Vivado Design Suite Quick Reference

Getting Help

For the most up-to-date information on every command available in the Vivado® Design Suite use the built-in help system. At the Tcl prompt type: `help`

This will list all the categories of commands available in the Vivado tool. You can then query the specific category of interest for a list of the commands in that category. For example, to see the list of XDC commands you would type: `help -category XDC`

See the *Vivado Design Suite Tcl Command Reference Guide* (UG835) for more information.
For Vivado tools video tutorials refer to: http://www.xilinx.com/training/vivado/index.htm

Simulation Commands

The Vivado simulator is an event-driven Hardware Description Language (HDL) simulator for functional and timing simulations of VHDL, Verilog, and mixed VHDL/Verilog designs.

Command	Purpose	Example	Output
xvlog **xvhdl**	Compile Verilog and VHDL files	xvlog file1.v file2.v xvhdl f1.vhd f2.vhd	Parsed dump into HDL library on disk.
xelab	Compile and Elaborate	xelab work.top1 work.top2 -s cpusim	Creates a snapshot.
xsim	Run simulation on the executable snapshot	xsim *<options> <exe>*	The xsim command loads a simulation snapshot to perform a batch mode simulation, or run simulation interactively in a GUI and/or a Tcl-based environment.

To create an example simulation script from Vivado IDE: `launch_xsim -scripts_only`

For details, refer to the *Vivado Design Suite User Guide: Logic Simulation (UG900)*

Vivado Hardware Manager

The Hardware Manager lets you interact with debug cores that are implemented on Xilinx FPGA devices. Tcl commands used to access features of the Hardware Manager include:

- **open_hw** - Opens the Hardware Manager in the Vivado Design Suite.
- **connect_hw_server** - Makes a connection to a local or remote hardware server application.
- **open_hw_target** - Opens a connection to the hardware target.
- **current_hw_device** – Sets or returns the Xilinx FPGA device to program and debug.
- **get_hw_ilas** – Get the Integrated Logic Analyzer debug core objects that are used to monitor signals in the design, trigger on hardware events, and capture system data in real-time. Use any of the `*_hw_ila*` TCL commands to interact with the ILA core.
- **get_hw_vios** - Get the Virtual I/O debug core objects that are used to drive control signals and/or monitor design status signals.
- **get_hw_axis** - Get the AXI debug core objects that are used to generate AXI transactions to interact with various AXI full and AXI lite slave cores in a system running on Xilinx FPGA devices.
- **get_hw_sio_iberts** - Get the SIO debug cores that are used to measure and optimize high-speed serial I/O transmit/receive settings, and measure transmission bit error rates.

For more information, refer to the *Vivado Design Suite User Guide: Programming and Debugging (UG908)*.

Vivado Design Suite Quick Reference

Vivado IDE Launch Modes

When launching Vivado from the command line there are three modes:

1. **GUI Mode** – The default mode. Launches the Vivado IDE for interactive design.
   Usage: `vivado` OR `vivado -mode gui`

2. **Tcl Shell Mode** - Launches the Vivado Design Suite Tcl shell for interactive design.
   Usage: `vivado -mode tcl`
   Note: Use the `start_gui` and `stop_gui` Tcl commands to open and close the Vivado IDE from the Tcl shell.

3. **Batch Process Mode** - Launches the Tcl shell, runs a Tcl script, and exits the tool.
   Usage: `vivado -mode batch -source <file.tcl>`

Vivado Command Options:

-mode	Invocation mode: gui, tcl, and batch. Default: gui
-init	Source vivado.tcl file during initialization.
-source	Source the specified Tcl file. Required for batch mode.
-nojournal	Do not write a journal file.
-appjournal	Append to the journal file instead of overwriting it.
-journal	Journal file name. The default is vivado.jou.
-nolog	Do not write a log file.
-applog	Append to the log file instead of overwriting it.
-log	Log file name. The default is vivado.log.
-ise	ISE Project Navigator integration mode.
-m32	Run 32-bit binary version
-m64	Run 64-bit binary version
-version	Output version information and exit
-tclargs	Arguments passed on to Tcl argc argv
-tempDir	Temporary directory name
-verbose	Suspend message limits during command execution
<project>]	Load specified project file (.xpr) or Design Check Point (.dcp)

Main Reporting Commands

The Vivado Design Suite includes many reporting commands which provide different levels of information as the design progresses through the design flow:

Category	Purpose	Examples
Timing	Design goals	report_timing check_timing report_clocks report_clock_interaction
Performance	Measurement against design goals	report_timing report_timing_summary report_power report_min_pulse_width
Resource Utilization	Logical to physical resource mapping	report_utilization report_clock_util report_io report_control_sets report_ram_configuration
Design Rule Checks	Physical verification	report_drc report_ssn report_sso
Design Data	Project-specific settings	report_param report_config_timing report_ip_status

Vivado Design Suite Quick Reference

Design Object Query Commands

Command	Description
get_cells	Get logic cell objects based on name/hierarchy or connectivity
get_pins	Get pin objects based name/hierarchy or connectivity
get_nets	Get net objects by name/hierarchy or connectivity
get_ports	Get top-level netlist ports by name or connectivity
all_inputs	Return all input ports in the current design
all_outputs	Return all output ports in the current design
all_ffs	Return all flip flops in current design
all_latches	Return all latches in current design
all_dsps	Return all DSP cells in the current design
all_rams	Return all ram cells in the current design

Timing-based Query Commands

Command	Description
all_clocks	Get a list of all defined clocks in the current design
get_clocks	Get clock objects by name or object traversed
get_generated_clocks	Get generated Clock objects
all_fanin	Get list of pins or cells in fanin of specified object in timing path
all_fanout	Get list of pins or cells in fanout of specified object in timing path
all_registers	Get list of all sequential / latched cells in the current design
get_path_groups	Path group objects
get_timing_paths	Timing path objects, equivalent to report_timing printout

Filtering

All get_* commands provide a -filter option. There is also an independent filter command. Filtering provides a mechanism to reduce lists of returned objects based on the object properties. For example, to filter on a LIB_CELL type you could do the following:

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

You can combine multiple filters together:

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

You can filter directly on Boolean properties:

> get_cells -filter {IS_PRIMITIVE && !IS_LOC_FIXED}

Valid operations are: ==, !=, =~, !~, <=, >=, >, < as well as && and || between filter patterns

XILINX ALL PROGRAMMABLE™

Vivado Design Suite Quick Reference

Tcl Examples

Tcl examples:

An iterative loop to display direction and IO standard for all ports:

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

A simple procedure to give the short version of help on a command:

```
proc short cmdName {
   help -short $cmdName
   }
```

Usage: short <command_name>

A procedure to return an array of unique prim/pinname count based on input pin object list:

```
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]
}
```

Usage: countPrimPin <pinObjs>

A procedure to return the primitive/libpinname of the pin:

```
proc getPrimPinName {pin} {
   set pinname  [regsub {.*/([^/]*)$} [get_property name $pin] {\1}]
   set primname [get_property LIB_CELL [get_cells -of $pin]]
   return "$primname/$pinname"
}
```

Usage: getPrimPinName pin

A procedure to return the number of characters of the longest string in a list:

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

Usage: returnMaxStringLength list

See the *Vivado Design Suite User Guide: Using Tcl Scripting (UG894)* for more information.

XILINX ALL PROGRAMMABLE

Vivado Design Suite Quick Reference

Batch Mode Script Examples

Example scripts for both Project Mode and Non-Project Mode, using the BFT example design.

Non-Project Mode:

When working in Non-Project Mode, sources are accessed from their current locations and the design is compiled in memory. You are viewing the active design in memory, so changes are automatically passed forward in the design flow. You can save design checkpoints and create reports at any stage of the design process using Tcl commands. In addition, you can open the Vivado IDE at each design stage for design analysis and constraints assignment.

```
set outputDir ./Tutorial_Created_Data/bft_output
file mkdir $outputDir
# 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 -part xc7k70tfbg484-2
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_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
```

Project Mode:

When working in Project Mode, a directory structure is created on disk in order to manage design source files, run results, and track project status. A runs infrastructure is used to manage the automated synthesis and implementation process and to track run status.

```
create_project project_bft ./project_bft -part xc7k70tfbg484-2
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 -norecurse
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 Quick Reference

Timing Constraints

create_clock: Create a physical or virtual clock object in the current design.

```
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: Set input or output delay on I/O ports.

```
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: Define a false timing path.

```
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: Specify maximum or minimum delay for timing paths.

```
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>
```

NOTE: datapath_only is only valid for set_max_delay

```
>set_max_delay -through s3_err_i 3.0
```

---

set_multicycle_path: Define multicycle path.

```
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>]
```

NOTE: In XDC, unlike UCF, you can specify setup and hold for multicycle paths.

```
> 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: Create a generated (derived) clock object.

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

---

set_clock_groups: Set exclusive or asynchronous clock groups.

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

Order of Precedence - Timing Exceptions:

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

Highest ↓ Lowest

Filter Matching:
-from pin / -to pin
-from pin
-to pin
-through pin
-from clock
-to clock