Vivado Design Suite User Guide

Hierarchical Design

UG905 (v2017.4) December 20, 2017

XILINX
ALL PROGRAMMABLE™

Revision History

12/20/2017: Released with Vivado® Design Suite 2017.4 without changes from 2017.2.

Date	Version	Revision
07/26/2017	2017.2	Updated links to Xilinx training courses.
04/05/2017	2017.1	Added Important note to Introduction, specifying that this document only applies to the 7 series device family. Removed references to UltraScale™ devices from Known Issues.

Table of Contents

Vivado Hierarchical Design

Introduction

Hierarchical Design (HD) flows enable you to partition a design into smaller, more manageable modules to be processed independently. In the Vivado® Design Suite, these flows are based on the ability to implement a partitioned module out-of-context (OOC) from the rest of the design. The following is a list of the current methodologies in the Vivado Design Suite.

- **Module Analysis**: This flow allows you to analyze the module independent of the rest of the design to determine resource utilization and perform timing analysis. No wrapper or dummy logic is required; just synthesize, optimize, place and route the module on its own. Perform resource usage analysis, inspect timing reports, and examine placement results just as you would for a full design.

  The Module Analysis flow implements a partitioned module or IP core out-of-context of the top level of the design. The module is implemented in a specific part/package combination, and with a fixed location in the device. I/O buffers, global clocks and other chip-level resources are not inserted, but can be instantiated within the module. The OOC implementation results can be saved as a design checkpoint (DCP) file.

- **Module Reuse**: This flow reuses placed and routed modules from the Module Analysis flow within a top-level design, locking down validated results. Users can iterate on a specific section of a design, achieving timing closure and other specific goals, then reuse those exact results while turning their attention to other parts of the design.

  Reuse of out-of-context modules requires knowledge of where the module pins and interface logic have been placed so that the connecting logic can be floorplanned accordingly. The preservation level of the imported OOC module can be selected, allowing for minor placement and routing changes if desired. This flow does not yet support moving or replicating the OOC implementation results to other areas of a device, or to a different device.

  The Module Reuse flow has two variations, with the difference between the two variations being the mechanism for establishing the module constraints. Context constraints (which define how a module connects in the full design) and timing constraints are critical for successfully assembling the top level design with one or more reused modules.

The variations of Module Reuse are:

- **Bottom-Up Reuse**: Using this methodology, the OOC implementation is done with little to no knowledge of the top-level design in which it is reused, and the OOC results drive the top-level implementation. This approach enables you to build a verified module (such as a piece of IP) through place and route for reuse in one or more top level designs. In this flow, the top-level design details are not known, so you must supply the context constraints. These define the physical location for the module, placement details for the module I/O, definitions of clock sources, timing requirements for paths in and out of the module, and information about unused I/O.
- **Top-Down Reuse**: Using this methodology, the top-level design and floorplan create the OOC implementation constraints, and the top-level design drives the OOC implementation. This approach enables a Team Design methodology, enabling parallel synthesis and implementation of one or more modules within the design. Team members can implement their portions of a design independently, reusing their exact results in the assembled design. In this flow, the top-level design details (pinout, floorplan, and timing requirements) are known, and are used to guide the OOC implementation. This allows for OOC module pin constraints, top-level input/output timing requirements, and boundary optimization constraints to all be created from the top-level design.

All of these flows result in overall run time reduction by enabling the tools to implement only one module of the design, instead of the whole design. This allows you to compile many more turns per day, reducing time to design, verify, and meet timing on a per module basis. It also allows designers to actively work on a module even if the rest of the design is not complete or available.

IMPORTANT: *This document applies only to 7 series devices, not UltraScale™ or UltraScale+™ devices. For more information about applications of hierarchical design in UltraScale and UltraScale+ devices, see Vivado Design Suite User Guide: Partial Reconfiguration (UG909)[Ref 6].*

Design Considerations

Hierarchical Design methodologies require some special considerations to achieve optimal results. The following sections provide information to be considered when planning the architecture for, designing, and constraining a design for Vivado Hierarchical Design flows.

Design for Performance

Implementing modules out-of-context from the rest of the design prevents the optimization across modules that might normally occur in a typical top-down flow. To limit the loss of performance due to these restrictions, follow these guidelines:

- Choose the module(s) to be implemented out-of-context carefully. Select modules that are logically isolated from other logic in the design, and that can be physically constrained to a contiguous area of the device.
- Build an effective hierarchy with the selected modules in mind. Structure the hierarchy for independent implementation. Design hierarchy is an important consideration. Where a design is partitioned can have significant effects on the quality of the results, and in some cases hierarchy might need to be added or modified to group appropriate modules together for an out-of-context implementation.
- Keep critical paths contained entirely within modules, either in submodules or in top.
- Register inputs and outputs between modules to maximize optimization within the modules, and to allow for maximum placer and router flexibility.
- Provide information on how the module will be used by defining context constraints. Context constraints define how the module will be connected in the top level, allowing for additional optimization and accurate timing analysis. See Out-of-Context Design Constraints, page 12 in the Commands and Constraints section of this document.
- Dedicated connections cannot always be handled properly across an OOC module boundary. All related design elements must be partitioned together. Examples of this are I/O components with dedicated connection such as transceivers or IOLOGIC components. See Dedicated Connections Between Components in the Design Considerations section.

Build an Effective Floorplan

Implementing a module out-of-context has the following requirements.

- Each module implementation must have a Pblock constraint to control the placement. If a Pblock is not used, placement conflicts are likely during the assembly phase.
- Add the `CONTAIN_ROUTING` property to all OOC Pblocks. Without this property, `lock_design` cannot lock the routing of an imported module because it cannot be guaranteed that there are no routing conflicts.
- Pblock ranges for each OOC module must not overlap. If a top-level design is to import multiple OOC module results, the modules must occupy separated regions in the device.
- Nested or child Pblocks are supported within the OOC implementation as long as the nested Pblock range is fully contained by the parent Pblock range, and the `PARENT` property is correctly set. See Table 5, Pblock Constraints.
- All clock buffers (both in top and the OOC module) must be locked. Buffers inside the OOC module must have LOC constraints, and the locations of buffers in the top level should be identified by `HD.CLK_SRC` constraints. See Out-of-Context Design Constraints, page 12, for a description of the `HD.CLK_SRC` constraint.

- If the OOC implementation results are to be reused, it is strongly recommended that the OOC module pins be locked down during the OOC implementation using `HD.PARTPIN_RANGE` or `HD.PARTPIN_LOCS` constraints.

  The partition pins serve two main purposes in an OOC implementation.

  - A partition pin creates a physical location to guide the placement of the associated interface logic. Timing constraints (`set_max_delay`) to and from the PartPins are needed to affect placement.
  - A partition pin gives the router a point to route the interface subnet. Without PartPins the interface nets cannot be routed, and it is possible to create an OOC result where routes internal to the module block routing resources needed by the interface nets during reuse. This prevents the preservation of all OOC nets without causing an unroutable design.

  See Table 6, Context Constraints, for a description of the `HD.PARTPIN_RANGE` and `HD.PARTPIN_LOCS` constraints, and Table 4, Timing Constraints, for a description of `set_max_delay`.

Dedicated Connections Between Components

It is recommended, and in some cases required, to keep components with dedicated connections in the same partition of the design. Having dedicated connections span the boundary of an OOC module can cause reduced performance and/or implementation errors. The following is a list of components with dedicated connections.

- **IOLOGIC and IOBUF** – This includes connections from registers placed in the ILOGIC or OLOGIC, IDDR, ODDR, ISERDES, and OSERDES to I/O components including IBUF, OBUF, IBUFDS, OBUFDS, IOBUF, and IOBUFDS.
- **GT transceiver components** – GTX and GTP transceivers and their dedicated I/O connections.
- Bidirectional ports should be avoided if possible. They do not receive `PP_LOCS`, so any `PP_RANGE` or `PP_LOCS` constraints on bidirectional ports are automatically removed in OOC mode. If bidirectional ports are required, floorplan the associated interface logic to avoid timing issues when the OOC module is imported during the Reuse flow.

Avoid placing any I/O components that connect to each other in different partitions of a design.

Use of IDELAYCTRL Groups

The use of IDELAYCTRL groups within an OOC module is supported. The OOC implementation inserts an IDELAYCTRL, and the OOC implementation results can be imported into Top. The following rules apply to the use of IDELAYCTRL groups:

- Multiple OOC modules, each with its own IDELAYCTRL, cannot share the same clock region.
- An OOC module with an IDELAYCTRL cannot be preserved 100%. This is a known limitation in this version of Vivado that will be addressed in a later release of the Vivado Design Suite.

I/O and Clock Buffers

I/O and clock buffers are supported inside OOC modules. However, some special considerations should be taken into account depending on their use.

- **I/O Buffers** – If an OOC port connects directly to an I/O buffer in the top level, it is recommended to move this buffer inside the OOC module for better results. This is not possible in all situations (for example, if an OOC port connects directly to an IBUF in the top level, but that IBUF also drives other logic not in the OOC module), and in those cases the logic inside the OOC module should be controlled with an `HD.PARTPIN_LOCS` constraint. See the Out-Of-Context Commands and Constraints section for more information.
- **Regional Clock Buffers** – If a BUFR or BUFHCE exists within the OOC module it should be locked down to a specific location. The tools then appropriately place logic driven by the buffer. However, if the BUFR or BUFHCE is in the top-level design, and the OOC Pblock spans more clock regions than the buffer has access to, then more information must be supplied. A nested Pblock must be created with a range that is a subset of the range defined by the OOC Pblock. The nested Pblock would contain all of the cells driven by the BUFR or BUFHCE, and can be created with the following commands:

  ```
  create_pblock -parent <parent_pblock_name> <nested_pblock_name>
  add_cells_to_pblock <nested_pblock_name> -cells [get_cells -of [get_nets
  -segments -of [get_ports [list <clock_port> <clock_port>]]] -filter
  "(IS_PRIMITIVE)"]
  resize_pblock <nested_pblock_name> -add {SLICE_Xx1Yy1:SLICE_Xx2Yy2}
  ```

  This should be done for each module port that is driven by a BUFR or BUFHCE in the top-level. If multiple OOC clock ports have loads in the same clock regions, all applicable ports can be listed in the `add_cells_to_pblock` command above. The range for the nested Pblock must correspond to the BUFR or BUFHCE location in the top-level implementation. A mismatch between buffer location at the top level and the corresponding Pblock range for the OOC module can lead to an unroutable condition during top-level implementation.
- **Global Clock Buffers** – Global buffers are supported inside an OOC module. When a BUFG is inside an OOC instance the clock net is routed on global routing in the OOC implementation. If an OOC port is driven by a clock net in the top level, the clock net is not routed during the OOC implementation, and timing estimations are used to determine clock delays/skew. The `HD.CLK_SRC` constraint should be used to help improve timing estimations in this case. This constraint allows the tools to know the driver location and type (for example, BUFG vs. BUFR), and to improve timing

estimation by calculating clock pessimism removal (CPR). For a description of CPR, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 1].

Design Rules for Out-of-Context Designs

Table 1 shows the Design Rule Checks (DRCs) that are performed while implementing an OOC design, and the design rules for which the DRCs test.

Table 1: **Design Rules for Hierarchical Designs**

DRC #	Severity	Design Rule
HDOOC-1	Error	Reconfigurable modules must have a Pblock defined.
HDOOC-2	Error	HD modules must have certain Pblock properties defined.
HDOOC-3	Error	Bitstream generation not allowed for OOC modules.
HDOOC-4	Error	No Pblock range or LOC for cell.

Checkpoints

To export and import results of a module implementation, Hierarchical Design flows use checkpoints. Checkpoints archive the logical design, physical design, and module constraints and are the only file needed to fully restore a design.

A saved checkpoint can only be read into the same part/package/speed grade combination in which it was originally generated.

RECOMMENDED: *The* ***-strict*** *option to the* `read_checkpoint` *command is recommended for HD flows to make sure that all data read in matches the module interface exactly, ensuring a robust implementation. For more information on Checkpoints, refer to this link in the Vivado Design Suite User Guide: Implementation (UG904)* [Ref 2].

Out-Of-Context Commands and Constraints

The HD flows are currently only supported through the Non-Project Mode batch/Tcl interface (no Vivado IDE (GUI) or Project Mode based commands). An example design and related scripts are available upon request. Please contact Xilinx support for information on accessing this document.

The following sections describe a few specialized out-of-context commands and constraints used by the HD flows. Examples of how to use these commands to run an HD flow are given.

For more information on individual commands, refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 3].

Out-of-Context Commands

Synthesizing or implementing a module out of context requires that the tools be run in an "out-of-context" mode. Other than that, the commands for running an out-of-context flow are the same as for any other flow. There are currently no unsupported commands for synthesis, optimization, or implementation.

Synthesis

There are several synthesis tools and methods supported for this flow. The following is a list of supported tools.

- **XST**: Bottom-up synthesis or Incremental synthesis using partitions (PXML file). This is supported for 7 series only.

  Note: XST is not recommended for new Vivado designs or for designs targeted to architectures newer than 7 series.

- **Synplify**: Bottom-up synthesis or Compile Points (using Hierarchical Projects to generate individual netlists)
- **Vivado synthesis**: Out-of-Context (OOC) synthesis only.

IMPORTANT: *OOC synthesis (also known as bottom-up synthesis) refers to a synthesis flow in which each module has its own synthesis run. This generally involves turning off automatic I/O buffer insertion for the lower level modules.*

This document only covers the Vivado synthesis flow. For information on the Synplify flow, refer to the Synopsys Synplify documentation.

Vivado synthesis for this flow is run in batch mode using the `synth_design` command:

```
synth_design -mode out_of_context -flatten_hierarchy rebuilt -top <top_module_name>
-part <part>
```

Table 2: **synth_design Options**

Command Option	Description
`-mode out_of_context`	Prevents I/O insertion for synthesis and downstream tools. The mode is saved in checkpoints if `write_checkpoint` is issued.
`-flatten_hierarchy rebuilt`	There are several values allowed for `-flatten_hierarchy`, but `rebuilt` is the recommended setting for HD flows.
`-top`	This is the module/entity name of the module being synthesized. This switch can be omitted if `set_property top <top_module_name> [current_fileset]` is issued prior to `synth_design`.
`-part`	This is the Xilinx part being targeted (e.g., `xc7k325tffg900-3`)

The `synth_design` command synthesizes the design and stores the results in memory. To write the results out to a file, a `write_checkpoint` command must be issued.

```
write_checkpoint <file_name>.dcp
```

Issuing the above command saves the synthesis results into a DCP file and allows you to close the in-memory design. You can reload the synthesis results at a later time using the `open_checkpoint` command. This allows you to run one or more implementation runs on the synthesized results without having to rerun synthesis each time.

Timing and physical constraints can be passed to OOC synthesis. The physical constraints are ignored by synthesis and are passed on to the final DCP. For timing constraints, it is important to make sure that any module level context constraints are properly marked as `USED_IN out_of_context`. See Constraints Designated for OOC Use Only, page 13, for more information on specifying that a constraints is for OOC use only.

Implementation

This section describes the necessary commands to implement a module instance in an out-of-context flow. Note that if this module has multiple instances instantiated in a top level design and the Assembly Flow is going to be used, multiple OOC implementations each with unique Pblock constraints are required to generate the necessary implementation results.

If `synth_design -mode out_of_context` was previously run, and the results are still in memory, then implementation can be run directly. For example, the following implementation commands can be used.

- `read_xdc` – Use this command if all constraints are not already loaded. It might be necessary to apply certain module-level XDC constraints to the OOC implementation only. See Constraints Designated for OOC Use Only, page 13, for a description of the OOC-only constraints).
- `set_property HD.PARTITION 1 [current_design]` – This identifies the cell as being implemented OOC, and enables DRCs and the required software features during the reuse flows. It is not required for the OOC implementation, but setting it is a good way to ensure the final DCP is imported correctly during the reuse flow. See Top-Level Reuse Commands, page 18, for more information on what HD.PARTITION does in the reuse flow.
- `opt_design` (optional, but recommended)
- `place_design`
- `phys_opt_design` (optional, but recommended)
- `route_design`

If there is currently no design in memory, then a design must be loaded. This can be done in one of several ways.

- **Method 1: Read Netlist Design**

```
read_edif <file_name>.edf/edn/ngc
link_design -mode out_of_context -top <top_module_name> -part <part>
```

Table 3: **link_design Options**

Command Option	Description
`-mode out_of_context`	Load a netlist design in an out-of-context mode. Enables special checking and optimization for downstream tools.
`-part`	This is the Xilinx part being targeted (e.g., `xc7k325tffg900-3`).
`-top`	This is the module/entity name of the module being implemented. This switch can be omitted if `set_property top <top_module_name> [current_fileset]` is issued prior to `link_design`.

If the `-mode out_of_context` option is not issued for `link_design` after reading the design netlist(s), subsequent implementation steps treat the design as a full design and trim any sourceless or loadless signals. The OOC mode must be defined during either `synth_design` or `link_design` to run the Module Analysis flow.

- **Method 2: Open Checkpoint**

```
open_checkpoint <file_name>.dcp
```

IMPORTANT: *The `open_checkpoint` command does not have a `-mode out_of_context` option. The mode is saved as part of the checkpoint, therefore it is critical to make sure that the tools are in the correct mode when writing out a checkpoint.*

- **Method 3: Add Various File Types**

The `add_files` command can be used in conjunction with `link_design` to load in modules that include multiple files and various file types.

```
add_files <file_name>.dcp
add_files <file_name>.edf
add_files <file_name>.xdc
link_design -mode out_of_context -top <top_module_name> -part <part>
```

Out-of-Context Design Constraints

To process a design using the Module Analysis flow, none of the following constraints are absolutely required. For more accurate timing analysis, use of `HD.CLK_SRC` and `create_clock` are strongly encouraged. All other constraints are optional.

When using a Module Reuse flow, these context constraints become much more important. For successful assembly of designs with OOC modules, these constraints ensure that the physical resources are appropriately allocated, clock interactions are understood, and information about the module interfaces are accurately set. Without establishing the constraints for each module, assembly become more difficult.

Constraints Designated for OOC Use Only

Some constraints that are necessary for the OOC implementation can cause undesirable results if imported into the top-level design. To prevent this behavior these constraints need to be specified in separated XDC file(s), and designated for OOC use only. There are two ways to specify that the XDC file should be used for the OOC flow only. Specifying that a particular XDC file's constraints should only be used in the OOC flow adds a marker to them. This makes the tools ignore the constraints when reading them into a non-OOC design.

- **Method 1: Using** `read_xdc`

  When reading in an XDC file with the `read_xdc` command, the `-mode out_of_context` switch can be used.

```
read_xdc -mode out_of_context <file>.xdc
```

TIP: *The `read_xdc` command can be issued prior to or after a design has been loaded with `link_design`.*

- **Method 2:** `USED_IN` **property**

  If files are being added using the `add_files` command, then a property can be set on a file to specify that it is to be used in OOC only. It is required to specify all flows in which the XDC file is to be used in (that is, synthesis and/or implementation).

```
add_files <file>.xdc
set_property USED_IN {synthesis implementation out_of_context} [get_files <file>]
```

IMPORTANT: *The `add_files` command must be issued prior to a design being loaded with `link_design`. Using `add_files` on a design that is already loaded has no effect.*

Constraint Syntax

The following tables list timing, placement, and context constraints that should be used for an out-of-context implementation. Many of these constraints are used for any design flow, and more information can be found in the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 4].

IMPORTANT: *All of the constraints listed in this section can be generated automatically for you through the Top-Down Reuse flow. The scripts and methodology to generate these constraints are described in the Vivado Design Suite Tutorial: Hierarchical Design (UG946). Please contact Xilinx support for information on accessing this document.*

Table 4: **Timing Constraints**

Constraint Name	Description
`set_max_delay`	Defines input and output delays to budget the time allowed for the OOC module. Helps control placement in the OOC implementation and ease timing closure at the top level.
`create_clock`	Defines clocks on the OOC module ports. A `create_clock` constraint should exist for every clock port, whether the clock buffer is instantiated in the top level or the OOC module.
`set_clock_uncertainty`	Defines clock uncertainty for a clock that is an input to an OOC module. This constraint should be defined for all clocks of an OOC module to ensure accurate timing analysis. Failure to do so could result in failing paths when a module is imported.
`set_system_jitter`	Defines system jitter value. This constraint should be set to zero when defining user clock uncertainty (`set_clock_uncertainty`) based on the top-level design. Otherwise, system jitter is factored into the uncertainty calculations for the OOC implementation, making the final value different from the user-defined value.
`set_clock_latency`	Defines latency for a clock that is an input to an OOC module. This constraint is needed to correctly model clock delay when the entire clock path is not known.
`set_clock_groups`	Defines asynchronous clocks (`-asynchronous`), or clocks driven by the same global buffer (`-physically_exclusive`).

Timing Constraint Examples:

- `create_clock -period 8.000 -name clk -waveform {0.000 4.000} [get_ports clk]`
- `set_max_delay -from [get_ports <ports>] -to [get_pins <synchronous pin>] -datapath_only <delay>`
- `set_max_delay -from [get_pins <synchronous pin>] -to [get_ports <ports>] -datapath_only <delay>`
- `set_system_jitter 0.0`
- `set_clock_latency -source -min 0.10`
- `set_clock_latency -source -max 0.20`
- `set_clock_groups -physically_exclusive -group [clk1] -group [clk2]`
- `set_clock_groups -asynchronous -group [clk1] -group [clk2]`

These timing constraints are applied to the OOC module itself. The OOC implementation should constrain all timing into, out of, and internal to the instance. This includes special case paths such as false paths and multicycle paths.

Table 5: **Pblock Constraints**

Command/Property Name	Description
`create_pblock`	Command that creates the initial Pblock for each OOC instance.
`resize_pblock`	Command that defines the site types (SLICE, RAMB36, etc.) and site locations to be owned by the Pblock.
`add_cells_to_pblock`	Command that specifies the instances that will belong to the Pblock. This is typically a level of hierarchy as opposed to individual instances. For OOC implementations, `-top` can be used to specify all cells under the OOC module instead of specifying cell names.
`CONTAIN_ROUTING`	Pblock property that controls the routing to prevent usage of routing resources not owned by the Pblock. Default value is `FALSE`. Only paths that are entirely owned by the Pblock range will be contained (for example, if no BUFGMUX range exists, paths from/to a BUFGMUX will not be contained. This is the desired behavior for many components like a BUFGMUX).
`EXCLUDE_PLACEMENT`	Pblock property that prevents the placement of any logic not belonging to the Pblock inside the defined Pblock range. The default value is `FALSE`. This property has no effect on the OOC implementation, but it affects the placement of the top-level logic during assembly. Xilinx recommends you leave this as `false` for the best results during assembly.
`PARENT`	Pblock property that identifies Pblock hierarchy. The value is the name of the parent Pblock in which the child Pblock is fully contained. Any nested Pblocks with the OOC implementation must use the Parent keyword to get correct behavior during Module Reuse.

Pblock Command and Property Examples:

- `create_pblock <pblock_name>`
- `add_cells_to_pblock [get_pblocks <pblock_name>] -top`
- `resize_pblock [get_pblocks <pblock_name>] -add {SLICE_X0Y0:SLICE_X100Y100}`
- `resize_pblock [get_pblocks <pblock_name>] -add {RAMB18_X0Y0:RAMB18_X2Y20}`
- `set_property CONTAIN_ROUTING true [get_pblocks <pblock_name>]`

Note that the cells added to the Pblock are specified using `-top`. This is because in an out-of-context implementation the OOC instance is the top level, and the entire OOC instance must be contained by the Pblock. Using `-top` also allows the Pblock to be properly translated to the correct level of hierarchy when the OOC module is imported into the top level design.

Nested Pblocks are permitted for floorplanning logic within an OOC module. Any child Pblock must be completely contained within the parent. The parent-child relationship between Pblocks is declared using the PARENT property, as shown here:

```
set_property PARENT <parent_pblock_name> [get_pblocks <child_pblock_name>]
```

Because the PARENT Pblock property references another Pblock, the order in which the constraints are processed does matter. The parent Pblock (which uses -top) must be defined before the child Pblock which references it.

In addition to the timing and physical constraints shown above, there are constraints to define context for an OOC implementation. Context constraints define the environment of the top level into which the OOC implementation is imported.

Table 6: **Context Constraints**

Command/Property Name	Description
HD.CLK_SRC	Used in the OOC implementation to tell the implementation tools if a clock buffer will be used outside of the out-of-context module. The value is the location of the clock buffer instance. This is applied on a clock port, and the port must have a clock defined (create_clock) prior to this constraint being applied.
HD.PARTPIN_LOCS	Defines a specific interconnect tile (INT) for the specified port to be routed. Overrides an HD.PARTPIN_RANGE value. Affects placement and routing of internal OOC logic. Do not use on clock ports because this assumes local routing for the clock. Do not use on dedicated connections.
HD.PARTPIN_RANGE	Defines a range of component sites (SLICE, DSP, block RAM) or interconnect tiles (INT) that can be used to route the specified pin(s)/port(s). This constraint is only valid for pins or ports that are do not have dedicated connections (for example, clocks or direct connections to top-level I/O pads). If applied to these pins or ports, the constraint is ignored.
set_logic_unconnected	Allows for additional optimization for any specified output ports that will be left unconnected in Top.
set_logic_one	Allows for additional optimization for any specified input ports that are driven by VCC in Top.
set_logic_zero	Allows for additional optimization for any specified input ports that are driven by GND in Top.

IMPORTANT: *Incorrectly specifying the set_logic boundary optimization constraints can lead to incorrect behavior and tool errors. For example, defining an output port as unconnected in the OOC module when it is actually used in the top-level can lead to errors such as: ERROR: [Opt 31-67] Problem: A LUT2 cell in the design is missing a connection on input pin I0, which is used by the LUT equation.*

Context Constraint Examples:

- `set_property HD.CLK_SRC BUFGCTRL_X0Y16 [get_ports <port_name>]`
- `set_property HD.PARTPIN_LOCS INT_R_X0Y0 [get_ports <port_name>]`
- `set_property HD.PARTPIN_RANGE SLICE_X0Y1:SLICE_X1Y3 [get_ports <port_name>]`
- `set_logic_unconnected [get_ports <port_name>]`
- `set_logic_one [get_ports <port_name>]`
- `set_logic_zero [get_ports <port_name>]`

By default, in the Module Analysis flow, interface nets (nets inside the module connected to the OOC module ports) are not routed. To have these interface nets routed, you must lock the module ports using `HD.PARTPIN` constraints. To get a quick placement of module ports (or partition pins), the `HD.PARTPIN_RANGE` can be used with a value of the OOC module Pblock `SLICE` range. To obtain more specific placement of these pins, tighter `HD.PARTPIN_RANGE` values can be used, or explicit `HD.PARTPIN_LOCS` values can be specified. To determine what an appropriate site or range might be, open the Device View in the Vivado IDE and enable Routing Resources by clicking this button:  .

When you zoom in, you see INT locations as shown in Figure 1 (routing resources are hidden to simplify this image):

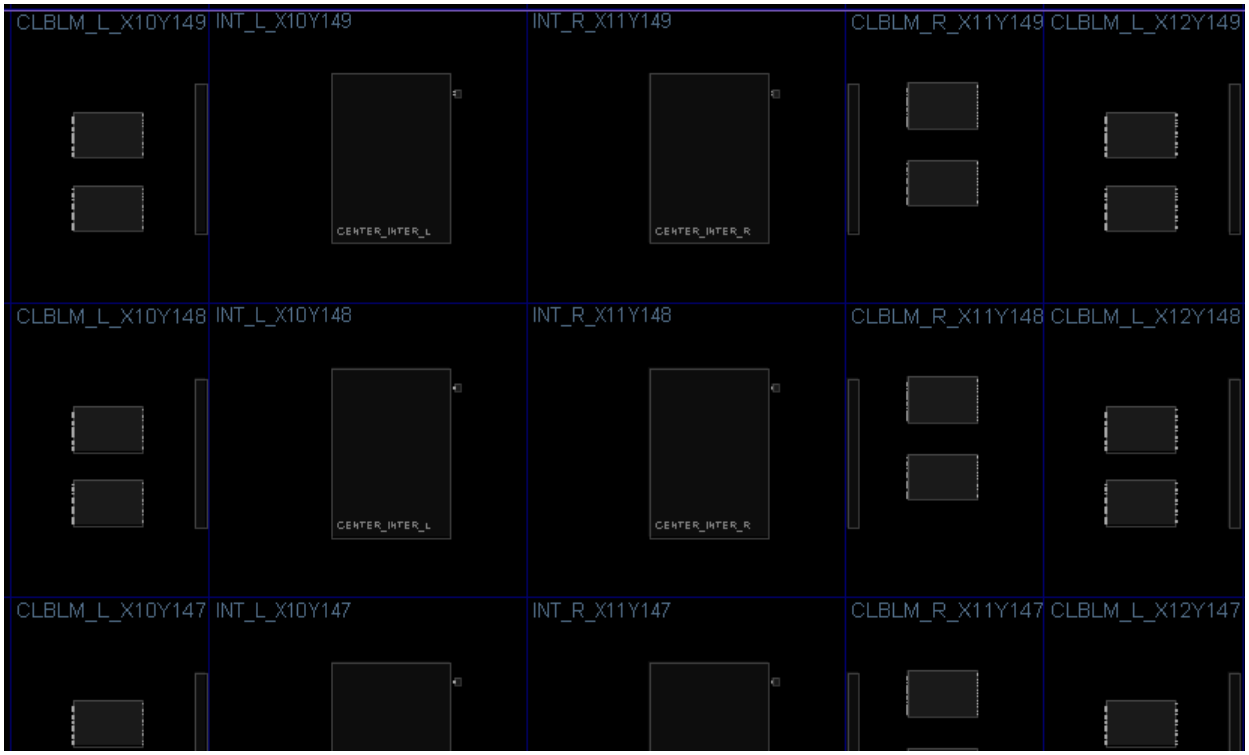

Figure 1: **INT Tile Locations**

Top-Level Reuse Commands and Constraints

The following sections describe commands and constraints used by the top-level design when importing out-of-context implementations. An example design and related scripts are available upon request. Please contact Xilinx support for information on accessing this document.

For more information on individual commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 3].

Top-Level Reuse Commands

Assembly requires the tools to read in a previously implemented module from the Module Analysis flow. A checkpoint for each partitioned instance must exist, and each is read into a top-level design, which has a black box for each OOC module. The OOC implementation results cannot be read into a cell that is not a black box. The standard implementation commands are then used to implement the portions of the design that are not already placed and routed (that is, Top).

Synthesis

You must have a top-level netlist with a black box for each partitioned instance. This requires the top-level synthesis to have module/entity declarations for the partitioned instances, but no logic.

The top level synthesis typically infers I/O buffers on all top level ports. However, if I/O buffers are specifically instantiated in an OOC module, you must turn off I/O buffer insertion in the top-level synthesis on a port-by-port basis. For Vivado synthesis, the attribute for this is `IO_BUFFER_TYPE = "none"`. For more information on `IO_BUFFER_TYPE` and other synthesis attributes, see this link in the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 5].

Implementation

Top level implementation is done the same as it is for standard designs, except for the addition of the following steps.

1. Set the `HD.PARTITION` property on each black box cell to be resolved with OOC implementation results.

   Note: The `HD.PARTITION` property is imported with the OOC module as well, assuming it was set in the OOC implementation. However, it helps to ensure the cell is property marked by setting it on the top-level black box instance.

2. Read in an OOC checkpoint for each partitioned instance.
3. Choose the level of preservation to maintain (logical, placement, or routing).

Mark Cells as Partitions

Marking a cell as a partition requires an `HD.PARTITION` cell property. Setting `HD.PARTITION` does these things.

- Sets `DONT_TOUCH` on the specified cell to prevent illegal optimization across and on the HD boundary
- Triggers HD-specific DRCs
- Enables special code in `read_checkpoint -cell` to clean up PartPins and remove redundant Pblocks

```
set_property HD.PARTITION 1 [get_cells <cell_name>]
```

Read in OOC Checkpoint

Reading in an OOC checkpoint is done using the `read_checkpoint` command with the `-cell` option. The top level design must already be opened, and must have a black box for each partitioned instance.

```
read_checkpoint -cell <cell_name> <file> [-strict]
```

Table 7: **read_checkpoint Switches**

Switch Name	Description
`-cell`	Specifies the full hierarchical name of the OOC module.
`-strict`	Requires exact ports match for replacing cell, and checks that part, package, and speed grade values are identical.
`<file>`	Specifies the OOC checkpoint to be read in.

When importing and locking down the results of an OOC module implementation, interface nets are not preserved. Also, any PartPin locations generated during the initial Top-Down or OOC implementation are likely not ideal, or impose unnecessary restrictions on the router. Therefore, the `read_checkpoint -cell` command automatically unroutes any interface nets, and removes all PartPins from the cell checkpoint.

Setting Preservation Level

After reading in the OOC checkpoint, the preservation level for this module must be defined.

To lock the placement and/or routing of an imported OOC checkpoint, use a `lock_design` command.

```
lock_design [-level <value>] [-unlock] [<cell>]
```

Within this command, the following preservation levels are available as values for the `-level` switch:

- **Logical** – Preserves the logical design. Any placement or routing information is still used, but can be changed if the tools can achieve better results.
- **Placement** (default) – Preserves the logical and placed design. Any routing information is still used, but can be changed if the tools can achieve better results.
- **Routing** – Preserves the logical, placed and routed design. Internal routes are preserved, but interface nets are not. To preserve routing, the `CONTAIN_ROUTING` property must have been used on the Pblock during the OOC implementation. This ensures that there are no routing conflicts when the OOC implementation is reused.

Note that regardless of the desired preservation level, the entire physical database will still be read in (including routing) but will not be modified unless the tools determine that better results can be obtained.

Table 8: **`lock_design` Arguments**

Argument Name	Description
`-level`	Specifies the preservation level. Values are `logical`, `placement`, or `routing`. The default value is `placement`.
`-unlock`	Unlock cells. If cells are not specified, the whole design is unlocked. The `-level` parameter must be specified for unlocking, and works the opposite of locking behavior. Specifying `routing` only unlocks routing, while specifying `placement` unlocks placement and routing.
`<cell>`	This is the hierarchical cell name to be locked. If cells are not specified, the whole design is locked.

Top-Level Reuse Constraints

When reusing an out-of-context module in a top-level design, all normal design constraints can be applied. The out-of-context constraints used in the out-of-context implementation are saved in the checkpoint, and are also applied (where applicable) to the design.

Tcl Scripts

Tcl scripts for this flow are available upon request. Please contact Xilinx support for information.

Known Issues

This section reports a few known issues with the current release for Hierarchical Design flows.

Limitations on Global Clock Routing

Clocks driven by a buffer in the top level will not be routed during the OOC implementation. Routing estimations will be used, and the use of `HD.CLK_SRC` will help improve routing estimates. Clock buffers within the OOC module will be routed during OOC implementation. With proper constraints on the OOC clocks, this timing estimation is sufficient to give accurate results for timing.

Limitations on OOC Module with IDELAYCTRL

An OOC module with IDELAYCTRL can be imported, but not locked. Therefore, the OOC module will not be preserved 100%.

No Project Mode Support

There is currently no project mode support for Hierarchical Design in the Vivado Design Suite.

Appendix A

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.

References

1. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)
2. *Vivado Design Suite User Guide: Implementation* (UG904)
3. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
4. *Vivado Design Suite User Guide: Using Constraints* (UG903)
5. *Vivado Design Suite User Guide: Synthesis* (UG901)
6. *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909)
7. 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 QuickTake Video Tutorials
2. Designing FPGAs Using the Vivado Design Suite 4 Training Course

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.