

## 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 behavioral, functional, and timing simulations of VHDL, Verilog, SystemVerilog, and mixed-language designs.

Command	Purpose	Example	Output
<b>xvlog</b> <b>xvhdl</b>	Compile Verilog and VHDL files	xvlog file1.v file2.v xvhdl f1.vhd f2.vhd	Parsed dump into HDL library on disk.
<b>xelab</b>	Compile and Elaborate	xelab work.top1 work.top2 -s cplusim	Creates a snapshot.
<b>xsim</b>	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 a simulation script from Vivado IDE: `launch_simulation -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.  
*Usage:* `vivado` OR `vivado -mode gui`
- 2. Tcl Shell Mode** - Launches the Vivado Design Suite Tcl shell.  
*Usage:* `vivado -mode tcl`  
*Note:* Use `start_gui` and `stop_gui` Tcl commands to open and close the Vivado IDE from the Tcl shell.
- 3. Batch Mode** - Launches the Tcl shell, runs a Tcl script, and then exits the tool.  
*Usage:* `vivado -mode batch -source <file.tcl>`

#### Vivado Command Options:

<code>-mode</code>	Invocation mode: gui, tcl, or batch. Default: gui
<code>-init</code>	Source vivado.tcl file during initialization.
<code>-source</code>	Source the specified Tcl file. Required for batch mode.
<code>-nojournal</code>	Do not write a journal file.
<code>-appjournal</code>	Append to the journal file instead of overwriting it.
<code>-journal</code>	Journal file name. The default is vivado.jou.
<code>-nolog</code>	Do not write a log file.
<code>-applog</code>	Append to the log file instead of overwriting it.
<code>-log</code>	Log file name. The default is vivado.log.
<code>-version</code>	Output version information and exit
<code>-tclargs</code>	Arguments passed on to Tcl argc argv
<code>-tempDir</code>	Temporary directory name
<code>-verbose</code>	Suspend message limits during command execution
<code>&lt;project   dcp&gt;</code>	Load the 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	check_timing report_timing report_clocks report_clock_interaction report_cdc report_synchronizer_mtbf
Performance	Measurement against design goals	report_timing_summary report_datasheet report_design_analysis report_power report_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
<code>get_cells</code>	Get logic cell objects based on name/hierarchy or connectivity
<code>get_pins</code>	Get pin objects based name/hierarchy or connectivity
<code>get_nets</code>	Get net objects by name/hierarchy or connectivity
<code>get_ports</code>	Get top-level netlist ports by name or connectivity
<code>all_inputs</code>	Return all input ports in the current design
<code>all_outputs</code>	Return all output ports in the current design
<code>all_ffs</code>	Return all flip flops in current design
<code>all_latches</code>	Return all latches in current design
<code>all_dsps</code>	Return all DSP cells in the current design
<code>all_rams</code>	Return all ram cells in the current design

### Timing-based Query Commands

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

## Vivado Design Suite Quick Reference

### 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 procedure that displays a list of commands supporting the specified option or argument:

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

Usage: findCmd <option>

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 specified pin:

```
proc getPrimPinName {pin} {  
    set pinname [regsub {"/*"} $pin {}]  
    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.

## 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  
set_part xc7k70tfg484-2  
# STEP#1: setup design sources and constraints  
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhd ]  
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_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 xc7k70tfg484-2  
add_files {./Sources/hdl/FifoBuffer.v ./Sources/hdl/async_fifo.v ./Sources/hdl/bft.vhdl}  
add_files [ glob ./Sources/hdl/bftLib/*.vhd ]  
set_property library bftLib [get_files [ glob ./Sources/hdl/bftLib/*.vhd ]]  
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\_multipler>]

**NOTE:** In XDC 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>] [-invert] [-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] [-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