

Vivado Design Suite クイックリファレンス

ヘルプの表示

Vivado® Design Suite で使用可能なすべてのコマンドの最新情報は、Tcl コンソールのヘルプを使用してください。h ヘルプを表示するには、Tcl プロンプトに「help」と入力します。

Vivado ツールで使用可能なコマンドのカテゴリがリストされるので、カテゴリを指定してそのカテゴリに含まれるコマンドをリストできます。たとえば、XDC コマンドのリストを表示するには、「help -category XDC」と入力します。

詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) を参照してください。Vivado ツールのビデオ チュートリアルは、<http://japan.xilinx.com/training/vivado/index.htm> を参照してください。

シミュレーション コマンド

Vivado シミュレータは、イベントドリブンのハードウェア記述言語 (HDL) シミュレータで、VHDL、Verilog、SystemVerilog、および混合言語のビヘイビア、論理、およびタイミング シミュレーションがサポートされます。

コマンド	目的	例	出力
xvlog xvhdl	Verilog および VHDL ファイルをコンパイル	xvlog file1.v file2.v xvhdl f1.vhd f2.vhd	解析済みデータをディスク上の HDL ライブラリに保存
xelab	コンパイルおよび エラポレーション	xelab work.top1 work.top2 -s cpusim	スナップショットを作成
xsim	実行可能なスナップショットでシミュレーションを実行	xsim <options> <exe>	シミュレーション スナップショットを読み込んでバッチ モードシミュレーションを実行するか、GUI または Tcl ベースの環境でインタラクティブにシミュレーションを実行

Vivado IDE からシミュレーション スクリプトを作成する場合は、「launch_simulation -scripts_only」と入力します。詳細は、『Vivado Design Suite ユーザー ガイド: ロジックシミュレーション』(UG900) を参照してください。

Vivado ハードウェア マネージャー

ハードウェア マネージャーでは、ザイリンクス FPGA デバイスにインプリメントされているデバッグ コアと通信できます。次に、ハードウェア マネージャーの機能にアクセスする Tcl コマンドを示します。

- **open_hw** : Vivado Design Suite でハードウェア マネージャーを開きます。
- **connect_hw_server** : ローカルまたはリモートのハードウェア サーバー アプリケーションに接続します。
- **open_hw_target** : ハードウェア ターゲットへの接続を開きます。
- **current_hw_device** : プログラムおよびデバッグするザイリンクス FPGA デバイスを設定または取得します。
- **get_hw_ilas** : デザインに含まれる信号の監視、ハードウェア イベントでのトリガー、リアルタイムでのデータキャプチャに使用する ILA デバッグ コア オブジェクトを取得します。ILA コアと通信するには、任意の `*_hw_ila*` Tcl コマンドを使用します。
- **get_hw_vios** : 制御信号を駆動し、デザインのステータス信号を監視するのに使用する仮想 I/O デバッグ コア オブジェクトを取得します。
- **get_hw_axis** : ザイリンクス FPGA デバイスに含まれるシステムで動作中のさまざまな AXI フルおよび AXI Lite スレーブ コアと通信する AXI トランザクションを生成するために使用するデバッグ コア オブジェクトを取得します。
- **get_hw_sio_iberts** : 高速シリアル I/O 転送および受信設定を計測、最適化したり、転送ビット エラー レートを計測するために使用する SIO デバッグ コアを取得します。

詳細は、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』(UG908) を参照してください。

Vivado Design Suite クイックリファレンス

Vivado IDE 起動モード

Vivado をコマンドラインから起動する場合、次の 3 つのモードがあります。

1. GUI モード : デフォルト モードで、Vivado IDE が起動します。
使用方法 : vivado または vivado -mode gui
2. Tcl シェル モード : Vivado Design Suite Tcl シェルから Tcl スクリプトを実行します。
使用方法 : vivado -mode tcl
注記 : Tcl シェルから Vivado IDE を開く場合は start_gui、閉じる場合は stop_gui を使用します。
3. バッチ モード : Tcl シェルを起動して、Tcl スクリプトを実行し、ツールを終了します。
使用方法 : vivado -mode batch -source <file.tcl>

Vivado コマンド オプション :

-mode 起動モード : gui, tcl, または batch。デフォルト : gui
-init 初期化中に vivado.tcl ファイルをソースにします。
-source 特定の Tcl ファイルをソースにします。バッチ モードが必要です。
-nojournal ジャーナル ファイルを書き出しません。
-appjournal ジャーナル ファイルに上書きせずに追加します。
-journal ジャーナル ファイル名を指定します。デフォルトは verilog.jou です。
-nolog ログ ファイルを書き出しません。
-applog ログ ファイルに上書きせずに追加します。
-log ログ ファイル名を指定します。デフォルトは vivado.log です。
-version バージョン情報を表示して終了します。
-tclargs Tcl argc argv に渡す引数を指定します。
-tempDir 一時的ディレクトリ名を指定します。
-verbose コマンド実行中のメッセージ制限を解除します。
<project / dcp> 指定のプロジェクト ファイル (.xpr) またはデザイン チェック ポイント (.dcp) を読み込みます。

主なレポートコマンド

Vivado Design Suite にはレポートコマンドが多数含まれており、デザイン フローを進んでいくにつれて、異なるレベルの情報が提供されます。

カテゴリ	目的	例
タイミング	デザイン目標	check_timing report_timing report_clocks report_clock_interaction report_cdc report_synchronizer_mtbf
パフォーマンス	デザイン目標に対する測定	report_timing_summary report_datasheet report_design_analysis report_power report_pulse_width
リソース使用率	論理/物理リソース マップ	report_utilization report_clock_util report_io report_control_sets report_ram_configuration
デザイン ルール チェック	物理検証	report_drc report_methodology report_ssn
デザイン データ	プロジェクト特有の設定	report_param report_config_timing report_ip_status

Vivado Design Suite クイックリファレンス

デザイン オブジェクト クエリ コマンド

コマンド	説明
get_cells	名前/階層または接続に基づいてロジック セル オブジェクトを取得します。
get_pins	名前/階層または接続に基づいてピン オブジェクトを取得します。
get_nets	名前/階層または接続に基づいてネット オブジェクトを取得します。
get_ports	名前または接続に基づいて最上位ネットリスト ポートを取得します。
all_inputs	現在のデザインの入力ポートすべてが返されます。
all_outputs	現在のデザインの入力ポートすべてが返されます。
all_ffs	現在のデザインのリップフロップすべてが返されます。
all_latches	現在のデザインのリッチすべてが返されます。
all_dsps	現在のデザインのリッチすべてが返されます。
all_rams	現在のデザインのリッチすべてが返されます。

タイミング ベースのクエリ コマンド

コマンド	説明
all_clocks	現在のデザインに含まれるすべての定義されたクロックのリストを取得します。
get_clocks	名前または通過したオブジェクトに基づいてクロック オブジェクトを取得します。
get_generated_clocks	生成クロック オブジェクトを取得します。
all_fanin	タイミング バスに含まれる指定したオブジェクトのファンインのピンまたはセルのリストを取得します。
all_fanout	タイミング バスに含まれる指定したオブジェクトのファンアウトのピンまたはセルのリストを取得します。
all_registers	現在のデザインに含まれるシークンシャルラッチ セルすべてのリストを取得します。
get_path_groups	バス グループ オブジェクトのリストを取得します。
get_timing_paths	report_timing のプリントアウト同等のタイミング バス オブジェクトを取得します。

フィルター

すべての get_* コマンドで -filter オプションを使用できます。また個別のフィルター コマンドもあります。フィルターを使用すると、オブジェクト プロパティに基づいて返されるオブジェクトのリスト数を減らすことができます。たとえば、LIB_CELL タイプに限定する場合は、次を入力します。

```
> get_cells -hier -filter {LIB_CELL == FDCE}
```

複数のフィルターを組み合わせたことも可能です。

```
> get_ports -filter {DIRECTION == in && NAME == *clk*}
```

ブール型プロパティでは直接フィルターできます。

```
> get_cells -filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

有効な演算 : ==, !=, ==, !=, <=, >=, <, && および || をフィルター パターンで使用できます。



Vivado Design Suite クイック リファレンス

Tcl の例

すべてのポートの方向および I/O 規格を表示する反復ループ例：

```
foreach x [get_ports] {puts "$x \  
    [get_property IOSTANDARD $x] \  
    [get_property DIRECTION $x]"}
```

指定したオプションまたは引数をサポートするコマンドのリストを表示する例：

```
proc findCmd {option} {  
    foreach cmd [lsort [info commands *]] {  
        catch {  
            if {[regexp "$option" [help -syntax $cmd]]} {  
                puts $cmd  
            }  
        }  
    }  
}; # End proc
```

使用方法：findCmd <option>

入力ピン オブジェクトリストに基づいてプリミティブ/ピン名の配列を返す例：

```
proc countPrimPin {pinObjs} {  
    array set count {}  
    foreach pin $pinObjs {  
        set primpinname [getPrimPinName $pin]  
        if {[info exist count($primpinname)]} {  
            incr count($primpinname)  
        } else {  
            set count($primpinname) 1  
        }  
    }  
    return [array get count]  
}
```

使用方法：countPrimPin <pinObjs>

指定ピンのプリミティブ/ピン名を返す例：

```
proc getPrimPinName {pin} {  
    set pinname [regsub {./}([/])"$} [get_property name $pin] {/}]  
    set primname [get_property LIB_CELL [get_cells -of $pin]]  
    return "$primname/$pinname"  
}
```

使用方法：getPrimPinName <pin>

リストに含まれる最長文字列の文字数を返す例：

```
proc returnMaxStringLength {list} {  
    set l [map {x} {return [string length $x]}] [lsort -unique $list]  
    return [expr max([join $l ])]  
}
```

使用方法：returnMaxStringLength <list>

詳細は、『Vivado Design Suite ユーザー ガイド：Tcl スクリプト機能の使用』(UG894) を参照してください。

Vivado Design Suite クイック リファレンス

バッチ モード スクリプトの例

BFT サンプル デザインを使用した、プロジェクト モードと非プロジェクト モードの両方のスクリプト例

非プロジェクト モード

このモードでは、ソースが現在の場所から読み込まれ、デザインはメモリ内でコンパイルされます。メモリ内でアクティブ デザインで作業しているため、変更は自動的にその後のフローに渡されます。デザイン プロセスのどの段階でも、Tcl コマンドを使用してデザイン チェックポイントを保存し、レポートを生成できます。また、各デザイン段階で Vivado IDE を開いてデザインの解析および制約の設定を実行できます。

```
set outputDir ./Tutorial_Created_Data/bft_output  
file mkdir $outputDir  
set_part xc7k70tfg484-2  
# STEP#1: setup design sources and constraints  
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]  
read_vhdl ./Sources/hdl/bft.vhdl  
read_verilog [ glob ./Sources/hdl/*.v ]  
read_xdc ./Sources/bft_full.xdc  
# STEP#2: run synthesis, report utilization and timing estimates, write checkpoint design  
synth_design -top bft  
write_checkpoint -force $outputDir/post_synth  
report_utilization -file $outputDir/post_synth_util.rpt  
report_timing -sort_by group -max_paths 5 -path_type summary \  
-file $outputDir/post_synth_timing.rpt  
# STEP#3: run placement and logic optimization, report utilization and timing estimates  
opt_design  
power_opt_design  
place_design  
phys_opt_design  
write_checkpoint -force $outputDir/post_place  
report_clock_utilization -file $outputDir/clock_util.rpt  
report_utilization -file $outputDir/post_place_util.rpt  
report_timing -sort_by group -max_paths 5 -path_type summary \  
-file $outputDir/post_place_timing.rpt  
# STEP#4: run router, report actual utilization and timing, write checkpoint design, run DRCs  
route_design  
write_checkpoint -force $outputDir/post_route  
report_timing_summary -file $outputDir/post_route_timing_summary.rpt  
report_utilization -file $outputDir/post_route_util.rpt  
report_power -file $outputDir/post_route_power.rpt  
report_methodology -file $outputDir/post_imp_checks.rpt  
report_drc -file $outputDir/post_imp_drc.rpt  
write_verilog -force $outputDir/bft_impl_netlist.v  
write_xdc -no_fixed_only -force $outputDir/bft_impl.xdc  
# STEP#5: generate a bitstream  
write_bitstream $outputDir/design.bit
```

プロジェクト モード

このモードでは、ディスク上にディレクトリ構造が作成され、このディレクトリでデザイン ソース ファイル、run の結果、およびプロジェクト ステータスが管理されます。run インフラストラクチャを使用して、合成およびインプリメンテーション プロセスおよび run ステータスが自動的に管理されます。

```
create_project project_bft ./project_bft -part xc7k70tfg484-2  
add_files {./Sources/hdl/FifoBuffer.v ./Sources/hdl/async_fifo.v ./Sources/hdl/bft.vhdl}  
add_files [ glob ./Sources/hdl/bftLib/*.vhdl ]  
set_property library bftLib [get_files [ glob ./Sources/hdl/bftLib/*.vhdl]]  
import_files -force -norecuse  
import_files -fileset constrs_1 ./Sources/bft_full.xdc  
set_property steps.synth_design.args.flatten_hierarchy full [get_runs synth_1]  
launch_runs synth_1  
wait_on_run synth_1  
launch_runs impl_1  
wait_on_run impl_1  
launch_runs impl_1 -to_step write_bitstream
```

Vivado Design Suite クイック リファレンス

タイミング制約

create_clock：現在のデザインに含まれる物理または仮想クロック オブジェクトを作成します。

create_clock -period <arg> [-name <arg>] [-waveform <args>] [-add] [<objects>]

> create_clock -period 10 -name sysClk [get_ports sysClk]

set_input_delay / set_output_delay：I/O ポートの入力または出力遅延を設定します。

```
set_input/output_delay [-clock <args>] [-reference_pin <args>] [-clock_fall] [-rise] [-fall] [-max] [-min] [-add_delay] [-network_latency_included] [-source_latency_included] <delay> <objects>
```

> set_input_delay -clock sysClk 3.0 [get_ports DataIn_pad_0_i[*]]

set_false_path：フォールス タイミング パスを定義します。

```
set_false_path [-setup] [-hold] [-rise] [-fall] [-reset_path] [-from <args>] [-rise_from <args>] [-fall_from <args>] [-to <args>] [-rise_to <args>] [-fall_to <args>] [-through <args>] [-rise_through <args>] [-fall_through <args>]
```

> set_false_path -from [get_ports GTPRESET_IN]

set_max_delay / set_min_delay：タイミング パスの最大または最小遅延を指定します。

```
set_max/min_delay [-rise] [-fall] [-reset_path] [-from <args>] [-rise_from <args>] [-fall_from <args>] [-to <args>] [-rise_to <args>] [-fall_to <args>] [-through <args>] [-rise_through <args>] [-fall_through <args>] [-datapath_only] <delay>
```

注記：datapath_only は set_max_delay でのみ有効です。

> set_max_delay -through s3_err_i 3.0

set_multicycle_path：マルチサイクル パスを定義します。

```
set_multicycle_path [-setup] [-hold] [-rise] [-fall] [-start] [-end] [-reset_path] [-from <args>] [-rise_from <args>] [-fall_from <args>] [-to <args>] [-rise_to <args>] [-fall_to <args>] [-through <args>] [-rise_through <args>] [-fall_through <args>] [-<path_multiplier>]
```

注記：XDC では、マルチサイクル パスにセットアップ およびホールドを指定できます。

```
> set_multicycle_path -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*] 2  
> set_multicycle_path -hold -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*] 1
```

create_generated_clock：既存のクロックからクロック オブジェクトを生成します。

```
create_generated_clock [-name <arg>] -source <args> [-edges <args>] [-divide_by <arg>] [-multiply_by <arg>] [-combinational] [-duty_cycle <arg>] [-invert] [-edge_shift <args>] [-add] [-master_clock <arg>] <objects>
```

set_clock_groups：排他的または非同期のクロック グループを設定します。

```
set_clock_groups [-name <arg>] [-logically_exclusive] [-physically_exclusive] [-asynchronous] [-group <args>]
```

優先順位

タイミング例外：

```
set_false_path / set_clock_groups  
set_max_delay / set_min_delay  
set_multicycle_path  
normal setup/hold check
```

フィルター：

```
-from pin / -to pin  
-from pin  
-to pin  
-through pin  
-from clock  
-to clock
```

