ISE to Vivado Design Suite Migration Guide

UG911 (v2020.2) November 18, 2020

XILINX

Revision History

11/18/2020: Released with Vivado® Design Suite 2020.2 without changes from 2019.2.

Section	Revision Summary
10/30/2019 Version 2019.2	
OUT_TERM	Updated to show this constraint is not supported in XDC.
Association of ELF Files	Updated reference to the Vitis™ software development platform.

Table of Contents

XILINX

Chapter 1

Introduction to ISE Design Suite Migration

Overview

The ISE® Design Suite is an industry-proven solution for all generations of Xilinx® devices, and extends the familiar design flow for projects targeting 7 series devices and Zynq®-7000 SoCs.

The Vivado® Design Suite supports the 7 series devices (including Virtex®-7, Kintex®-7, Artix®-7), UltraScale™ architectures, Zynq-7000 SoCs, and the UltraFast™ methodology and offers enhanced tool performance, especially on large or congested designs.

Because both the ISE Design Suite and the Vivado Design Suite support 7 series devices, you have time to migrate to the Vivado Design Suite.

The Vivado Design Suite provides reuse of all or part of a previous design through the ability to import projects and sources into a Vivado Design Suite project, and command mapping to Tcl scripts.

See the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4] for more information about the Tcl commands and syntax.

On the Tcl Console command line in the Vivado Integrated Design Environment (IDE), you can also type the following for information about any Tcl command:

```
<command_name> -help
```

Design Flows

You can launch the Vivado Design Suite and run the tools using different methods depending on your preference. For example, you can choose a Tcl script-based compilation style method in which you manage sources and the design process yourself, also known as *Non-Project Mode*.

Alternatively, you can use a project-based method to automatically manage your design process and design data using projects and project states, also known as *Project Mode*. Either of these methods can be run using a Tcl scripted batch mode or run interactively in the Vivado IDE. For more information on the different design flow modes, see the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1].

This guide covers migration considerations and procedures for both design flow modes in the Vivado Design Suite.

XILINX

Chapter 2

Migrating ISE Design Suite Designs to Vivado Design Suite

Overview

This chapter describes how to migrate ISE® Design Suite designs to the Vivado® Design Suite.

The Vivado Design Suite offers the same level of retargeting as the ISE Design Suite for Virtex-class devices. For Spartan-class devices, some manual migration is required.

Importing an XISE Project Navigator Project

You can use the Vivado Integrated Design Environment (IDE), which is the GUI, to import an `XISE` project file, as follows:

1. Select **File > Project > New**.
2. Select Project name and location.
3. In the New Project Wizard, select the **Imported Project** option.
4. Select **ISE** and choose the appropriate XISE file for import.

IMPORTANT: *Vivado Design Suite does not support older ISE Design Suite projects (`.ise`) files.*

After you have imported the project file:

- Review the `Import XISE Summary` report for important information about the imported project.
- Ensure the selected device meets your requirements, and if it does not, select a new device. If the ISE project does not have an equivalent supported device in Vivado, a default device is selected.
- Review the files in the Sources window to ensure all design files were imported properly.

If the design contains a User Constraints File (UCF), the file appears as an unsupported constraint file.

IMPORTANT: *The UCF must be converted to Xilinx® Design Constraints (XDC) format to apply any timing or physical constraints in the design. For more information, see Chapter 3, Migrating UCF Constraints to XDC.*

For more details on using the Vivado Design Suite interface for design creation, see the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 2].

For information on the next steps in design flow, see the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1].

For information on constraints, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 3].

for information on Tcl commands, see *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

For information on design properties, see *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 13].

Converting a PlanAhead Tool Project

To convert a PlanAhead™ tool project to a Vivado IDE project, open the PlanAhead project file (PPR extension) in the Vivado IDE. When prompted, type in a new project name and location for the converted project.

The project conversion effects the following changes:

- Targets the Vivado Design Suite default 7 series device on projects with pre-7 series devices.
- Resets all runs. They are generated after implementing the design in the tool.
- Replaces run strategies with Vivado Design Suite default strategies.
- Moves UCF files to an Unsupported Constraints Folder because UCF constraints are not supported.

Note: Conversion of designs with Partitions is not supported.

Note: Zynq®-7000 SoC processor designs using XPS are not supported.

IMPORTANT: *The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for new embedded processor designs, including designs targeting Zynq devices and MicroBlaze™ processors. XPS is no longer integrated with the Vivado Design Suite. However, DCP and NGC files created without constraints in XPS are supported as source files in the Vivado Design Suite.*

Importing an XST Project File

If you do not have an existing or up-to-date ISE® Project Navigator (`.xise`) project file or PlanAhead tool (`.ppr`) project file, you can use the Xilinx Synthesis Technology (XST) project file to import initial settings for a Vivado Design Suite project. To import an XST project file:

1. Select **File > Project > New**.
2. Select **Project name and location**.
3. In the New Project Wizard, select **Imported Project**.
4. Select **XST**, and select the appropriate project file with a `.xst` extension.

After you have imported the project file:

- Review the Import XST Summary Report for important information about the imported project.
- Ensure the selected device meets your requirements, and if it does not, select a new device. If the XST project does not have an equivalent supported device in the Vivado Design Suite, a default device is selected.
- Review the files in the Sources view to ensure all design files were imported properly.

For information on the next steps in design flow, see the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 1].

Migrating Source Files

When you import a project or convert a project into the Vivado IDE, as specified, you can also add all source files that are supported in Vivado Design Suite to the project.

- **IP**: You can migrate existing ISE Design Suite projects and IP to Vivado Design Suite projects and IP. The Vivado Design Suite can use ISE Design Suite IP during implementation. For more information, see Chapter 4, Migrating Designs with Legacy IP to the Vivado Design Suite.
- **Source files**: You can add all source files, with the exception of Schematic (SCH) and Architecture Wizard (XAW) source files, from an existing ISE Design Suite project to new project in the Vivado Design Suite. For example, you can add CORE Generator™ tool project files (`.xco` file extension) and netlist files (`.ngc` file extension) as design sources.

  Note: The Vivado tools do not support the use of hierarchical NGC files.

For UltraScale™ devices and later architectures, NGC format netlists are no longer supported. If your existing design contains NGC netlists, you must convert them to Electronic Data Interchange Format (EDIF) before targeting an UltraScale device. To convert an NGC netlist to EDIF, open Vivado Design Suite and type the following in the Tcl console:

```
exec ngc2edif <name>.ngc <name>.edf
```

Replace `<name>` with the name of the NGC netlist. When this command completes successfully, you can add the resulting EDIF file to the project in place of the NGC file.

- **Constraints**: User Constraint Format (UCF) files used for the design or IP must be converted to Xilinx Design Constraints (XDC) format for use with Vivado Design Suite. For more information about UCF to XDC migration, see Chapter 3, Migrating UCF Constraints to XDC in this guide.

CAUTION! *Do not migrate from ISE® Design Suite and Vivado Design Suite in the middle of a current ISE Design Suite project because design constraints and scripts are not compatible between the environments.*

Mapping ISE Design Suite Command Scripts

This section covers the Vivado Design Suite non-project design flow mode only, and is intended for those who plan to use Tcl scripting with the Vivado tools.

You can use Tcl scripts to migrate your ISE Design Suite scripts to implement your design. Similar to the ISE Design Suite, the compilation flow in the Vivado Design Suite translates the design, maps the translated design to device-specific elements, optimizes the design, places and routes the design, and then generates a BIT file for programming.

Table 2-1 shows the main differences between the two design flows.

Table 2-1: **ISE Design Suite versus Vivado Design Suite Design Flow**

ISE Design Suite	Vivado Design Suite
Separate command line applications	Tcl commands in a shell.
XCF/NCF/UCF/PCF constraints	XDC timing and physical constraints
Design constraints (timing or physical) only applied at beginning of the flow	Constraints (timing or physical) can be applied, changed, or removed at any point in the flow.
Multiple database files (NGC, NGD, NCD) required	A single design database (checkpoint with a `.dcp` extension) written out on-demand at any point in the flow.
Reports generated by applications	Reports generated on-demand at any point in the flow, where applicable.

The table below shows the mapping of ISE Design Suite commands to corresponding Vivado Design Suite Tcl commands. You can run the Tcl commands using any of the following:

- In the Vivado IDE Tcl Console
- At the Tcl prompt (`vivado -mode tcl`)
- Through a batch script (`vivado -mode batch -source my.tcl`)

Table 2-2: **ISE Design Suite Commands and Vivado Design Suite Tcl Commands**

ISE Design Suite Command	Vivado Design Suite Tcl Command

`xst`	`read_verilog`
	`read_vhdl`
	`read_xdc`
	`synth_design`
	Note: The commands must be run in this order.

`ngdbuild`	`read_edif`
	`read_xdc`
	`link_design`
	Note: You need these commands if you are importing from third-party synthesis. If using `synth_design`, omit these steps.
	Note: If you are using the Non-Project flow, you need to use the `read_edif` command to include the NGC file.

`map`	`opt_design`
	`power_opt_design` (optional)
	`place_design`
	`phys_opt_design` (optional)

`par`	`route_design`
`trce`	`report_timing` `report_timing_summary`
`xpwr`	`read_saif` `report_power`
`drc`	`report_drc`

`netgen`	`write_verilog`
	`write_vhdl`
	`write_sdf`

`bitgen`	`write_bitstream`
`xinfo`	`report_environment`

The ISE Design Suite and the Vivado Design Suite use different algorithms; consequently, a one-to-one mapping is not always possible between the two tool flows. Table 2-3 provides a mapping of frequently-used options between the two implementation flows.

Table 2-3: **ISE to Vivado Implementation Flow Mappings**

ISE Design Suite	Vivado Design Suite
`ngdbuild -p partname`	`link_design -part`
`ngdbuild -a` (insert pads)	`synth_design -no_iobuf` (opposite)
`ngdbuild -u` (unexpanded blocks)	Allowed by default, generates critical warnings.
`ngdbuild -quiet`	`link_design -quiet`
`map -detail`	`opt_design -verbose`
`map -lc auto`	Feature is no longer needed. Vivado automatically uses less area in quest for better routability and QOR.
`map -logic_opt`	`opt_design, phys_opt_design`
`map -mt`	`place_design` automatically runs MT with four cores in Linux or two cores in Windows.
`map -ntd`	`place_design -non_timing_driven`
`map -ol`	`place_design -directive *`
`map -power`	`power_opt_design`
`map -u`	`link_design -mode out_of_context, opt_design -retarget` (skip constant propagation and sweep)
`par -pl`	`place_design -directive *`
`par -rl`	`route_design -directive *`
`par -mt`	`route_design` automatically runs MT with four cores in Linux or two core in Windows
`par -k` (keep existing placement and routing)	Default behavior for `route_design`
`par -nopad`	`report_io` generates pad report
`par -ntd`	`route_design -no_timing_driven`

* See the Tcl help for more information on the `-directive` option.

Retrieving Tcl Command Information

For information about additional Tcl commands available to use for design implementation and analysis, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835). To get help at the Tcl command prompt, type:

- `help <command>`
- `<command> -help`

For help with a category of Tcl commands, type:

- `help` (lists the categories)
- `help -category <category>`

Note: The interactive help has an auto-complete feature and is not case-sensitive, so the following categories are the same in Tcl: `end`, `EndGroup`. To save time, you can use all lowercase and auto-complete when getting help on commands or categories (for example, type `end` rather than `endgroup`).

Command Line Examples

Following is an example of a typical ISE Design Suite command line run that can be put in a `run.cmd` file. This is followed by the same run using Vivado Design Suite Tcl commands that can be put into a `run.tcl` file.

Example 1: Mapping ISE Design Suite Commands to Vivado Design Suite Tcl Commands

ISE Design Suite Command Line

```
xst -ifn design_top.xst

#-ifn (input file name with project settings and options)
ngdbuild -sd .. -dd . -p xc7v585tffg1157-2 -uc design_top.ucf design_top.ngd

#-sd (search directory), -dd (destination directory), -p (part), -uc (UCF
#file)
map -xe c -w -pr b -ol high -t 2 design_top_map.ncd design_top.pcf
#-xe c (extra effort), -w (overwrite existing file), -pr b (push registers
#into IOBs), -ol (overall effort), -t (placer cost table)
par -xe c -w -ol high -t 2 design_top_map.ncd design_top.ncd
#-xe c (extra effort), -w (overwrite existing file), -ol (overall effort), -t
#(placer cost table)
trce -u -e 10 design_top.ncd design_top.pcf
#-u (report uncovered paths), -e (generate error report)
bitgen -w design_top.ncd design_top.pcf
```

Similar Vivado Design Suite Tcl Command

```
set design_name design_top
#read inputs
read_verilog { $design_name.v source2.v source3.v }
read_vhdl -lib mylib { libsource1.vhdl libsource2.vhdl }
read_xdc $design_name.xdc
#run flow and save the database
synth_design -top $design_name -part xc7v585tffg1157-21
write_checkpoint -force ${design_name}_post_synth.dcp
opt_design
place_design
write_checkpoint -force ${design_name}_post_place.dcp
report_utilization -file post_place_util.txt
route_design
#Reports are not generated by default
report_timing_summary -file post_route_timing.txt
#Save the database after post route
write_checkpoint -force ${design_name}_post_route.dcp
#Check for DRC
report_drc -file post_route_drc.txt
# Write Bitstream
write_bitstream -force ${design_name}.bit
```

Example 2: Vivado Design Suite Tcl Commands for Third-Party Synthesis (starting from EDIF)

```
set design_name design_top
#read inputs
read_edif { source1.edf source2.edf $design_name.edf }
read_xdc $design_name.xdc
link_design -part xc7v585tffg1157-2 -top $design_name
#Reports are not generated by default
report_timing_summary -file post_synth_timing_summ.txt
opt_design
place_design
write_checkpoint -force ${design_name}_post_place.dcp
report_utilization -file post_place_util.txt
route_design
#Reports are not generated by default
report_timing_summary -file post_route_timing.txt
#Save the database after post route
write_checkpoint -force ${design_name}_post_route.dcp
#Check for DRC
```

```
report_drc -file post_route_drc.txt
# Write Bitstream
write_bitstream -force ${design_name}.bit
```

Mapping Makefiles

A makefile is a text file that is referenced by the make command and controls how the make command compiles and links a program. The makefile consists of rules for when to carry out an action and contains information such as source-level and build-order dependencies. To determine the sequence of the compilation commands, the makefile checks the timestamps of dependent files. The following example shows how to write makefiles.

Example: Mapping an ISE Design Suite Makefile to a Vivado Design Suite Makefile

Sample Makefile used in the ISE Design Suite

```
DESIGN = test
DEVICE = xc7v585tffg1157-2
UCF_FILE = ../Src/${DESIGN}.ucf
EDIF_FILE = ../Src/${DESIGN}.edf

# Make all runs to place & route
all : place_n_route

# bitstream : Creates device bitstream
bitstream : ./${DESIGN}.bit

# place_n_route: Stops after place and route for analysis prior to bitstream
generation
place_n_route : ${DESIGN}.ncd

# translate: Stops after full design elaboration for analysis and floorplanning prior
to place and route step
translate : ${DESIGN}.ngd

# Following executes the ISE run

${DESIGN}.bit : ${DESIGN}.ncd
bitgen -f ${DESIGN}.ut ${DESIGN}.ncd

${DESIGN}.ncd : ${DESIGN}_map.ncd
```

```
par -w -ol high ${DESIGN}_map.ncd ${DESIGN}.ncd ${DESIGN}.pcf

${DESIGN}_map.ncd : ${DESIGN}.ngd
map -w -ol high -o ${DESIGN}_map.ncd ${DESIGN}.ngd ${DESIGN}.pcf

${DESIGN}.ngd : ${EDIF_FILE} ${UCF_FILE}
ngdbuild -uc ${UCF_FILE} -p ${DEVICE} ${EDIF_FILE} ${DESIGN}.ngd

# Clean up all the files from the Vivado run
clean :
rm -rf *.ncd *.ngd *.bit *.mrp *.map *.par *.bld *.pcf *.xml *.bgn *.html \
           *.lst *.ngo *.xrpt *.unroutes *.xpi *.txt *.pad *.csv *.ngm xlnx_auto* \
            _xmsgs *.ptwx

# Tar and compress all the files
tar :
tar -zcvf ${DESIGN}.tar.gz *.ncd *.ngd *.mrp *.map *.par *.bld *.pcf *.bgn \
               Makefile
```

Equivalent Makefile Used in the Vivado Design Suite

```
DESIGN = test
DEVICE = xc7v585tffg1157-2
XDC_FILE = ../Src/${DESIGN}.xdc
EDIF_FILE = ../Src/${DESIGN}.edf

# Make all runs to place & route
all : place_n_route

# bitstream : Creates device bitstream
bitstream : ./${DESIGN}.bit

# place_n_route: Stops after place and route for analysis prior to bitstream
generation
place_n_route : ./${DESIGN}_route.dcp

# translate: Stops after full design elaboration and initial optimization for
analysis and floorplanning prior to place and route step
translate : ./${DESIGN}_opt.dcp

# Following calls Tcl files for each desired portion of the Vivado run
# Design checkpoint files and bit file used for dependency management

./${DESIGN}.bit : ./run_vivado_place_n_route.tcl ./${DESIGN}_route.dcp
```

```
vivado -mode batch -source run_vivado_bitstream.tcl -tclargs ${DESIGN}

./${DESIGN}_route.dcp : ./run_vivado_place_n_route.tcl ./${DESIGN}_opt.dcp
vivado -mode batch -source run_vivado_place_n_route.tcl -tclargs \
            ${DESIGN}

./${DESIGN}_opt.dcp : ./run_vivado_opt.tcl ${EDIF_FILE} ${XDC_FILE}
vivado -mode batch -source run_vivado_opt.tcl -tclargs ${DESIGN} ${DEVICE}
${EDIF_FILE} ${XDC_FILE}

# Clean up all the files from the Vivado run
clean :
rm -f *.jou *.log *.rpt *.dcp *.bit *.xml *.html

# Tar and compress all the files
tar :
tar -zcvf ${DESIGN}.tar.gz *.jou *.log *.rpt *.dcp *.tcl Makefile
```

Associated Tcl Files for the Vivado Design Suite Makefile

run_vivado_opt.tcl

```
# Gathering TCL Args
set DESIGN [lindex $argv 0]
set DEVICE [lindex $argv 1]
set EDIF_FILE [lindex $argv 2]
set XDC_FILE [lindex $argv 3]

# Reading EDIF/NGC file
read_edif ../Src/${DESIGN}.edf

# Linking Design
link_design -part ${DEVICE} -edif_top_file ../Src/${DESIGN}.edf

# Running Logic Optimization
opt_design

# Adding Constraints
read_xdc ${XDC_FILE}

# Saving Run
write_checkpoint -force ./${DESIGN}_opt.dcp

# Creating opt reports
report_utilization -file ${DESIGN}_utilization_opt.rpt
```

```
report_timing_summary -max_paths 10 -nworst 1 -input_pins
report_io -file ${DESIGN}_io_opt.rpt
report_clock_interaction -file ${DESIGN}_clock_interaction_opt.rpt

exit
```

run_vivado_place_n_route.tcl

```
# Gathering TCL Arg
set DESIGN [lindex $argv 0]

read_checkpoint ./${DESIGN}_opt.dcp
link_design
# Placing Design
place_design
write_checkpoint -force ./${DESIGN}_place.dcp

# Routing Design
route_design

# Saving Run
write_checkpoint -force ./${DESIGN}_route.dcp

# Creating route reports
report_timing_summary -max_paths 10 -nworst 1 -input_pins
report_drc -file ${DESIGN}_drc_route.rpt

exit
```

run_vivado_bitstream.tcl

```
# Gathering TCL Arg
set DESIGN [lindex $argv 0]

read_checkpoint ./${DESIGN}_route.dcp
# Create bitstream
write_bitstream -force ${DESIGN}.bit

exit
```

Note: This flow exits and re-enters Vivado tools for the defined steps in the makefile. While this allows greater run control from the make infrastructure, it is not the most efficient in execution time because you must exit and restart the software, and then reload the design for each defined step. Building this entire run in Tcl could be more efficient in runtime because the design can remain in memory from step to step if that is more desirable than having the makefile control the steps.

Understanding the Differences in Messages

The Vivado Design Suite uses the same ISE Design Suite concept of Info, Warning, and Error with unique identifying numbers for messages (for example, ngdbuild:604). Applications have messages with unique identifying numbers (HDL-189). Additionally, the Vivado Design Suite includes two new message types: Status and Critical Warning.

- Status provides information about the current tool process.
- Critical Warnings in Vivado Design Suite are equivalent to Errors in the ISE Design Suite, except the Vivado design process is not stopped. Critical Warnings in the design are upgraded to Error at the bitstream generation (`write_bitstream`) stage, which stops the design process.

RECOMMENDED: *Resolve any Critical Warnings before moving forward with your design.*

Table 2-4 shows the five message types in the Vivado Design Suite, indicates whether use intervention is required, and describes the message purpose.

Table 2-4: **Vivado Design Suite Message Types**

Type	Intervention?	Purpose
STATUS	No	Provides a general status of the process and feedback regarding design processing. A *STATUS* message is the same as an *INFO* message, excluding the severity and message ID tag.
INFO	No	Provides you with general status of the process and feedback regarding design processing. An *INFO* message is the same as a *STATUS* message but includes a severity and message ID tag for filtering or searching.
WARNING	Optional	Indicates that design results might be sub-optimal because constraints or specifications might not be applied as intended. The processing continues to completion and valid results are generated.
CRITICAL WARNING	Recommended	Indicates an issue that likely results in improperly functioning hardware and result in an *ERROR* later in the flow. The processing continues until an *ERROR* is encountered.
ERROR	Yes	Indicates an issue that renders design results unusable and cannot be resolved without user intervention. Further processing is not possible.

Understanding Reporting Differences

In the ISE Design Suite, reports are automatically generated when each application is run through the design flow. This includes but is not limited to the following:

- `.syr` for xst
- `.bld` for ngdbuild
- `.mrp` for map
- `.par` for par
- `.twr` for trce
- `.pwr` for xpwr

In the Vivado Design Suite, you can generate reports at any design stage. The benefits of on-demand report generation are:

- **Better runtime**: You can better manage runtime by generating reports only when needed.
- **More available reports**: At any point in the design flow, you can generate a report, making more reports available to you. For example, you can generate a utilization report for your design post-synthesis, post-optimization or post-route to see current information.

When you use Project Mode in the Vivado IDE, a fixed set of reports automatically generate that you can access from the Reports window.

When you use Non-Project Mode with Tcl commands or a script, you must add Tcl reporting commands to generate reports at stages where you require them while the design is in memory.

Specific `report_*` commands report different types of information, for example, utilization, timing, and DRC results. By default, the report output is sent to the tool transcript and the `vivado.log` file, but it can also be directed to a file. For a list of reports and their descriptions, type `help -category report` at the Tcl Command prompt.

TIP: *The xinfo command from ISE is replaced by the report_enviroment Tcl command in Vivado.*

Table 2-5 shows the relationship between ISE Design Suite report information and Vivado Design Suite reporting commands.

Table 2-5: **ISE Design Suite Reports and Vivado Design Suite Reports**

ISE Design Suite Information (Report)	Vivado Design Suite Command
Utilization Information (.syr, .mrp, .par)	report_utilization, report_clock_utilization
I/O Information (.pad)	report_io
Timing Information (.par, .twr)	report_timing, report_timing_summary
Power Information (.pwr)	report_power

Understanding Log File Differences

The ISE Design Suite tools generate status and output information as a part of the individual command log files. For example, the output status and progress of the Map executable is placed into the .map file where the output of PAR is placed in the .par file.

The Vivado Design Suite uses a single log file to capture all tool commands and output. The file is named vivado.log by default, which you can change by using the vivado -log option. The Vivado Design Suite log file displays flow progress using phases. Each phase has a name and number and a single-line performance summary. Following is an example:

```
report_timing: Time (s): cpu = 00:03:57 ; elapsed = 00:03:55 . Memory (MB): peak =
6526.066 ; gain = 64.125
```

Where:

- cpu is total run time for all processors.
- elapsed is the actual time spent running the process.
- peak is the maximum memory usage up to that particular design step.
- gain is the incremental contribution of a design step to the peak memory usage. For example, report_timing added 64.125 MB to the peak memory usage.

Migrating UCF Constraints to XDC

Overview

The Vivado® Integrated Design Environment (IDE) does not support the User Constraint File (UCF) constraints used in the ISE® Design Suite.

IMPORTANT: *You must migrate the designs with UCF constraints to the Xilinx® Design Constraint (XDC) format.*

- For information on XDC constraints, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 3].
- For information on UCF constraints, see the *Constraints Guide* (UG625) [Ref 11].
- For information on timing, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 7].

As with UCF, XDC consists of:

- Timing constraints, for which XDC timing constraints are based on the Synopsys Design Constraint (SDC)
- Physical constraints

IMPORTANT: *The migration method described in Converting UCF to XDC in the PlanAhead Tool is good for migrating physical constraints, such as I/Os; timing constraints are often better recreated from scratch.*

Differences Between XDC and UCF Constraints

The fundamental differences between XDC and UCF constraints are:

- XDC is a sequential language, with clear precedence rules.
- UCF constraints are typically applied to nets: XDC constraints are typically applied to pins, ports, and cell objects.
- UCF PERIOD constraints and XDC `create_clock` command are not always equivalent and can lead to different timing results.
- UCF by default does not time between asynchronous clock groups, while in XDC, all clocks are considered related and timed unless otherwise constrained (`set_clock_groups`).
- In XDC, multiple clocks can exist on the same object.

UCF to XDC Mapping

For information, see the following guides:

- *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 3]
- *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4]
- *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 13]

Table 3-1 shows the main mapping between UCF constraints to XDC commands.

Table 3-1: **UCF to XDC Mapping**

UCF	XDC
`TIMESPEC PERIOD`	`create_clock` `create_generated_clock`
`OFFSET = IN <x> BEFORE <clk>`	`set_input_delay`
`OFFSET = OUT <x> BEFORE <clk>`	`set_output_delay`
`FROM:TO "TS_"*2`	`set_multicycle_path`
`FROM:TO`	`set_max_delay`
`TIG`	`set_false_path`
`NET "clk_p" LOC = AD12`	set_property `LOC AD12 [get_ports clk_p]`
`NET "clk_p" IOSTANDARD = LVDS`	set_property `IOSTANDARD LVDS [get_ports clk_p]`

Constraint Sequence

Whether you use one or more XDC files for your design, Xilinx recommends that you organize your constraints in the following sequence:

```
## Timing Assertions Section
# Primary clocks
# Virtual clocks
# Generated clocks
# Clock Groups
# Input and output delay constraints

## Timing Exceptions Section (sorted by precedence)
# False Paths
# Max Delay / Min Delay
# Multicycle Paths
# Case Analysis
# Disable Timing

## Physical Constraints Section
# located anywhere in the file, preferably before or after the timing constraints
# or stored in a separate XDC file
```

Converting UCF to XDC in the PlanAhead Tool

The PlanAhead™ tool assists in converting UCF constraints to XDC when you open an ISE Design Suite or PlanAhead tool project that contains UCF constraints.

When a design is loaded into the database, you use the `write_xdc` command to convert a large percentage of the UCF constraints. You need to manually verify the output file and manually convert some constraints to XDC to ensure that all the design constraints are correct.

The Tcl command `write_xdc` requires that a synthesized netlist be open with one or more UCF files loaded. From the PlanAhead tool, do the following:

1. Open your project that contains UCF constraints.
2. Click **Open Synthesized Design**.
3. In the Tcl Console, type:

   ```
   write_xdc <filename>.xdc
   ```

The `write_xdc` command is not a file converter. The command writes out the constraints that were successfully applied to the design as an XDC file. The output XDC file contains:

- A comment with the file and line number from the UCF for each converted UCF.
- A comment for each conversion not done.

IMPORTANT: *Pay attention to Critical Warnings that indicate which constraints were not successfully converted.*

This conversion is only a starting point for migration to XDC-based constraints.

RECOMMENDED: *Create XDC timing constraints without using the conversion process, because fundamental differences between UCF and XDC make automation less than optimal.*

- Using the PlanAhead tool for converting UCF is best for physical constraints and basic timing constraints. Timing constraints for simple clock definitions and I/O delays typically translate well.

IMPORTANT: *Convert timing exceptions manually. Many do not translate and others can produce sub-optimal results.*

- Fundamental differences between the timer in the Vivado IDE (XDC/SDC) and the timer in ISE Design Suite (UCF) make direct translation impossible. For that reason the UCF constraint must be re-evaluated and a new approach might be required with XDC. Conversion can be done with an elaborated RTL design; however, many objects referenced in a typical UCF do not exist at that stage and thus are not applied to the database.
- Only constraints that are successfully applied to the database can be written out as XDC. Consequently simple clock and I/O delay constraints typically can be translated from an elaborated RTL design.

TimeGROUP

You can use Tcl variables with timing exceptions to accomplish the same effect as an INST/TNM and a TIMESPEC. The following example illustrates the point.

UCF Example:

```
INST "DUT/BLOCK_A/data_reg[*]" TNM = "from_data_reg_0";
INST "DUT/BLOCK_A/addr_reg[*]" TNM = "from_data_reg_0";
INST "DUT/BLOCK_B/data_sync[*]" TNM = "to_data_reg_0";
INST "DUT/BLOCK_B/addr_sync[*]" TNM = "to_data_reg_0";
TIMESPEC "TS_MCP" = FROM "from_data_reg_0" TO "to_data_reg_0" TS_FSCLK * 3;
```

Tcl Equivalent:

```
set from_data_reg_0 [get_cells {DUT/BLOCK_A/data_reg[*]\
DUT/BLOCK_A/addr_reg[*]}];
set to_data_reg_0 [get_cells {DUT/BLOCK_B/data_sync[*]\
DUT/BLOCK_B/addr_sync[*]}];
set_multicycle_path -setup 3 -from $from_data_reg_0 -to $to_data_reg_0;
set_multicycle_path -hold 2 -from $from_data_reg_0 -to $to_data_reg_0;
```

Timing Constraints

The following ISE Design Suite timing constraints can be represented as XDC timing constraints in the Vivado Design Suite. Each constraint description contains a UCF example and the equivalent XDC example.

UCF and XDC differ when creating clocks on a net that is not directly connected to the boundary of the design (such as port). In XDC, when defining a primary clock with `create_clock` on a net the source point is the driving pin of the net.

The clock insertion delay before that point is ignored. This can be an issue when timing the clock with another related clock; the skew will not be accurate.

Using the create_clock Tcl Command

First, be aware of how the different start points of the `create_clock` command affect timing accuracy.

In ISE

In ISE, the following period constraints are equivalent:

```
NET "clk" TNM_NET = sys_clk;
TIMESPEC TS_sys_clk = PERIOD "sys_clk" 10 ns HIGH 50%;

NET "clk_IBUF_BUFG" TNM_NET = sys_clk;
TIMESPEC TS_sys_clk = PERIOD "sys_clk" 10 ns HIGH 50%;
```

In the Vivado Design Suite

In the Vivado Design Suite, the following `create_clock` constraints differ.

- create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports clk]
- create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_pins clk_IBUF_BUFG_inst/O]

The constraints differ because they use different start points to define the time zero that the Vivado IDE assigns when computing the clock latency and uncertainty for the slack equation. See the figure below.

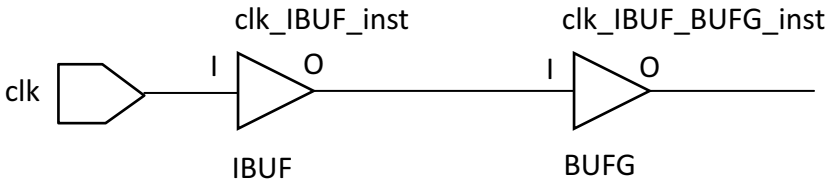

Figure 3-1: **Clocking Structure Example from a Clock Pad through BUFG**

The Vivado IDE ignores all clock tree delays coming from cells located upstream from the point at which the primary clock is defined. If you define a primary clock on a pin in the middle of the design, only part of its latency is used for timing analysis. This can be an issue if this clock communicates with other related clocks in the design because the skew, and consequently the slack, value between the clocks can be inaccurate.

Use the `create_clock` command where the clock trees originate not in the middle of the design (for example, input port or GT clock output pin). Create generated clocks in the middle of the design only.

Clock Constraints

Period

UCF Example

```
NET "clka" TNM_NET = "clka";
TIMESPEC "TS_clka" = PERIOD "clka" 13.330 ns HIGH 50.00%;
```

XDC Example

```
create_clock -name clka -period 13.330 -waveform {0 6.665}\
[get_ports clka]
```

Period Constraints with Uneven Duty Cycle

UCF Example

```
NET "clka" TNM_NET = "clka";
TIMESPEC "TS_clka" = PERIOD "clka" 13.330 ns HIGH 40.00%;
```

XDC Example

```
create_clock -name clka -period 13.330 -waveform {0 5.332}\
[get_ports clka]
```

Generated Clocks Constraints

UCF Example

```
NET "gen_clk" TNM_NET = "gen_clk";
TIMESPEC "TS_gen_clk" = PERIOD "gen_clk" "TS_clka" * 0.500 HIGH
50.00%;
```

XDC Example

```
create_generated_clock -source [get_ports clka] -name gen_clk\
-multiply_by 2 [get_ports gen_clk]
```

Period Constraints with LOW Keyword

UCF Example
```
NET "clka" TNM_NET = "clka";
TIMESPEC "TS_clka" = PERIOD "clka" 13.330 ns LOW 50.00%;
```

XDC Example
```
create_clock -name clka -period 13.330 -waveform {6.665 13.330}\
[get_ports clka]
```

Net PERIOD Constraints

UCF Example
```
NET "clk_bufg" PERIOD = 10 ns;
```

XDC Example
```
create_clock -name clk_bufg -period 10 -waveform {0 5}\
[get_pins clk_bufg/O}
```

Note: Unless there is specific reason to define the clock on `bufg/O`, define it at an upstream top-level port.

OFFSET IN

BEFORE

UCF Example
```
OFFSET = IN 8 BEFORE clka;
```

XDC Example
```
set_input_delay -clock clka 2 [all_inputs]
```

Note: This assumes the clock period is 10 ns.

AFTER

UCF Example
```
OFFSET = IN 2 AFTER clka;
```

XDC Example
```
set_input_delay -clock clka 2 [all_inputs]
```

Note: This assumes the clock period is 10 ns.

BEFORE an Input Port Net

UCF Example
```
NET enable OFFSET = IN 8 BEFORE clka;
```

XDC Example
```
set_input_delay 2 [get_ports enable]
```

Note: This assumes the clock period is 10 ns.

BEFORE an Input Port Bus

UCF Example
```
INST "processor_data_bus[*]" TNM = "processor_bus";
TIMEGRP "processor_bus" OFFSET = IN 8ns BEFORE "clka";
```

XDC Example
```
set_input_delay 2 [get_ports {processor_data_bus[*]}]
```

Note: Offset is applied to ports only.

To TIMEGROUP

UCF Example	`INST "input_ffs[*]" TNM = "input_ffs";` `OFFSET = IN 8ns BEFORE "clka" TIMEGRP "input_ffs";`
XDC Example	Manual conversion is required. For more information, see TimeGROUP.

FALLING/RISING Edge

UCF Example	`OFFSET = IN 8ns BEFORE "clka" FALLING;`
XDC Example	`set_input_delay -clock clka 2 [all_inputs]` ***Note:*** This assumes the clock period is 10 ns.

LOW/HIGH Keyword

UCF Example	`OFFSET = IN 8ns BEFORE "clka" HIGH;`
XDC Example	Requires manual conversion. ***Note:*** HIGH/LOW keywords are precursors to RISING/FALLING. RISING/FALLING is the preferred method.

VALID Keyword

UCF Example	`OFFSET = IN 1ns VALID 2ns BEFORE clka;`

XDC Example	`set_input_delay -clock clka -max 9 [all_inputs]`
	`set_input_delay -clock clka -min 1[all_inputs]`
	Note: This assumes the clock period is 10 ns.

OFFSET OUT

RECOMMENDED: *In the ISE Design Suite, `OFFSET = OUT` performs only maximum delay analysis. In the Vivado Design Suite, `set_output_delay` performs both maximum and minimum delay analysis. To constrain output ports in XDC files, Xilinx recommends using `set_output_delay` with the `-max` and `-min` options.*

AFTER

UCF Example	`OFFSET = OUT 12 AFTER clkc;`
XDC Example	`set_output_delay -clock clkc -max 8 [all_outputs]` ***Note:*** This assumes the clock period is 20 ns.

BEFORE

UCF Example	`OFFSET = OUT 8 BEFORE clkc;`
XDC Example	`set_output_delay -clock clkc 8 [all_outputs]` ***Note:*** This assumes the clock period is 20 ns.

Output Net

UCF Example	`NET out_net OFFSET = OUT 12 AFTER clkc;`
XDC Example	`set_output_delay 8 [get_port out_net]`
	Note: This assumes the clock period is 20 ns.

Group of Outputs

UCF Example	`TIMEGRP outputs OFFSET = OUT 12 AFTER clkc;`
XDC Example	`set_output_delay -clock clkc 8 [get_ports outputs*]`
	Note: This assumes the clock period is 20 ns.

From a TIMEGROUP

UCF Example	`OFFSET = OUT 1.2 AFTER clk TIMEGRP from_ffs;`
XDC Example	Manual conversion is required.

FALLING/RISING Edges

UCF Example	`OFFSET = OUT 12 AFTER clkc FALLING;`
XDC Example	`set_output_delay -clock clkc -clock_fall 8 [all_outputs]`

LOW Keyword

UCF Example	`OFFSET = OUT 12 AFTER clkc LOW;`
XDC Example	Requires manual conversion.
	Note: HIGH/LOW keywords are precursors to RISING/FALLING. RISING/FALLING is the preferred method.

REFERENCE_PIN

UCF Example	`TIMEGRP mac_ddr_out;` OFFSET = OUT AFTER clk REFERENCE_PIN clk_out RISING;
XDC Example	Requires manual conversion.
	Note: `REFERENCE_PIN` acts as a reporting switch to instruct `TRACE` to output a bus skew report. The Vivado Design Suite does not support this feature.

From:To Constraints

Generally, UCF From:To constraints are converted to either `set_max_delay` or `set_min_delay` XDC constraints, with the `-from`, `-to` and `-through` design-dependent arguments.

The intent of UCF constraints is to use the equivalent XDC constraints. While most UCF constraints are net-based, XDC constraints must be constructed to ports and pins.

Helpful XDC commands for these constraints are: `all_fanout`, `get_cells`, and `get_pins` as well as the `-from`, `-to`, and `-through` arguments.

Assigning Timing Group to an Area Group

UCF Example	`TIMEGRP clock_grp = AREA_GROUP clock_ag;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

EXCEPT

UCF Example	`TIMEGRP my_group = FFS EXCEPT your_group;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

Between Groups

UCF Example	`TIMESPEC TS_TIG = FROM reset_ff TO FFS TIG;`
XDC Example	Manual conversion is required. Construct a `set_false_path` that covers the desired paths.

By Net

UCF Example	`NET reset TIG;`

XDC Example	`set_false_path -through [get_nets reset]`
	A better approach is to find the primary reset port and use:
	`set_false_path -from [get_ports reset_port]`

By Instance

UCF Example	`INST reset TIG;`

XDC Example	`set_false_path -from [get_cells reset]`
	`set_false_path -through [get_cells reset]`
	`set_false_path -to [get_cells reset]`

By Pin

UCF Example	PIN ff.d TIG;

XDC Example	`set_false_path -to [get_pins ff/D]`
	`set_false_path -from [get_pins ff/C]`
	`set_false_path -through [get_pins lut/I0]`

Specific Time Constraints

UCF Example	`NET reset TIG = TS_fast TS_even_faster;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. ***Note:*** Constraint-specific TIG tries to disable timing through the net, but only for analysis of the two referenced constraints.

MAXSKEW

UCF Example	`NET local_clock MAXSKEW = 2ns;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

MAXDELAY

UCF Example	`NET local_clock MAXDELAY = 2ns;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. You can, however, use `set_max_delay` for specifying the timing requirement for a valid timing path (synchronous start point to synchronous Endpoint).

Physical Constraints

The following ISE Design Suite physical constraints can be represented as XDC constraints in the Vivado Design Suite. Each constraint description contains:

- Target object type
- Constraint value type
- UCF example
- Equivalent XDC example

For information, see the following guides:

- *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 3]
- *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4]
- *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 13]

Placement-Related Constraints

AREA_GROUP

Applied To	Cells
Constraint Values	String
UCF Example	`INST bmg0 AREA_GROUP = AG1;`
XDC Example	`create_pblock ag1; add_cells_to_pblock [get_pblocks ag1]\ [get_cells [list bmg0]]`

AREA_GROUP RANGE

SLICE

Applied To	Area groups and Pblocks
Constraint Values	SLICE_XnYn[:SLICE_XnYn]
UCF Example	`AREA_GROUP AG1 RANGE = SLICE_X0Y44:SLICE_X27Y20;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {SLICE_X0Y44:SLICE_X27Y20}`

RAMB18

Applied To	Area groups and Pblocks
Constraint Values	RAMB18_XnYn:RAMB18_XnYn
UCF Example	`AREA_GROUP AG1 RANGE = RAMB18_X0Y86:RAMB18_X3Y95;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {RAMB18_X0Y86:RAMB18_X3Y95}`

RAMB36

Applied To	Area groups and Pblocks
Constraint Values	RAMB36_XnYn:RAMB36_XnYn
UCF Example	`AREA_GROUP AG1 RANGE = RAMB36_X0Y11:RAMB36_X3Y18;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {RAMB36_X0Y11:RAMB36_X3Y18}`

CLOCKREGION (1)

Applied To	Area groups and Pblocks
Constraint Values	CLOCKREGION_XnYn
UCF Example	`area_group ag1 range = CLOCKREGION_X0Y0;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {CLOCKREGION_X0Y0:CLOCKREGION_X0Y0}`

CLOCKREGION (2)

Applied To	Area groups and Pblocks
Constraint Values	CLOCKREGION_XnYn[:CLOCKREGION_XnYn]
UCF Example	`area_group ag1 range = CLOCKREGION_X0Y0:CLOCKREGION_X1Y0;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {CLOCKREGION_X0Y0:CLOCKREGION_X0Y0}`

CLOCKREGION (3)

Applied To	Area groups and Pblocks
Constraint Values	CLOCKREGION_XnYn,CLOCKREGION_XnYn, . . .
UCF Example	`area_group ag1 range = CLOCKREGION_X0Y0, CLOCKREGION_X1Y0;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {CLOCKREGION_X0Y0:CLOCKREGION_X0Y0\ CLOCKREGION_X1Y0:CLOCKREGION_X1Y0}`

DSP48

Applied To	Area groups and Pblocks
Constraint Values	DSP48_XnYn:DSP48_XnYn
UCF Example	`AREA_GROUP D1 RANGE = DSP48_X2Y0:DSP48_X2Y9;`
XDC Example	`resize_pblock [get_pblocks D1] -add {DSP48_X2Y0:DSP48_X2Y9}`

BUFGCTRL

Applied To	Area groups and Pblocks
Constraint Values	BUFGCTRL_XnYn:BUFGCTRL_XnYn
UCF Example	`AREA_GROUP ag1 range = BUFGCTRL_X0Y24:BUFGCTRL_X0Y31;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {BUFGCTRL_X0Y24:BUFGCTRL_X0Y31}`

BUFHCE

Applied To	Area groups and Pblocks
Constraint Values	BUFHCE_XnYn:BUFHCE_XnYn
UCF Example	`AREA_GROUP ag1 range = BUFHCE_X0Y72:BUFHCE_X1Y77;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {BUFHCE_X0Y72:BUFHCE_X1Y77}`

BUFR

Applied To	Area groups and Pblocks
Constraint Values	BUFR_XnYn:BUFR_XnYn
UCF Example	`AREA_GROUP ag1 range = BUFR_X0Y20:BUFR_X1Y23;`
XDC Example	`resize_pblock [get_pblocks ag1] -add {BUFR_X0Y0:BUFR_X1Y2}`

BUFIO

Applied To	Area groups and Pblocks
Constraint Values	BUFIO_XnYn:BUFIO_XnYn
UCF Example	`AREA_GROUP ag1 range = BUFIO_X0Y8:BUFIO_X0Y11;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {BUFIO_X0Y8:BUFIO_X0Y11}`

IOB Range

Applied To	Area groups and Pblocks
Constraint Values	IOB_XnYn:IOB_XnYn
UCF Example	`AREA_GROUP ag1 range = IOB_X0Y341:IOB_X1Y349;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {IOB_X0Y341:IOB_X1Y349}`

IN_FIFO

Applied To	Area groups and Pblocks
Constraint Values	IN_FIFO_XnYn:IN_FIFO_XnYn
UCF Example	`AREA_GROUP ag1 range = IN_FIFO_X0Y24:IN_FIFO_X1Y27;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {IN_FIFO_X0Y24:IN_FIFO_X1Y27}`

OUT_FIFO

Applied To	Area groups and Pblocks
Constraint Values	OUT_FIFO_XnYn:OUT_FIFO_XnYn
UCF Example	`AREA_GROUP ag1 range = OUT_FIFO_X0Y24:OUT_FIFO_X1Y27;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {OUT_FIFO_X0Y24:OUT_FIFO_X1Y27}`

ILOGIC

Applied To	Area groups and Pblocks
Constraint Values	ILOGIC_XnYn:ILOGIC_XnYn
UCF Example	`AREA_GROUP ag1 range = ILOGIC_X0Y76:ILOGIC_X0Y79;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {ILOGIC_X0Y76:ILOGIC_X0Y79}`

OLOGIC

Applied To	Area groups and Pblocks
Constraint Values	OLOGIC_XnYn:OLOGIC_XnYn
UCF Example	`AREA_GROUP ag1 range = OLOGIC_X0Y76:OLOGIC_X0Y79;`
XDC Example	`resize_pblock [get_pblocks ag1] -add\ {OLOGIC_X0Y76:OLOGIC_X0Y79}`

LOC

IOB

Applied To	Port nets
Constraint Values	IOB site
UCF Example	`NET p[0] LOC = H1;`
XDC Example	`set_property PACKAGE_PIN H1 [get_ports p[0]]`

TIP: *To assign pins in the Vivado Design Suite, use the* `PACKAGE_PIN` *port property, and not the* `LOC` *property, which is used for cells.*

SLICE (1)

Applied To	Cells
Constraint Values	Site range
UCF Example	`INST a_reg[*] LOC = SLICE_X25Y*;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

SLICE (2)

Applied To	Cells
Constraint Values	SLICE_XnYn
UCF Example	`INST a_reg[0] LOC = SLICE_X4Y4;`
XDC Example	`set_property LOC SLICE_X4Y4 [get_cells a_reg[0]]`

RAMB18

Applied To	Cells
Constraint Values	RAMB18_XnYn
UCF Example	`INST ram0 LOC = RAMB18_X0Y5;`
XDC Example	`set_property LOC RAMB18_X0Y5 [get_cells ram0]`

RAMB36

Applied To	Cells
Constraint Values	RAMB36_XnYn
UCF Example	`INST ram0 LOC = RAMB36_X0Y0;`
XDC Example	`set_property LOC RAMB36_X0Y0 [get_cells ram0]`

DSP48

Applied To	Cells
Constraint Values	DSP48_XnYn
UCF Example	`INST dsp0 LOC = DSP48_X0Y10;`
XDC Example	`set_property LOC DSP48_X0Y10 [get_cells dsp0]`

BUFGCTRL

Applied To	Cells
Constraint Values	BUFGCTRL_XnYn
UCF Example	`INST cb[0] LOC = BUFGCTRL_X0Y24;`
XDC Example	`set_property LOC BUFGCTRL_X0Y24 [get_cells cb[0]]`

BUFHCE

Applied To	Cells
Constraint Values	BUFHCE_XnYn
UCF Example	`INST cb[0] LOC = BUFHCE_X0Y72;`
XDC Example	`set_property LOC BUFHCE_X0Y72 [get_cells cb[0]]`

BUFR

Applied To	Cells
Constraint Values	BUFR_XnYn
UCF Example	`INST cb[0] LOC = BUFR_X0Y20;`
XDC Example	`set_property LOC BUFR_X0Y20 [get_cells cb[0]]`

BUFIO

Applied To	Cells
Constraint Values	BUFIO_XnYn
UCF Example	`INST cb[0] LOC = BUFIO_X0Y8;`
XDC Example	`set_property LOC BUFIO_X0Y8 [get_cells cb[0]]`

KEEP_HIERARCHY

Applied To	Cells
Constraint Values	• TRUE • FALSE • YES • NO
UCF Example	`INST u1 KEEP_HIERARCHY = TRUE;`
XDC Example	`set_property DONT_TOUCH true [get_cells u1]`

IOB

Applied To	Cells
Constraint Values	IOB_XnYn
UCF Example	`INST ib[0] LOC = IOB_X0Y341;`
XDC Example	`set_property LOC IOB_X0Y341 [get_cells ib[0]]`

IN_FIFO

Applied To	Cells
Constraint Values	IN_FIFO_XnYn
UCF Example	`INST infifo_inst LOC = IN_FIFO_X0Y24;`
XDC Example	`set_property LOC IN_FIFO_X0Y24 [get_cells infifo_inst]`

OUT_FIFO

Applied To	Cells
Constraint Values	OUT_FIFO_XnYn
UCF Example	`INST outfifo_inst LOC = OUT_FIFO_X0Y24;`
XDC Example	`set_property LOC OUT_FIFO_X0Y24 [get_cells outfifo_inst]`

ILOGIC

Applied To	Cells
Constraint Values	ILOGIC_XnYn
UCF Example	`INST ireg LOC = ILOGIC_X0Y76;`
XDC Example	`set_property LOC ILOGIC_X0Y76 [get_cells ireg]`

OLOGIC

Applied To	Cells
Constraint Values	OLOGIC_XnYn
UCF Example	`INST oreg LOC = OLOGIC_X0Y76`
XDC Example	`set_property LOC OLOGIC_X0Y76 [get_cells oreg]`

IDELAY

Applied To	Cells
Constraint Values	IDELAY_XnYn
UCF Example	`INST idelay0 LOC = IDELAY_X0Y21;`
XDC Example	`set_property LOC IDELAY_X0Y21 [get_cells idelay0]`

IDELAYCTRL

Applied To	Cells
Constraint Values	IDELAYCTRL_XnYn
UCF Example	`INST idelayctrl0 LOC = IDELAYCTRL_X0Y0;`
XDC Example	`set_property LOC IDELAYCTRL_X0Y0 [get_cells idelayctrl0]`

BEL

A5LUT, B5LUT, C5LUT, D5LUT

Applied To	Cells
Constraint Values	A5LUT, B5LUT, C5LUT, D5LUT
UCF Example	`INST a0 BEL = A5LUT;`
XDC Example	`set_property BEL A5LUT [get_cells a0]`

A6LUT, B6LUT, C6LUT, D6LUT

Applied To	Cells
Constraint Values	A6LUT, B6LUT, C6LUT, D6LUT
UCF Example	`INST a0 BEL = D6LUT;`
XDC Example	`set_property BEL D6LUT [get_cells a0]`

AFF, BFF, CFF, DFF

Applied To	Cells
Constraint Values	AFF, BFF, CFF, DFF
UCF Example	`INST a_reg[0] BEL = CFF;`
XDC Example	`set_property BEL CFF [get_cells a_reg[0]]`

A5FF, B5FF, C5FF, D5FF

Applied To	Cells
Constraint Values	A5FF, B5FF, C5FF, D5FF
UCF Example	`INST a_reg[0] BEL = B5FF;`
XDC Example	`set_property BEL B5FF [get_cells a_reg[0]]`

F7AMUX, F7BMUX

Applied To	Cells
Constraint Values	F7AMUX, F7BMUX
UCF Example	`INST m0 BEL = F7BMUX;`
XDC Example	`set_property BEL F7BMUX [get_cells m0]`

IOB

TRUE

Applied To	FF cells
Constraint Values	TRUE
UCF Example	`INST a1_reg[*] IOB = TRUE;`
XDC Example	`set_property IOB TRUE [get_ports DataOut_pad[*]]`

TIP:
XDC example 1, above, puts the property on the port itself and causes the flop that drives the IO buffer to be placed inside the pad.

FALSE

Applied To	FF cells
Constraint Values	FALSE
UCF Example	`INST b1_reg[*] IOB = FORCE;`
XDC Example	`set_property IOB TRUE [get_cells a1_reg[*]]`

FORCE

Applied To	FF cells
Constraint Values	FORCE
UCF Example	`INST q_reg[*] IOB = FALSE;`
XDC Example	`set_property IOB TRUE [get_cells q_reg[*]]`
	Note: The Vivado Design Suite does not support this constraint in XDC. Use TRUE instead.

H_SET

Applied To	Cells
Constraint Values	Tool-generated string
UCF Example	N/A
XDC Example	N/A
	Note: For more information, see Relative Location (RLOC) in the *Constraints Guide* (UG625) [Ref 11]. In the Vivado Design Suite, H_SET cells have a property called RPM.

U_SET

Applied To	Cells
Constraint Values	String
UCF Example	`INST u0 U_SET = h0; (usually set in UCF)`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. U_SET must be placed in HDL code as an attribute. For more information, see Relative Location (RLOC) in the *Constraints Guide* (UG625) [Ref 11].

RLOC

Applied To	Cells
Constraint Values	XnYn
UCF Example	`INST u0 RLOC = X2Y1;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. RLOC must be placed in HDL code as an attribute. For more information, see Relative Location (RLOC) in the *Constraints Guide* (UG625) [Ref 11].

RLOC_ORIGIN

Applied To	Cells
Constraint Values	XnYn
UCF Example	`INST u0 RLOC_ORIGIN = X144Y255;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. `RLOC_ORIGIN` must be placed in HDL code as an attribute. For more information, see Relative Location (RLOC) in the *Constraints Guide* (UG625) [Ref 11].

RPM_GRID

Applied To	Cells
Constraint Values	GRID
UCF Example	`INST u0 RPM_GRID = GRID;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. RPM_GRID must be placed in HDL code as an attribute. For more information, see Relative Location (RLOC) in the *Constraints Guide* (UG625) [Ref 11].

USE_RLOC

Applied To	Cells
Constraint Values	TRUE, FALSE
UCF Example	`INST u0 USE_RLOC = FALSE;`

USE_RLOC

XDC Example	The Vivado Design Suite does not support this constraint in XDC.

RLOC_RANGE

Applied To	Cells
Constraint Values	XnYn:XnYn
UCF Example	`INST u0 RLOC_RANGE = X1Y1:X3Y3;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. Create a `Pblock` with the desired range, and add the RPM cells to the `Pblock`.

BLKNM

Applied To	Cells
Constraint Values	String
UCF Example	`INST u0 BLKNM = blk0;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

HBLKNM

Applied To	Cells, Nets
Constraint Values	String
UCF Example	`INST u0 HBLKNM = blk0;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

XBLKNM

Applied To	Cells, Nets
Constraint Values	String
UCF Example	`INST u0 XBLKNM = blk0;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC. Use `BEL PROHIBIT` to keep out unrelated logic.

HLUTNM

Applied To	LUT cells
Constraint Values	String
UCF Example	UCF is not allowed, only HDL.
XDC Example	`set_property HLUTNM h0 [get_cells {LUT0 LUT1}]`

LUTNM

Applied To	LUT cells
Constraint Values	String

LUTNM

UCF Example	UCF is not allowed, only HDL.
XDC Example	`set_property LUTNM h0 [get_cells {LUT0 LUT1}]`

USE_LUTNM

Applied To	LUT cells
Constraint Values	TRUE, FALSE
UCF Example	`INST lut0 USE_LUTNM = FALSE;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

CLOCK_DEDICATED_ROUTE

TRUE(1)

Applied To	Nets
Constraint Values	TRUE
UCF Example	`net clk0 CLOCK_DEDICATED_ROUTE = TRUE;`
XDC Example	`set_property CLOCK_DEDICATED_ROUTE TRUE [get_nets clk0]`

TRUE(2)

Applied To	Pins
Constraint Values	TRUE
UCF Example	`PIN clkbuf0.O CLOCK_DEDICATED_ROUTE = TRUE;`
XDC Example	`set_property CLOCK_DEDICATED_ROUTE TRUE [get_pins\ clkbuf0/O]`

FALSE(1)

Applied To	Nets
Constraint Values	FALSE
UCF Example	`NET clk0 CLOCK_DEDICATED_ROUTE = FALSE;`
XDC Example	`set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk0]`

FALSE(2)

Applied To	Pins
Constraint Values	FALSE
UCF Example	`PIN clkbuf0.O CLOCK_DEDICATED_ROUTE = FALSE;`
XDC Example	`set_property CLOCK_DEDICATED_ROUTE FALSE [get_pins\ clkbuf0/O]`

BACKBONE(1)

Applied To	Nets

BACKBONE(1)

Constraint Values	BACKBONE
UCF Example	`NET clk0 CLOCK_DEDICATED_ROUTE = BACKBONE;`
XDC Example	`set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_nets clk0]`

BACKBONE(2)

Applied To	Pins
Constraint Values	BACKBONE
UCF Example	`PIN clkbuf0.O CLOCK_DEDICATED_ROUTE = BACKBONE;`
XDC Example	`set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins\ clkbuf0/O]`

I/O-Related Constraints

IODELAY_GROUP

Applied To	IDELAY and IDELAYCTRL cells
Constraint Values	String
UCF Example	`INST idelay0 IODELAY_GROUP = group0;`
XDC Example	`set_property IODELAY_GROUP group0 [get_cells idelay0]`

DCI_VALUE

Applied To	I/O buffer cells
Constraint Values	Integer. Resistance in Ohms
UCF Example	`INST a_IBUF[0]_inst DCI_VALUE = 75;`
XDC Example	`set_property DCI_VALUE 75 [get_cells {a_IBUF[0]_inst}]`

DIFF_TERM

Applied To	I/O buffer cells
Constraint Values	Boolean
UCF Example	`INST a_IBUF[0]_inst DIFF_TERM = TRUE;`
XDC Example	`set_property DIFF_TERM true [get_cells {a_IBUF[0]_inst}]`

DRIVE

Applied To	Inout and output buffer cells
Constraint Values	Integer: 2, 4, 6, 8, 12, 16, 24
UCF Example	`INST q_OBUF[0]_inst DRIVE = 24;`
XDC Example	`set_property DRIVE 24 [get_ports q[0]]`
	LVTTL allows a value of 24.

IOSTANDARD

Applied To	I/O buffer cells
Constraint Values	I/O standard string
UCF Example	`INST q_OBUF[0]_inst IOSTANDARD = LVCMOS25;`
XDC Example	`set_property IOSTANDARD LVCMOS25 [get_ports q[0]]`
	For more information, see the *Constraints Guide* (UG625) [Ref 11].

SLEW

Applied To	Inout and output buffer cells
Constraint Values	SLOW or FAST
UCF Example	`INST q_OBUF[0]_inst SLEW = FAST;`
XDC Example	`set_property SLEW FAST [get_ports q[0]]`

FAST

Applied To	Inout and output buffer cells
Constraint Values	N/A
UCF Example	`INST q_OBUF[0]_inst FAST;`
XDC Example	`set_property SLEW FAST [get_ports q[0]]`

SLOW

Applied To	Inout and output buffer cells
Constraint Values	N/A
UCF Example	`INST q_OBUF[0]_inst SLOW;`
XDC Example	`set_property SLEW SLOW [get_ports q[0]]`

PORTS

IN_TERM

Applied To	Ports
Constraint Values	• NONE • UNTUNED_SPLIT_40 • UNTUNED_SPLIT_50 • UNTUNED_SPLIT_60
UCF Example	`NET a[0] IN_TERM = UNTUNED_SPLIT_50;`
XDC Example	`set_property IN_TERM UNTUNED_SPLIT_50 [get_ports [list\ clk]]`

OUT_TERM

Applied To	Ports
Constraint Values	• NONE • UNTUNED_25 • UNTUNED_50 • UNTUNED_75
UCF Example	`net q[0] OUT_TERM = UNTUNED_50;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

IOBDELAY

Applied To	Port nets
Constraint Values	NONE
UCF Example	`net b[0] IOBDELAY = NONE;`
XDC Example	`set_property IOBDELAY NONE [get_nets b[0]]` ***Note:*** You cannot set IOBDELAY on ports. However, you can set IOBDELAY on cells such as input buffers.

BOTH

Applied To	Port nets
Constraint Values	BOTH
UCF Example	`net b[0] IOBDELAY = BOTH;`
XDC Example	`set_property IOBDELAY BOTH [get_nets b[0]]` ***Note:*** You cannot set IOBDELAY on ports. However, you can set IOBDELAY on cells such as input buffers.

IBUF

Applied To	Port nets
Constraint Values	IBUF
UCF Example	`net b[0] IOBDELAY = IBUF;`
XDC Example	`set_property IOBDELAY IBUF [get_nets b[0]]` ***Note:*** You cannot set IOBDELAY on ports. However, you can set IOBDELAY on cells such as input buffers.

IFD

Applied To	Port nets
Constraint Values	IFD
UCF Example	`net b[0] IOBDELAY = IFD;`

IFD

XDC Example	`set_property IOBDELAY IFD [get_nets b[0]]`
	Note: You cannot set IOBDELAY on ports. However, you can set IOBDELAY on cells such as input buffers

KEEPER

Applied To	Port nets
Constraint Values	• TRUE • FALSE • YES • NO
UCF Example	`NET n1 KEEPER = TRUE;`
XDC Example	`set_property KEEPER true [get_ports n1]`

PULLDOWN

Applied To	Port nets
Constraint Values	• TRUE • FALSE • YES • NO
UCF Example	`NET n1 PULLDOWN = TRUE;`
XDC Example	`set_property PULLDOWN true [get_ports n1]`

PULLUP

Applied To	Port nets
Constraint Values	• TRUE • FALSE • YES • NO
UCF Example	`NET n1 PULLUP = TRUE;`
XDC Example	`set_property PULLUP true [get_ports n1]`

VCCAUX_IO

Applied To	Ports
Constraint Values	• NORMAL • HIGH • DONTCARE
UCF Example	`NET d[0] VCCAUX_IO = HIGH;`
XDC Example	`set_property VCCAUX_IO HIGH [get_ports d[0]]`

Miscellaneous Net-Related Constraints

KEEP

Applied To	Nets
Constraint Values	• TRUE • FALSE
UCF Example	`net x_int KEEP = TRUE;`
XDC Example	`set_property DONT_TOUCH true [get_nets x_int]`

SAVE NET FLAG

Applied To	Nets
Constraint Values	N/A
UCF Example	`net x_int S;`
XDC Example	`set_property DONT_TOUCH true [get_nets x_int]`

LOCK_PINS

Applied To	LUT cell
Constraint Values	CSV string: `I[0-5]:A[6-1]`
UCF Example	`INST LUT1 LOCK_PINS = I3:A6, I2:A5;`
XDC Example	`set_property LOCK_PINS {I3:A6 I2:A5} [get_cells LUT1]`

ROUTE

Applied To	Nets
Constraint Values	Directed Routing String (DIRT)
UCF Example	`NET n85 ROUTE={2;1;-4!-1;-53320; . . .16;-8!};`
XDC Example	`set_property FIXED_ROUTE {EE2BEG0 NR1BEG0\ CLBLL_LL_AX} [get_nets n85]` ***Note:*** ISE Design Suite directed routing strings and Vivado Design Suite net route properties are incompatible. Vivado uses a unique, un-encoded format

Configuration-Related Constraints

CONFIG PROHIBIT

Pin site

Applied To	Sites
Constraint Values	Pin site
UCF Example	`CONFIG PROHIBIT = K24, K26, K27, K28;`
XDC Example	`set_property PROHIBIT true [get_sites {K24 K26 K27 K28}]`

Bank number

Applied To	Sites
Constraint Values	Bank number
UCF Example	`CONFIG PROHIBIT = BANK34, BANK35, BANK36;`
XDC Example	`set_property PROHIBIT true [get_sites -of [get_iobanks 34\ 35 36]]`

RAM(1)

Applied To	Sites
Constraint Values	RAMs
UCF Example	`CONFIG PROHIBIT = RAMB18_X0Y0;`
XDC Example	`set_property PROHIBIT true [get_sites RAMB18_X0Y0]`

RAM(2)

Applied To	Sites
Constraint Values	RAMs
UCF Example	`CONFIG PROHIBIT = RAMB18_X0Y1, RAMB18_X0Y3, RAMB18_X0Y5;`
XDC Example	`set_property PROHIBIT true [get_sites {RAMB18_X0Y1\ RAMB18_X0Y3 RAMB18_X0Y5}]`
	Note: The comma-separated list shown here uses RAM sites but can use any supported site type.

RAM(3)

Applied To	Sites
Constraint Values	RAMs
UCF Example	`CONFIG PROHIBIT = RAMB36_X1Y1:RAMB36_X2Y2;`
XDC Example	`set_property PROHIBIT true [get_sites -range {RAMB36_X1Y1\ RAMB36_X2Y2}]`

RAM(4)

Applied To	Sites
Constraint Values	RAMs
UCF Example	`CONFIG PROHIBIT = RAMB36_X3Y*;`
XDC Example	`set_property PROHIBIT true [get_sites RAMB36_X3Y*]`

DSP48

Applied To	Sites
Constraint Values	DSP48s
UCF Example	`CONFIG PROHIBIT = DSP48_X0Y*;`
XDC Example	`set_property PROHIBIT true [get_sites DSP48_X0Y*]`

SLICE

Applied To	Sites
Constraint Values	Slices
UCF Example	`CONFIG PROHIBIT = SLICE_X0Y0:SLICE_X47Y49;`
XDC Example	`set_property PROHIBIT true [get_sites -range {SLICE_X0Y0\ SLICE_X47Y49}]`

ILOGIC

Applied To	Sites
Constraint Values	ILOGIC
UCF Example	`CONFIG PROHIBIT = ILOGIC_X0Y0:ILOGIC_X0Y49;`
XDC Example	`set_property PROHIBIT true [get_sites -range {ILOGIC_X0Y0\ ILOGIC_X0Y49}]`

OLOGIC

Applied To	Sites
Constraint Values	OLOGIC
UCF Example	`CONFIG PROHIBIT = OLOGIC_X0Y0:OLOGIC_X0Y49;`
XDC Example	`set_property PROHIBIT true [get_sites -range {OLOGIC_X0Y0\ OLOGIC_X0Y49}]`

BUFGCTRL

Applied To	Sites
Constraint Values	BUFGCTRL
UCF Example	`CONFIG PROHIBIT = BUFGCTRL_X0Y0:BUFGCTRL_X0Y15;`
XDC Example	`set_property PROHIBIT true [get_sites -range\ {BUFGCTRL_X0Y0 BUFGCTRL_X0Y15}]`

BUFR

Applied To	Sites
Constraint Values	BUFR
UCF Example	`CONFIG PROHIBIT = BUFR_X0Y0:BUFR_X0Y3;`
XDC Example	`set_property PROHIBIT true [get_sites -range {BUFR_X0Y0\ BUFR_X0Y3}]`

BUFIO

Applied To	Sites
Constraint Values	BUFIO
UCF Example	`CONFIG PROHIBIT = BUFIO_X0Y0:BUFIO_X0Y3;`
XDC Example	`set_property PROHIBIT true [get_sites -range {BUFIO_X0Y0\ BUFIO_X0Y3}]`

BUFHCE

Applied To	Sites
Constraint Values	BUFHCE
UCF Example	`CONFIG PROHIBIT = BUFHCE_X0Y0:BUFHCE_X1Y11;`
XDC Example	`set_property PROHIBIT true [get_sites -range {BUFHCE_X0Y0\ BUFHCE_X1Y11}]`

Voltage

Applied To	I/O bank
Constraint Values	Voltage
UCF Example	`CONFIG INTERNAL_VREF_BANK14 = 0.75;`
XDC Example	`set_property INTERNAL_VREF 0.75 [get_iobanks 14]`

NONE

Applied To	I/O bank
Constraint Values	NONE
UCF Example	`CONFIG INTERNAL_VREF_BANK0 = NONE;`
XDC Example	`reset_property INTERNAL_VREF  [get_iobanks 0]`

CONFIG DCI_CASCADE

Applied To	I/O banks
Constraint Values	Bank sequence
UCF Example	`CONFIG DCI_CASCADE = 17 15 14;`
XDC Example	`set_property DCI_CASCADE {15 14} [get_iobanks 17]`

CONFIG CONFIG_MODE

M_SERIAL

Applied To	Global
Constraint Values	M_SERIAL
UCF Example	`CONFIG CONFIG_MODE = M_SERIAL;`
XDC Example	`set_property CONFIG_MODE M_SERIAL [current_design]`

S_SERIAL

Applied To	Global
Constraint Values	S_SERIAL
UCF Example	`CONFIG CONFIG_MODE = S_SERIAL;`
XDC Example	`set_property CONFIG_MODE S_SERIAL [current_design]`

B_SCAN

Applied To	Global
Constraint Values	B_SCAN
UCF Example	`CONFIG CONFIG_MODE = B_SCAN;`
XDC Example	`set_property CONFIG_MODE B_SCAN [current_design]`

B_SCAN+READBACK

Applied To	Global
Constraint Values	B_SCAN+READBACK
UCF Example	`CONFIG CONFIG_MODE = B_SCAN+READBACK;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

M_SELECTMAP

Applied To	Global
Constraint Values	M_SELECTMAP
UCF Example	`CONFIG CONFIG_MODE = M_SELECTMAP;`
XDC Example	`set_property CONFIG_MODE M_SELECTMAP [current_design]`

M_SELECTMAP+READBACK

Applied To	Global
Constraint Values	M_SELECTMAP+READBACK
UCF Example	`CONFIG CONFIG_MODE = M_SELECTMAP+READBACK;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

S_SELECTMAP

Applied To	Global
Constraint Values	S_SELECTMAP
UCF Example	`CONFIG CONFIG_MODE = S_SELECTMAP;`
XDC Example	`set_property CONFIG_MODE S_SELECTMAP [current_design]`

S_SELECTMAP+READBACK

Applied To	Global
Constraint Values	S_SELECTMAP+READBACK
UCF Example	`CONFIG CONFIG_MODE = S_SELECTMAP+READBACK;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

S_SELECTMAP16

Applied To	Global
Constraint Values	S_SELECTMAP16
UCF Example	`CONFIG CONFIG_MODE = S_SELECTMAP16;`
XDC Example	`set_property CONFIG_MODE S_SELECTMAP16 [current_design]`

S_SELECTMAP16+READBACK

Applied To	Global
Constraint Values	S_SELECTMAP16+READBACK
UCF Example	`CONFIG CONFIG_MODE = S_SELECTMAP16+READBACK;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

S_SELECTMAP32

Applied To	Global
Constraint Values	S_SELECTMAP32
UCF Example	`CONFIG CONFIG_MODE = S_SELECTMAP32;`
XDC Example	`set_property CONFIG_MODE S_SELECTMAP32 [current_design]`

S_SELECTMAP32+READBACK

Applied To	Global
Constraint Values	S_SELECTMAP32+READBACK
UCF Example	`CONFIG CONFIG_MODE = S_SELECTMAP32+READBACK;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

SPIx1

Applied To	Global
Constraint Values	SPIx1
UCF Example	`CONFIG CONFIG_MODE = SPIx1;`
XDC Example	`set_property CONFIG_MODE SPIx1 [current_design]`

SPIx2

Applied To	Global
Constraint Values	SPIx2
UCF Example	`CONFIG CONFIG_MODE = SPIx2;`
XDC Example	`set_property CONFIG_MODE SPIx2 [current_design]`

SPIx4

Applied To	Global
Constraint Values	SPIx4
UCF Example	`CONFIG CONFIG_MODE = SPIx4;`
XDC Example	`set_property CONFIG_MODE SPIx4 [current_design]`

BPI8

Applied To	Global
Constraint Values	BPI8
UCF Example	`CONFIG CONFIG_MODE = BPI8;`
XDC Example	`set_property CONFIG_MODE BPI8  [current_design]`

BPI16

Applied To	Global
Constraint Values	BPI16
UCF Example	`CONFIG CONFIG_MODE = BPI16;`
XDC Example	`set_property CONFIG_MODE BPI16 [current_design]`

CONFIG POST_CRC Commands

ENABLE

Applied To	Global
Constraint Values	ENABLE
UCF Example	`CONFIG POST_CRC = ENABLE;`
XDC Example	`set_property POST_CRC ENABLE [current_design]`

DISABLE

Applied To	Global
Constraint Values	DISABLE
UCF Example	`CONFIG POST_CRC = DISABLE;`
XDC Example	`set_property POST_CRC DISABLE [current_design]`

CONFIG POST_CRC_ACTION Commands

HALT

Applied To	Global
Constraint Values	HALT
UCF Example	`CONFIG POST_CRC_ACTION = HALT;`
XDC Example	`set_property POST_CRC_ACTION HALT [current_design]`

CONTINUE

Applied To	Global
Constraint Values	CONTINUE
UCF Example	`CONFIG POST_CRC_ACTION = CONTINUE;`
XDC Example	`set_property POST_CRC_ACTION CONTINUE [current_design]`

CORRECT_AND_CONTINUE

Applied To	Global
Constraint Values	CORRECT_AND_CONTINUE
UCF Example	`CONFIG POST_CRC_ACTION = CORRECT_AND_CONTINUE;`
XDC Example	`set_property POST_CRC_ACTION CORRECT_AND_CONTINUE\ [current_design]`

CORRECT_AND_HALT

Applied To	Global
Constraint Values	CORRECT_AND_HALT
UCF Example	`CONFIG POST_CRC_ACTION = CORRECT_AND_HALT;`
XDC Example	`set_property POST_CRC_ACTION correct_and_halt\ [current_design]`

CONFIG POST_CRC_FREQ

Applied To	Global
Constraint Values	Integer; frequency in MHz
UCF Example	`CONFIG POST_CRC_FREQ = 50;`
XDC Example	`set_property POST_CRC_FREQ 50 [current_design]`

CONFIG POST_CRC_INIT_FLAG

ENABLE

Applied To	Global
Constraint Values	ENABLE
UCF Example	`CONFIG POST_CRC_INIT_FLAG = ENABLE;`
XDC Example	`set_property POST_CRC_INIT_FLAG ENABLE [current_design]`

DISABLE

Applied To	Global
Constraint Values	DISABLE
UCF Example	`CONFIG POST_CRC_INIT_FLAG = DISABLE;`
XDC Example	`set_property POST_CRC_INIT_FLAG DISABLE [current_design]`

CONFIG POST_CRC_SOURCE

FIRST_READBACK

Applied To	Global
Constraint Values	FIRST_READBACK
UCF Example	`CONFIG POST_CRC_SOURCE = FIRST_READBACK;`
XDC Example	`set_property POST_CRC_SOURCE FIRST_READBACK\ [current_design]`

PRE_COMPUTED

Applied To	Global
Constraint Values	PRE_COMPUTED
UCF Example	`CONFIG POST_CRC_SOURCE = PRE_COMPUTED;`
XDC Example	`set_property POST_CRC_SOURCE PRE_COMPUTED [current_design]`

DEFAULT Commands

Note: DEFAULT is not supported. I/O ports must be individually configured.

DEFAULT FLOAT

Applied To	Global
Constraint Values	Boolean
UCF Example	`DEFAULT FLOAT = TRUE;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

DEFAULT KEEPER

Applied To	Global
Constraint Values	Boolean
UCF Example	`DEFAULT KEEPER = TRUE;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

DEFAULT PULLDOWN

Applied To	Global
Constraint Values	Boolean
UCF Example	`DEFAULT PULLDOWN = TRUE;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

DEFAULT PULLUP

Applied To	Global
Constraint Values	Boolean
UCF Example	`DEFAULT PULLUP = TRUE;`
XDC Example	The Vivado Design Suite does not support this constraint in XDC.

XILINX

Chapter 4

Migrating Designs with Legacy IP to the Vivado Design Suite

Overview

Vivado® Design Suite lets you migrate IP designs from the CORE Generator™ tool. You can also migrate IP to the latest version in the Vivado Design Suite.

IMPORTANT: *The Vivado Integrated Development Environment (IDE) requires that IP, instantiation, and port names are all lowercase. Rename any uppercase or mixed-case filenames to lowercase.*

You can reuse IP in the Vivado Design Suite from the following sources:

- ISE® Design Suite project using CORE Generator IP
- PlanAhead™ tool project using CORE Generator IP
- IP from a CORE Generator project
- IP from the Vivado IDE **ADD IP** option (`.xci` files)
- IP from the Embedded Development Kit (EDK) using the Create and Package New IP Wizard.

IMPORTANT: *Before migrating your design to Vivado Design Suite, make sure that your design uses the latest version of available IP.*

When migrating a project with IP (either an older Vivado project, an ISE Design Suite `xise` project) or adding IP stored externally (either from Core Generator or Vivado) into Vivado, the IP can be in one of the following states:

- IP is current. The IP can be re-customized and output products can be generated.
- IP is locked because the version cannot be found in the catalog and there is an upgrade path available. If you do not wish to upgrade then there are two possible scenarios:
  - If the output products were present when adding/importing they are available and can be used by the Vivado tools.

You cannot re-customize or generate any additional output products. If the output products required for synthesis (RTL) or implementation (NGC) are present you can proceed.

Note: Simulation targets are required for behavioral simulation.

- If the output products are not present you cannot regenerate them in the Vivado Design Suite. You must either go back to the version of the software in which you created the IP and generate them, or upgrade to the latest version because there is an upgrade path.

- IP is locked because the version cannot be found in the catalog and there is no upgrade path available. There are two possible scenarios:
  - If the output products were present when adding/importing, they are available and can be used by the Vivado Design Suite. You cannot re-customize or generate any additional output products. If the output products required for synthesis (RTL) or implementation (netlist) are present you can proceed.

    Note: Simulation targets are required for behavioral simulation.
  - If the output products are not present you cannot regenerate them. Either go back to the version of the software you used to create the IP and generate them, or recreate the IP using currently available IP in the Vivado Design Suite. This might require interface and design changes.

RECOMMENDED: *When working with IP, keep the IP in a remote location outside of a project. This makes IP more portable and easier to maintain. When customizing IP, generate the output products. This would be an NGC for CORE Generator and the synthesis, simulation, test bench, example, and possible other products for Vivado. This allows you to have a usable IP for synthesis and/or implementation even if the IP is removed or if the IP requires an update in the Vivado tools before re-customizing or generation can be done.*

Migrating CORE Generator IP to the Vivado Design Suite

Migrate CORE Generator IP to Vivado Design Suite IP in two steps:

1. Migrate a design using CORE Generator IP
2. Migrate IP to the latest version

Step 1: Migrating Design Using CORE Generator IP Sources

You can migrate a project with IP to Vivado Design Suite. To do so, you can do one of the following:

- Import an ISE Design Suite project into Vivado Design Suite project (see Importing an XISE Project Navigator Project)
- Convert PlanAhead tool project to Vivado Design Suite project (see Converting a PlanAhead Tool Project)
- Add the IP core source files (.xco files) from a CORE Generator project to a Vivado Design Suite project

Step 2: Migrating IP to Latest Version

Use the latest version of IP in your design. To migrate IP, update your current IP as follows:

1. In the Sources window, click the **IP Sources** tab.
2. Right-click on an IP core source.
3. Select **Upgrade IP** from the right-click menu.

Note: You can re-customize IP after you upgrade the IP to the latest version.

IMPORTANT: *For IP that is no longer available in the IP catalog, you can continue to reuse existing IP netlists and sources (such as NGC netlist or simulation files) with Vivado synthesis and implementation flows.*

RECOMMENDED: *You can also use the* ***Reports*** *>* ***Report IP Status*** *option to get a report of all the IP in your project with upgrade recommendations and quick access to the IP change logs. After reviewing the IP Status Report, you can select the IP to upgrade in the report.*

VIDEO: *For more information, see the* Vivado Design Suite QuickTake Video: Managing Vivado IP Version Upgrades.

Migrating EDK IP to the Vivado Design Suite

You can convert an XPS processor core, or Pcore, to a Vivado Design Suite native IP for use in the IP integrator.

To do so, you must manually run the **Tools > Create and Package New IP**. This process creates an IP-XACT definition file, `component.xml`, using the Package IP wizard. You can complete this through the **Manage IP** flow, working directly with the Pcore, or within your design project.

Feature Differences between Vivado Design Suite IP and ISE CORE Generator IP

IP delivered with the Vivado® Design Suite have the following characteristics over IP delivered in the ISE® Design Suite CORE™ Generator tool:

- Are accessible in a single unified IP Catalog
- Uses the new Xilinx® Design Constraints (XDC file) for physical and timing constraints which are applied automatically
- Generates a Xilinx Constraints Interface (XCI) file, various output products and by default a synthesis Design Checkpoint (DCP file).
  - The DCP contains a netlist for the IP and when referencing the XCI the netlist any constraints generated for the IP are used during implementation.
  - The DCP can be used directly, similar to an NGC, as it contains both the netlist and resolved constraints, but it is not recommended.
  - If an IP delivers BMM, ELF, Tcl script, or certain other files, they are not contained in the DCP. Using the XCI ensures all output products, including a synthesized netlist, are used.
- Places each IP (XCI file) in a separate directory (see the documentation on the Managed IP Flow and In Project Flow in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6].
- No longer uses the `XilinxCoreLib` for simulation (unless using older IP) as each IP delivers its own simulation sources as an output product

XILINX

Chapter 5

Migrating from XPS to IP Integrator

Overview

The Vivado® Design Suite IP integrator lets you stitch together a design containing Xilinx® IP or your custom IP in a relatively short time, working in a GUI environment.

Just as in Xilinx Platform Studio, you can quickly create an embedded processor design (using, for example, a Zynq® -7000 device or a MicroBlaze™ processor), along with peripherals, in IP integrator. You can migrate the following types design types to IP integrator:

- Zynq platform processor-based designs
- Microblaze processor-based designs
- Custom IP created in ISE® or PlanAhead™ software

IMPORTANT: *The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for new embedded processor designs, including designs targeting Zynq devices and MicroBlaze™ processors. XPS is no longer integrated with the Vivado Design Suite. However, DCP and NGC files created without constraints in XPS are supported as source files in the Vivado Design Suite.*

Key Feature Comparison between XPS and IP Integrator

Xilinx Platform Studio (XPS) and IP integrator are system-level tools that enable easier design creation using Xilinx IP or custom IP. The primary differences between these tool flows are highlighted in Table 5-1.

Table 5-1: **Tool Flow Differences Between XPS and IP Integrator**

Feature	XPS	IP Integrator
IP catalog	Embedded-only IP catalog, which is separate from other Xilinx IP (CORE Generator™ catalog)	Integrated Vivado Design Suite catalog
Design capture format	• XMP file for project information (device, flow) • MHS file for design information (IPs and their connections)	• Project information stored as part of design in the Vivado Design Suite • Design information stored in BD (XML format)
Text-based editing	MHS file editing in text editor/XPS editor	Tcl-based edit/design creation facilities in-line with the Vivado Design Suite
Integration with design tool	• Loose coupling with Vivado Design Suite • Supports both ISE Design Suite (PlanAhead tool) and Vivado Design Suite	• Tightly integrated into Vivado Design Suite • Only supported with Vivado Design Suite
Domains addressed	Embedded (processor based)	All
Graphical User Interface	Patch Panel-based connectivity for interfaces	Schematic/Block-based editing
Design Flows	Makefile-based	Integrated Vivado Design Suite flow
Device Family Support	• PlanAhead™ design tool: Spartan®-3, Virtex®-4, Virtex-5, Virtex-6, 7 Series devices, Zynq devices • Vivado: 7 series devices only (no Zynq device support)	• 7 series devices • Zynq device • UltraScale devices • New architectures

Tips for Converting Designs from XPS to IP Integrator

Xilinx does not provide an automated method for converting XPS designs to IP integrator. You must therefore create designs with IP integrator from scratch, referring to the XPS design.

The key tasks to perform in the system-level tool are as follows:

- IP Instantiation
- IP Customization
- Design Connectivity
- Address Mapping

- Clock and Resets
- Interconnect Configuration
- Setting up for Debug
- Association of ELF Files
- Migrating Zynq-7000 SoC-Based Designs
- Migrating a MicroBlaze Processor-Based Design

IP Instantiation

Most of the 7 series IP from the XPS catalog map directly to another IP in the Vivado IDE IP catalog. For each IP in XPS, you must instantiate an equivalent IP in the IP integrator canvas.

Tcl Command for XPS IP Instantiation

You can also accomplish XPS IP instantiation into the IP integrator by using the `create_bd_cell` Tcl command with the appropriate options.

Some IP from the ISE Design Suite IP catalog were modified to suit the embedded design needs and delivered in the XPS IP catalog. With unification of the IP catalogs, certain IP transitions require additional attention.

1. AXI 7 series DDRx to MIG
2. Block RAM to Block Memory Generator
3. Clock Generator to Clocking Wizard
4. AXI Interconnect is used to interconnect the IP and processor
5. Debug IP: Most of the Debug IP are available in the Vivado Design Suite tools, just as in XPS. The available debug IP are:
    - AXI Performance Monitor
    - Vivado Integrated Logic Analyzer (ILA)
    - Virtual I/O
    - MicroBlaze Debug Module

IP Customization

While attempts have been made to have similar parameter names for embedded IP as were available in XPS, you are encouraged to look at the data sheets for Vivado Design Suite IP to ensure proper customization. Figure 5-1, shows a snippet of an MHS file with parameter specifications. Figure 5-2 highlights appropriate instantiation options.

```
BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 8.50.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0xa8000000
 PARAMETER C_ICACHE_HIGHADDR = 0xafffffff
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_CACHE_BYTE_SIZE = 8192
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BASEADDR = 0xa8000000
 PARAMETER C_DCACHE_HIGHADDR = 0xafffffff
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_DCACHE_BYTE_SIZE = 8192
 PARAMETER C_DCACHE_ALWAYS_USED = 1
```

Figure 5-1: **Snippet from MHS File Showing Parameter Specification**

```
  # Create instance: microblaze_0, and set properties
  set microblaze_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze:9.2 microblaze_0 ]
  set_property -dict [ list CONFIG.C_DEBUG_ENABLED {1} CONFIG.C_D_AXI {1} CONFIG.C_D_LMB {1} CONFIG.
C_FAULT_TOLERANT {0} CONFIG.C_I_LMB {1} CONFIG.C_USE_INTERRUPT {0}  ] $microblaze_0
```

Figure 5-2: **Tcl Command to Instantiate a MicroBlaze Processor with Appropriate Options**

Design Connectivity

The IP integrator offers Designer Assistance to help you through the process of connecting up the design. Figure 5-3 shows an example from MHS, and Figure 5-4 shows the design assistance available in the IP integrator.

```
BUS_INTERFACE ILMB = microblaze_0_ilmb
BUS_INTERFACE DLMB = microblaze_0_dlmb
BUS_INTERFACE M_AXI_DP = axi4lite_0
BUS_INTERFACE M_AXI_DC = axi4_0
BUS_INTERFACE M_AXI_IC = axi4_0
BUS_INTERFACE DEBUG = microblaze_0_debug
BUS_INTERFACE INTERRUPT = microblaze_0_interrupt
PORT MB_RESET = proc_sys_reset_0_MB_Reset
PORT CLK = clk_100_0000MHzPLLE0
```

Figure 5-3: **Code Snippet from MHS File Showing Interface and Port Connectivity**

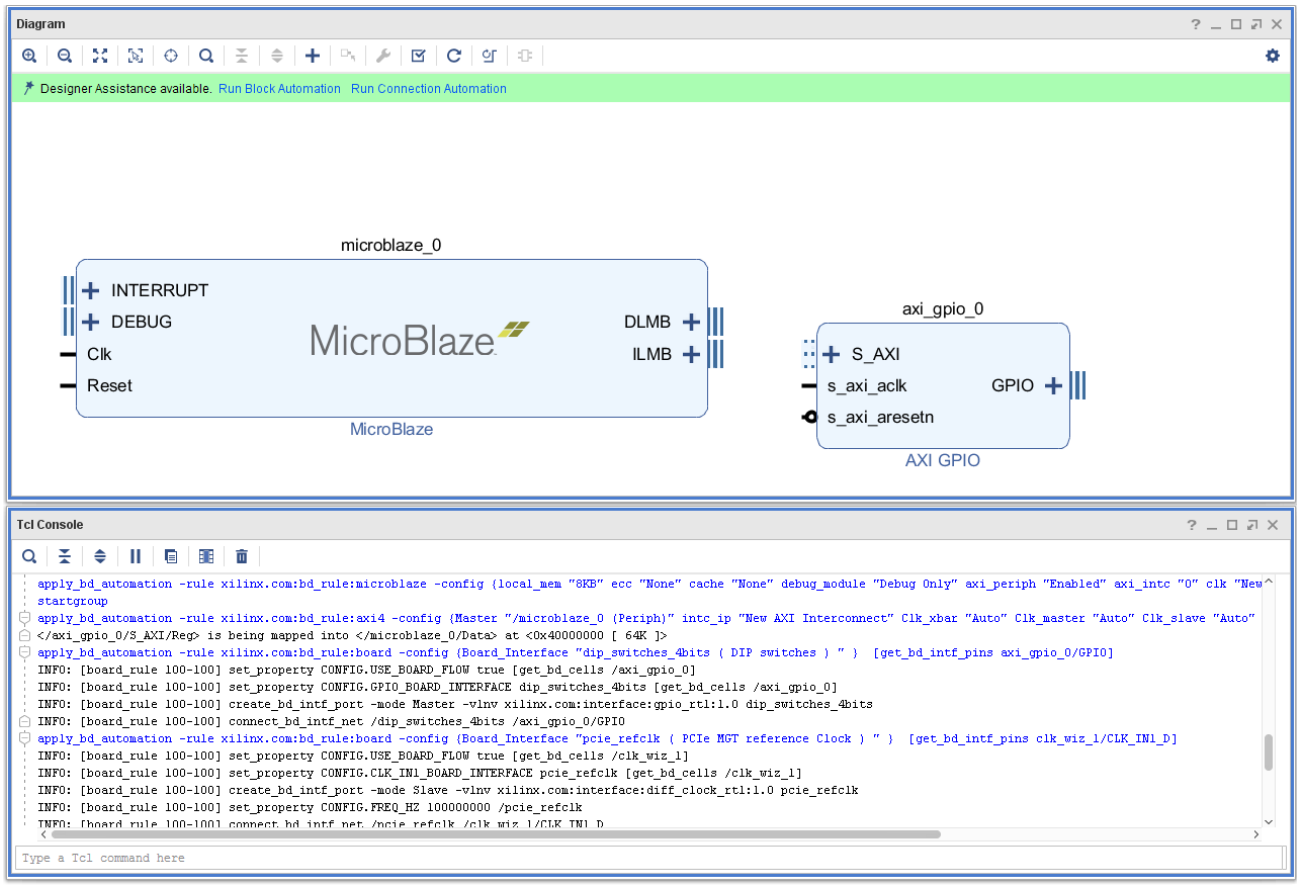

Figure 5-4: **Designer Assistance Available in GUI and the Corresponding Tcl Commands**

Address Mapping

In XPS, every slave had the same address, regardless of the Master accessing the Slave IP. The IP integrator provides support for master-based addressing. As a result, the same slave can have two different addresses as seen by two different masters. Figure 5-5 and Figure 5-6 illustrate this addressing change from MHS to IP integrator.

```
PARAMETER C_BASEADDR = 0x40600000
PARAMETER C_HIGHADDR = 0x4060FFFF
```

Figure 5-5: **MHS Snippet Showing Address Mapping in XPS**

```
# Create address segments
create_bd_addr_seg -range 0x10000 -offset 0x40000000 [get_bd_addr_spaces microblaze_0/Data] [get_bd_addr_segs axi_gpio_0/
s_axi/Reg] SEG_axi_gpio_0_Reg
create_bd_addr_seg -range 0x10000 -offset 0x40600000 [get_bd_addr_spaces microblaze_0/Data] [get_bd_addr_segs
axi_uartlite_0/s_axi/Reg] SEG_axi_uartlite_0_Reg
create_bd_addr_seg -range 0x8000 -offset 0x0 [get_bd_addr_spaces microblaze_0/Data] [get_bd_addr_segs
microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB/Mem] SEG_dlmb_bram_if_cntlr_Mem
create_bd_addr_seg -range 0x8000 -offset 0x0 [get_bd_addr_spaces microblaze_0/Instruction] [get_bd_addr_segs
microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB/Mem] SEG_ilmb_bram_if_cntlr_Mem
```

Figure 5-6: **Tcl Commands to Create Address Mapping in IP Integrator**

Clock and Resets

XPS provides a central clocking mechanism through use of the Clock Generator IP. The clock generator recognizes the clock requirements for all IP and generates the desired MMCM/PLL configurations as part of the IP.

In the IP integrator, use the `clocking_wizard` for clock configuration.

IMPORTANT: *You must enter the desired frequency as part of the clocking wizard IP. The properties of the generated clocks (such as frequency and phase) are propagated from the Clocking IP to the individual IP through use of the parameter propagation methods implemented by the IP.*

See the following documents for more information:

- *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898) [Ref 9]
- *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 10]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6]

Interconnect Configuration

The LogiCORE™ IP Advanced eXtensible Interface (AXI) Interconnect core connects one or more AXI memory-mapped master devices to one or more memory-mapped slave devices.

Setting up for Debug

You can mark a signal for debug by selecting a net in the block design, right-clicking, and selecting **Mark Debug** in the IP integrator. This preserves the nets marked for debug by putting the appropriate net-preserving attributes in the generated HDL code. Then the design can be synthesized, and debug core(s) can be inserted in the synthesized netlist. An ILA can also be instantiated in the Block Design and AXI interfaces or individual I/O ports can be hooked up to the ILA for monitoring later.

Zynq-7000 platform processor-based and MicroBlaze processor-based designs also support cross-trigger functionality. This essentially means that the processors have the ability to trigger and be triggered by the Vivado Integrated Logic Analyzer.

Refer to the following documents for more information:

- *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* (UG940) [Ref 8]
- *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 5]

Association of ELF Files

In a microprocessor-based design (such as a MicroBlaze processor design or Zynq-7000 device design containing a MicroBlaze processor), an ELF file generated in the Vitis™ software development platform (or in another software development tool) can be imported and associated with a block design in Vivado. You can then program the bitstream along with the ELF file from Vivado and run it on target hardware. The process is same between XPS and the IP integrator.

See the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 10] for more information.

Migrating Zynq-7000 SoC-Based Designs

The Vivado IDE uses the IP integrator tool for embedded development. A variety of IP are available in the Vivado IDE IP catalog to accommodate complex designs. You can also add custom IP to the IP catalog.

You can migrate a Zynq-7000 platform processor-based design into the Vivado Design Suite using the following steps.

1. Generate the system infrastructure.
   a. Create a Vivado project with the desired board or programmable device.
   b. In the Flow Navigator, click **IP Integrator** and select **Create Block Design**.
   c. Enter the Design name: **design_1**. This generates the block design.
2. Add the ZYNQ7 processing system and import the XML file from the XPS design.
   a. In the block diagram, right-click anywhere and select **Add IP** to open the IP Catalog.
   b. In the IP catalog, double-click **ZYNQ7 Processing System.** This instantiates a `processing_system7_0` instance on the block design.
   c. Double-click the **processing_system7_0** instance.
   d. At the top of the **Re-customize IP** dialog box, click **Import XPS Settings**.
   e. Click **browse** and select the directory of the XML file that was used for XPS.
   f. Click **OK**.

TIP: *Typically the XML file is located in the* `<XPS_Project>/data/ps7_system_prj.xml`. *The XML file stores information dealing with Zynq device Peripherals, MIO settings, DDR settings, and clocking including fabric clocks. You must enable AXI and other interfaces for Zynq devices manually.*

3. Open the MHS file and look at the `processing_system7` instance parameters and ports.
   a. In the Page Navigator, select the **PS-PL Configuration**.

b. Do a search in the MHS file and set the following options based on the options listed in Table 5-2.

Table 5-2: **PS/PL Configuration Options Settings**

Parameter or Port	Exists	Does Not Exist
C_USE_M_AXI_GP0 = 1	GP Master AXI Interface/M AXI GP0 Interface: **Selected**	GP Master AXI Interface/M AXI GP0 Interface: **Unselected**
C_USE_M_AXI_GP1 = 1	GP Master AXI Interface/M AXI GP1 Interface: **Selected**	GP Master AXI Interface/M AXI GP1 Interface: **Unselected**
C_USE_S_AXI_GP0 = 1	GP Slave AXI Interface/S AXI GP0 Interface: **Selected**	GP Slave AXI Interface/S AXI GP0 Interface: **Unselected**
C_USE_S_AXI_GP1 = 1	GP Slave AXI Interface/S AXI GP1 Interface: **Selected**	GP Slave AXI Interface/S AXI GP1 Interface: **Unselected**
C_USE_S_AXI_HP0 = 1	HP Slave AXI Interface/S AXI HP0 Interface: **Selected**	HP Slave AXI Interface/S AXI HP0 Interface: **Unselected**
C_USE_S_AXI_HP1 = 1	HP Slave AXI Interface/S AXI HP1 Interface: **Selected**	HP Slave AXI Interface/S AXI HP1 Interface: **Unselected**
C_USE_S_AXI_HP2 = 1	HP Slave AXI Interface/S AXI HP2 Interface: **Selected**	HP Slave AXI Interface/S AXI HP2 Interface: **Unselected**
C_USE_S_AXI_HP3 = 1	HP Slave AXI Interface/S AXI HP3 Interface: **Selected**	HP Slave AXI Interface/S AXI HP3 Interface: **Unselected**
C_USE_S_AXI_ACP = 1	ACP Slave AXI Interface/S AXI ACP Interface: **Selected**	ACP Slave AXI Interface/S AXI ACP Interface: **Unselected**
FCLK_CLKTRIG0_N	General/Enable Clock Triggers/FLCK_CLKTRIG0: **Selected**	General/Enable Clock Triggers/FLCK_CLKTRIG0: **Unselected**
FCLK_CLKTRIG1_N	General/Enable Clock Triggers/FLCK_CLKTRIG1: **Selected**	General/Enable Clock Triggers/FLCK_CLKTRIG1: **Unselected**
FCLK_CLKTRIG2_N	General/Enable Clock Triggers/FLCK_CLKTRIG2: **Selected**	General/Enable Clock Triggers/FLCK_CLKTRIG2: **Unselected**
FCLK_CLKTRIG3_N	General/Enable Clock Triggers/FLCK_CLKTRIG3: **Selected**	General/Enable Clock Triggers/FLCK_CLKTRIG3: **Unselected**
FCLK_RESET0_N	General/Enable Clock Resets/FCLK_RESET0_N: **Selected**	General/Enable Clock Resets/FCLK_RESET0_N: **Unselected**
FCLK_RESET1_N	General/Enable Clock Resets/FCLK_RESET1_N: **Selected**	General/Enable Clock Resets/FCLK_RESET1_N: **Unselected**

4. In the Page Navigator, select **Clock Configuration**. Search the MHS file and set the following options based on the selections in Table 5-3.

   The **Requested Frequencies** are set automatically based upon the imported XML file.

Table 5-3: **CLock Configurations**

Port	Exists	Does Not Exist
FCLK_CLK0	PL Fabric Clocks/FCLK_CLK0: **Selected**	PL Fabric Clocks/FCLK_CLK0: **Unselected**
FCLK_CLK1	PL Fabric Clocks/FCLK_CLK1: **Selected**	PL Fabric Clocks/FCLK_CLK1: **Unselected**
FCLK_CLK2	PL Fabric Clocks/FCLK_CLK2: **Selected**	PL Fabric Clocks/FCLK_CLK2: **Unselected**
FCLK_CLK3	PL Fabric Clocks/FCLK_CLK3: **Selected**	PL Fabric Clocks/FCLK_CLK3: **Unselected**

5. If interrupts are used:
   a. In the Page Navigator, select **Interrupts**.
   b. Check **Fabric Interrupts** and select the interrupts used by the Zynq device. With BSB designs, `IRQ_F2P[15:0]` under PL-PS Interrupts Ports were used. Check **IRQ_F2P[15:0]** under PL-PS Interrupt Ports.
6. In the Re-customize IP dialog box, click **OK** to save the imported settings.

ZYNQ7 Processing System Block Automation

1. Click **Run Block Automation** for `/processing_system7_0`. This makes the proper external connections for the top level of the design.
2. Click **Apply Board Preset** to match the board, if applicable.
3. Click **OK**.

Connect Fabric Clocks to processing_system7 instance

Open the MHS in a text editor to determine if AXI interfaces clocks were connected for Design Automation to function properly. A search for these clocks in the MHS includes:

```
M_AXI_GP0_ACLK          S_AXI_GP1_ACLK          S_AXI_HP1_ACLK
M_AXI_GP1_ACLK          S_AXI_ACP_ACLK          S_AXI_HP2_ACLK
S_AXI_GP0_ACLK          S_AXI_HP0_ACLK          S_AXI_HP3_ACLK
```

Typically, these clocks are connected to `FCLK_CLK(0-3)` on the `processing_system7` instance. If not, you must connect them to the external `clk port` or `clk_wiz` in the design that matches the configuration in the MHS. Connect these clocks based on connections in the MHS file.

In addition, if DMA Controller Peripheral request interface is included in the `processing_system7` instance in the MHS, connect the following clocks, if applicable: `DMA0_ACLK, DMA1_ACLK, DMA2_ACLK, DMA3_ACLK`.

Typically these clocks are connected to `FCLK_CLK(0-3)` on the `processing_system7` instance. If not, you must connect these clocks to the external `clk port` or `clk_wiz` in the design that matches the configuration in the MHS.

Adding IP to the Base Design and Design Automation

This section covers adding AXI IP to the design. Design Automation provides (1) connections to Proc Sys Reset IP for reset capability and (2) proper clocks to the IP AXI Interface and to the generated AXI Interconnect instance. With the ZYNQ7 Processing System Block, there can be up to 9 AXI3 Interfaces (2xAXI3 Master interfaces/6xAXI3 Slave interfaces).

Adding AXI Slave IPs (AXI4-Lite and AXI4) Example

Add the equivalent IP to the block diagram. The following are example steps for AXI GPIO:

1. Right-click anywhere in the block diagram and select **Add IP**.
2. Search for and double-click **AXI GPIO** to add the IP.
3. Double-click the instance (`axi_gpio_0`) to configure the IP.
4. Open the MHS and match the settings as closely as possible; parameters could have been added, modified, or removed.
5. If possible, select **Generate Board based IO Constraints** to set parameters based upon the board.
6. Click **OK**.

   Note: Parameter verification is discussed in the section Verifying Parameters between XPS and IP Integrator Designs for AXI Masters and Slaves in a Zynq Platform Processor-Based Design.

7. In the MHS, in the AXI slave instance section, find the `BUS_INTERFACE S_AXI` line. Note that `S_AXI` could have a slightly different name.
8. Based on the right side of the equation on that line (`BUS_INTERFACE S_AXI = AXI_INTERCONNECT_GP0_MASTER` in this case `AXI_INTERCONNECT_GP0_MASTER`), do a search and trace back into the `processing_system_7` instance.

   Consider, for example, `BUS_INTERFACE M_AXI_GP0 = AXI_INTERCONNECT_GP0_MASTER`.

   This AXI Slave interface is connected to the `M_AXI_GP0` interface. AXI Slave interfaces for Zynq devices can be connected to `M_AXI_GP0`, `M_AXI_GP1`, depending on the connection in the MHS.
9. Run **Connection Automation** on the AXI Slave interface(s) on the IP. Consider, for example, the `/axi_gpio_0/S_AXI` interface. For the `/axi_gpio_0/S_AXI` interface, the interface connection should be for the Master: `/processing_system7_0/M_AXI_GP0`.
10. Select **OK**.

    This creates or modifies the `processing_system7_0_axi_periph` instance.

For other clocks and resets, and for other internal or external signals, the method for making connections is similar to the method used in XPS designs. If possible, use the Connection Automation on external interfaces such as the `/axi_gpio_0/GPIO` interface.

Adding AXI Master IP (AXI4-Lite and AXI4) Example

Add the equivalent IP to the block diagram. The following are example steps for AXI Central Direct Memory Access (CDMA):

1. Right-click anywhere in the block diagram and select **Add IP**.
2. Search for and double-click **AXI Central Direct Memory Access**.

   The AXI Masters (SG engine and AXI4 Data Master) on the IP is connected only in this section.
3. Double-click the instance (`axi_cdma_0`) to configure the IP.
4. Open the MHS and match the settings as closely as possible; parameters could have been added, modified or removed.
5. Click **OK**.

   Note: Parameter verification is discussed in the section Verifying Parameters between XPS and IP Integrator Designs for AXI Masters and Slaves in a Zynq Platform Processor-Based Design.
6. In the MHS, in the AXI Master instance section (`axi_cdma_0` instance), find the `BUS_INTERFACE M_AXI = AXI_INTERCONNECT_HP0_SLAVE` line.

Note: `M_AXI` could have a slightly different name.

7. Based upon the right side of the equation on that line (`BUS_INTERFACE M_AXI = AXI_INTERCONNECT_HP0_SLAVE`, in this case, `AXI_INTERCONNECT_HP0_SLAVE`) do a search and trace back into the `processing_system_7` instance.

   For example, `BUS_INTERFACE S_AXI_HP0 = AXI_INTERCONNECT_HP0_SLAVE`.

   This AXI Master interface is connected to the `S_AXI_HP0` interface.

   AXI Master interfaces for the Zynq-7000 device are connected to `S_AXI_GP0`, `S_AXI_GP1`, `S_AXI_HP0`, `S_AXI_HP1`, `S_AXI_HP2`, `S_AXI_HP3`, `S_AXI_ACP`, depending on the connection in the MHS.

8. If the Zynq device AXI slave interface does not have an associated AXI Interconnect (for example on first time running Connection Automation on this interface), run **Connection Automation** for the Zynq-7000 device AXI slave interface.

   For this example the AXI slave interface is: `/processing_system_7_0/S_AXI_HP0`.

   The AXI Master `/axi_cdma_0/M_AXI` or `/axi_cdma_0/M_AXI_SG` displays in the Run Connection Automation dialog box.

9. Select `/axi_cdma_0/M_AXI`.
10. Click **OK**.
11. If the Zynq device AXI Slave interface does contain an AXI Interconnect instance, run **Connection Automation** on the AXI Master interface (for this example, `/axi_cdma_0/M_AXI_SG`).
12. Select `/processing_system7_0/S_AXI_HP0`.
13. Click **OK**.

    The Connection Automation creates or modifies the `axi_mem_intercon` instance for the AXI Interconnect.

    For other clocks and resets, and for other internal or external signals, the method for making connections is similar to the method used in XPS designs.

14. If possible, use **Run Connection Automation** on external interfaces.

Connecting Interrupts

IMPORTANT: *For each IP that needs interrupt support: make sure that interrupt support is enabled by a parameter in the IP GUI.*

1. Right-click anywhere in the block diagram and select **Add IP**.
2. Search for and double-click `Concat` to add the IP.

3. Double-click the `xlconcat_0` instance.
4. Modify the number of ports to match the number of interrupts in the design.
5. Click **OK**.
6. Connect `dout[1:0]` on the `xlconcat_0` instance to `IRQ_F2P[0:0]` on the `processing_system7_0` instance.
7. Connect the interrupt from the IP to the `xlconcat_0 InX` input port. For example, the `ip2intc_irpt` on the `axi_gpio_0` instance to the `In0[0:0]` port on the `xlconcat_0`.

Use this method to connect each interrupt in the design.

TIP: *The* `In0[0:0]` *port is the lowest interrupt priority, and in the MHS, the left-most signal connected for* `PORT IRQ_F2P` *for the* `processing_system_7` *instance is the lowest interrupt priority.*

Customizing Addresses to Match the XPS Design

It is necessary to match addresses for the design because **Design Automation** sets the addresses and address size. This ensures compatibility with existing software.

1. Open the MHS file in a text editor, and select **Address Editor** in the block design.
2. For each AXI slave in the MHS, obtain the `PARAMETER C_BASEADDR` and `PARAMETER C_HIGHADDR`.
   a. Enter the `C_BASEADDR` value in the **Offset Address** field for the AXI slave under `processing_system7_0/Data`.
   b. Ensure the `C_HIGHADDR` matches the **High Address** column for the AXI slave.

      If there is a mismatch, adjust the **Range** column for the AXI slave to match the `C_HIGHADDR` with the Zynq-7000 device AXI Master Interfaces (`M_AXI_GP0/M_AXI_GP1`). Address spaces are between 0x40000000 to 0x7FFFFFFF and 0x80000000 to 0xBFFFFFFF, respectively.

      Note: For Zynq-7000 device AXI slave interfaces, address ranges are set automatically between 0x00000000 and 0x3FFFFFFF.

Strategies

By default, the **Interconnect Optimization Strategy** is set to **Custom**. This lets you add register-slicing data FIFOs on every master and slave interface. However, full crossbar support is enabled (masters have direct connection to all slaves). AXI Master issuance is set to 2, and AXI Slave acceptance is set to 4.

IMPORTANT: *If the XPS design includes sparse crossbar support (certain masters access a subset of slaves), be aware that this feature is not available in IP integrator. Each master connects to all the slaves.*

Use the **Minimize Area** strategy if a portion of the design only connects AXI4-Lite slave peripherals.

This puts the AXI Interconnect in a Shared Access mode (masters share connection to slaves), which reduces system resources for the AXI Interconnect and sets the **AXI Master Issuance** to 1 and **AXI Slave Acceptance** to 1. Use this option with the `processing_system7_0_axi_periph` or `processing_system7_0_axi_periph_1` instance.

IMPORTANT: *If a high-speed AXI slave, such as AXI MIG or AXI block RAM , is connected to the AXI Interconnect instance, leave the strategy at* ***Custom****.*

Use the **Maximize Performance** strategy with high-performance portions of the design. This adds 512-deep FIFOs for every master, as well as setting **AXI Master Issuance** to 4 and **AXI Slave Acceptance** to 4. Leave the `axi_mem_intercon` instances at **Custom**, unless all AXI Masters need Packet AXI Data FIFOs. These are discussed in the following subsections.

Setting AXI Master(s) Register Slicing and AXI Data FIFO

Open the MHS file, and for each AXI Master, set the Enable Register Slice/Enable Data FIFO. To determine the correct settings, search the MHS using the information in Table 5-4. When doing the search, replace `<intf_name>` with the associated `BUS_INTERFACE` name.

For example, the `<intf_name>` for `BUS_INTERFACE M_AXI_MM2S` would be `M_AXI_MM2S`, and the `<intf_name>` for `BUS_INTERFACE M_AXI_GP0` would be `M_AXI_GP0`.

Notes:

- There could be multiple AXI master interfaces per IP.
- AXI masters connect to the AXI interconnect slave connection. Make the selection in the **Slave Interfaces** tab.

Table 5-4: **AXI Master Interconnect Settings with Zynq Devices**

Parameter	Exists	Does Not Exist
C_INTERCONNECT_<intf_name>_AR_REGISTER C_INTERCONNECT_<intf_name>_R_REGISTER C_INTERCONNECT_<intf_name>_AW_REGISTER C_INTERCONNECT_<intf_name>_W_REGISTER C_INTERCONNECT_<intf_name>_B_REGISTER	SXX_AXI: Enable Register Slice **Auto**	SXX_AXI: Enable Register Slice **None**
C_INTERCONNECT_<intf_name>_WRITE_FIFO_DEPTH C_INTERCONNECT_<intf_name>_READ_FIFO_DEPTH	Parameter = 32 SXX_AXI: Enable Data FIFO **32 deep** Parameter = 512 SXX_AXI: Enable Data FIFO **512 deep**	SXX_AXI: Enable Data FIFO **None**

Setting AXI Slave(s) Register Slicing and AXI Data FIFO

Open the MHS file, and for each AXI slave, set the **Enable Register Slice/Enable Data FIFO**. To determine the correct settings, search the MHS using the information in Table 5-5.

Replace the <intf_name> with the associated BUS_INTERFACE name. For example, the <intf_name> for BUS_INTERFACE S_AXI would be S_AXI, and the <intf_name> for BUS_INTERFACE S_AXI_HP0 would be S_AXI_HP0.

Notes:

- There could be multiple AXI slave interfaces per IP.
- AXI slaves connect to the AXI Interconnect master connection. You make the selection in the Master Interfaces tab.

Table 5-5: **AXI Slave Interconnect Settings with Zynq Devices**

Parameter	Exists	Does Not Exist
C_INTERCONNECT_<intf_name>_AR_REGISTER C_INTERCONNECT_<intf_name>_R_REGISTER C_INTERCONNECT_<intf_name>_AW_REGISTER C_INTERCONNECT_<intf_name>_W_REGISTER C_INTERCONNECT_<intf_name>_B_REGISTER	MXX_AXI: Enable Register Slice **Auto**	MXX_AXI: Enable Register Slice **None**
C_INTERCONNECT_<intf_name>_WRITE_FIFO_DEPTH C_INTERCONNECT_<intf_name>_READ_FIFO_DEPTH	Parameter = 32 MXX_AXI: Enable Data FIFO **32 deep** Parameter = 512 MXX_AXI: Enable Data FIFO **512 deep**	MXX_AXI: Enable Data FIFO **None**

Validating the Design

Click the **Validate Design** button to help fix issues with the design that could include signals, connections, or other issues.

Verifying Parameters between XPS and IP Integrator Designs for AXI Masters and Slaves in a Zynq Platform Processor-Based Design

1. Ensure that the XPS design has generated a netlist. This creates top-level IP wrapper files for every IP in the `<xps_project>/hdl` directory.

   For example, `<xps_project>/hdl /system_leds_4bits_wrapper.vhd` contains the wrapper for `AXI_GPIO` in the EDK design. The important portions of the wrapper are the mappings to parameters, as shown below.

   ```
   C_GPIO_WIDTH => 4,
   C_GPIO2_WIDTH => 32,
   C_ALL_INPUTS => 0,
   C_ALL_INPUTS_2 => 0,
   C_INTERRUPT_PRESENT => 1,
   C_DOUT_DEFAULT => X"00000000",
   C_TRI_DEFAULT => X"ffffffff",
   C_IS_DUAL => 0,
   C_DOUT_DEFAULT_2 => X"00000000",
   C_TRI_DEFAULT_2 => X"ffffffff"
   ```

2. In the Tcl Console, get the properties for the IP. For example, in the Tcl Console use the following command:

   ```
   report_property [get_bd_cells axi_gpio_0]
   ```

   From the report in the Tcl Console, the important lines contain `C_`, which are for parameters. The following example shows lines that are used for comparison.

   ```
   CONFIG.C_ALL_INPUTS           string  false      true     0
   CONFIG.C_ALL_INPUTS_2         string  false      true     0
   CONFIG.C_ALL_OUTPUTS          string  false      true     1
   CONFIG.C_ALL_OUTPUTS_2        string  false      true     0
   CONFIG.C_DOUT_DEFAULT         string  false      true     0x00000000
   CONFIG.C_DOUT_DEFAULT_2       string  false      true     0x00000000
   CONFIG.C_GPIO2_WIDTH          string  false      true     32
   CONFIG.C_GPIO_WIDTH           string  false      true     4
   CONFIG.C_INTERRUPT_PRESENT    string  false      true     1
   CONFIG.C_IS_DUAL              string  false      true     0
   CONFIG.C_TRI_DEFAULT          string  false      true     0xFFFFFFFF
   CONFIG.C_TRI_DEFAULT_2        string  false      true     0xFFFFFFFF
   ```

   Comparing the parameters, all parameter values match except `C_ALL_OUTPUTS` and `C_ALL_OUTPUTS_2`, which are new parameters on the latest AXI GPIO IP for IP integrator.

3. When setting these parameters, look for parameter additions or differences in the parameter values of the appropriate IP Product Specification.

If changes to parameters are necessary:

Double-click the IP instance in the block diagram and make the proper parameter changes to update the XCI file.

If you cannot find the parameters in the GUI:

1. Select the IP in the block diagram.
2. Select the **Properties** tab under Block Properties.
3. Expand **Config** and enter the value for the parameter.
4. Redo the `report_property` command and modify if necessary.
5. Click **Validate Design** to ensure no issues exist with the design.

Finishing the Design

1. In the Sources tab, right-click `design_1` and select **Generate Output Products**.
2. Click **Generate**.
3. In the Sources tab, after generation, right-click `design_1` and select **Create HDL Wrapper.**
4. Select **OK** for Vivado to manage the top-level wrapper.
5. Create an XDC file that locks down (1) pins in the design that are not board-related or (2) additional constraints for MIG. For example, the XDC file might lock down the location for the Reset Pin and `DCI_CASCADE` settings for MIG.

Migrating a MicroBlaze Processor-Based Design

Generating System Infrastructure (MicroBlaze, AXI_Interconnect, Clk_Wiz, Proc_Sys_Reset)

To generate a system infrastructure (MicroBlaze, AXI_Interconnect, Clk_Wiz, and Proc_Sys_Reset):

1. Create a Vivado project with the desired board or programmable device.
2. In the **Flow Navigator**, click **IP Integrator** and select **Create Block Design**.
3. Enter the Design name: for example: **design_1**.

   This generates the block design.

IMPORTANT: *If MIG is in the design, follow the steps on Migrating AXI MIG before proceeding further.*

Determining MicroBlaze Interfaces/Base System Configuration

1. Open the MHS file for the XPS design in a text editor.
2. Determine which interfaces are used on MicroBlaze. Search the MHS file for `BUS_INTERFACE ILMB`, `BUS_INTERFACE DLMB`, `BUS_INTERFACE M_AXI_DP`, `BUS_INTERFACE M_AXI_DC`, `BUS_INTERFACE M_AXI_IC`, `BUS_INTERFACE DEBUG`, `BUS_INTERFACE INTERRUPT`.

   Typically, MicroBlaze designs built by the Base System Builder (BSB) contain LMB interfaces for local block RAM,

   MicroBlaze Data Port interface (for slave registers, such as `AXI GPIO`), debug interface (using MDM with or without UART based upon the `C_USE_UART` parameter in MDM IP), and interrupt support (using the `AXI_INTC` slave IP). In addition, BSB adds clock and reset support with the `clock_generator` and `proc_sys_reset` IP.

   More complex interfaces use the MicroBlaze Instruction Cache Port and MicroBlaze Data Cache Port. Typically, these interfaces are used for high performance portions of the design that use AXI MIG or AXI block RAM (AXI4 slaves). The `C_CACHE_BYTE_SIZE` and `C_DCACHE_BYTE_SIZE` parameters determine the size of caches for the MicroBlaze MHS instance.

3. Refer to Table 5-6 and set MicroBlaze Block Automation, based on the interfaces in the MHS file for the MicroBlaze instance.

Table 5-6: **Bus Interfaces Used in XPS MicroBlaze Design**

Interface	Exists	Does Not Exist
BUS_INTERFACE ILMB BUS_INTERFACE DLMB	Local Memory: Select KB size based upon `C_BASEADDR/C_HIGHADDR` for LMB BRAM if a CNTLR instance is in the MHS	Local Memory: **None**
BUS_INTERFACE DEBUG	`C_USE_UART = 0` on MDM instance in MHS Debug Module: Debug Only `C_USE_UART = 1` on MDM instance in MHS Debug Module: Debug and UART	Debug Module: **None**
BUS_INTERFACE M_AXI_DP	Peripheral AXI Port: Checked	Peripheral AXI Port: **Disabled**
BUS_INTERFACE M_AXI_IC BUS_INTERFACE M_AXI_DC	Cache Configuration: Select cache size in KB based upon `C_CACHE_BYTE_SIZE` or `C_DCACHE_BYTE_SIZE` for MicroBlaze Instance in MHS	Cache Configuration: **None**
BUS_INTERFACE INTERRUPT	Interrupt Controller: Checked	Interrupt Controller: **Unchecked**

MicroBlaze Block Automation

MicroBlaze Block Automation is similar to BSB, in that it lets you add portions of the design automatically. The automation can:

- Enable the MicroBlaze Cache Interface (sets only the cache size; no other connections are made).
- Add:
  - Local Memory (LMB)
  - A Debug Module, with or without UART
  - An AXI Interconnect for AXI4Lite slave peripherals
  - An Interrupt Controller (`AXI INTC`)
  - Clock connections from other IP
  - External clock or `clk wiz`, and resets, using the `proc_sys_reset` IP for the design.

Running MicroBlaze Block Automation

1. In the block design, add the MicroBlaze IP.
2. Select **Run Block Automation** for `MicroBlaze_0`.
3. In the Run Block Automation dialog box, make selections based on the information provided in Table 5-6.

TIP: *The AXI Interconnect instance for AXI4Lite slave peripherals is called* `microblaze_0_axi_periph` *if Interrupt or UART is enabled for Debug from the table. Connection Automation adds this AXI Interconnect instance when connecting the first AXI Slave (typically AXI4Lite interface).*

4. For the Clock Connection, select **Clocking Wizard** or **New External Port**, depending on the connections in XPS.
5. If MIG is used:
   a. Instantiate and configure the MIG.
   b. Select the clock (typically, the 100 MHz clock) that is determined for the MicroBlaze clock.

Using CLK Wiz/Proc Sys Reset

IMPORTANT: *Follow the steps in this section only if AXI MIG is* not *included in the design.*

Connecting Interfaces for the Clocking Wizard

Use the following steps to design with a platform board selected as the part or to design with a part selected, as appropriate to your project.

Designing with a platform board selected as the part:

1. From the Board tab drag and drop the desired clock, such as the **System differential clock**, for the KC705 board on the block design canvas.

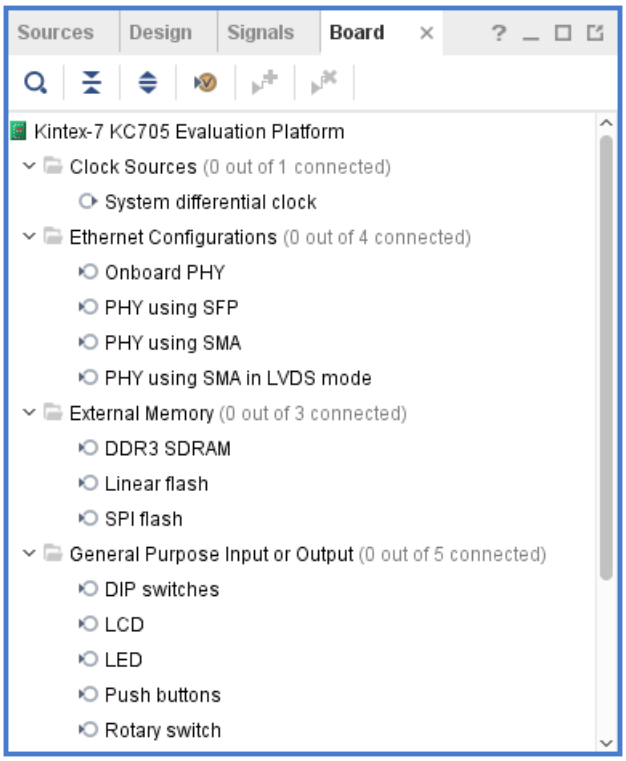

Figure 5-7: **Board Tab**

2. Likewise drag and drop the **FPGA Reset** from the Board tab to the block design canvas**.**

   For more information on the platform board flow, refer to this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 10].

Designing with a part selected:

Manually configure the clocking wizard based on the board in use:

1. Double-click **clk_wiz_1**.

2. On the **clk_wiz_1** instance, right-click **CLK_IN1_D** and select **Make External**.
3. Right-click **reset** and select **Make External**.
4. Connect the reset external pin to **ext_reset_in** on **proc_sys_reset_1**.

Using AXI MIG/Proc Sys Reset

IMPORTANT: *Perform the following steps only if AXI MIG is included in your design.*

Connecting Interfaces for AXI MIG

1. Select **Run Connection Automation** on `/mig_7series_0/S_AXI`.
2. Select the `/microblaze_0 (Cached)` or `/microblaze_0 (Periph)` option:
   a. Select the `/microblaze_0 (Cached)` option if it is available.
   b. If the `/microblaze_0 (Cached)` option is *not* available, it means the design does not contain caching. In this case, select `/microblaze_0 (Periph)`.

   Note: When you select `/microblaze_0 (Cached)`, another AXI Interconnect instance called `axi_mem_intercon` is generated.

Migrating AXI MIG

IMPORTANT: *Before performing the AXI MIG migration steps, be sure the XPS project has generated a netlist using XPS 14.7.*

1. Copy `<EDK_PROJECT>/__xps/<MIG_INST_NAME>/mig.prj` to another location.
2. In a text editor, open the `mig.prj` file.
3. Make sure that the `<TargetFPGA>` section matches the device settings in the Vivado project. If the settings do not match, modify the package and the speed appropriately.
4. If using DDR3, do a global replace of `ddr_` with `ddr3_`.
5. If using DDR2, do a global replace of `ddr_` with `ddr2_`.

Adding MIG to the Block Diagram

1. In the block diagram, add the **Memory Interface Generator** IP.
2. Click the `mig_7series_0` instance.

Configuring AXI MIG

1. In the Block Properties, click the Properties tab, expand **CONFIG**, and scroll down to **XML_INPUT_FILE**.
2. Provide the absolute path of the modified `mig.prj` file and press **Enter**.

The tool loads the XML and provides error messages if the PRJ file is incorrect.

TIP: *It might take a 30 seconds or so for this operation to complete.*

3. Double-click the `mig_7series_0` instance in the block diagram to verify the MIG settings.

   Because the clocking/reset for the memory controller is handled in the IP, MIG must configure the input and output clocks and reset for the design.

4. Verify that the **Options** for Controller 0 are correct (such as memory speed and memory selection). If the settings do not look correct for the board memory, change them.
5. Verify the **AXI Parameter Options**, C0, are correct. Ensure that **Narrow Burst Support** is set correctly, based on the XPS design.If you are unsure, set this parameter to 1.
6. In the **Memory Options** for Controller 0, configure the input clocking/reset.
   a. For the **Input Clock Period**, select the differential or single-ended clock input frequency, based upon the board type. On the KC705 board, the correct setting is **5000 ps (200 MHz)**.
   b. If the existing XPS design contained MicroBlaze or other IP, additional fabric clocks are needed. Click the **Select Additional Clocks** check box.

      Note: Typically, additional clocks are not needed if the original XPS design was created using BSB.

   c. Typically, a MicroBlaze device design has 100 MHz clocks. For **Clock 0**, select **10000 ps (100.00000 MHz)**. This is used for the MicroBlaze portion of the design.
   d. Verify the other options for the memory controller.
   e. Click **Next**.

   The system clock selects the method used to deliver the input clock to the memory controller. The KC705 board uses a Differential clock input.

7. Select **Differential** for the System Clock.

   You can generate the reference clock from the internal MMCM if it can generate a 200 MHz clock.

8. Select **Use System Clock** for the Reference Clock.
9. Set the **System Reset Polarity** based upon the board type (active-Low or active-High).
   a. Select the appropriate setting for the board in use. Active-High is correct for the KC705 board.
   b. Verify the other options for the memory controller.
   c. Click **Next**.

10. Keep validating settings and clicking **Next** until the Pin/Bank Selection Mode dialog box opens.
    a. In the Pin/Bank Selection Mode dialog box, select **Fixed Pin Out** and click **Next**.

       The pinout is already defined if the MIG settings are correct for the memory, and the project file was modified correctly.

    b. Select **Validate**.
    c. After the pinout is validated correctly, click **OK**, even if `INFO` messages exist.
    d. Click **Next**.
11. In the **System Signals Selection**
    a. Select the **sys_clk_p/n** pins that are used on the board. (The KC705 uses Bank Number 33 Pin AD12/AD11.) The `clk_ref` is already connected to an internal signal.
    b. Connect the Status Signals as needed, based on the design.
12. Click **Next** until you reach the Memory Model License agreement.
    a. Accept the Memory Model License agreement.
    b. Click **Generate**.

Connecting AXI MIG Interfaces

Right-click the following interfaces in the block diagram and select **Make External** for each:

- `SYS_CLK`
- `sys_rst`
- `DDR3`

This provides all DDR3 signals, clocks and resets to be connected to the top-level board pins.

See the MicroBlaze Block Automation section for instructions on connecting AXI MIG to the rest of the design.

Adding IP to the Base Design

This section covers adding AXI IP to the design.

Adding Low Speed Peripherals (AXI4-Lite)

1. Add the equivalent IP to the block diagram. For example, for AXI GPIO:
   a. Right-click anywhere in the block diagram and select **Add IP**.

b. Search for, and double-click **AXI GPIO** to add the IP.

2. Double-click the instance (for example, `axi_gpio_0`) to configure the IP. Open the MHS and match the settings as closely as possible. Parameters might have been added, modified or removed. If possible, apply the peripheral IP settings (Board-based I/O Constraints) based upon the board (such as `led_8bits`).
3. Click **OK**.

Note: Parameter verification is discussed in Verifying Parameters between XPS and IP Integrator Designs for AXI Masters and Slaves in a MicroBlaze Processor-Based Design.

4. Run **Connection Automation** on the slave interface of the IP. For example, the `/axi_gpio_0/S_AXI` interface.
5. Connect to the proper AXI Interconnect instance. This would be `/microblaze_0` (Periph), using the `microblaze_0_axi_periph` instance.
6. Connect any other clocks, resets, or other signals. The method for external signal connection is similar to that used for XPS designs. If possible, use Connection Automation on external interfaces. For example, the `/axi_gpio_0/GPIO` interface.

Adding High Speed Peripherals (AXI4)

1. Add the equivalent IP to the block diagram. For example, to add AXI block RAM:
   a. Right-click anywhere in the block diagram and select **Add IP**.
   b. Search for, and double-click **AXI BRAM Controller** to add the IP.
2. Double-click the instance (for example, **axi_bram_ctrl_0**) to configure the IP.
3. Open the MHS and match the settings best as closely possible. Parameters might have been added, modified, or removed.
4. Click **OK**.

Note: Parameter verification is discussed Verifying Parameters between XPS and IP Integrator Designs for AXI Masters and Slaves in a MicroBlaze Processor-Based Design.

5. Run **Connection Automation** on the AXI master or AXI slave interface on the IP. For example, the `/axi_bram_ctrl_0/S_AXI` interface.
6. Connect to the proper AXI Interconnect instance. This should be `/microblaze_0 (Cached)`, using the `axi_mem_intercon` instance.
7. Connect any other clocks, resets, or other signals.

The method for external signal connection is similar to that used for XPS designs.

8. If possible, use **Connection Automation** on external interfaces.

   For example, the `/axi_bram_ctrl_0/BRAM_PORTA` and `/axi_bram_ctrl_0/BRAM_PORTB` interfaces. Typically, a True Dual Port RAM is used for `BRAM_PORTA` and `BRAM_PORTB`.

Connecting Interrupts

In the IP, make sure that interrupt support is enabled by a parameter for each IP where it is necessary.

1. Double-click the **microblaze_0_xlconcat** instance created by Block Automation.
2. Modify the number of ports to match the number of interrupts in the design.
3. Click **OK**.
4. Connect the interrupt from the IP to the `microblaze_0_xlconcat InX` input port. For example, connect the `ip2intc_irpt` on the `axi_gpio_0` instance to the `In0[0:0]` port on the `microblaze_0_xlconcat`.
5. Repeat step 4 for each interrupt in the design.

Note: The In0[0:0] port is the lowest interrupt priority and in the MHS, the left-most signal connected for `PORT INTR` for the `axi_intc` instance is the lowest interrupt priority.

Customizing Addresses to Match the XPS Design

IMPORTANT: *You must perform the steps in this section to match addresses for the design because Design Automation sets the addresses and address size. These steps ensure compatibility with existing software.*

1. In a text editor, open the MHS file and select **Address Editor** in the block design.
2. For each AXI slave or LMB block RAM in the MHS, obtain:
   - `PARAMETER C_BASEADDR`
   - `PARAMETER C_HIGHADDR`
3. Enter the `C_BASEADDR` value under the **Offset Address** for the AXI slave or LMB block RAM.
4. Ensure the `C_HIGHADDR` matches the **High Address** column for the AXI slave or LMB block RAM. If there is a mismatch, adjust the **Range** column for the AXI Slave or LMB block RAM to match the `C_HIGHADDR`.

   If there are multiple AXI masters connected to an AXI slave, be sure to change the address on the multiple AXI masters for the **Offset Address**. For example, `microblaze_0/Data` and `microblaze_0/Instruction`.

Strategies

By default, the Interconnect Optimization Strategy is set to **Custom**. This lets you add register-slicing Data FIFOs on every master/slave interface; however, if the full crossbar is enabled (masters have direct connection to all slaves), AXI Master issuance is set to 2, and AXI Slave acceptance is set to 4.

IMPORTANT: *If your XPS design included Sparse Crossbar support (certain masters access a subset of slaves), be aware that this feature is not available in the IP Integrator. In the IP Integrator, each master connects to all the slaves.*

Use the **Minimize Area** strategy if a portion of the design connects only AXI4-Lite slave peripherals. This puts the AXI Interconnect in a Shared Access mode (masters share connection to slaves), which reduces system resources for the AXI Interconnect and sets the AXI Master issuance to 1 and AXI Slave acceptance to 1. Use this option with the `microblaze_0_axi_periph` instance.

Use the **Maximize Performance** strategy with high-performance portions of the design. This adds 512-deep FIFOs for every master, sets AXI Master issuance to 4, and AXI Slave acceptance to 4. Leave the `axi_mem_intercon` set to Custom unless all AXI Masters need Packet AXI Data FIFOs.

IMPORTANT: *The AXI Interconnect in the IP integrator does not support sparse connectivity.*

Setting AXI Master(s) Register Slicing and AXI Data FIFO

1. Open the MHS file, and for each AXI Master set the **Enable Register Slice/Enable Data FIFO** based upon searching the MHS in Table 5-7.

   When doing the search, replace `<intf_name>` with the associated `BUS_INTERFACE` name. For example, `<intf_name>` for `BUS_INTERFACE M_AXI_MM2S` would be `M_AXI_MM2S`.

TIP: *AXI Masters connect to the AXI Interconnect Slave connection. You can make the selection in the Slave Interfaces tab.*

Table 5-7: **AXI Master Interconnect Settings with MicroBlaze Devices**

Parameter	Exists	Does Not Exist
C_INTERCONNECT_<intf_name>_AR_REGISTER C_INTERCONNECT_<intf_name>_R_REGISTER C_INTERCONNECT_<intf_name>_AW_REGISTER C_INTERCONNECT_<intf_name>_W_REGISTER C_INTERCONNECT_<intf_name>_B_REGISTER	`SXX_AXI`: Enable Register Slice **Auto**	`SXX_AXI`: Enable Register Slice **None**
C_INTERCONNECT_<intf_name>_WRITE_FIFO_DEPTH C_INTERCONNECT_<intf_name>_READ_FIFO_DEPTH	Parameter = 32 `SXX_AXI`: Enable Data FIFO **32 deep** Parameter = 512 `SXX_AXI`: Enable Data FIFO **512 deep**	`SXX_AXI`: Enable Data FIFO **None**

Setting AXI Slave(s) Register Slicing and AXI Data FIFO

1. Open the MHS file, and for each AXI slave set the Enable Register Slice/Enable Data FIFO, based upon searching the MHS in the Table 5-8.

   When doing the search, replace `<intf_name>` with the associated `BUS_INTERFACE` name. For example, `<intf_name>` for `BUS_INTERFACE S_AXI` would be `S_AXI`.

TIP: *AXI Slaves connect to the AXI Interconnect Master connection. You can make the selection in the Master Interfaces tab.*

Table 5-8: **AXI Slave Interconnect Settings with MicroBlaze Devices**

Parameter	Exists	Does Not Exist
C_INTERCONNECT_<intf_name>_AR_REGISTER C_INTERCONNECT_<intf_name>_R_REGISTER C_INTERCONNECT_<intf_name>_AW_REGISTER C_INTERCONNECT_<intf_name>_W_REGISTER C_INTERCONNECT_<intf_name>_B_REGISTER	`MXX_AXI`: Enable Register Slice **Auto**	`MXX_AXI`: Enable Register Slice **None**
C_INTERCONNECT_<intf_name>_WRITE_FIFO_DEPTH C_INTERCONNECT_<intf_name>_READ_FIFO_DEPTH	Parameter = 32 `MXX_AXI`: Enable Data FIFO **32 deep** Parameter = 512 `MXX_AXI`: Enable Data FIFO **512 deep**	`MXX_AXI`: Enable Data FIFO **None**

Validating the Design

1. Click the **Validate Design** button.
2. Make any needed design changes. Changes might include signal connections or other issues.

Verifying Parameters between XPS and IP Integrator Designs for AXI Masters and Slaves in a MicroBlaze Processor-Based Design

1. Ensure that the XPS design has generated a netlist. This creates IP top-level wrapper files for every IP in the `<xps_project>/hdl` directory.

   For example, `<xps_project>/hdl /system_leds_8bits_wrapper.vhd` contains the wrapper for `AXI_GPIO` in the EDK design. The important portions of the wrapper are the mappings to parameters, as shown in the following example:

```
C_GPIO_WIDTH => 8,
C_GPIO2_WIDTH => 32,
C_ALL_INPUTS => 0,
C_ALL_INPUTS_2 => 0,
C_INTERRUPT_PRESENT => 1,
```

```
C_DOUT_DEFAULT => X"00000000",
C_TRI_DEFAULT => X"ffffffff",
C_IS_DUAL => 0,
C_DOUT_DEFAULT_2 => X"00000000",
C_TRI_DEFAULT_2 => X"ffffffff"
```

2. In the Tcl Console, get the properties for the IP, using the following command:

   ```
   report_property [get_bd_cells axi_gpio_0]
   ```

   From the report in the TCL console, the important lines contain `C_`, which are for parameters. The following example shows lines that are used for comparison.

   ```
   CONFIG.C_ALL_INPUTS            string  false      true     0
   CONFIG.C_ALL_INPUTS_2          string  false      true     0
   CONFIG.C_ALL_OUTPUTS           string  false      true     1
   CONFIG.C_ALL_OUTPUTS_2         string  false      true     0
   CONFIG.C_DOUT_DEFAULT          string  false      true     0x00000000
   CONFIG.C_DOUT_DEFAULT_2        string  false      true     0x00000000
   CONFIG.C_GPIO2_WIDTH           string  false      true     32
   CONFIG.C_GPIO_WIDTH            string  false      true     8
   CONFIG.C_INTERRUPT_PRESENT     string  false      true     1
   CONFIG.C_IS_DUAL               string  false      true     0
   CONFIG.C_TRI_DEFAULT           string  false      true     0xFFFFFFFF
   CONFIG.C_TRI_DEFAULT_2         string  false      true     0xFFFFFFFF
   ```

3. Compare the parameters.

   Notice that all parameter values match except `C_ALL_OUTPUTS` and `C_ALL_OUTPUTS_2`, which are new parameters on the latest AXI GPIO IP for IP integrator. For parameter additions, differences in parameter values, or for information on setting these parameters, see the IP Product Guide.

4. If changes are necessary for parameters, double-click the IP instance in the block diagram and make the appropriate parameter changes.
5. If parameters cannot be found in the GUI:
   a. In the block diagram, under **Block Properties**, select the IP.
   b. Select the Properties tab, expand **Config** and enter the value for the parameter.
   c. Redo the `report_property` command and modify it, if necessary.
   d. Click the **Validate Design** button to ensure there are no design issues.

Finishing the Design

1. In the Sources tab, right-click **design_1** and select **Generate Output Products**, and click **Generate**.
2. After generation, right-click **design_1** in the Sources tab, and select **Create HDL Wrapper**.
3. Select **OK** for Vivado Design Suite to manage the top-level wrapper.

4. Create an XDC file that locks down pins in the design that are not board-related or additional constraints for MIG, for example, the location for the Reset pin and `DCI_CASCADE` settings for MIG.
5. In the Sources tab, right-click and select **Associate ELF Files**.
6. Select the appropriate ELF file from the project for **Design Sources** and/or **Simulation Sources** by clicking the ellipsis (**...**) under Associated ELF Files.
7. Run the design through the Vivado Design Suite Implementation tools.

Migrating Pcores into a Vivado Design Suite Project

You can migrate processor cores (pcore IP that was packaged in the ISE tools or PlanAhead tools) by re-packaging them into a Vivado Design Suite project.

For step-by-step instructions for migration, see the *Vivado Design Suite Tutorial: Creating and Packaging Custom IP* (UG1119) [Ref 14].

Managing Location Constraints

In XPS, constraints were all a part of top-level XDC (with the exception of MIG).

In IP integrator, the physical and timing constraints are generated as a part of the output products of individual IP.

Migrating ISE Simulator Tcl to Vivado Simulator Tcl

Tcl Command Migration

Table 6-1 lists the ISE® Simulator (ISim) Tcl commands and the equivalent Tcl commands in the Vivado® simulator.

Table 6-1: **ISE Simulator (ISim) Tcl to Vivado Tcl Mappings**

ISim Tcl	Vivado Design Suite Tcl
`bp add <file_name> <line_number>`	`add_bp file_name line_number`
`bp clear`	`remove_bps`
`bp del <index> [<index>…]`	`remove_bp indexlist…`
`bp list`	`report_bps`
`bp remove <file_name> <line_number>`	`remove_bps [get_bps -filter {file_name==<file_name> && line_number == <line_number>}]`
`describe <name>`	`describe name`
`dump`	`report_values`
`dump -p <process_scope_name>`	`report_values process_scope_name/*`
`isim condition add <condition_expression> <command> [-label <label_name>]`	`add_condition [-label name] <condition_expression> <command>`
`isim condition remove [<label_names>…] [<indexlist>…] [-all]`	`remove_conditions [names_indices_objects…]`
`isim condition list`	`report_conditions`
`isim force add <object_name> <value> [-radix <radix>] [-time <time_offset>] { [ -value <value> [-radix <radix>] -time <time_offset>] } <[-cancel <time_offset>] [-repeat <time_offset>]`	`add_force [-radix radix] [-cancel_after <time_offset>] [-repeat_every <time_duration>] <object_name> {<value> [<time>] } [{ <value> <time>}…]`
`isim force remove`	`remove_force`

Table 6-1: **ISE Simulator (ISim) Tcl to Vivado Tcl Mappings *(Cont'd)***

ISim Tcl	Vivado Design Suite Tcl
isim get <property> Properties: arraydisplaylength, radix, userunit, maxtraceablesize, ltrace, ptrace	get_property property_name [current_sim] Properties: array_display_limit, radix, time_unit, trace_limit, line_tracing, process_tracing
isim set <property> <value> properties: arraydisplaylength, radix, userunit, maxtraceablesize, ltrace, ptrace	set_property property_name property_value [current_sim] Properties: array_display_limit, radix, time_unit, trace_limit, line_tracing, process_tracing
onerror {tcl_commands}	onerror {tcl_commands}
put [-radix <radix>] name value	set_value [-radix radix] Design_object value
quit [-f|-force] [-s|-sim]	quit [-f|-force]
restart	restart
resume	resume
run [all | continue | <time> <unit>]	run [-all] [time unit]
saif open [-scope <path_name>] [-file <file_name>] [-level <nesting_level>] [-allnets]	open_saif file_name; log_saif hdl_objects
saif_close	close_saif [SaifObj]
scope [<path>]	current_scope hdl_scope
sdfanno	SDF annotation an option for the simulation xelab (elaborator) command. sdfanno is no longer supported.
show time	current_time
show port	report_objects [get_objects * -filter {type == port}]
show scope	report_scope
show signal	report_objects [get_objects * -filter {type == signal}]
show variable	report_objects [get_objects * -filter {type == variable}]
show constant	report_objects [get_objects * -filter {type == constant}]
show child [-r]	report_scopes [get scopes -r *]
show driver <hdl_object_name>	report_drivers hdl_object (not supported)
show load <hdl_object_name>	report_readers hdl_object (not supported)
show value [-radix <radix>] <hdl_object_name>	report_value [-radix radix] hdl_object
step	step [-over]

Table 6-1: **ISE Simulator (ISim) Tcl to Vivado Tcl Mappings *(Cont'd)***

ISim Tcl	Vivado Design Suite Tcl
`test [-radix radix] <hdl_object_name> <test_value>`	No longer supported. Use Tcl built-in command as follows: `expr {[get_value -radix radix hdl_object] == test_value}`
`vcd dumpfile <file_name>`	`open_vcd file_name`
`vcd dumpvars -m <hdl_scope_name> [-l <level>]`	`log_vcd hdl_objects`
`vcd dumplimit <size>`	`limit_vcd [VCDObject] filesize`
`vcd dumpon`	`start_vcd [VCDObject]`
`vcd dumpoff`	`stop_vcd [VCDObject]`
`vcd dumpflush`	`flush_vcd [VCDObject]`
`wave log [-r] name`	`log_wave hdl_objects`

Compiling Simulation Libraries

In the ISE Design Suite, the `compxlib` tool compiled simulation libraries for third-party simulation tools. In the Vivado Design Suite, you can use the `compile_simlib` Tcl command. For more information, see:

- *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12]

Note: The `compile_simlib` Tcl command options are in the Vivado Design Suite. You can also change the defaults by using the `config_compile_simlib` Tcl command. See the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4], and *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12].

XILINX

Chapter 7

Migrating ISE ChipScope Logic Analyzer to Vivado Hardware Manager

Introduction

This chapter describes the Vivado® Design Suite hardware manager, how these tools relate to the ISE® Design Suite ChipScope™ Logic Analyzer tools, and how you can migrate IP cores from the ISE ChipScope environment into the Vivado Design Suite.

Vivado hardware manager is the term that represents all of the programming and debugging tools that are available in the Vivado Design Suite. The features that are included in the Vivado hardware manager include:

- Vivado device programmer
- Vivado logic analyzer
- Vivado serial I/O analyzer

Table 7-1 provides the Vivado Integrated Design Environment (IDE) nomenclature, and lists what ISE tool the Vivado hardware manager replaces.

Table 7-1: **Vivado IDE Feature, Description, and ISE Tool Replacement**

Vivado IDE Feature	Descriptions	Replaces ISE Tool
Vivado device programmer	The Vivado device programmer feature represents the functionality in the Vivado IDE that is used to program and configure Xilinx® devices. This feature also programs any nonvolatile (NV) memory storage devices that are attached to Xilinx devices. NV memory devices store configuration information used to program Xilinx devices.	Replaces the iMPACT device programmer tool.
Vivado logic analyzer	Vivado logic analyzer feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx devices in hardware. The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including: • ILA 2.0 (and later versions) • VIO 2.0 (and later versions)	ChipScope Logic Analyzer
Vivado serial I/O analyzer	Vivado serial I/O analyzer feature represents the functionality in the Vivado IDE that is used for debugging and validating the high-speed serial I/O links of Xilinx devices in hardware. The Vivado serial I/O analyzer is used to interact with the serial I/O debug LogiCORE IP cores, including: • IBERT 7 Series GTZ 3.0 (and later) • IBERT 7 Series GTH 3.0 (and later) • IBERT 7 Series GTX 3.0 (and later) • IBERT 7 Series GTP 3.0 (and later)	

Legacy IP Core Support

Xilinx recommends that you move to the new Vivado debug IP cores.

IMPORTANT: The ChipScope Pro debug IP core XCO files are not compatible with the Vivado tools. *Do not add an XCO file to a Vivado project.*

- In a Vivado project, add the following to a project:
  - The NGC file generated from the core
  - The XDC file
  - The synthesis template file (.V or .VHD, depending on the HDL language)
- Set the `USED_IN_SYNTHESIS` property to `false` for the ChipScope debug core XDC files.
- Set the `SCOPED_TO_REF` property to the appropriate cell name.

  The following is an example for a design that contains the `icon_v1_06a`, `ila_v1_05a`, and the `vio_v1_05a` ChipScope Pro debug IP cores:

```
set_property USED_IN_SYNTHESIS false [get_files icon_v1_06a.xdc ila_v1_05a.xdc
vio_v1_05a.xdc]
set_property SCOPED_TO_REF {ila_v1_05a} [get_files ila_v1_05a.xdc]
```

- The legacy ChipScope Pro debug IP cores, listed in Table 7-2, require the ChipScope Pro Analyzer tool for interaction during run time debugging and are NOT compatible with the Vivado hardware manager.

Table 7-2: **Legacy Cores, Compatibility, and New Vivado Debug IP Cores**

Legacy ChipScope Pro Debug IP Core and Version	Compatible with Vivado 2013.1 (and later)	New Compatible Vivado Debug IP Core
Agilent Trace Core 2 (ATC2), v1.05a	No	N/A
AXI ChipScope Monitor, v3.05a	Yes	N/A
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTH, v2.01a	No	Integrated Bit Error Ratio Tester (IBERT) 7 Series GTH, v3.0 (or later)
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTP, v2.00a	No	Integrated Bit Error Ratio Tester (IBERT) 7 Series GTP, v3.0 (or later)
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTX, v2.02a	No	Integrated Bit Error Ratio Tester (IBERT) 7 Series GTX, v3.0 (or later)
Integrated Bit Error Ratio Tester (IBERT) Spartan6 GTP, v2.02a	No	N/A

Table 7-2: **Legacy Cores, Compatibility, and New Vivado Debug IP Cores *(Cont'd)***

Legacy ChipScope Pro Debug IP Core and Version	Compatible with Vivado 2013.1 (and later)	New Compatible Vivado Debug IP Core
Integrated Bit Error Ratio Tester (IBERT) Virtex5 GTX, v2.01a	No	N/A
Integrated Bit Error Ratio Tester (IBERT) Virtex6 GTX, v2.03a	No	N/A
Integrated Bit Error Ratio Tester (IBERT) Virtex6 GTH, v2.06a	No	N/A
Integrated Controller (ICON), v1.06a	Yes	N/A
Integrated Logic Analyzer (ILA), v1.05a	Yes	Integrated Logic Analyzer (ILA), v2.0 (or later)
Virtual Input/Output (VIO), v1.05a	Yes	Virtual I/O (VIO), v2.0 (or later)

ChipScope Pro Analyzer Core Compatibility

The following subsections describe the compatibility of the ChipScope Pro Analyzer with new Vivado debug IP cores.

ILA and VIO Debug IP Cores

Use the Vivado logic analyzer to interact with the ILA v2.0 (or later) and/or VIO v2.0 (or later) debug IP cores.

Table 7-3 shows the logic debug IP core compatibility with run time tools.

Table 7-3: **Debug IP Core and Run Time Tool Requirements**

Debug IP Core and Version	Run Time Tool Requirement
AXI ChipScope Monitor, v3.05a (or earlier)	ChipScope Pro Analyzer
Integrated Controller (ICON), v1.06a (or earlier)	ChipScope Pro Analyzer
Integrated Logic Analyzer (ILA), v1.05a (or earlier)	ChipScope Pro Analyzer
Integrated Logic Analyzer (ILA), v2.0 (or later)	Vivado logic analyzer
Virtual Input/Output (VIO), v1.05a (or earlier)	ChipScope Pro Analyzer
Virtual Input/Output (VIO), v2.0 (or later)	Vivado logic analyzer

IBERT 7 Series GTH/GTP/GTX/GTZ v3.0 (or later) Debug IP Cores

Use the Vivado serial I/O analyzer to interact with the IBERT 7 Series GTH/GTP/GTX/GTZ v3.0 (or later) debug IP cores.

Table 7-4 shows the serial I/O debug IP core compatibility with run time tools.

Table 7-4: **IBERT 7 Series Debug IP Cores and Run Time Tool Requirements**

Debug IP Core and Version	Run-time Tool Requirement
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTH, v2.01a (or earlier)	ChipScope Pro Analyzer
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTH, v3.0 (or later)	Vivado serial I/O analyzer
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTP, v2.00a (or earlier)	ChipScope Pro Analyzer
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTP, v3.0 (or later)	Vivado serial I/O analyzer
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTX, v2.02a	ChipScope Pro Analyzer
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTX, v3.0 (or later)	Vivado serial I/O analyzer
Integrated Bit Error Ratio Tester (IBERT) 7 Series GTZ, v2.0	ChipScope Pro Analyzer or Vivado serial I/O analyzer

Combining legacy ChipScope Pro and Vivado Debug IP Cores within a Design

You can combine legacy ChipScope cores with Vivado core using the following rules:

- You can either instantiate Vivado debug IP cores in your HDL code or you can insert the ILA v2.0 core into the netlist of the Vivado design.

  Note: The `dbg_hub` core that connects your Vivado debug IP cores to the JTAG infrastructure is automatically inserted into your design.

- You must instantiate the legacy ChipScope Pro debug IP cores into your HDL code.

  Note: Debug core insertion into the Vivado design netlist is not supported for legacy ChipScope Pro debug IP cores.

- Instantiate an ICON core in your design that is used to connect the other legacy ChipScope Pro debug IP cores to the JTAG chain infrastructure.

IMPORTANT: *Make sure that the ICON and* `dbg_hub` *cores do not use the same JTAG user scan chain; doing so produces errors during* `write_bitstream` *DRC checking.*

To change the JTAG user scan chain of the `dbg_hub` core:

1. Open the synthesized design
2. In the Netlist window, select the **dbg_hub** core.
3. In the Cell Properties window, select the **Debug Core Options**.
4. Modify the C_USER_SCAN_CHAIN property value to a value that does not conflict with the ICON cores in your design.

XILINX

Chapter 8

Migrating Additional Command Line Tools to the Vivado IDE

Introduction

This chapter describes how to migrate various Xilinx® command line tools for use in the Vivado® Integrated Design Environment (IDE).

Migrating the ISE Partgen Command Line Tool

The ISE® Design Suite Partgen tool obtains:

- Information on all the devices installed on the system
- Detailed package information

You can retrieve the same type of information in the Vivado Design Suite using Tools Command Language (Tcl) commands. Table 8-1 lists the Vivado Tcl commands that retrieve the equivalent information that is stored in the Partgen Partlist file (`.xct`).

For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

Partlist File Contents

Table 8-1: **Tcl Command to Partgen Partlist Content Mapping**

Partlist Content	Tcl Command
Device	`get_parts`
Package	get_property `PACKAGE [get_parts <part_name>]`
Speedgrade	get_property `SPEED [get_parts <part_name>]`
NBIOBS	`llength [get_sites -filter {IS_BONDED==1 && SITE_TYPE =~ IOB*}]`
SLICES_PER_CLB	`llength [get_sites -of_objects [lindex [get_tiles CLBLM_L_*] 0] -filter {NAME=~SLICE*}]`

Table 8-1: **Tcl Command to Partgen Partlist Content Mapping *(Cont'd)***

Partlist Content	Tcl Command
NUM_BLK_RAMS	`llength [get_sites RAMB36*]`

NUM_BLK_RAM_COLS	`set looplimit [llength [get_sites RAMB36*]];`
	`for {set i 0} {$i <= $looplimit} {incr i} {`
	`set BLK_PER_COL [llength [get_sites RAMB36_X${i}Y*]]`
	`if {$BLK_PER_COL > 0} {`
	`puts "Number of BlockRAM per Column for RAMB36_X${i}, $BLK_PER_COL"}}`
	`for {set x 0} {$x <= $looplimit} {incr x} {`
	`set BLK_COLS [llength [get_sites RAMB36_X*Y$x]]`
	`if {$BLK_COLS > 0 } {`
	`puts "Number of BlockRAM Columns for RAMB36_Y$x, $BLK_COLS"}}`

FF_PER_SLICE	`llength [get_bels -of [get_sites SLICE_X0Y0] -fil` `ter {NAME=~*FF*}]`
NUM_MMCM	`llength [get_sites MMCM*]`
NUM_LUTS_PER_SLICE	`set SLICEM [lindex [get_sites * -filter SITE_TYPE==SLICEM] 0]` `llength [get_bels -of $SLICEM -filter "TYPE=~LUT_OR_MEM*"]`

LUT_NAME ENUMERATION and LUT_SIZE_ENUMERATION	`set SLICEM [lindex [get_sites * -filter SITE_TYPE==SLICEM] 0]`
	`foreach bel [get_bels -of $SLICEM -filter "TYPE=~LUT_OR_MEM*"] {`
	`set name [lindex [split $bel /] 1]`
	`set fields [split [get_property TYPE $bel] "M"]`
	`puts "LUT_NAME=$name and LUT_SIZE=[lindex $fields 2]"`
	`}`
	`set SLICEL [lindex [get_sites * -filter SITE_TYPE==SLICEL] 0]`
	`foreach bel [get_bels -of $SLICEL -filter "TYPE=~LUT*"] {`
	`set name [lindex [split $bel /] 1]`
	`set fields [split [get_property TYPE $bel] "T"]`
	`puts "LUT_NAME=$name and LUT_SIZE=[lindex $fields 1]"`
	`}`

NUM_GLOBAL_BUFFERS	`llength [get_sites BUFGCTRL*]`
GLOBAL_BUFFERS ENUMERATION	`get_sites BUFGCTRL`

Table 8-1: **Tcl Command to Partgen Partlist Content Mapping *(Cont'd)***

Partlist Content	Tcl Command
GLOBAL_BUFFER IOBS ENUMERATION	`get_sites -of [get_package_pins -filter {IS_CLK_CAPABLE==1 && IS_MASTER==1}]`
NUM_BUFIO_BUFFERS	`llength [get_sites BUFIO*]`
BUFIO_BUFFERS ENUMERATION	`get_sites BUFIO`
NUM_DSP	`llength [get_sites DSP*]`
NUM_PCIE	`llength [get_sites PCIE*]`
NUM_PLL	`llength [get_sites PLL*]`
NUM_CLB	`llength [get_tiles CLB*]`
CLKRGN ENUMERATION	`get_clock_regions`
NUM_OF_SLR	`llength [get_slrs]`
NUM_DSP_COLUMNS	`llength [get_sites DSP48_X*Y1]`
NUM_DSP_PER_COLUMN	`llength [get_sites DSP48_X1Y*]`

NUM_BRAM_PER_COLUMN	`set looplimit [llength [get_sites RAMB36*]]`
	`for {set i 0} {$i <= $looplimit} {incr i} {`
	`set BLK_PER_COL [llength [get_sites RAMB36_X${i}Y*]]`
	`if {$BLK_PER_COL > 0} {`
	`puts "Number of BlockRAM per Column for RAMB36_X${i}, $BLK_PER_COL"}}`
	`for {set x 0} {$x <= $looplimit} {incr x} {`
	`set BLK_COLS [llength [get_sites RAMB36_X*Y$x]]`
	`if {$BLK_COLS > 0 } {`
	`puts "Number of BlockRAM Columns for RAMB36_Y$x, $BLK_COLS"}}`

HEIGHT_OF_DSP	`foreach region [get_clock_regions] { puts "Height of DSP48 in $region, [llength [get_sites -filter "CLOCK_REGION==$region" DSP48*]]" }`
SLR ENUMERATION	`get_slrs`

Package Information

Table 8-2 lists the Partgen Package file content to Vivado Tcl commands.

Table 8-2: **Tcl Command to Partgen Package File Content Mapping**

Package File	Tcl Command
Pin Type	`foreach pin [get_package_pins] {` `  puts "Pin Type = [get_property CLASS $pin]"}`
Pin Name	`foreach pin [get_package_pins] {puts "Pin Name = $pin"}`
Pin Function	`foreach pin [get_package_pins] {` `  puts "Pin Function = [get_property PIN_FUNC $pin]"}`

PAD Name	`foreach pin [get_package_pins] {`
	`  set site [get_sites -of $pin]`
	`  if {$site ne ""} {`
	`  puts "PAD Name = [get_property NAME [get_sites -of $pin]]"} else {`
	`  puts "PAD Name = N.A."}}`

Bank Number of Pin	`foreach pin [get_package_pins ] {` `  puts "Bank Number = [get_property BANK $pin]"}`
Differential Pair	`foreach pin [get_package_pins] {` `  puts "DIFF Pair = [get_property DIFF_PAIR_PIN $pin]"}`

IO Bank Type	`foreach pin [get_package_pins] {`
	`  set bank [get_property BANK $pin]`
	`  if {$bank ne ""} {`
	`  puts "Bank Type = [get_property BANK_TYPE [get_iobanks [get_property BANK $pin]]]"} else {`
	`  puts "Bank Type = N.A."}}`

Write Package Pin and Port Placement Information (including package trace delay information for every pin on the part)	`write_csv`

ISE BitGen Command Line Tool

The ISE Design Suite `Bitgen` tool generates bitstreams.

In the Vivado Design Suite, use the `write_bitstream` Tcl command. For more information, see:

- *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4]
- *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 5]

Note: The BitGen command options are Tcl properties in the Vivado Design Suite. See this link in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 5] for details on the new properties and values.

Speedprint Command Line Tool

The ISE Design Suite Speedprint tool lists the speed data for all device components. The Speedprint tool has been replaced by the `get_speed_models` Tcl command in the Vivado Design Suite.

For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

ISE PROMGen Command Line Tool

The ISE PROMGen tool creates PROM files for programming.

IMPORTANT: *The Vivado Design Suite provides* `write_cfgmem` *command that replaces the PROMGen capability.*

ISE BSDLAnno Command Line Tool

In ISE, the ISE BSDLAnno tool creates post-configuration Boundary Scan Description Language (BSDL) files.

In the Vivado IDE Tcl console, you can use the Tcl command:

```
write_bsdl
```

This command creates BSDL files. For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

ISE Data2MEM Command Line Tool

UpdateMEM in Vivado is the equivalent to Data2MEM in ISE. For information on UpdateMEM, see this link in the *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898).

Migrating from compxlib to compile_simlib

The `compile_simlib` command replaces the `compxlib` command in the Vivado Design Suite.

Obsolete Primitives

Introduction

The following primitives are not native to 7 series devices. Consequently, these primitives are obsolete in the Vivado® Design Suite and are not retargeted:

A

- AFIFO35_INTERNAL
- ARAMB_36_INTERNAL

B

- BSCAN_FPGACORE
- BSCAN_SPARTAN3
- BSCAN_SPARTAN3A
- BUFCF
- BUFDS
- BUFE
- BUFGDLL
- BUFIO2
- BUFIO2_2CLK
- BUFIO2FB
- BUFIODQS
- BUFPLL
- BUFPLL_MCB
- BUFT

C

- CAPTURE_FPGACORE
- CLKDLL
- CLKDLLE
- CLKDLLHF
- CONFIG
- CRC32
- CRC64

D

- DCM_CLKGEN
- DCIRESET
- DSP48A
- DSP48A1

E

- EMAC

F

- FDDRCPE
- FDDRRSE
- FIFO36_EXP
- FIFO36_72_EXP
- FMAP
- FRAME_ECC_VIRTEX4
- FRAME_ECC_VIRTEX5

G

- GT11
- GT11CLK
- GT11_CUSTOM
- GT11_DUAL
- GT11CLK_MGT
- GTHE1_QUAD
- GTPA1_DUAL
- GTP_DUAL
- GTX_DUAL
- GTXE1

I

- IBUF_DLY_ADJ
- IBUFDS_DLY_ADJ
- IBUFDS_GTHE1
- IBUFDS_GTXE1
- IFDDRCPE
- IFDDRRSE
- IODELAY2
- IODRP2
- IODRP2_MCB
- ISERDES2

J

- JTAGPPC
- JTAGPPC440
- JTAG_SIM_SPARTAN3A
- JTAG_SIM_VIRTEX4
- JTAG_SIM_VIRTEX5

M

- MCB

O

- OFDDRCPE
- OFDDRRSE
- OFDDRTCPE
- OFDDRTRSE
- ORCY
- OSERDES2

P

- PCIE_2_0
- PCIE_A1
- PCIE_EP
- PCIE_INTERNAL_1_1
- PMCD
- POST_CRC_INTERNAL
- PPC405_ADV
- PPC440

R

- RAMB32_S64_ECC
- RAMB36_EXP
- RAMB36SDP_EXP
- RAMB4_S1
- RAMB4_S1_S1
- RAMB4_S1_S16
- RAMB4_S1_S2
- RAMB4_S1_S4
- RAMB4_S1_S8

- RAMB4_S16
- RAMB4_S16_S16
- RAMB4_S2
- RAMB4_S2_S16
- RAMB4_S2_S2
- RAMB4_S2_S4
- RAMB4_S2_S8
- RAMB4_S4
- RAMB4_S4_S16
- RAMB4_S4_S4
- RAMB4_S4_S8
- RAMB4_S8
- RAMB4_S8_S16
- RAMB4_S8_S8
- ROC
- ROCBUF
- RAMB16BWER
- RAMB16BWE
- RAMB8BWER

S

- SIM_CONFIG_S3A_SERIAL
- SIM_CONFIG_S3A
- SIM_CONFIG_S6_SERIAL
- SIM_CONFIG_S6
- SIM_CONFIG_V5_SERIAL
- SIM_CONFIG_V5
- SIM_CONFIG_V6_SERIAL
- SIM_CONFIG_V6

- SPI_ACCESS
- STARTUP_FPGACORE
- STARTUP_SPARTAN3E

T

- TBLOCK
- TEMAC
- TEMAC_SINGLE
- TIMEGRP
- TIMESPEC
- TOC
- TOCBUF

Appendix B

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. *Vivado Design Suite User Guide: Design Flows Overview (*UG892*)*
2. *Vivado Design Suite User Guide: System Design Entry (*UG895*)*
3. *Vivado Design Suite User Guide: Using Constraints (*UG903*)*
4. *Vivado Design Suite Tcl Command Reference Guide (*UG835*)*
5. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
6. *Vivado Design Suite User Guide: Designing with IP* (UG896)
7. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)
8. *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* (UG940)
9. *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898)
10. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
11. *Constraints Guide* (UG625)
12. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
13. *Vivado Design Suite Properties Reference Guide* (UG912)
14. *Vivado Design Suite Tutorial: Designing with IP* (UG1119)
15. Vivado Design Suite Documentation

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. Vivado Design Suite for ISE Software Project Navigator Users Training Course
2. Vivado Design Suite Advanced XDC and Static Timing Analysis for ISE Software Users
3. Vivado Design Suite QuickTake Video: Managing Vivado IP Version Upgrades
4. Vivado Design Suite QuickTake Video Tutorials

Please Read: Important Legal Notices