます。

- コンポーネントのパラメータに対して論理グループを作成することができます。例えば、コンポーネントのタイミング、サイズ、およびシミュレーション・パラメータに対して個別のグループを作成する場合があります。コンポーネントは、**_hw.tcl** ファイルで指定された表示アイテムの順でグループおよびパラメータを表示します。
- 複数列の表を作成してコンポーネントのパラメータを表示することができます。複数列の表を示す例については、23 ページの例 8–12 を参照してください。
- 画像を挿入して、パラメータまたはパラメータ・グループを図的に表現することができます。
- `action` タイプの表示アイテムの追加でボタンを作成することができます。この表示アイテムは、動作が実行される時のコールバックの名前を含みます。

表示アイテムを追加することで、表示グループを作成することができます。

add_display_item		
コールバック可能なフェーズ	Main Program	
使用方法	`add_display_item <groupName> <id> <type> [<additionalInfo>]`	
戻り値	`String`	
引数	`groupName`	表示アイテムの属するグループを指定します。
	`id`	グループで表示されるパラメータまたはアイコンを指定します。コンポーネントに関連付けられる表示アイテムごとに、異なる ID が必要です。
	type	表示アイテムのカテゴリを指定します。次のタイプが定義されます。 ■ `icon`– **.gif**、**.jpg**、または **.png** ファイル ■ `parameter`– インスタンス内のパラメータ ■ `text`– テキストのブロック ■ `group`– グループです。*`groupName`* も定義されている場合、この新しいグループは *`groupName`* グループの子になります。*`groupName`* が空の文字列の場合、このグループがトップレベルになります。 ■ `action`– *`actionName`* によってラベル付けされたボタンをクリックする際にコールバックで定義されている動作です。
	additionalInfo	表示アイテムに必要な追加情報を提供します。次の例では、様々なタイプに対する `additionalInfo` 引数の使用方法を説明します。 ■ `add_display_item` *`groupName id`* `icon` *`path-to-image-file`* ■ `add_display_item` *`groupName parameterName`* `parameter` (`additionalInfo` は不要) ■ `add_display_item` *`groupName id`* `text "your-text"` `your-text` 引数は、GUI に表示されるテキストのブロックです。テキストを `"html>"` で開始すると、`<b>` と `<i>` などの単純な HTML フォーマットが使用可能になります。 ■ `add_display_item` *`parentGroupName childGroupName`* `group [tab]` `tab` はオプションのパラメータです。`tab` を使用すると、グループはインスタンスの GUI 内の別のタブで表示されるようになります。 ■ `add_display_item` *`parentGroupName actionName`* `action buttonClickCallbackProc`
例 s	`add_display_item timing read_latency parameter` `add_display_item sound speaker icon speaker.jpg`	

get_display_items

このコマンドは、パラメータ設定 GUI の一部として表示されるすべてのアイテムの一覧を返します。

get_display_items	
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition
使用方法	`get_display_items`
戻り値	`List of strings`
引数	なし
例	`get_display_items`

get_display_item_properties

このコマンドは、パラメータ設定 GUI の一部として表示されるすべてのアイテムのプロパティ名の一覧を返します。

get_display_item_properties	
コールバック可能なフェーズ	Main
使用方法	`get_display_item_properties`
戻り値	`List of strings`
引数	なし
例	`get_display_item_properties`

get_display_item_property

このコマンドは、パラメータ設定 GUI の一部として表示されるアイテムの特定のプロパティ名の値を返します。

get_display_item_property		
コールバック可能なフェーズ	Main	
使用方法	`get_display_item_property <itemName> <propertyName>`	
戻り値	`String`	
引数	`itemName`	プロパティ値が取得されるアイテムです。
	`propertyName`	値が読み出されるプロパティです。
例	`set my_label [get_display_item_property my_action DISPLAY_NAME]`	

set_display_item_property

このコマンドは、パラメータ設定 GUI の一部として表示されるアイテムの特定のプロパティ名の値を設定します。

set_display_item_property		
コールバック可能なフェーズ	Main	
使用方法	set_display_item_property <*itemName*> <*propertyName*> <*value*>	
戻り値	String	
引数	itemName	プロパティ値が設定されるアイテムです。
	propertyName	値が設定されるプロパティです。
	value	設定する値です。
例	set_display_item_property my_action DISPLAY_NAME "Click Me" set_display_item_property my_action DESCRIPTION "clicking this button runs the click_me_callback proc in the hw.tcl file"	

インタフェースおよびポート

インタフェース・コマンドおよびポートコマンドを使用して、インタフェースとポートを定義するか、またはインタフェースとポートのプロパティを取得することができます。

add_interface

このコマンドは、モジュールにインタフェースを追加します。コンポーネントの作成者がインタフェースの名前を決めます。デフォルトでは、インタフェースがイネーブルされます。インタフェース・プロパティ ENABLED を false に設定し、コンポーネント・インタフェースをディセーブルすることができます。インタフェースがディセーブルされると、そのインタフェースが隠され、ポートが自動的にデフォルト値に戻されます。*_n* を追加することでアクティブ Low に指定された信号は、1 に戻されます。その他の信号は 0 に戻されます。

各インタフェース・タイプに使用可能なプロパティはそれぞれ異なります。ENABLED プロパティはすべてのインタフェース・タイプに適用されます。ほかのプロパティの説明については、*「Avalon Interface Specifications」*を参照してください。

add_interface		
コールバック可能なフェーズ	Main program、Elaboration、および Compose	
使用方法	add_interface <*interfaceName*> <*interfaceType*> <*direction*> [<*associatedClock*>] *(1)*	
戻り値	String	
引数	interfaceName	インタフェースを識別する名前です。
	interfaceType and direction	7 種類の interfaceTypes があります。次に、これらの interfaceTypese に使用可能な方向を示します。 インタフェースのタイプ — 方向 avalon — master, slave *(2)* avalon_conduit_tristate — master avalon_streaming — source, sink interrupt — sender, receiver conduit — end clock — source, sink reset — source,sink nios_custom_instruction — slave
	associatedClock	インタフェースに関連付けられているクロックを定義します。この引数は、クロック・インタフェースを除くすべてのインタフェースに必要です。
例	add_interface mm_slave avalon slave clock0	

注：

(1) クロック・インタフェース自身など、クロックに関連付けられていないインタフェースの場合、associatedClock は省略されます。別の方法は、associatedClock 引数を非同期に設定することです。

(2) *master*、*source*、および *start* は互いに交換できます。*slave*、*sink*、および *end* は互いに交換できます。

get_interfaces

このコマンドは、以前に add_interface で定義されるすべてのインタフェースの名前をスペース区切りのリストとして返します。

get_interfaces	
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition
使用方法	get_interfaces
戻り値	List of strings
引数	なし
例	set all_interfaces [get_interfaces]

get_interface_properties

このコマンドは、特定のインタフェースのすべての使用可能なインタフェース・プロパティをスペース区切りのリストとして返します。

get_interface_properties		
コールバック可能なフェーズ	Main program、Elaborations、および Composition	
使用方法	get_interface_properties *<interfaceName>*	
戻り値	List of strings	
引数	interfaceName	定義されているインタフェースの名前です。
例	get_interface_properties mm_slave	

各インタフェースの使用可能なプロパティはそれぞれ異なります。インタフェース・プロパティについて詳しくは、「*Avalon Interface Specifications*」を参照してください。

表 8–6 に、すべてのインタフェース・タイプに共通のインタフェース・プロパティを示します。

表 8-6. すべてのインタフェース・タイプに共通インタフェース・プロパティ

プロパティ名	プロパティ・タイプ	説明
EXPORT_OF	String	合成 **_hwl.tcl** ファイルに対しては、EXPORT_OF プロパティは、子インスタンスのどちらのインタフェースがこのインタフェースを通じてエクスポートされるかを指定します。このコマンドを使用する前に、add_interface を使用して境界インタフェースを作成する必要があります。エクスポートとされるインタフェースの形式は、*<instanceName.interfaceName>* となります。 例：set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port
ENABLED	Boolean	インタフェースがイネーブルされるかどうかを指定します。

get_interface_property

このコマンドは、特定のインタフェースから 1 つのインタフェース・プロパティの値を返します。

get_interface_property		
コールバック可能なフェーズ	Main Program、Compose、および Elaboration	
使用方法	get_interface_property <interfaceName> <propertyName>	
戻り値	プロパティに応じて、string、boolean、または units になります。インタフェース・プロパティについて詳しくは、「*Avalon Interface Specifications*」を参照してください。	
引数	interfaceName	情報を取得しようとするインタフェースの名前です。
	propertyName	値を取得しようとするプロパティの名前です。このプロパティは、ENABLED または ASSOCIATED_CLOCK 、あるいはインタフェースによって定義されているプロパティ名となります。
例	get_interface_property mm_slave readWaitTime	

set_interface_property

このコマンドはインタフェースの 1 つのインタフェース・プロパティを設定します。

set_interface_property		
コールバック可能なフェーズ	Main、Compose、および Elaboration	
使用方法	`set_interface_property <interfaceName> <propertyName> <value>`	
戻り値	`String`	
引数	`interfaceName`	このプロパティが属するインタフェースの名前です。
	`propertyName`	設定しようとするプロパティの名前です。このプロパティは、`ENABLED` または `ASSOCIATED_CLK`、あるいは 「*Avalon Interface Specifications*」 からのプロパティ名となります。
	`value`	設定しようとする特定のプロパティの値です。
例	`set_interface_property mm_slave linewrapBursts false`	

add_interface_port

このコマンドは、モジュール上のインタフェースにポートを追加します。コンポーネントの作成者がポートの名前を決めます。ポートの幅と方向を Elaboration フェーズの最後に設定しなければなりません。ポート幅は、次のいずれかの方法で設定できます。

- Main Program フェーズで、定数の幅または幅の式を設定
- Elaboration コールバックで、定数の幅を設定

☞ Elaboration コールバックが使用されていない場合、`quartus_map` が HDL からスタティック・コンポーネントのポート幅を決定します。

add_interface_port		
コールバック可能なフェーズ	Main Program および Elaboration	
使用方法	`add_interface_port <interfaceName> <portName> <portRole> [<direction> <width_expr>]`	
戻り値	`String`	
引数	`interfaceName`	ポートが属するインタフェースの名前です。
	`portName`	コンポーネントの作成者が選択した、ポートの名前です。
	`portRole`	このポートのインタフェースにおける役割です。ポートの役割は、*Avalon Interface Specification* で `signal types` として記述されています。各インタフェースに使用可能な `signal types` については、「*Avalon Interface Specifications*」 を参照してください。
	`direction`	`direction` は、`input`、`output`、または `bidir` のいずれかとなります。
	`width_expr`	ポート幅の式です。単純なケースでは、この式が幅のビット数となる場合もあります。
例	`add_interface_port mm_slave s0_rdata readdata output 32`	

get_interface_ports

このコマンドは、特定のインタフェースに追加されたすべてのポートの名前を返します。インタフェース名が省略された場合、すべてのインタフェースのあらゆるポートが返されます。

get_interface_ports		
コールバック可能なフェーズ	Main、Elaboration、および Generation	
使用方法	`get_interface_ports [<interfaceName>]`	
戻り値	String	
引数	interfaceName	ポートを表示しようとするインタフェースの名前です。（オプション）
例	`get_interface_ports mm_slave`	

get_port_properties

このコマンドは、すべての使用可能なポート・プロパティをリストとして返します。

get_port_properties		
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition	
使用方法	`get_port_properties <portName>`	
戻り値	プロパティに応じて、String、boolean、または units になります。25 ページの表 8–4 を参照してください。	
引数	portName	プロパティが必要とされるポートの名前です。次のポート・プロパティがサポートされます。 ■ DIRECTION ■ TERMINATION ■ TERMINATION_VALUE ■ VHDL_TYPE ■ WIDTH ■ WIDTH_EXPR ■ DRIVEN_BY ■ ROLE これらのプロパティの説明については、表 8–7 を参照してください。
例	`get_port_properties mm_slave`	

表 8–7 に、使用可能なポート・プロパティを説明します。

表 8-7. ポート・プロパティ

名前	タイプ	説明
DIRECTION	input, output, bidir	コンポーネントの視点からの、ポートの方向です。
TERMINATION	boolean	true に設定されると、ポートは Qsys システムに接続する代わりに、output と bidir の場合に未接続のままにするか、または input. の場合に固定値に設定されます。デフォルトのラッパー生成の代わりに Generation コールバックを実装するコンポーネントには効果がありません。
TERMINATION_VALUE	integer	入力ポートをドライブする定数値です。
VHDL_TYPE	std_logic std_logic_vector auto	VHDL ポートのタイプを指定します。デフォルト値の auto は、幅が 1 に固定されると std_logic を選択し、それ以外の場合は std_logic_vector を選択します。
WIDTH	integer	ポートの幅（ビット単位）。
WIDTH_EXPR	string	ポート幅を表す式です。width および width_expr の設定は同じ効果があります。両方も有効な幅の式を更新します。width/width_expr プロパティは、いつでも integer に設定できます。ただし、演算式に設定するのは、Main Program フェーズでのみ実行できます。
DRIVEN_BY	integer, input	この出力ポートが常に固定値または入力ポートによってドライブされるように指定します。コンポーネント上のすべての出力の driven_by プロパティが有効な値に設定される場合、コンポーネントの HDL は自動的に生成されます。
ROLE	string	waitrequest、readdata、または read など、Avalon 信号の種類を指定します。信号の種類の完全なリストについては、 *「Avalon Interface Specifications」* を参照してください。

get_port_property

このコマンドは、特定のポートの 1 つのポート・プロパティの値を返します。

get_port_property		
コールバック可能なフェーズ	Main、Elaboration、および Generation	
使用方法	get_port_property *<portName> <propertyName>*	
戻り値	プロパティの種類に依存します。	
引数	portName	ポート名
	propertyName	表 8–7 に示すサポートされるプロパティのいずれか
例	get_port_property rdata WIDTH	

set_port_property

このコマンドは、1 つのポート・プロパティを設定します。

set_port_property		
コールバック可能なフェーズ	Main program および Elaboration	
使用方法	set_port_property *<portName> <propertyName> [<value>]*	
戻り値	プロパティに応じて、String、boolean、または units になります。25 ページの表 8–4 を参照してください。	
引数	portName	ポート名
	propertyName	表 8–7 に示すサポートされるプロパティのいずれか
	value	設定値
例	set_port_property rdata WIDTH 32	

get_interface_assignments

このコマンドは、特定のインタフェースのすべてのインタフェース・アサインメントの値を返します。

get_interface_assignments		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	get_interface_assignments *<interfaceName>*	
戻り値	String	
引数	interfaceName	アサインメントが読み出される Avalon インタフェースの名前です。
例	get_interface_assignments s1	

get_interface_assignment

このコマンドは、特定のインタフェースの特定のアサインメント名を返します。

get_interface_assignment		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	get_interface_assignments *<interfaceName> <name>*	
戻り値	String	
引数	interfaceName	アサインメントが読み出される Avalon インタフェースの名前です。
	name	値が読み出されるアサインメントの名前です。
例	get_interface_assignment s1 embeddedsw.configuration.isFlash	

set_interface_assignment

このコマンドは、特定のインタフェースの特定のアサインメントの値を設定します。

set_interface_assignment		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	`set_interface_assignment <interfaceName> <name> [<value>]`	
戻り値	なし	
引数	`interfaceName`	アサインメントが設定される Avalon インタフェースの名前です。
	`name`	値が設定されるアサインメントの名前です。
	`value`	設定値
例	`set_interface_assignment s1 embeddedsw.configuration.isFlash 1`	

`set_interface_assignment` コマンドの使用について詳しくは、「*Nios II Software Developer's Handbook*」の「*Publishing Component Information to Embedded Software*」の章を参照してください。

Composition

この項では、複数のコンポーネントを組み合わせて新しいコンポーネントを構築するためのコマンドについて説明します。また、システム内のモジュール・インスタンスにクエリーを行うコマンドについても説明されています。

add_instance

`add_instance` コマンドは、定義済みのモジュールのインスタンス（*child* または *child module*）を新しいコンポーネントに追加します。このコマンドを使用して、複数のコンポーネントから合成されたコンポーネントを作成することができます。

add_instance		
コールバック可能なフェーズ	Main および Compose	
使用方法	`add_instance` *<instanceName> <type> [<version>]*	
戻り値	`String`	
引数	`instanceName`	モジュールを扱うのに使用される固有のローカル名を指定します。この名前は、モジュールを識別するために生成された HDL に使用されます。
	`type`	`type` は、ライブラリで使用可能なモジュールを示します。例えば、`altera_avalon_uart` です。
	`version`	特定のモジュールに必要なバージョンです。バージョンが指定されていない場合、最新のバージョンが使用されます。
例	`add_instance my_uart altera_avalon_uart`	

get_instances

このコマンドは、システム内のすべてのモジュールの名前を返します。

get_instances		
コールバック可能なフェーズ	Main、Elaboration、および Composition	
使用方法	`get_instances`	
戻り値	`List of strings`	
引数	なし	
例	`get_instances`	

get_instance_parameters

このコマンドは、親インスタンスに操作可能な子インスタンス上のすべてのパラメータの名前を返します。派生したパラメータ、および SYSTEM_INFO パラメータ・プロパティが設定されているパラメータは省略されます。

get_Instance_parameters		
コールバック可能なフェーズ	Main、Elaboration、および Compose	
使用方法	`get_instance_parameters` *<instanceName>*	
戻り値	`List of strings`	
引数	`instanceName`	パラメータが取得されるインスタンスの名前を指定します。
例	`get_instance_parameters pixel_converter`	

set_instance_parameter_value

このパラメータは、子モジュールのパラメータを設定します。子モジュールの派生したパラメータおよび SYSTEM_INFO パラメータは、このコマンドによって設定できません。

set_instance_parameter_value		
コールバック可能なフェーズ	Main および Compose	
使用方法	`set_instance_parameter_value` *<instanceName>* `<parameterName> <parameterValue>`	
戻り値	なし	
引数	`instanceName`	子モジュールの名前を指定します。
	`parameterName`	設定されるパラメータを指定します。
	`parameterValue`	設定されるパラメータの値を指定します。
例	`set_instance_parameter_value pixel_converter input_DPI 1200`	

get_instance_parameter_value

このコマンドは、特定のパラメータの値を返します。このコマンドは、値が他のパラメータから派生したパラメータ、または SYSTEM_INFO パラメータ・プロパティによって定義されているパラメータに使用できません。

get_instance_parameter_value		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_parameter_value *<instanceName> <parameterName>*	
戻り値	プロパティに応じて、String、boolean、または units になります。25 ページの表 8–4 を参照してください。	
引数	instanceName	パラメータが取得されるインスタンスの名前を指定します。
	parameterName	値が取得されるパラメータを指定します。
例	get_instance_parameter_value pixel_converter input_DPI	

get_instance_parameter_property

このコマンドは、特定のインスタンス・パラメータ・プロパティの名前を返します。親インスタンスに見える、子インスタンス上のパラメータは、TYPE、WIDTH、DERIVED、VISIBLE、ENABLED、UNITS、DISPLAY_NAME、ALLOWED_RANGES、および SYSTEM_INFO となります。

get_Instance_parameter_property		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_parameter_property *<instanceName> <parameterName> <propertyName>*	
戻り値	プロパティに応じて、String、boolean、または units になります。25 ページの表 8–4 を参照してください。	
引数	instanceName	モジュールのインスタンス名を指定します。
	parameterName	プロパティが取得されるパラメータを指定します。
	propertyName	値が取得されるプロパティを指定します。
例	get_instance_parameter_property my_stereo separate_control DISPLAY_NAME	

get_instance_interfaces

このコマンドは、子モジュールのすべてのインタフェースの名前を返します。モジュールのパラメータ設定が変更された場合、インタフェースが変更されることがあります。

get_Instance_interfaces		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_interfaces <*instanceName*>	
戻り値	String	
引数	instanceName	モジュールのインスタンス名を指定します。
例	get_instance_interfaces my_ColorSpaceConverter	

get_instance_interface_properties

このコマンドは、特定のインタフェースのすべてのプロパティの名前を返します。

get_Instance_interface_properties		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_interface_properties <*instanceName*> <*interfaceName*>	
戻り値	String	
引数	instanceName	モジュールのインスタンス名を指定します。
	interfaceName	インスタンスのインタフェースを指定します。
例	get_instance_interface_properties my_ColorSpaceConverter inputInterface	

get_instance_interface_property

このコマンドは、特定のモジュール・インタフェースに関連付けられているプロパティの値を返します。

get_Instance_interface_property		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_interface_property <*instanceName*> <*interfaceName*> <*propertyName*>	
戻り値	String	
引数	instanceName	モジュールのインスタンス名を指定します。
	interfaceName	インスタンスのインタフェースを指定します。
	propertyName	値が取得されるプロパティを指定します。
例	get_instance_interface_property my_component s1 setupTime	

get_instance_interface_ports

このコマンドは、特定のインタフェース上のポートの名前をリストとして返します。

get_Instance_interface_ports		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_interface_ports <*instanceName*> <*interfaceName*>	
戻り値	List of Strings	
引数	instanceName	モジュールのインスタンス名を指定します。
	interfaceName	インスタンスのインタフェースを 1 つ指定します。
例	get_instance_interface_ports my_ColorSpaceConverter outputInterface	

get_instance_port_property

このコマンドは、指定されたポート・プロパティに関する情報を返します。

get_instance_port_property		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_instance_port_property <*instanceName*> <*portName*> <*propertyName*>	
戻り値	String	
引数	instanceName	モジュールのインスタンス名を指定します。
	portName	ポートを指定します。
	property	情報が取得されるプロパティを指定します。一部のポート・プロパティのみが親に見えます。見えるものは、ROLE、DIRECTION、WIDTH、WIDTH_EXPR および VHDL_TYPE となります。
例	get_instance_port_property my_uart width	

add_connection

このコマンドは、適切な接続タイプを使用して、指定された複数のインタフェースを接続させることができます。両方のインタフェース名は、子インスタンスの名前とそれに続く、そのモジュールに提供されたインタフェースの名前から構成されています。例えば、mux0.out は mux0 と呼ばれるインスタンス上の out と呼ばれるインタフェースです。このコマンドは、新たに追加された接続を start.point/end.point のフォーマットとして返します。開始インタフェースから終了インタフェースまでの順で接続するように注意してください。

add_connection		
コールバック可能なフェーズ	Main program および Compose	
使用方法	add_connection <*start.Interface*> [<*end.Interface*>] [*kind*] [*name*]	
戻り値	String	
??	start.interface	接続される開始インタフェース。 <*instance_name*>.<*interface_name*>
	end.interface	接続される終了インタフェース。 <*instance_name*>.<*interface_name*>
	kind	インタフェースのタイプを示します。インタフェース・タイプのリストについては、8-36 ページの「add_interface」を参照してください。
	name	接続の名前を指定します。省略される場合、名前は start-module.start-interface/end-module.end-interface の形式となります。
例	add_connection dma.read_master sdram.s1	

get_connections

エレメントが指定されていない場合、このコマンドは、システム内のすべての接続をリストとして返します。インタフェースが指定されている場合（例えば cpu.instruction_master）、そのインタフェースへの接続が返されます。コンポーネント・インスタンスが指定されている場合、そのインスタンスの任意のインタフェースへの接続はすべて返されます。

get_connections	
コールバック可能なフェーズ	Main および Compose
使用方法	get_connections *[<interfaceName or instanceName>]*
戻り値	List of strings
引数	なし
例	get_connections cpu.instruction_master

get_connection_parameters

このコマンドは、指定された接続のすべてのパラメータの名前を取得します。

get_connection_parameters		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_connection_parameters <*connectionName*>	
戻り値	`List of strings`	
引数	`connectionName`	接続パラメータが必要とされる接続を指定します。
例	`get_connection_parameters cpu0.data_master/dma0.csr`	

get_connection_parameter_value

このコマンドは、接続のパラメータの値を取得します。

get_connection_parameter_value		
コールバック可能なフェーズ	Main および Compose	
使用方法	get_connection_parameters <*connectionName*> <*parameterName*>	
戻り値	`String`	
引数	`connectionName`	接続パラメータが必要とされる接続を指定します。
	parameterName	プロパティ値が取得されるパラメータの名前です。
例	`get_connection_parameters cpu0.data_master/dma0.csr`	

set_connection_parameter_value

このコマンドは、接続のプロパティを設定します。開始と終了はそれぞれ、*<instance>.<interface>* フォーマットの各インタフェース名となります。接続パラメータは接続のタイプに依存します。Avalon-MM の場合、ベース・アドレスとアービトレーション・プライオリティが含まれます。

set_connection_parameter_value		
コールバック可能なフェーズ	Main program および Compose	
使用方法	set_connection_parameter_value <*connName*> <*parameterName*> <*parameterValue*>	
戻り値	なし	
引数	connName	add_conection コマンドによって返される接続の名前を指定します。形式は start.point/end.point となります。
	parameterName	設定されるパラメータを指定します。
	parameterValue	パラメータの値を指定します。
例	set_connection_parameter_value cpu0.data_master/dma0.csr baseAddress 0x1000	

ファイル設定および生成

この項では、コンポーネントを定義し、ダウンストリーム・ツールに情報を提供するファイルを生成するコマンドについて説明します。生成プロパティについても説明します。

get_files

このコマンドは、このモジュールに追加されたすべてのファイルのリストを返します。

get_files	
コールバック可能なフェーズ	Main、Elaboration、および Generation
使用方法	get_files
戻り値	List of strings
引数	なし
例	set list_of_files [get_files]

add_file

このコマンドは、モジュールに合成ファイル、シミュレーション・ファイル、または TimeQuest 制約ファイルを追加します。Main Program で追加されたファイルは削除できません。Generation コールバックでファイルを追加することにより、含まれたファイルがパラメータ・セットの関数または生成の結果になることができます。コールバックで追加されたファイルは、Main Program で追加されたファイルに追加されるものです。

add_file		
コールバック可能なフェーズ	Main、Elaborate、および Generation	
使用方法	`add_file filename [<fileProperties> . . . ]`	
戻り値	`String`	
引数	`filename`	追加されるファイル名です。パスは **_hw.tcl** を含むディレクトリを基準にします。
	`fileProperties`	ファイルは次の 3 つのプロパティをサポートします。 ■ `SIMULATION`— シミュレーション用のファイル ■ `SYNTHESIS`— 合成用のファイル ■ SDC—TimeQuest 制約ファイル (SDC は合成ファイルと同様に動作します)
例	`add_file my_component.v {SIMULATION SYNTHESIS}`	

add_fileset

このコマンドは、`<filesetKind>` によって指定されるとおり、特定のターゲットに対して生成用のファイルセットを追加します。このターゲット（`SIM_VHDL`、`SIM_VERILOG`、`QUARTUS_SYNTH`、または `EXAMPLE_DESIGN`）は、指定されたターゲットが要求されるときに Qsys に呼び出されます。各種のファイルセットに対して複数のファイルセットを定義することができます。指定されたコールバックには、1 つの引数が必要です。この引数の値は生成された名前であり、コンポーネントのトップレベル・モジュールまたはエンティティ宣言に使用されなければなりません。この生成された名前を上書きするには、`STATIC_TOP_LEVEL_MODULE_NAME` モジュール・プロパティを設定する必要があります。

☞ 生成された名前の上書きは、コアのすべてのパラメータ設定が同様な HDL を得る場合にのみ安全です。

add_fileset		
コールバック可能なフェーズ	Main	
使用方法	add_fileset <filesetName> <filesetKind> <callbackProcName> [<displayName>]	
戻り値	String	
引数	filename	ファイルセットの名前です。
	filesetKind	ファイルは次の種類をサポートします。 ■ SIM_VHDL ■ SYSTEM_VERILOG ■ QUARTUS_SYNTH ■ EXAMPLE_DESIGN
	calbackProcName	コールバックの名前を識別する文字列です。
	displayName	ファイルセットを識別するのに表示される文字列です。
例	add_fileset PCIE_SYNTHESIS QUARTUS_SYNTH mySynthProc	

add_fileset_file

このコマンドは、生成用のディレクトリ内の特定の位置にファイルを追加します。絶対パスまたはコンポーネントの **_hw.tcl** ファイルを基準にしたパスを使用してソース・ファイルの位置を指定できます。

add_fileset_file		
コールバック可能なフェーズ	File Set Generation	
使用方法	add_fileset_file <fileDestination> <fileKind> <fileSource> <contentsOrPath>	
戻り値	String	
引数	fileDestination	Qsys 生成後にファイルを格納する位置を指定します。
	fileKind	ファイルは次の種類をサポートします。 ■ VERILOG ■ SYSTEM_VERILOG ■ SYSTEM_VERILOG_INCLUDE ■ VHDL ■ SDC ■ MIF ■ HEX ■ DAT ■ OTHER
	fileSource	次のソースが定義されています。 ■ PATH—filePath にコピーされるオリジナルのソース・ファイルを指定します。 ■ TEXT— ファイルの内容に対して任意のテキスト文字列を指定します。
	contentsOrPath	fileSource が PATH の場合、filePath にコピーされるファイルを指定します。 fileSource が TEXT の場合、このファイルに格納されるテキスト文字列を指定します。
例	add_fileset_file "./implementation/rx_pma.sv" SYSTEM_VERILOG PATH rx_pma.sv add_fileset_file gui.sv SYSTEM_VERILOG TEXT "Customize your IP core"	

get_file_properties

このコマンドは、ファイルに対して定義されているすべてのプロパティのリストを返します。

get_file_properties	
コールバック可能なフェーズ	Main、Elaboration、Generation、および Composition
使用方法	get_file_properties
戻り値	List of strings
引数	なし
例	get_file_properties

get_file_property

このコマンドは、1 つのファイル・プロパティの値を返します。引数として渡されるファイル名がユニークである限り、一部のみ入力しても構いません。例えば、フルのファイル名が **/components/my_file.v** の場合、**my_file.v** だけで十分です。

get_file_property		
コールバック可能なフェーズ	Main、Elaboration、および Generation	
使用方法	`get_file_property <filename> <propertyName>`	
戻り値	`Boolean`	
引数	`filename`	プロパティが取得されるファイルの名前です。
	`propertyName`	値が取得されるプロパティの名前です。
例	`set forSynthesis [get_file_property my_file.v SYNTHESIS]`	

set_file_property

このコマンドは、1 つのファイル・プロパティの値を設定します。関数に渡されるファイル名がユニークである限り、一部のみ入力しても構いません。例えば、フルのファイル名が **/components/my_file.v** の場合、**my_file.v** だけで十分です。使用可能なプロパティは、`add_files` コマンドで説明されています。

set_file_property		
コールバック可能なフェーズ	Main、Elaboration、および Generation	
使用方法	`set_file_property <filename> <propertyName> <propertyValue>`	
戻り値	`Boolean`	
引数	`filename`	プロパティが取得されるファイルの名前です。
	`propertyName`	値が取得されるファイル・プロパティの名前です。
	`propertyValue`	ファイル・プロパティに設定する値です。
例	`set_file_property my_file.v SYNTHESIS true`	

create_temp_file

このコマンドは、**_hw.tcl** ファイルの生成コールバックで操作できる一時ファイルを作成します。この一時ファイルは、スクラッチ・パッドとして使用されるか、または `add_fileset_file` コマンドによって追加される場合に生成の出力に含めることができます。

create_temp_file		
コールバック可能なフェーズ	File Set Generation	
使用方法	`create_temp_file <fileName>`	
戻り値	`String`	
引数	`fileName`	作成されるファイルの名前です。
例	`set filelocation [create_temp_file "./hdl/compute_frequency.v"` `add_fileset_file compute_frequency.v VERILOG PATH ${filelocation}`	

get_generation_properties

このコマンドは、すべての使用可能な生成プロパティの名前をスペース区切りのリストとして返します。これらのプロパティはモジュールによって変更することができません。インスタンスごとの HDL 生成をサポートするために、生成プロパティが生成コールバックに提供されます。

get_generation_properties	
コールバック可能なフェーズ	Main、Elaboration、Compose、および Generation
使用方法	`get_generation_properties`
戻り値	`String`。次の生成プロパティがサポートされます。 ■ `hdl_language` ■ `output_directory` ■ `output_name` 生成プロパティの説明については、表 8–8 を参照してください。
引数	なし
例	`get_generation_properties`

表 8–8 に、生成プロパティについて説明します。

表 8-8. 生成プロパティ

名前	タイプ	説明
`HDL_LANGUAGE`	`enum`	生成する HDL 言語です。`verilog` または `vhdl` (小文字) です。モジュールが特定の言語で生成することができない場合、その他の言語で生成しても構いません。
`OUTPUT_DIRECTORY`	`file`	ファイルが生成される位置です。ディレクトリ名内のファイル名コンポーネントは、スラッシュで区切られています。
`OUTPUT_NAME`	`string`	`OUTPUT_NAME` が `module_0` で、`HDL_LANGUAGE` が `verilog` の場合、**module_0.v** ファイルまたは **module_O.sv** _ ファイルを生成する必要があり、`module_0` モジュールを含む必要があります。

get_generation_property

このコマンドは、1 つの生成プロパティの値を返します。

get_generation_property		
コールバック可能なフェーズ	Generation	
使用方法	`get_generation_property` *<propertyName>*	
戻り値	プロパティに応じて、String、boolean、または units になります。25 ページの表 8–4 を参照してください。	
引数	`propertyName`	下記 3 つの生成プロパティのいずれかになります。 ■ HDL_LANGUAGE ■ OUTPUT_DIRECTORY ■ OUTPUT_NAME
例	`get_generation_property OUTPUT_DIRECTORY`	

改訂履歴

表 8–9 に、本資料の改訂履歴を示します。

表 8-9. 改訂履歴

日付	バージョン	変更内容
2011 年 5 月	11.0.0	■ ベータのステータスを削除 ■ HDL および合成コンポーネントの実装について説明するセクションを改訂 ■ 例でのリセット・インタフェースおよびクロック・インタフェースを分ける ■ TRISTATECONDUIT_INFO、GENERATION_ID、および SYSTEM_INFO プロパティを追加 ■ WIDTH および SYSTEM_INFO_ARG パラメータ・プロパティを追加 ■ add_documentation_link コマンドから doc_type 引数を削除 ■ get_instance_parameter_properties コマンドを削除 (get_instance_parameter_property コマンドが使用可能) ■ add_fileset、add_fileset_file および create_temp_file コマンドを追加 ■ クロック・インタフェースおよびリセット・インタフェースを分けるように Tcl 例を更新
2010 年 12 月	10.1.0	初版

Quartus II ハンドブックの以前のバージョンについては、Quartus II Handbook Archive を参照してください。

このハンドブックの章について、「online survey」でフィードバックを提供してください。