Vivado Design Suite Tutorial

Partial Reconfiguration

UG947 (v2016.1) April 6, 2016

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
04/06/2016	2016.1	Added two additional Partial Reconfiguration labs: • Lab 2: UltraScale Devices • Lab 3: Partial Reconfiguration Controller IP Updated Tutorial Design Description to account for new labs.

Table of Contents

Overview

This tutorial covers the Partial Reconfiguration (PR) software support in Vivado® Design Suite release 2016.1. The tutorial steps through basic information about the current Partial Reconfiguration (PR) design flow, example Tcl scripts, and shows results within the Vivado integrated design environment (IDE). You run scripts for part of the tutorial and work interactively with the design for other parts. You can also script the entire flow, and a completed script is included with the tutorial files. The focus of this tutorial is specifically the software flow from RTL to bitstream, demonstrating how to process a Partial Reconfiguration design. For more information about design considerations and techniques, further details about the commands and constraints, and other aspects of building a partially reconfigurable design, see the *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909).

VIDEO:
The following videos provide an overview of Vivado Partial Reconfiguration solutions:

- *Vivado Design Suite QuickTake Video Tutorial: Partial Reconfiguration in Vivado*
- *Vivado Design Suite QuickTake Video Tutorial: Partial Reconfiguration for UltraScale*

TRAINING: *Xilinx provides training courses that can help you learn more about the concepts presented in this document. Use the link to explore related courses:*

- *Xilinx Partial Reconfiguration Tools & Techniques*
- *Partial Reconfiguration Flow on Zynq using Vivado*

Hardware and Software Requirements

This tutorial requires that the 2016.1 Vivado Design Suite software release or later is installed. A Partial Reconfiguration license is required to run the PR software tools in the Vivado Design Suite. If necessary, request access by sending an email to pr_access@xilinx.com for a 30-day evaluation license.

For Operating Systems support, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) for a complete list and description of the system and software requirements.

Tutorial Design Description

Designs for the tutorial labs are available as a zipped archive on the Xilinx website. Each lab in this tutorial has its own folder within the zip file. To access the tutorial design files:

1. Download the Reference Design Files from the Xilinx website.
2. Extract the zip file contents to any write-accessible location.

Lab 1: 7 Series Basic Flow

The sample design used throughout this tutorial is called `led_shift_count_k7`. The design targets an xc7k325t device for use on the KC705 demonstration board, Rev 1.0 or 1.1.

This design is very small, which (1) helps minimize data size and (2) allows you to run the tutorial quickly, with minimal hardware requirements.

Lab 2: UltraScale Basic Flow

The sample design used throughout this tutorial is called `led_shift_count_us`. The design targets an xcku040-2FFVA1156E device for use on the KCU105 demonstration board, Rev 1.0.

Lab 3: Partial Reconfiguration Controller IP

The sample design used throughout this tutorial is called `prc_k7`. The design targets an xc7k325t device for use on the KC705 demonstration board, rev 1.0 or 1.1. While this lab targets a 7 Series device, the concepts apply to UltraScale™ devices as well.

Step 1: Extract the Tutorial Design Files

1. To obtain the tutorial design file, see Tutorial Design Description.
2. Navigate to `\led_shift_count_k7` in the extracted files. The `led_shift_count_k7` data directory is referred to in this tutorial as the `<Extract_Dir>`.

Step 2: Examine the Scripts

Start by reviewing the scripts provided in the design archive. The files `design.tcl` and `design_complete.tcl` are located at the root level. Both files contain the same information, but `design.tcl` has parameters set such that only synthesis runs, while `design_complete.tcl` runs the entire flow for two configurations.

The Main Script

In the `<Extract_Dir>`, open `design.tcl` in a text editor. This is the master script where you define the design parameters, design sources, and design structure. This is the only file you have to modify to compile a complete Partial Reconfiguration design. Find more details regarding `design.tcl` and the underlying scripts in the `README.txt` located in the `Tcl` subdirectory.

Note the following details in this file:

- Visualization scripts are requested using `set_param hd.visual 1` (on line 5, the `set_param` command is called). This command creates scripts in the root directory that you use later in the tutorial to identify the frames to be included in the partial bitstreams.
- Under **flow control**, you can control what phases of synthesis and implementation are run. In the tutorial, only synthesis is run by the script; implementation, verification, and bitstream generation are run interactively. To run these additional steps via the script, set the flow variables (e.g., `run.prImpl`) to **1**.
- The **Output Directories** and **Input Directories** set the file structure expected for design sources and results files. You must reflect any changes to your file structure here.

- The **Top Definition** and **RP Module Definitions** sections allow you to reference all source files for each part of your design. Top Definition covers all sources needed for the static design, including constraints and IP. The RP Module Definitions section does the same for Reconfigurable Partitions (RP). Complete a section for each RP and list all Reconfigurable Module (RM) variants for each RP.
    - This design has two Reconfigurable Partitions (`inst_shift` and `inst_count`), and each RP has two module variants.
- The **Configuration Definition** sections define the sets of static and reconfigurable modules that make up a configuration.
    - This design has two configurations, `Config_shift_right_count_up_implement` and `Config_shift_left_count_down_import`. You can create more configurations by adding RMs or by combining existing RMs.

The Supporting Scripts

Underneath the `Tcl` subdirectory, several supporting Tcl scripts exist. The scripts are called by `design.tcl`, and they manage specific details for the Partial Reconfiguration flow. Provided below are some details about a few of the key PR scripts.

CAUTION! *Do not modify the supporting Tcl scripts.*

- `step.tcl`
  Manages the current status of the design by monitoring checkpoints.
- `synth.tcl`
  Manages the details regarding the synthesis phase.
- `impl.tcl`
  Manages the details regarding the module implementation phase.
- `pr_impl.tcl`
  Manages the details regarding the top-level implementation of a PR design.
- `run.tcl`
  Launches the actual runs for synthesis and implementation.
- `log.tcl`
  Handles report file creation at key points during the flow.

Remaining scripts provide details within these scripts (such as the `*_utils.tcl` scripts) or manage other Hierarchical Design flows (such as `ooc_impl.tcl`).

Step 3: Synthesize the Design

The `design.tcl` script automates the synthesis phase of this tutorial. Five iterations of synthesis are called, one for the static top-level design and one for each of four Reconfigurable Modules.

1. Open the Vivado Tcl shell:
   - On Windows, select the Xilinx Vivado desktop icon or **Start > All Programs > Xilinx Design Tools> Vivado 2016.x > Vivado 2016.x Tcl Shell**.
   - On Linux, simply type, `vivado -mode tcl`.
2. In the shell, navigate to the <`Extract_Dir`> directory.
3. Run the `design.tcl` script by entering:

```
source design.tcl -notrace
```

After all five passes through Vivado Synthesis have completed, the Vivado Tcl shell is left open. You can find log and report files for each module, alongside the final checkpoints, under each named folder in the `Synth` subdirectory.

***TIP:** In the <`Extract_Dir`> directory, multiple log files have been created:*

- *`run.log` shows the summary as posted in the Tcl shell window*
- *`command.log` echoes all the individual steps run by the script*
- *`critical.log` reports all critical warnings produced during the run*

Step 4: Assemble the Design

Now that the synthesized checkpoints for each module, plus top, are available, you can assemble the design. Because project support for Partial Reconfiguration flows is not yet in place, you do not use the project infrastructure from within the IDE.

You will run all flow steps from the Tcl Console, but you can use features within the IDE (such as the floorplanning tool) for interactive events.

***TIP:** Copy and paste commands directly from this document to avoid redundant effort and typos in the Vivado IDE. Copy and paste only one full command at a time. Note that some commands are long and therefore span multiple lines.*

Implement the Design

1. Open the Vivado IDE. You can open the IDE from the open Tcl shell by typing `start_gui` or by launching Vivado with the command `vivado -mode gui`.
2. Navigate to the `<Extract_Dir>` directory if you are not already there. The pwd command can confirm this.
3. Load the static design by issuing the following command in the Tcl Console:

   ```
   open_checkpoint Synth/Static/top_synth.dcp
   ```

   You can see the design structure in the Netlist pane, but black boxes exist for the `inst_shift` and `inst_count` modules. Note that the Flow Navigator pane is not present. You are working in non-project mode.

   TIP: *Place the IDE in floorplanning mode by selecting **Layout > Floorplanning**. Make sure the Device view is visible.*

   Two critical warnings are issued regarding unmatched instances. These instances are the Reconfigurable Modules that have yet to be loaded, and you can therefore ignore these warnings safely.
4. Load the synthesized checkpoints for first Reconfigurable Module variants for each of reconfigurable partitions:

   ```
   read_checkpoint -cell inst_shift Synth/shift_right/shift_synth.dcp
   read_checkpoint -cell inst_count Synth/count_up/count_synth.dcp
   ```

   Note: *The `inst_shift` and `inst_count` modules have been filled in with logical resources. You can now traverse the entire hierarchy within the Netlist pane.*
5. Define each of these submodules as partially reconfigurable by setting the `HD.RECONFIGURABLE` property:

   ```
   set_property HD.RECONFIGURABLE 1 [get_cells inst_shift]
   set_property HD.RECONFIGURABLE 1 [get_cells inst_count]
   ```
6. Save the assembled design state for this initial configuration:

   ```
   write_checkpoint ./Checkpoint/top_link_right_up.dcp
   ```

Step 5: Build the Design Floorplan

Next, you must create a floorplan to define the regions that will be partially reconfigured.

1. Select the `inst_count` instance in the Netlist pane. Right-click and select **Floorplanning > Draw Pblock** and draw a tall narrow box on the left side of the X0Y3 clock region. The exact size and shape do not matter at this point, but keep the box within the clock region.

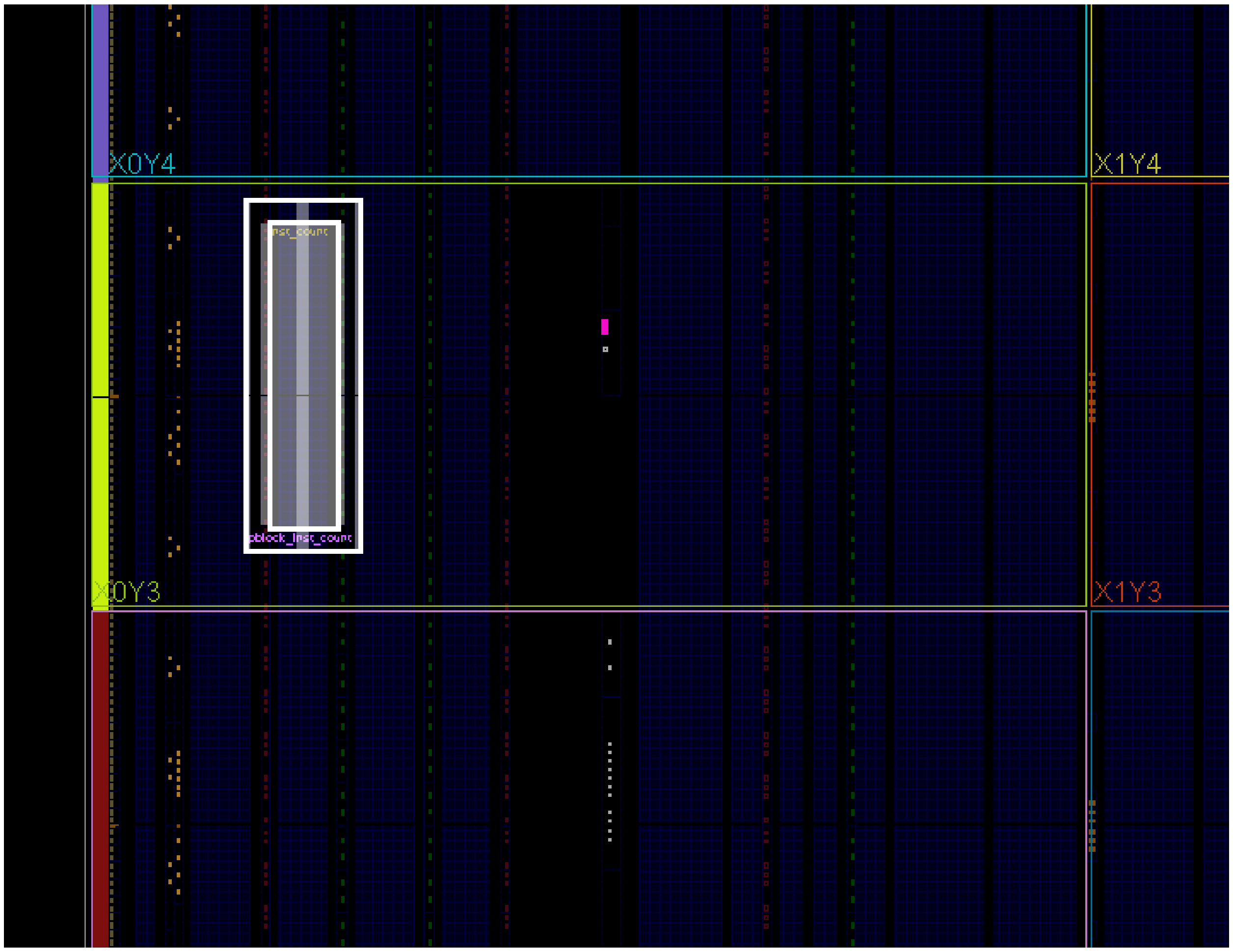


Figure 1: Pblock for the inst_count Reconfigurable Partition

   Although this Reconfigurable Module only requires CLB resources, also include RAMB16, RAMB32, or DSP48 resources if the box encompasses those types. This allows the routing resources for these block types to be included in the reconfigurable region. The **General** tab of the Pblock Properties pane can be used to add these if needed. The **Statistics** tab shows the resource requirements of the currently loaded Reconfigurable Module.

2. In the Properties pane, select the checkbox for **RESET_AFTER_RECONFIG** to utilize the dedicated initialization of the logic in this module after reconfiguration has completed.

3. Repeat steps 1 and 2 for the `inst_shift` instance, this time targeting the right side of clock region X1Y1. This Reconfigurable Module includes block RAM instances, so the resource type must be included. If omitted, the RAMB details in the **Statistics** tab are shown in red.

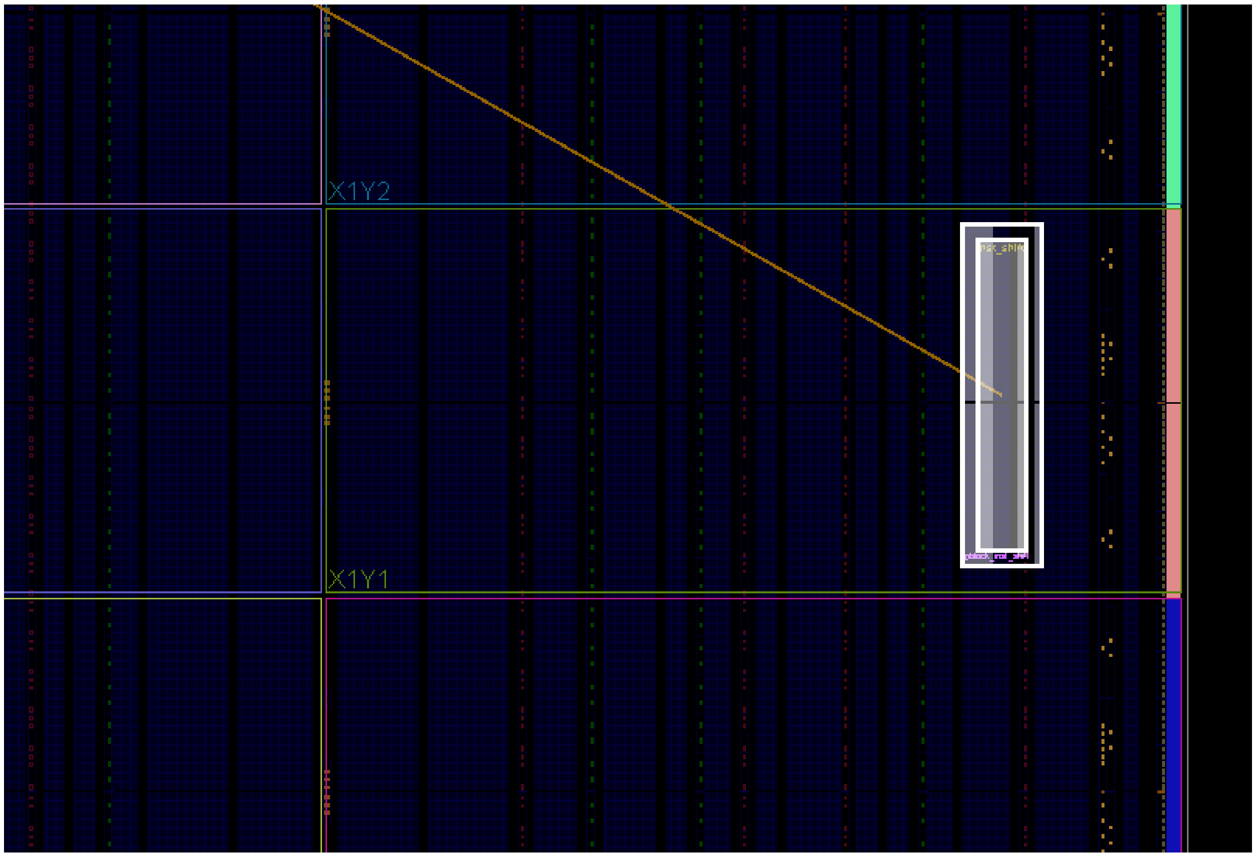
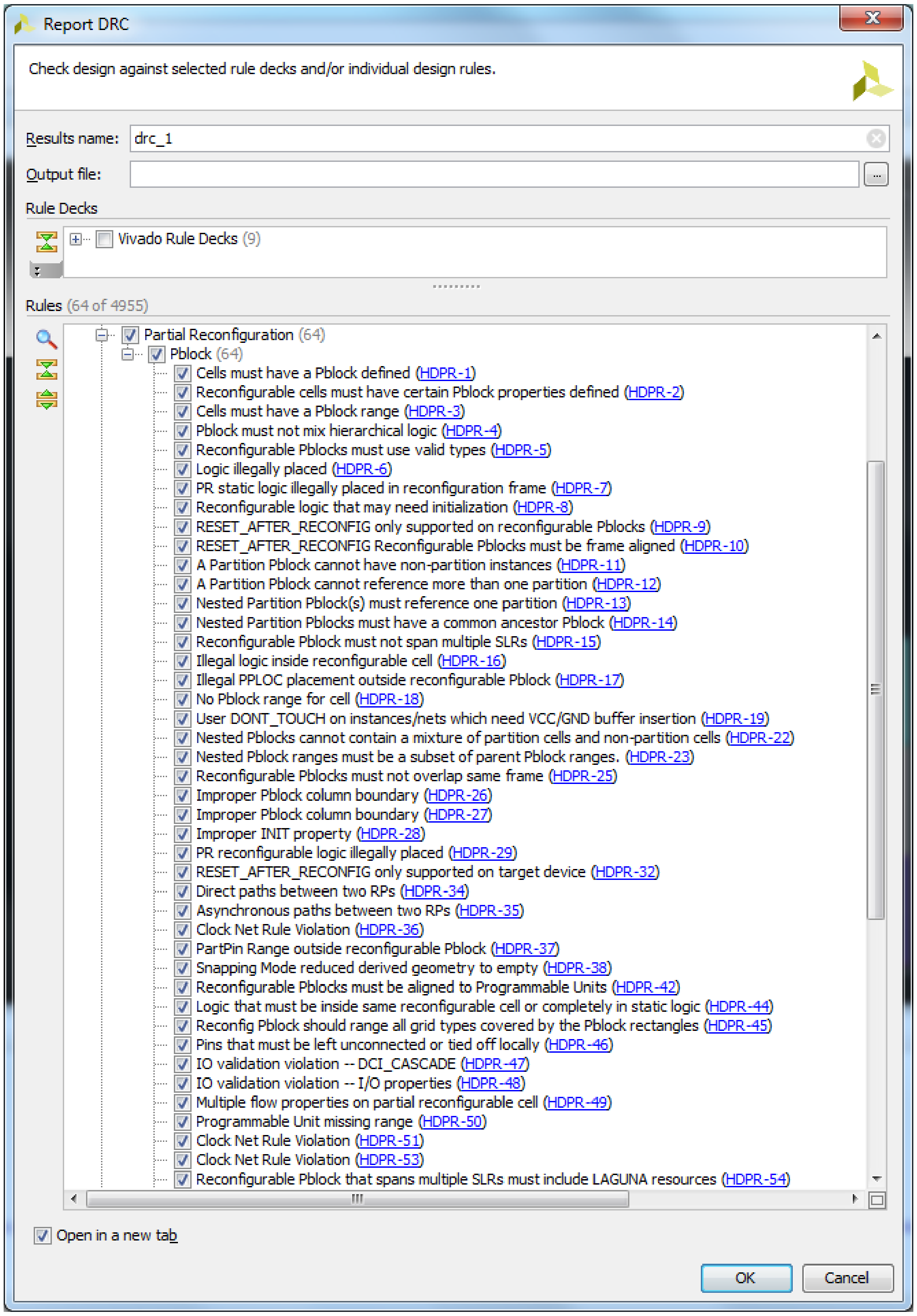


Figure 2: Pblock for the inst_shift Reconfigurable Partition

4. Run Partial Reconfiguration Design Rule Checks by selecting **Tools > Report > Report DRC**. You can uncheck **All Rules** and then check **Partial Reconfiguration** to focus this report strictly on PR DRCs.


Report DRC

Check design against selected rule decks and/or individual design rules.

Results name: drc_1

Output file:

Rule Decks

Vivado Rule Decks (9)

Rules (64 of 4955)

- ✓ Partial Reconfiguration (64)
  - ✓ Pblock (64)
    - ✓ Cells must have a Pblock defined (HDPR-1)
    - ✓ Reconfigurable cells must have certain Pblock properties defined (HDPR-2)
    - ✓ Cells must have a Pblock range (HDPR-3)
    - ✓ Pblock must not mix hierarchical logic (HDPR-4)
    - ✓ Reconfigurable Pblocks must use valid types (HDPR-5)
    - ✓ Logic illegally placed (HDPR-6)
    - ✓ PR static logic illegally placed in reconfiguration frame (HDPR-7)
    - ✓ Reconfigurable logic that may need initialization (HDPR-8)
    - ✓ RESET_AFTER_RECONFIG only supported on reconfigurable Pblocks (HDPR-9)
    - ✓ RESET_AFTER_RECONFIG Reconfigurable Pblocks must be frame aligned (HDPR-10)
    - ✓ A Partition Pblock cannot have non-partition instances (HDPR-11)
    - ✓ A Partition Pblock cannot reference more than one partition (HDPR-12)
    - ✓ Nested Partition Pblock(s) must reference one partition (HDPR-13)
    - ✓ Nested Partition Pblocks must have a common ancestor Pblock (HDPR-14)
    - ✓ Reconfigurable Pblock must not span multiple SLRs (HDPR-15)
    - ✓ Illegal logic inside reconfigurable cell (HDPR-16)
    - ✓ Illegal PPLOC placement outside reconfigurable Pblock (HDPR-17)
    - ✓ No Pblock range for cell (HDPR-18)
    - ✓ User DONT_TOUCH on instances/nets which need VCC/GND buffer insertion (HDPR-19)
    - ✓ Nested Pblocks cannot contain a mixture of partition cells and non-partition cells (HDPR-22)
    - ✓ Nested Pblock ranges must be a subset of parent Pblock ranges. (HDPR-23)
    - ✓ Reconfigurable Pblocks must not overlap same frame (HDPR-25)
    - ✓ Improper Pblock column boundary (HDPR-26)
    - ✓ Improper Pblock column boundary (HDPR-27)
    - ✓ Improper INIT property (HDPR-28)
    - ✓ PR reconfigurable logic illegally placed (HDPR-29)
    - ✓ RESET_AFTER_RECONFIG only supported on target device (HDPR-32)
    - ✓ Direct paths between two RPs (HDPR-34)
    - ✓ Asynchronous paths between two RPs (HDPR-35)
    - ✓ Clock Net Rule Violation (HDPR-36)
    - ✓ PartPin Range outside reconfigurable Pblock (HDPR-37)
    - ✓ Snapping Mode reduced derived geometry to empty (HDPR-38)
    - ✓ Reconfigurable Pblocks must be aligned to Programmable Units (HDPR-42)
    - ✓ Logic that must be inside same reconfigurable cell or completely in static logic (HDPR-44)
    - ✓ Reconfig Pblock should range all grid types covered by the Pblock rectangles (HDPR-45)
    - ✓ Pins that must be left unconnected or tied off locally (HDPR-46)
    - ✓ IO validation violation -- DCI_CASCADE (HDPR-47)
    - ✓ IO validation violation -- I/O properties (HDPR-48)
    - ✓ Multiple flow properties on partial reconfigurable cell (HDPR-49)
    - ✓ Programmable Unit missing range (HDPR-50)
    - ✓ Clock Net Rule Violation (HDPR-51)
    - ✓ Clock Net Rule Violation (HDPR-53)
    - ✓ Reconfigurable Pblock that spans multiple SLRs must include LAGUNA resources (HDPR-54)

✓ Open in a new tab

Figure 3: Partial Reconfiguration Design Rule Checks (DRCs)

One or two DRCs are reported at this point, and there are two ways of resolving them. We will use one method for `inst_shift` and the other for `inst_count`.

The first DRC is an error, HDPR-10, reporting that `RESET_AFTER_RECONFIG` requires Pblock frame alignment.

5. To resolve the first DRC error, make sure that the height of the Pblock aligns with the clock region boundaries. Using the Pblock for `inst_shift`, stretch the top and bottom edges to match the clock region boundaries of X1Y1 as shown in the figure below. See that the shading of the Pblock is now more uniform.

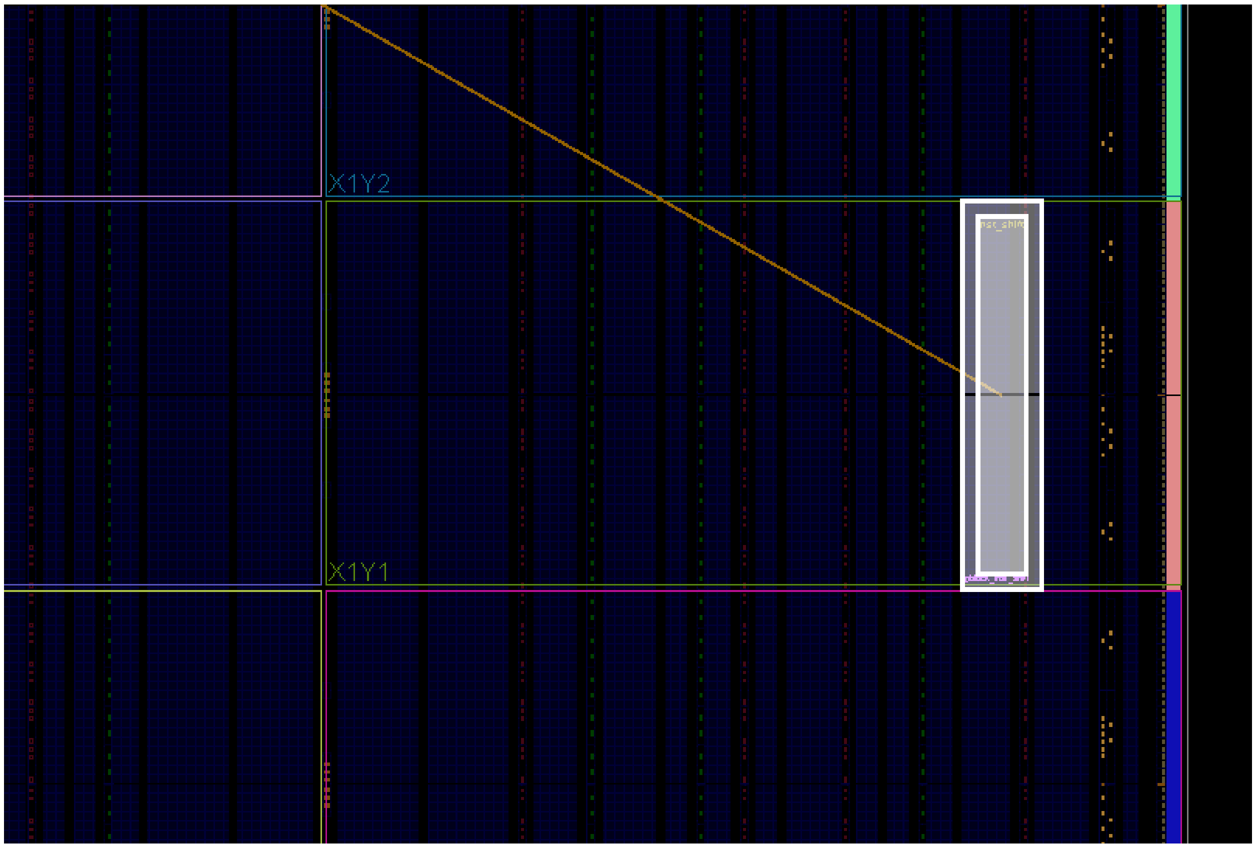


Figure 4: Pblock for the aligned `inst_shift` Reconfigurable Partition

The other possible DRC is a warning, HDPR-26, reporting that a left or right edge of a reconfigurable Pblock terminates on an improper boundary. Left or right edges must not split interconnect (INT) columns. More information on this requirement can be found in the *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909), in the section entitled Reconfigurable Partition Pblock Sizes and Shapes.

6. To manually avoid this DRC warning, zoom into the upper or lower corner on the reported edge of `inst_shift` (or `inst_count`, if `inst_shift` did not report an issue) to see where the violation has occurred. Move this edge left or right one column, as shown by the yellow arrows in Figure 5, so it lands between two resource types (CLB-CLB or CLB-RAMB, for example) instead landing between CLB-INT or BRAM-INT.

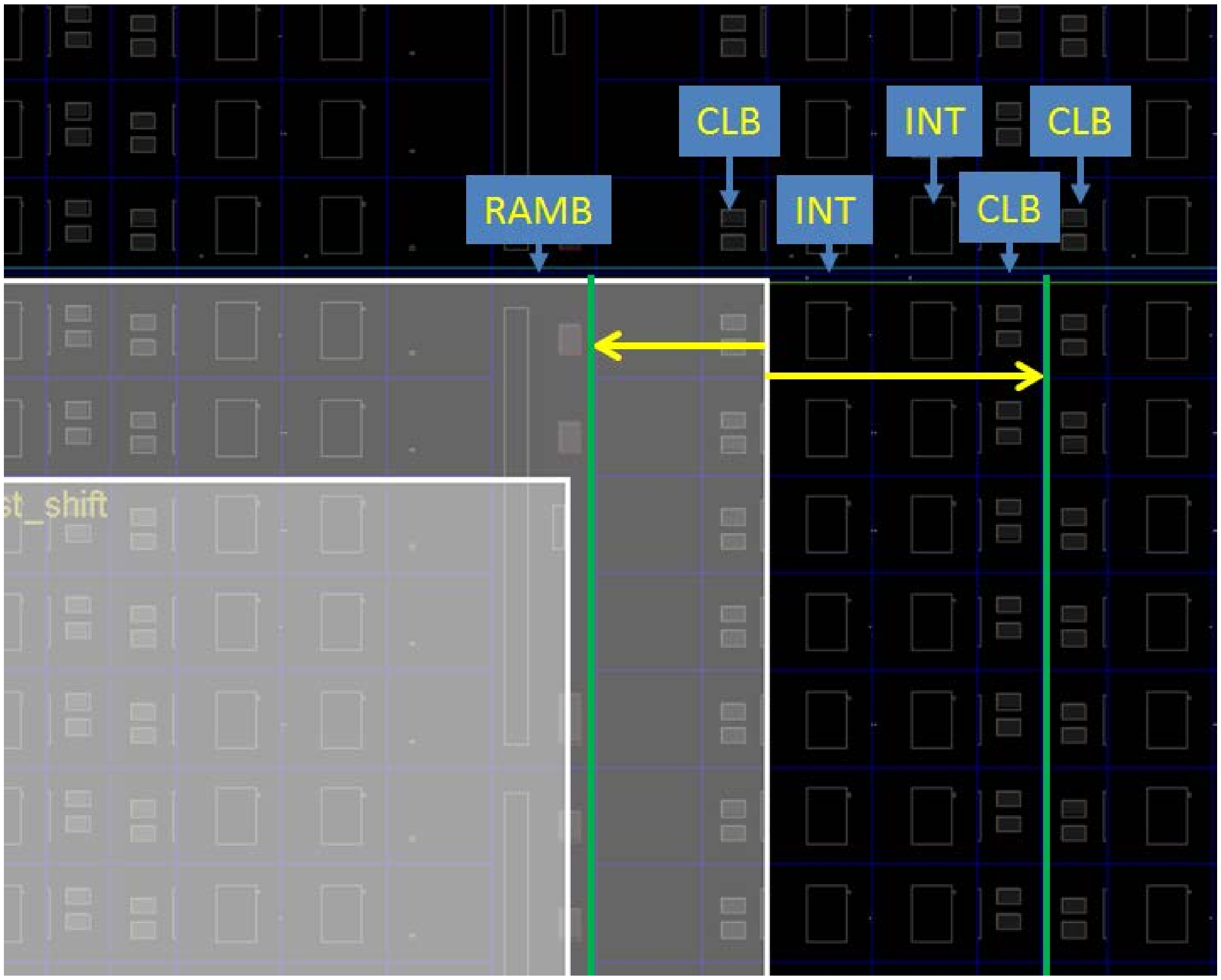


Figure 5: Adjusting the Edges of a Reconfigurable Pblock

7. Run the PR DRCs again to confirm that the errors and warnings that you have addressed have been resolved for the `inst_shift` instance.

   An alternative to manually adjusting the size and shape of reconfigurable Pblocks is to use the `SNAPPING_MODE` feature. This feature automatically adjusts edges to align with legal boundaries. It will make the Pblock taller, aligning with clock region boundaries, if the `RESET_AFTER_RECONFIG` feature is selected. It makes the Pblock narrower, adjusting left and/or right edges as needed. Note that the number and type of resources available are altered if `SNAPPING_MODE` makes changes to the Pblock.

8. Select the Pblock for `inst_count`, and in the **Properties** tab of the Pblock Properties pane, change the value of `SNAPPING_MODE` from OFF to ROUTING (or ON).

   Note that the original Pblock does not change. The adjustments to the Pblock needed for it to conform to PR rules are done automatically, without modifying your source constraints.

9. Run the PR DRCs once again to confirm that all issues have been resolved.

10. Save these Pblocks and associated properties by issuing this command in the Tcl Console:

    ```
    write_xdc ./Sources/xdc/fplan.xdc
    ```

    This exports all the current constraints in the design. These constraints can be managed in their own XDC file, merged with another XDC file (such as `top_io.xdc`), or managed within a run script (as is typically done with `HD.RECONFIGURABLE`).

Now that the floorplan is established, the next step implementing the design.

Step 6: Implement the First Configuration

In this step, you place and route the design and prepare the static portion of the design for reuse with new Reconfigurable Modules.

1. Load the top-level constraint file by issuing the command:

   ```
   read_xdc Sources/xdc/top_io.xdc
   ```

   This sets the device pinout and top-level timing constraints. This XDC file is not accessible from the IDE – it will not appear as a design source.

   This top-level XDC file should only contain constraints that reference objects in the static design. Constraints for logic or nets inside of the RP can be applied for specific Reconfigurable Modules if needed.

2. Optimize, place, and route the design by issuing the following command:

   ```
   opt_design
   ```

   This is the point at which the Partial Reconfiguration license is checked. If you have a valid license, you see this message:

   ```
   Feature available: PartialReconfiguration
   ```

   If you have no license with the `PartialReconfiguration` feature, contact your local Xilinx sales office for more information. Evaluation licenses are available.

   ```
   place_design
   route_design
   ```

   After both `place_design` and `route_design`, examine the state of the design in the Device view (See Figure 6). One thing to note after `place_design` is the introduction of Partition Pins. These are the physical interface points between static and reconfigurable logic. They are anchor points within an interconnect tile through which each I/O of the Reconfigurable Module must route. They appear as white boxes in the placed design view.

For `pblock_shift`, they appear in the top of that Pblock, as the connections to static are just outside the Pblock in that area of the device. For `pblock_count`, they appear outside the user-defined region, as SNAPPING_MODE has vertically collected more frames to be added to the Reconfigurable Partition.

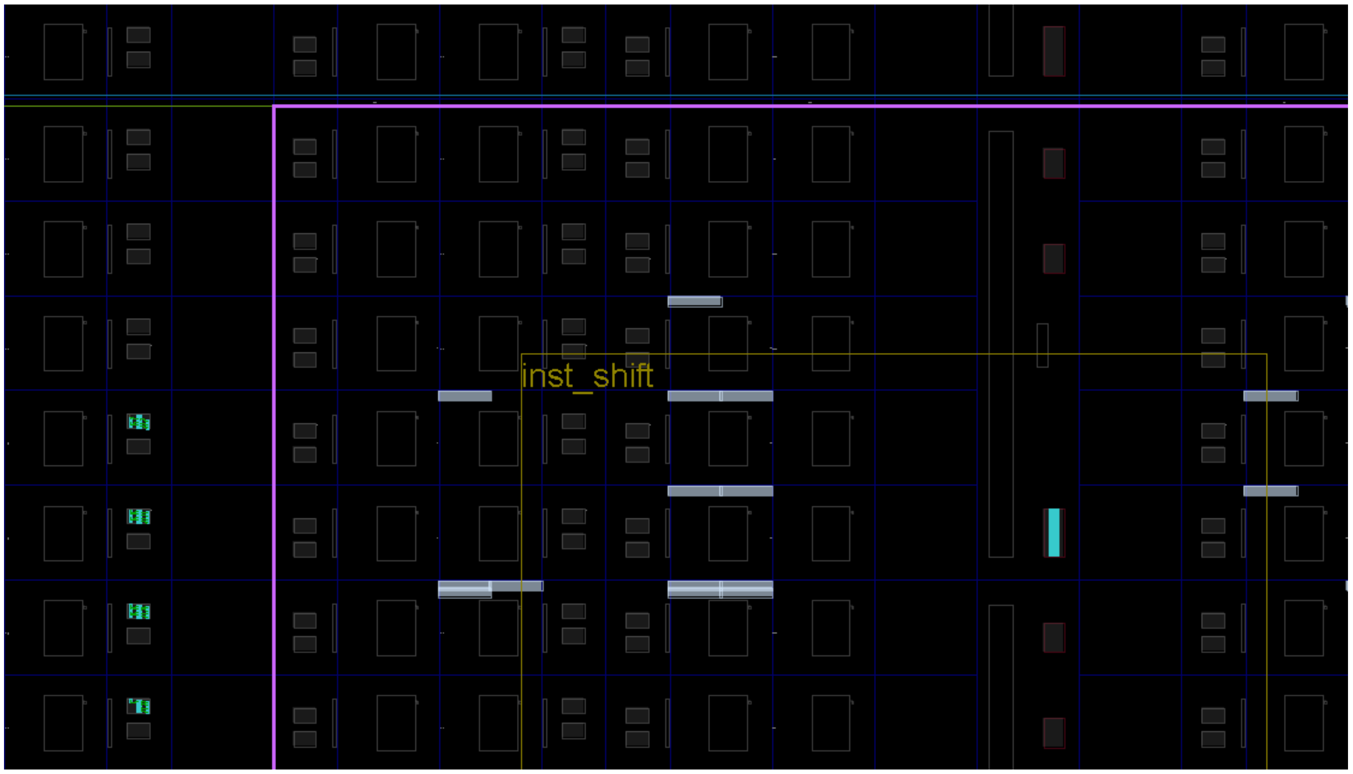


Figure 6: Partition Pins within Placed Design

3. To find these partition pins in the GUI easily:
   a. Select the Reconfigurable Module (for example, `inst_shift`) in the Netlist pane.
   b. Select the **Cell Pins** tab in the Cell Properties pane.
4. Select any pin to highlight it, or use Ctrl+A to select them all. The Tcl equivalent of the latter is:

   ```
   select_objects [get_pins inst_shift/*]
   ```

5. In the routed design view, click the **Show/Hide Nets** icon to display all routes by type (Fully Routed, Partially Routed, or Unrouted), as shown in Figure 7.

   Use the Routing Resources icon to toggle between abstracted and actual routing information, and to change the visibility of the routing resources themselves. All nets in the design are fully routed at this point.

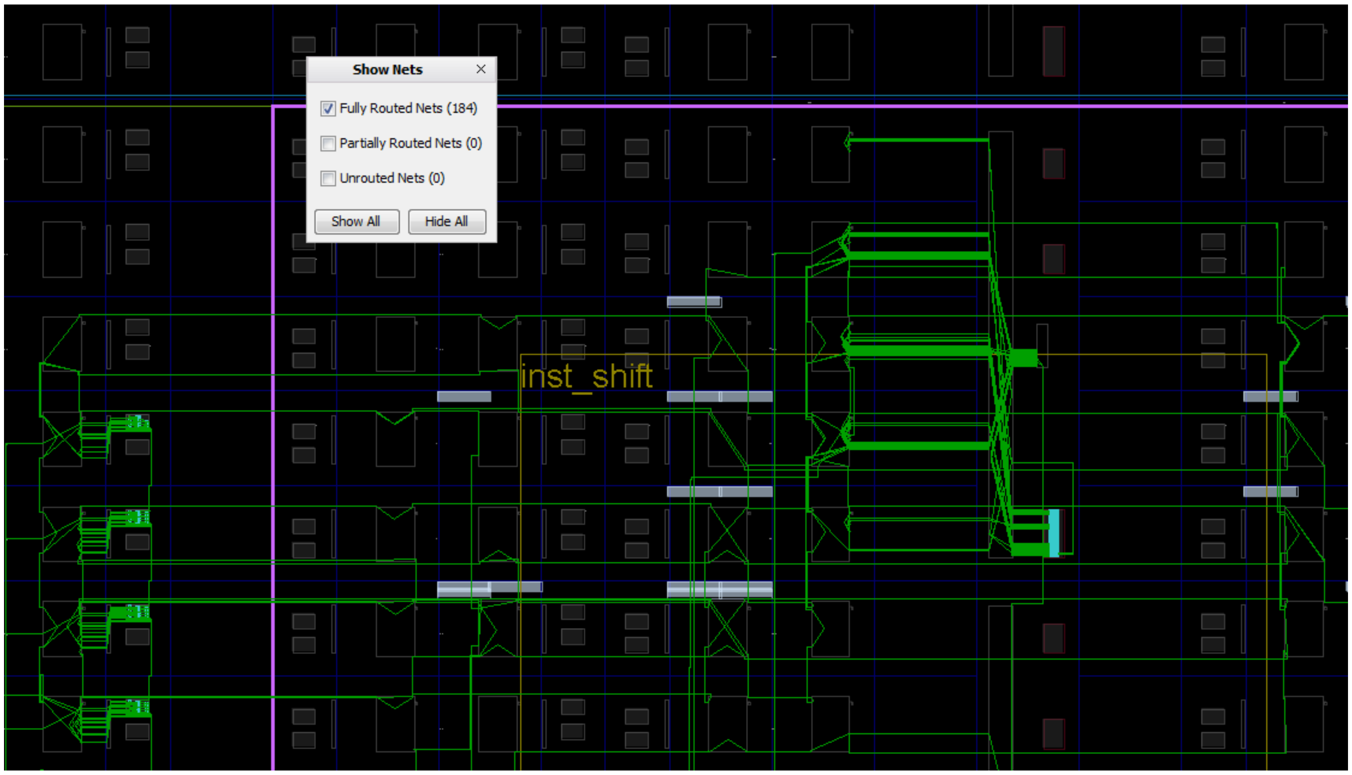


Figure 7: Close-up of First Configuration Routed

Save the Results

1. Save the full design checkpoint and create report files by issuing these commands:

```
write_checkpoint -force\
Implement/Config_shift_right_count_up_implement/top_route_design.dcp

report_utilization -file\
Implement/Config_shift_right_count_up_implement/top_utilization.rpt

report_timing_summary -file\
Implement/Config_shift_right_count_up_implement/top_timing_summary.rpt
```

2. [Optional] Save checkpoints for each of the Reconfigurable Modules by issuing these two commands:

```
write_checkpoint -force -cell inst_shift\
Checkpoint/shift_right_route_design.dcp

write_checkpoint -force -cell inst_count\
Checkpoint/count_up_route_design.dcp
```

***TIP:** When running `design_complete.tcl` to process the entire design in batch mode; design checkpoints, log files, and report files are created at each step of the flow.*

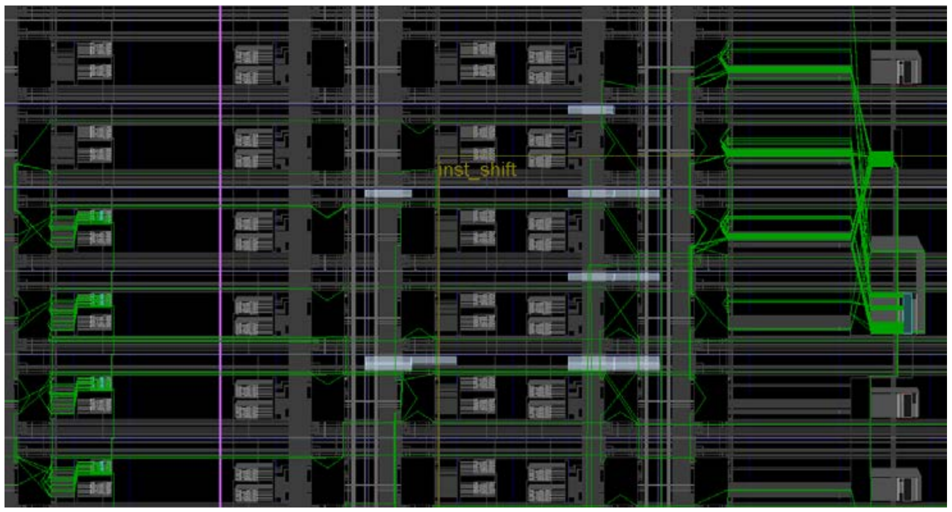
At this point, you have created a fully implemented partial reconfiguration design from which you can generate full and partial bitstreams. The static portion of this configuration is used for all subsequent configurations, and to isolate the static design, the current Reconfigurable Modules must be removed. Make sure routing resources are enabled, and zoom in to an interconnect tile with partition pins.

3. Clear out Reconfigurable Module logic by issuing the following commands:

```
update_design -cell inst_shift -black_box
update_design -cell inst_count -black_box
```

Issuing these commands results in many design changes as shown in the figure below:

- The number of Fully Routed nets (green) has decreased.
- `inst_shift` and `inst_count` now appear in the Netlist view as empty.



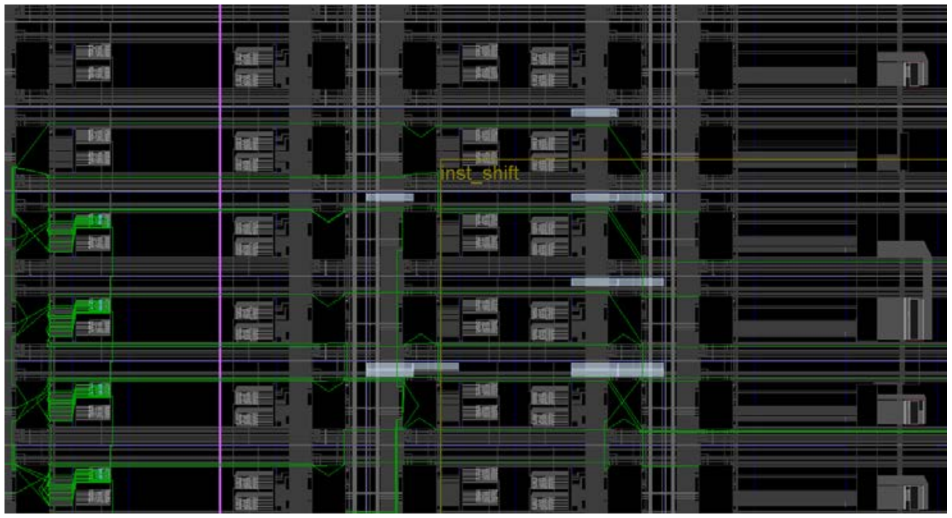


Figure 8: The inst_shift module before (top) and after (bottom) `update_design -black_box`

4. Issue the following command to lock down all placement and routing:

```
lock_design -level routing
```

Because no cell was identified in the `lock_design` command, the entire design in memory (currently consisting of the static design with black boxes) is affected. All routed nets are now displayed as locked, as indicated by dashed lines in the figure below. All placed components changed from blue to orange to show they are also locked.

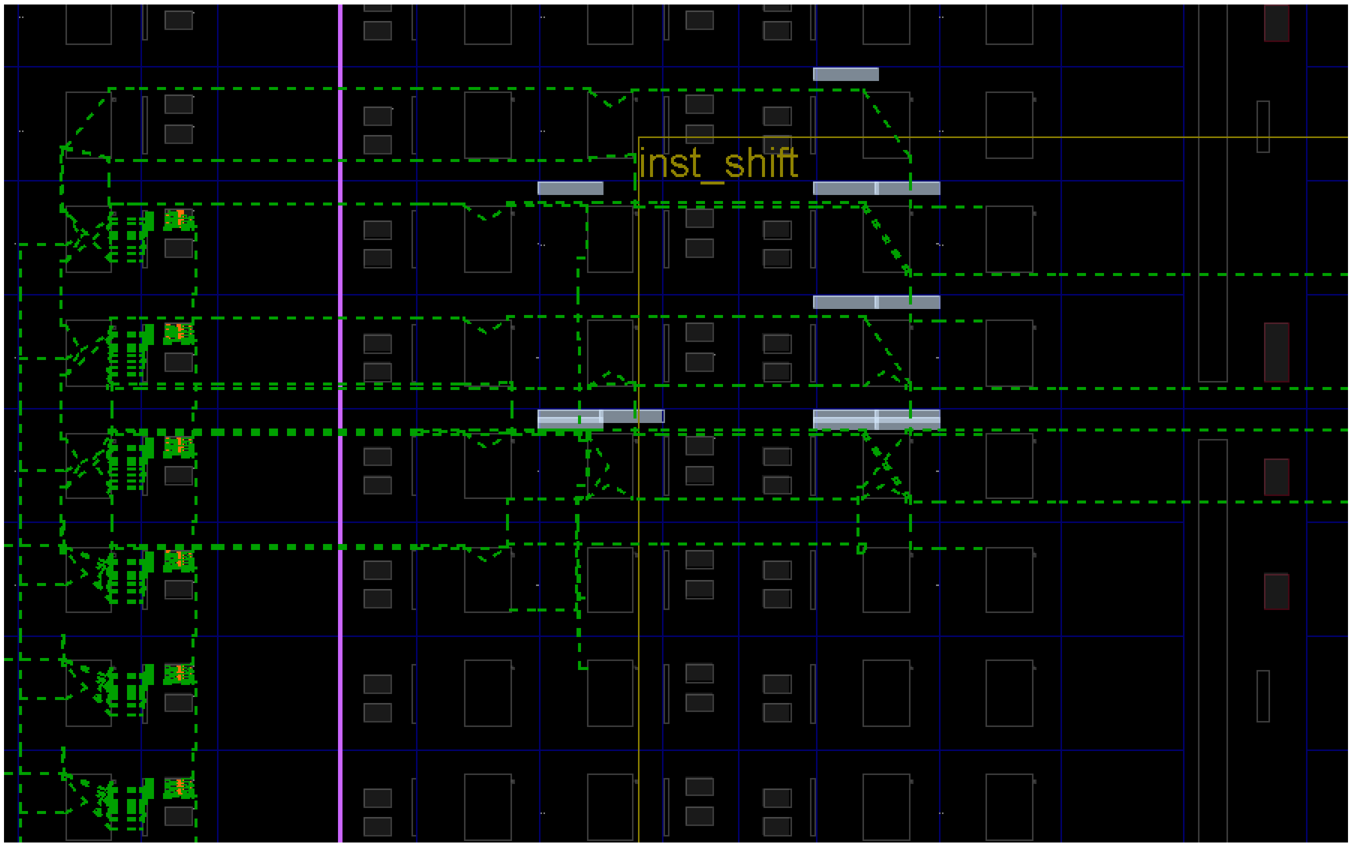


Figure 9: Close-up of Static-Only Design with Locked Routing

5. Issue the following command to write out the remaining static-only checkpoint:

```
write_checkpoint -force Checkpoint/static_route_design.dcp
```

This static-only checkpoint would be used for any future configurations. In this tutorial, however, you simply keep this design open in memory.

Step 7: Implement the Second Configuration

Now that the static design result is established and locked, and you can use it as context for implementing further Reconfigurable Modules.

Implement the Design

1. With the locked static design open in memory, read in post-synthesis checkpoints for the other two Reconfigurable Modules.

```
read_checkpoint -cell inst_shift Synth/shift_left/shift_synth.dcp
```

```
read_checkpoint -cell inst_count Synth/count_down/count_synth.dcp
```

2. Optimize, place and route the new RMs in the context of static by issuing these commands:

```
opt_design
```

```
place_design
```

```
route_design
```

The design is again fully implemented, now with the new Reconfigurable Module variants. The routing is a mix of dashed (locked) and solid (new) routing segments, as shown below.

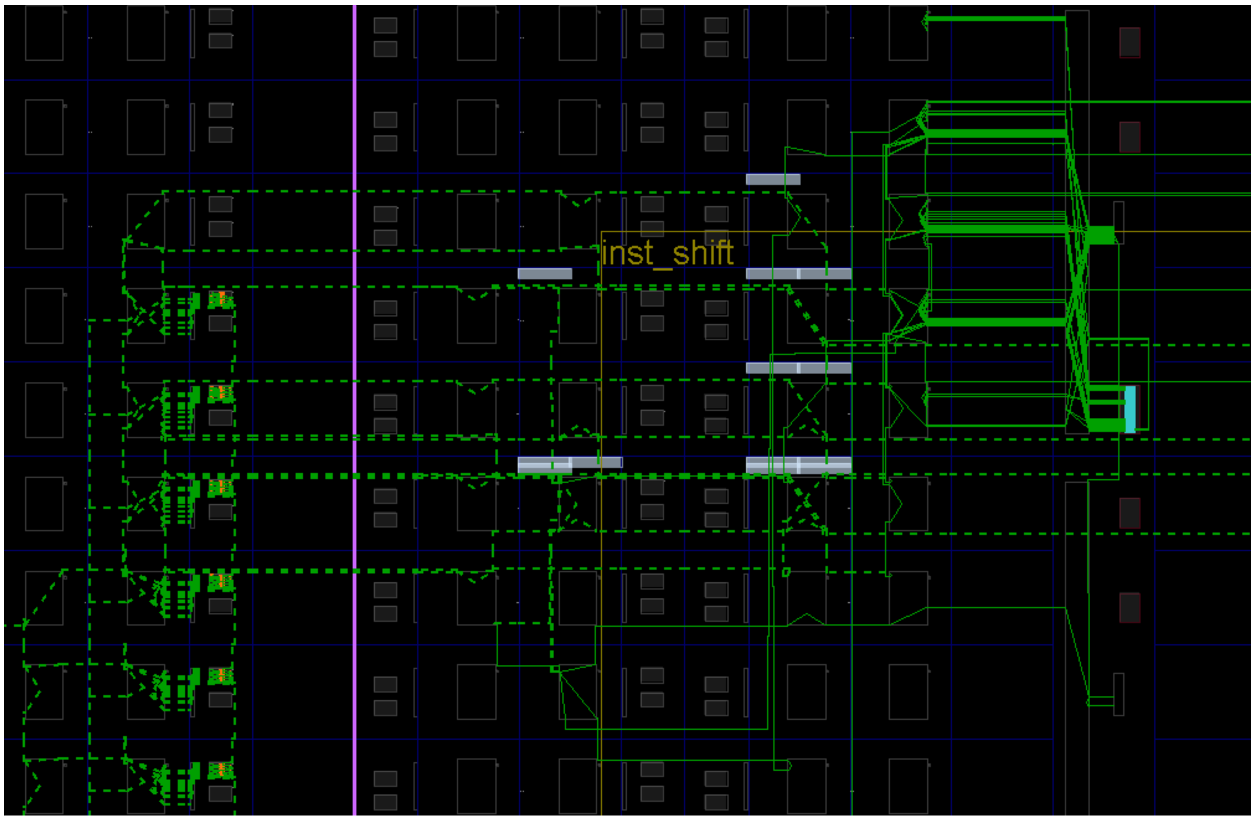


Figure 10: Second Configuration Routed, Showing Locked and New Routes

Save Results

1. Save the full design checkpoint and report files by issuing these commands:

```
write_checkpoint –force\
Implement/Config_shift_left_count_down_import/top_route_design.dcp

report_utilization –file\
Implement/Config_shift_left_count_down_import/top_utilization.rpt

report_timing_summary –file\
Implement/Config_shift_left_count_down_import/top_timing_summary.rpt
```

2. [Optional] Save checkpoints for each of the Reconfigurable Modules by issuing these two commands:

```
write_checkpoint -force –cell\
inst_shift Checkpoint/shift_left_route_design.dcp

write_checkpoint -force –cell\
inst_count Checkpoint/count_down_route_design.dcp
```

   At this point, you have implemented the static design and all Reconfigurable Module variants. This process would be repeated for designs that have more than two Reconfigurable Modules per Reconfigurable Partition.

Step 8: Examine Results

Use Highlighting Scripts

With the routed configuration open in the IDE, run some visualization scripts to highlight tiles and nets. These scripts identify the resources allocated for partial reconfiguration, and are automatically generated when the `hd.visual` parameter is enabled.

1. In the Tcl Console, issue the following commands from the <`Extract_Dir`> directory:

```
source hd_visual/pblock_inst_shift_AllTiles.tcl

highlight_objects -color blue [get_selected_objects]
```

2. Click somewhere in the Device view to deselect the frames (or enter `unselect_objects`), then issue the following commands:

```
source hd_visual/pblock_inst_count_AllTiles.tcl

highlight_objects -color yellow [get_selected_objects]
```

   The partition frames appear highlighted in the Device view, as shown in Figure 11.

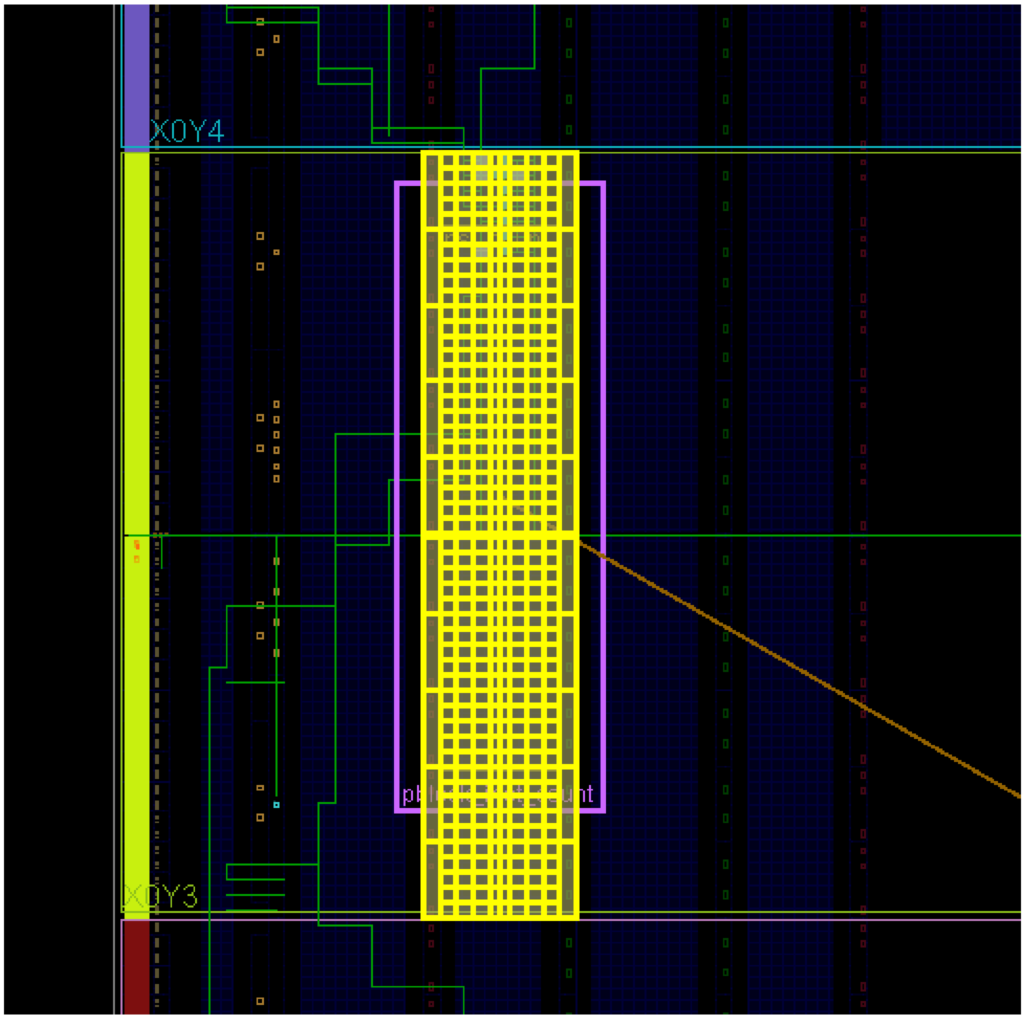


Figure 11: Reconfigurable Partition Frames Highlighted

These highlighted tiles represent the configuration frames that are sent to bitstream generation to create the partial bitstreams. As shown above, the `SNAPPING_MODE` feature has adjusted all four edges of `pblock_count` to account for `RESET_AFTER_RECONFIG` and legal reconfigurable partition widths.

The other "tile" scripts are variations on these. If you had not created Pblocks that vertically aligned to the clock region boundaries, the FrameTiles script would highlight the explicit Pblock tiles, while the AllTiles script extends those tiles to the full reconfigurable frame height. Note that these leave gaps where unselected frame types (for example: global clocks) exist.

The GlitchTiles script is a subset of frame sites, avoiding dedicated silicon resources; the other scripts are more informative than this one.

Finally, the Nets scripts are created for Tandem Configuration and do not apply to PR.

3. Close the current design:

```
close_project
```

Step 9: Generate Bitstreams

Verify Configurations

RECOMMENDED: *Before generating bitstreams, verify all configurations to ensure that the static portion of each configuration match identically, so the resulting bitstreams are safe to use in silicon. The PR Verify feature examines the complete static design up to and including the partition pins, confirming that they are identical. Placement and routing within the Reconfigurable Modules is not checked, as different module results are expected here.*

1. Run the `pr_verify` command from the Tcl Console:

```
pr_verify\
Implement/Config_shift_right_count_up_implement/top_route_design.dcp\
Implement/Config_shift_left_count_down_import/top_route_design.dcp
```

If successful, this command returns the following message.

```
INFO: [Vivado 12-3253] PR_VERIFY: check points
Implement/Config_shift_right_count_up/top_route_design.dcp and
Implement/Config_shift_left_count_down/top_route_design.dcp are compatible
```

By default, only the first mismatch (if any) is reported. To see all mismatches, use the `-full_check` option.

Generate Bitstreams

Now that the configurations have been verified, you can generate bitstreams and use them to target the KC705 demonstration board.

1. First, read the first configuration into memory:

```
open_checkpoint\
Implement/Config_shift_right_count_up_implement/top_route_design.dcp
```

2. Generate full and partial bitstreams for this design. Be sure to keep the bit files in a unique directory related to the full design checkpoint from which they were created.

```
write_bitstream –force -file Bitstreams/Config_RightUp.bit

close_project
```

Notice that three bitstreams have been created:

- `Config_RightUp.bit`
  This is the power-up, full design bitstream.
- `Config_RightUp_pblock_inst_shift_partial.bit`
  This is the partial bit file for the `shift_right` module.
- `Config_RightUp_pblock_inst_count_partial.bit`
  This is the partial bit file for the `count_up` module.

IMPORTANT: *The names of the bit files currently do not reflect the name of the Reconfigurable Module variant to clarify which image is loaded. The current solution uses the base name given by the* `-file` *option and appends the Pblock name of the reconfigurable cell. It is critical to provide enough description in the base name to be able to identify the reconfigurable bit files clearly. All partial bit files have the* `_partial` *postfix.*

3. Generate full and partial bitstreams for the second configuration, again keeping the resulting bit files in the appropriate folder.

```
open_checkpoint Implement/Config_shift_left_count_down_import/top_route_design.dcp
write_bitstream –force -file Bitstreams/Config_LeftDown.bit
close_project
```

   Similarly, you see three bitstreams created, this time with a different base name.

4. Generate a full bitstream with black boxes, plus blanking bitstreams for the Reconfigurable Modules. Blanking bitstreams can be used to "erase" an existing configuration to reduce power consumption.

```
open_checkpoint Checkpoint/static_route_design.dcp
update_design -cell inst_count -buffer_ports
update_design -cell inst_shift -buffer_ports
place_design
route_design
write_checkpoint –force Checkpoint/Config_black_box.dcp
write_bitstream –force -file Bitstreams/config_black_box.bit
close_project
```

   The base configuration bitstream has no logic for either reconfigurable partition. The `update_design` commands here insert constant drivers (ground) for all outputs of the Reconfigurable Partitions, so these outputs do not float. The `place_design` and `route_design` commands ensure they are completely implemented.

Step 10: Partially Reconfigure the FPGA

The `count_shift_led` design targets the KC705 demonstration board. The current design supports board revisions Rev 1.0 and Rev 1.1.

Configure the Device with a Full Image

1. Connect the KC705 to your computer via the Platform Cable USB and power on the board.
2. From the main Vivado IDE, select **Flow > Open Hardware Manager**.
3. Select **Open a new hardware target** on the green banner. Follow the steps in the wizard to establish communication with the board.
4. Select **Program device** on the green banner, and select the XC7K325T_0. Navigate to the `Bitstreams` folder to select `Config_RightUp.bit`, then click **OK** to program the device.

   You should now see the bank of GPIO LEDs performing two tasks. Four LEDs are performing a counting-up function (MSB is on the left), and the other four are shifting to the right. Note the amount of time it took to configure the full device.

Partially reconfigure the device

At this point, you can partially reconfigure the active device with any of the partial bitstreams that you have created.

1. Select **Program device** on the green banner again. Navigate to the `Bitstreams` folder to select `Config_LeftDown_pblock_inst_shift_partial.bit`, then click **OK** to program the device.

   The shift portion of the LEDs has changed direction, but the counter kept counting up, unaffected by the reconfiguration. Note the much shorter configuration time.

2. Select **Program device** on the green banner again. Navigate to the `Bitstreams` folder to select `Config_LeftDown_pblock_inst_count_partial.bit`, then click OK to program the device.

   The counter is now counting down, and the shifting LEDs were unaffected by the reconfiguration. This process can be repeated with the `Config_RightUp` partial bit files to return to the original configuration, or with the blanking partial bit files to stop activity on the LEDs (that will stay on).

Conclusion

In this tutorial, you:

- Synthesized a design bottom-up to prepare for partial reconfiguration implementation
- Created a valid floorplan for a partial reconfiguration design
- Created two configurations with common static results
- Implemented these two configurations, saving the static design to be used in each
- Created checkpoints for static and reconfigurable modules for later reuse
- Examined framesets and verified the two configurations
- Created full and partial bitstreams
- Configured and partially reconfigured an FPGA

Step 1: Extract the Tutorial Design Files

1. To obtain the tutorial design file, see the Tutorial Design Description.
2. Navigate to `\led_shift_count_us` in the extracted files. The `led_shift_count_us` data directory is referred to in this tutorial as the `<Extract_Dir>`.

Step 2: Examine the Scripts

Start by reviewing the scripts provided in the design archive. The files `design.tcl` and `design_complete.tcl` are located at the root level. Both files contain the same information, but `design.tcl` has parameters set such that only synthesis runs, while `design_complete.tcl` runs the entire flow for two configurations.

The Main Script

In the `<Extract_Dir>`, open `design.tcl` in a text editor. This is the master script where you define the design parameters, design sources, and design structure. This is the only file you have to modify to compile a complete Partial Reconfiguration design. Find more details regarding `design.tcl` and the underlying scripts in the `README.txt` located in the `Tcl` subdirectory.

Note the following details in this file:

- Visualization scripts are requested using the `set_param hd.visual 1` (on line 5, the `set_param` command is called). This command creates scripts in the root directory that you use later in the tutorial to identify the frames to be included in the partial bitstreams.
- Under **flow control**, you can control what phases of synthesis and implementation are run. In the tutorial, only synthesis is run by the script; implementation, verification, and bitstream generation are run interactively. To run these additional steps via the script, set the flow variables (e.g., `run.prImpl`) to **1**.
- The **Output Directories** and **Input Directories** set the file structure expected for design sources and results files. You must reflect any changes to your file structure here.

- The **Top Definition** and **RP Module Definitions** sections let you reference all source files for each part of your design. **Top Definition** covers all sources needed for the static design, including constraints and IP. The **RP Module Definitions** section does the same for Reconfigurable Partitions (RP). Complete a section for each RP and list all Reconfigurable Module (RM) variants for each RP.
  - This design has two Reconfigurable Partitions (`inst_shift` and `inst_count`), and each RP has two module variants.
- The **Configuration Definition** sections define the sets of static and reconfigurable modules that make up a configuration.
  - This design has two configurations defined within the master script: `config_shift_right_count_up_implement` and `config_shift_left_count_down_import`.
  - You can create more configurations by adding RMs or by combining existing RMs.

The Supporting Scripts

Underneath the `Tcl` subdirectory, several supporting Tcl scripts exist. The scripts are called by `design.tcl`, and they manage specific details for the Partial Reconfiguration flow. Provided below are some details about a few of the key PR scripts.

CAUTION! *Do not modify the supporting Tcl scripts.*

- `step.tcl`
  Manages the current status of the design by monitoring checkpoints.
- `synth.tcl`
  Manages all the details regarding the synthesis phase.
- `impl.tcl`
  Manages all the details regarding the module implementation phase.
- `pr_impl.tcl`
  Manages all the details regarding the top-level implementation of a PR design.
- `run.tcl`
  Launches the actual runs for synthesis and implementation.
- `log.tcl`
  Handles report file creation at key points during the flow.

Remaining scripts provide details within these scripts (such as the `*_utils.tcl` scripts) or manage other Hierarchical Design flows (such as `ooc_impl.tcl`).

Step 3: Synthesize the Design

The `design.tcl` script automates the synthesis phase of this tutorial. Five iterations of synthesis are called, one for the static top-level design and one for each of four Reconfigurable Modules.

1. Open the Vivado Tcl shell:
   - On Windows, select the Xilinx Vivado desktop icon or **Start > All Programs > Xilinx Design Tools> Vivado 2016.x > Vivado 2016.x Tcl Shell**.
   - On Linux, type: `vivado -mode tcl`.
2. In the shell, navigate to the `<Extract_Dir>` directory.
3. Run the `design.tcl` script by entering:

```
source design.tcl -notrace
```

After all five passes through Vivado Synthesis have completed, the Vivado Tcl shell is left open. You can find log and report files for each module, alongside the final checkpoints, under each named folder in the `Synth` subdirectory.

TIP: *In the `<Extract_Dir>` directory, multiple log files have been created:*

- `run.log` *shows the summary as posted in the Tcl shell window*
- `command.log` *echoes all the individual steps run by the script*
- `critical.log` reports all critical warnings produced during the run.

Step 4: Assemble the Design

Now that the synthesized checkpoints for each module, plus top, are available, you can assemble the design. Because project support for Partial Reconfiguration flows is not yet in place, you do not use the project infrastructure from within the IDE.

You will run all flow steps from the Tcl Console, but you can use features within the IDE (such as the floorplanning tool) for interactive events.

TIP: *Copy and paste commands directly from this document to avoid redundant effort and typos in the Vivado IDE. Copy and paste only one full command at a time. Note that some commands are long and therefore span multiple lines.*

Implement the Design

1. Open the Vivado IDE. You can open the IDE from the open Tcl shell by typing `start_gui` or by launching Vivado with the command `vivado -mode gui`.
2. Navigate to the `<Extract_Dir>` directory if you are not already there. The pwd command can confirm this.
3. Load the static design by issuing the following command in the Tcl Console:

   ```
   open_checkpoint Synth/Static/top_synth.dcp
   ```

   You can see the design structure in the Netlist pane, but black boxes exist for the `inst_shift` and `inst_count` modules. Note that the Flow Navigator pane is not present. You are working in non-project mode.

   TIP: *Place the IDE in floorplanning mode by selecting* ***Layout*** *>* ***Floorplanning****. Make sure the Device view is visible..*

   Two critical warnings are issued regarding unmatched instances. These instances are the Reconfigurable Modules that have yet to be loaded, and you can therefore ignore these warnings safely.

4. Load the synthesized checkpoints for first Reconfigurable Module variants for each of reconfigurable partitions:

   ```
   read_checkpoint -cell inst_shift Synth/shift_right/shift_synth.dcp
   read_checkpoint -cell inst_count Synth/count_up/count_synth.dcp
   ```

   Note*: The `inst_shift` and `inst_count` modules have been filled in with logical resources. You can now traverse the entire hierarchy within the Netlist pane.*

5. Define each of these submodules as partially reconfigurable by setting the `HD.RECONFIGURABLE` property:

   ```
   set_property HD.RECONFIGURABLE 1 [get_cells inst_shift]
   set_property HD.RECONFIGURABLE 1 [get_cells inst_count]
   ```

6. Save the assembled design state for this initial configuration:

   ```
   write_checkpoint ./Checkpoint/top_link_right_up.dcp
   ```

Step 5: Build the Design Floorplan

Next, you must create a floorplan to define the regions that will be partially reconfigured.

1. Select the `inst_count` instance in the Netlist pane. Right click and select **Floorplanning > Draw Pblock** and draw a tall narrow box on the left side of the X0Y4 clock region, which is the upper left corner of the device. The exact size and shape do not matter at this point, but keep the box within the clock region.

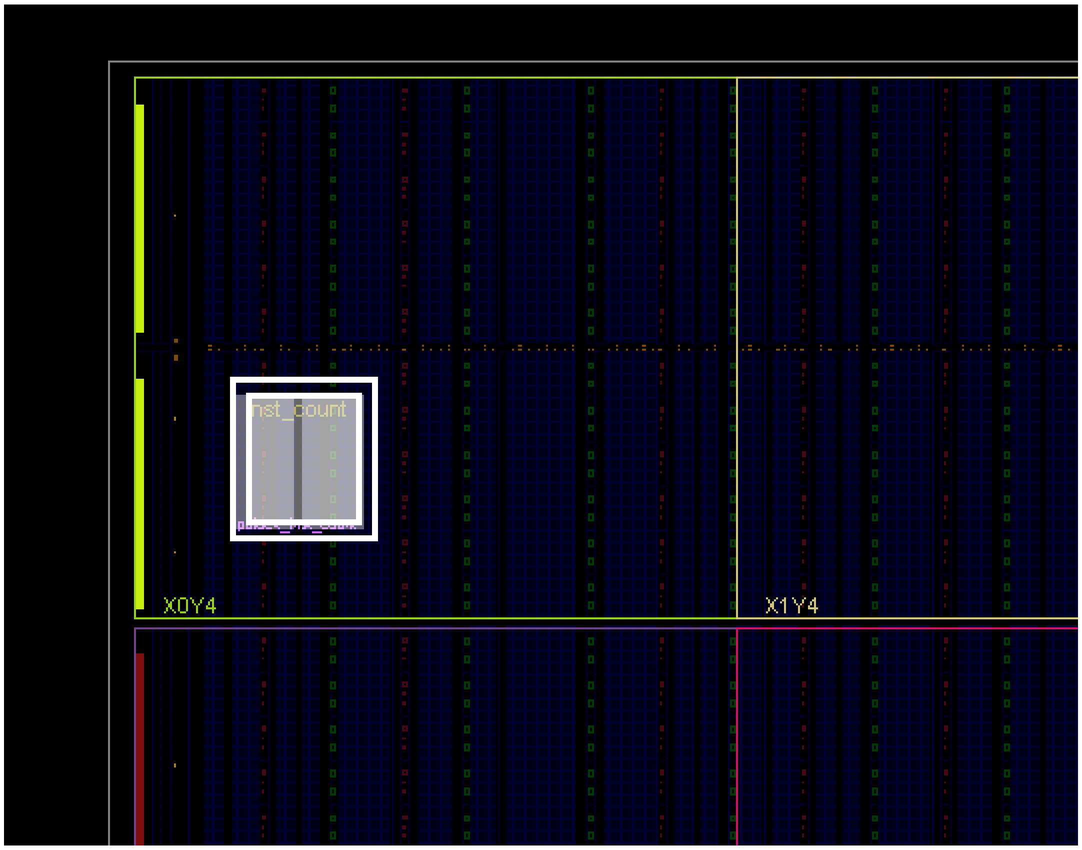


Figure 12: Pblock for the inst_count Reconfigurable Partition

Although this Reconfigurable Module only requires CLB resources, also include RAMB16, RAMB32, or DSP48 resources if the box encompasses those types. This allows the routing resources for these block types to be included in the reconfigurable region. The **General** tab of the Pblock Properties pane can be used to add these if needed. The **Statistics** tab shows the resource requirements of the currently loaded Reconfigurable Module.

2. Repeat the previous step for the `inst_shift` instance, this time targeting clock region X0Y3. This Reconfigurable Module includes block RAM instances, so the resource type must be included. If omitted, the RAMB details in the **Statistics** tab will be shown in red.

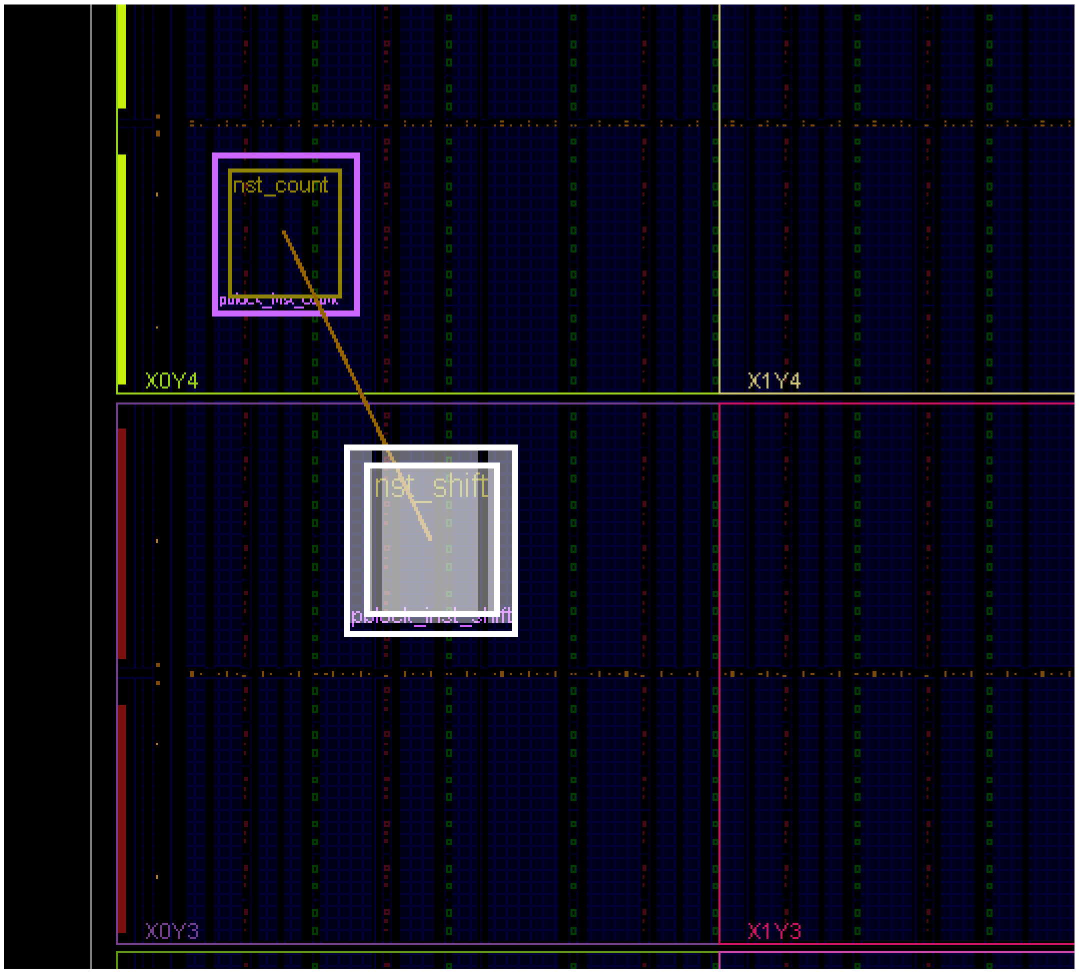
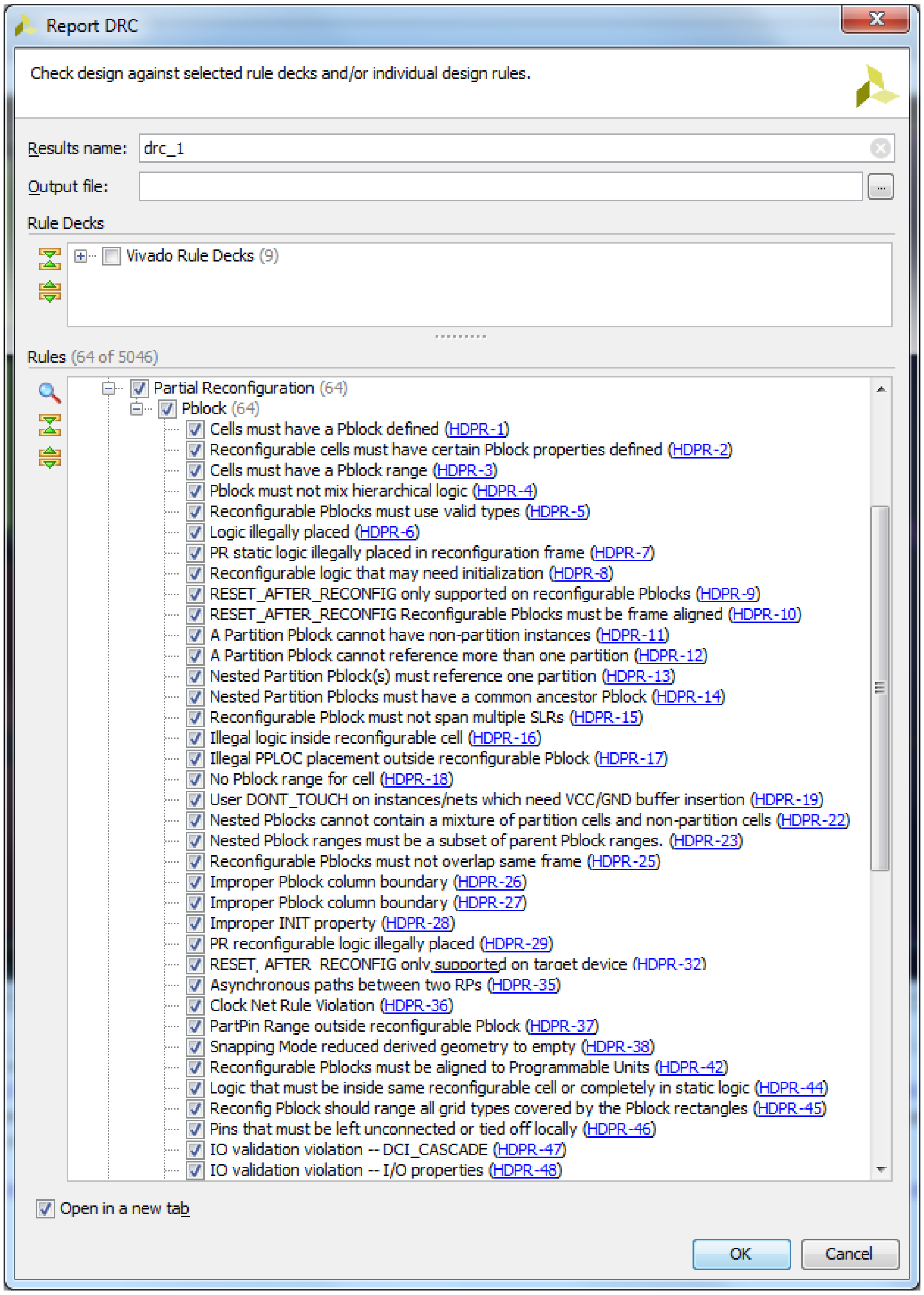


Figure 13: Pblock for the inst_shift Reconfigurable Partition

3. Run Partial Reconfiguration Design Rule Checks by selecting **Tools > Report > Report DRC**. You can uncheck **All Rules** and then check **Partial Reconfiguration** to focus this report strictly on PR DRCs.



Report DRC

Check design against selected rule decks and/or individual design rules.

Results name: drc_1

Output file:

Rule Decks

- ☐ Vivado Rule Decks (9)

Rules (64 of 5046)

- ☑ Partial Reconfiguration (64)
  - ☑ Pblock (64)
    - ☑ Cells must have a Pblock defined (HDPR-1)
    - ☑ Reconfigurable cells must have certain Pblock properties defined (HDPR-2)
    - ☑ Cells must have a Pblock range (HDPR-3)
    - ☑ Pblock must not mix hierarchical logic (HDPR-4)
    - ☑ Reconfigurable Pblocks must use valid types (HDPR-5)
    - ☑ Logic illegally placed (HDPR-6)
    - ☑ PR static logic illegally placed in reconfiguration frame (HDPR-7)
    - ☑ Reconfigurable logic that may need initialization (HDPR-8)
    - ☑ RESET_AFTER_RECONFIG only supported on reconfigurable Pblocks (HDPR-9)
    - ☑ RESET_AFTER_RECONFIG Reconfigurable Pblocks must be frame aligned (HDPR-10)
    - ☑ A Partition Pblock cannot have non-partition instances (HDPR-11)
    - ☑ A Partition Pblock cannot reference more than one partition (HDPR-12)
    - ☑ Nested Partition Pblock(s) must reference one partition (HDPR-13)
    - ☑ Nested Partition Pblocks must have a common ancestor Pblock (HDPR-14)
    - ☑ Reconfigurable Pblock must not span multiple SLRs (HDPR-15)
    - ☑ Illegal logic inside reconfigurable cell (HDPR-16)
    - ☑ Illegal PPLOC placement outside reconfigurable Pblock (HDPR-17)
    - ☑ No Pblock range for cell (HDPR-18)
    - ☑ User DONT_TOUCH on instances/nets which need VCC/GND buffer insertion (HDPR-19)
    - ☑ Nested Pblocks cannot contain a mixture of partition cells and non-partition cells (HDPR-22)
    - ☑ Nested Pblock ranges must be a subset of parent Pblock ranges. (HDPR-23)
    - ☑ Reconfigurable Pblocks must not overlap same frame (HDPR-25)
    - ☑ Improper Pblock column boundary (HDPR-26)
    - ☑ Improper Pblock column boundary (HDPR-27)
    - ☑ Improper INIT property (HDPR-28)
    - ☑ PR reconfigurable logic illegally placed (HDPR-29)
    - ☑ RESET_AFTER_RECONFIG only supported on target device (HDPR-32)
    - ☑ Asynchronous paths between two RPs (HDPR-35)
    - ☑ Clock Net Rule Violation (HDPR-36)
    - ☑ PartPin Range outside reconfigurable Pblock (HDPR-37)
    - ☑ Snapping Mode reduced derived geometry to empty (HDPR-38)
    - ☑ Reconfigurable Pblocks must be aligned to Programmable Units (HDPR-42)
    - ☑ Logic that must be inside same reconfigurable cell or completely in static logic (HDPR-44)
    - ☑ Reconfig Pblock should range all grid types covered by the Pblock rectangles (HDPR-45)
    - ☑ Pins that must be left unconnected or tied off locally (HDPR-46)
    - ☑ IO validation violation -- DCI_CASCADE (HDPR-47)
    - ☑ IO validation violation -- I/O properties (HDPR-48)

☑ Open in a new tab

Figure 14: Partial Reconfiguration Design Rule Checks (DRCs)

No DRCs should be reported, as long as the `inst_shift pblock` includes RAMB18 and RAMB36 resources. Note that for both pblocks, `SNAPPING_MODE` is set to `ON`, as reported in the **Properties** tab of the Pblock Properties pane. This is always enabled for all UltraScale devices given the fine granularity of programmable units in this architecture.

4. Save these Pblocks and associated properties by issuing this command in the Tcl Console:

   ```
   write_xdc ./Sources/xdc/fplan.xdc
   ```

   This exports all the current constraints in the design. These constraints can be managed in their own XDC file, merged with another XDC file (such as `top_io.xdc`), or managed within a run script (as is typically done with `HD.RECONFIGURABLE`).

Now that the floorplan has been established, the next step is implementing the design.

Step 6: Implement the First Configuration

In this step you will place and route the design and prepare the static portion of the design for reuse with new Reconfigurable Modules.

1. Load the top-level constraint file by issuing the command:

   ```
   read_xdc Sources/xdc/top_io.xdc
   ```

   This sets the device pinout and top-level timing constraints. This XDC file is not accessible from the IDE – it will not appear as a design source.

   This top-level XDC file should only contain constraints that reference objects in the static design. Constraints for logic or nets inside of the RP can be applied for specific Reconfigurable Modules if needed.

2. Optimize, place, and route the design by issuing the following commands:

   ```
   opt_design
   ```

   This is the point at which the Partial Reconfiguration license is checked. If you have a valid license, you see this message:

   ```
   Feature available: PartialReconfiguration
   ```

   If you have no license with the `PartialReconfiguration` feature, contact your local Xilinx sales office for more information. Evaluation licenses are available.

   ```
   place_design
   ```

   ```
   route_design
   ```

After both `place_design` and `route_design`, examine the state of the design in the Device view (see Figure 15). One thing to note after `place_design` is the introduction of Partition Pins. These are the physical interface points between static and reconfigurable logic. They are anchor points within an interconnect tile through which each I/O of the Reconfigurable Module must route. They appear as white boxes in the placed design view. For `pblock_shift`, they appear in the top of that Pblock, as the connections to static are just outside the Pblock in that area of the device. For `pblock_count`, they appear outside the user-defined region, as `SNAPPING_MODE` has vertically collected more frames to be added to the Reconfigurable Partition.

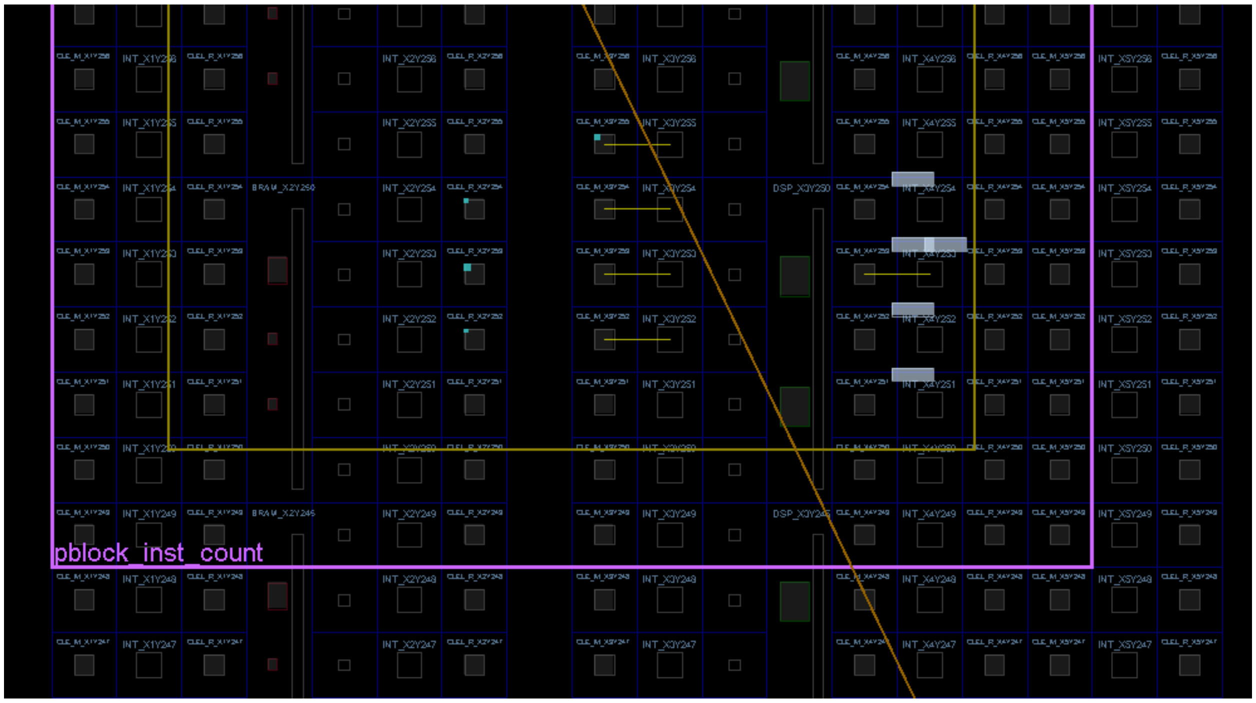


Figure 15: Partition Pins within Placed Design

3. To find these partition pins in the GUI easily:
    a. Select the Reconfigurable Module (for example, `inst_shift`) in the Netlist pane.
    b. Select the **Cell Pins** tab in the Cell Properties pane.
4. Select any pin to highlight it, or use **Ctrl+A** to select all. The Tcl equivalent of the latter is:

```
select_objects [get_pins inst_shift/*]
```

5. In the routed design view, click the **Show/Hide Nets** icon to display all routes by type (Fully Routed, Partially Routed, or Unrouted), as shown in Figure 16.

   Use the **Routing Resources** icon to toggle between abstracted and actual routing information, and to change the visibility of the routing resources themselves. All nets in the design are fully routed at this point.

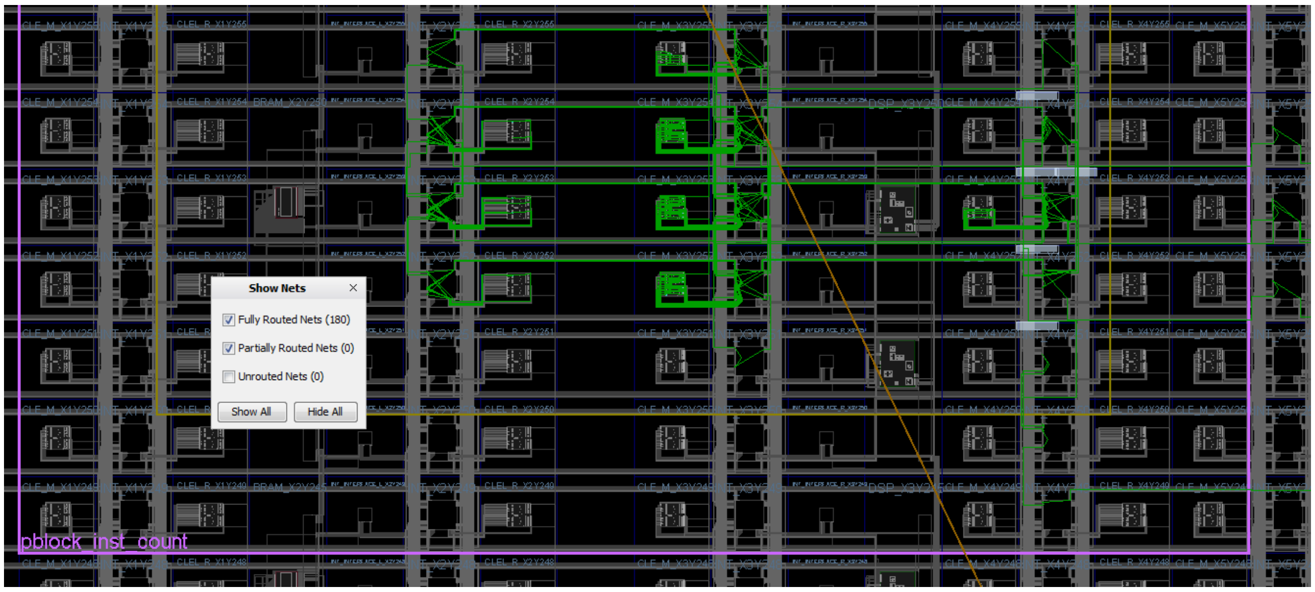


Figure 16: Close up of First Configuration Routed

Save the Results

1. Save the full design checkpoint and create report files by issuing these commands:

```
write_checkpoint -force\
Implement/Config_shift_right_count_up_implement/top_route_design.dcp

report_utilization -file\
Implement/Config_shift_right_count_up_implement/top_utilization.rpt

report_timing_summary -file\
Implement/Config_shift_right_count_up_implement/top_timing_summary.rpt
```

2. [Optional] Save checkpoints for each of the Reconfigurable Modules by issuing these two commands:

```
write_checkpoint -force\
-cell inst_shift Checkpoint/shift_right_route_design.dcp

write_checkpoint -force\
-cell inst_count Checkpoint/count_up_route_design.dcp
```

TIP: *When running `design_complete.tcl` to process the entire design in batch mode, design checkpoints, log files, and report files are created at each step of the flow.*

At this point, you have created a fully implemented partial reconfiguration design from which you can generate full and partial bitstreams. The static portion of this configuration is used for all subsequent configurations, and to isolate the static design, the current Reconfigurable Modules must be remove. Make sure routing resources are enabled, and zoom in to an interconnect tile with partition pins.

3. Clear out Reconfigurable Module logic by issuing the following commands:

```
update_design -cell inst_shift -black_box
update_design -cell inst_count -black_box
```

Issuing these commands results in many design changes as shown in Figure 17:

- The number of Fully Routed nets (green) has decreased.
- `inst_shift` and `inst_count` now appear in the Netlist view as empty.

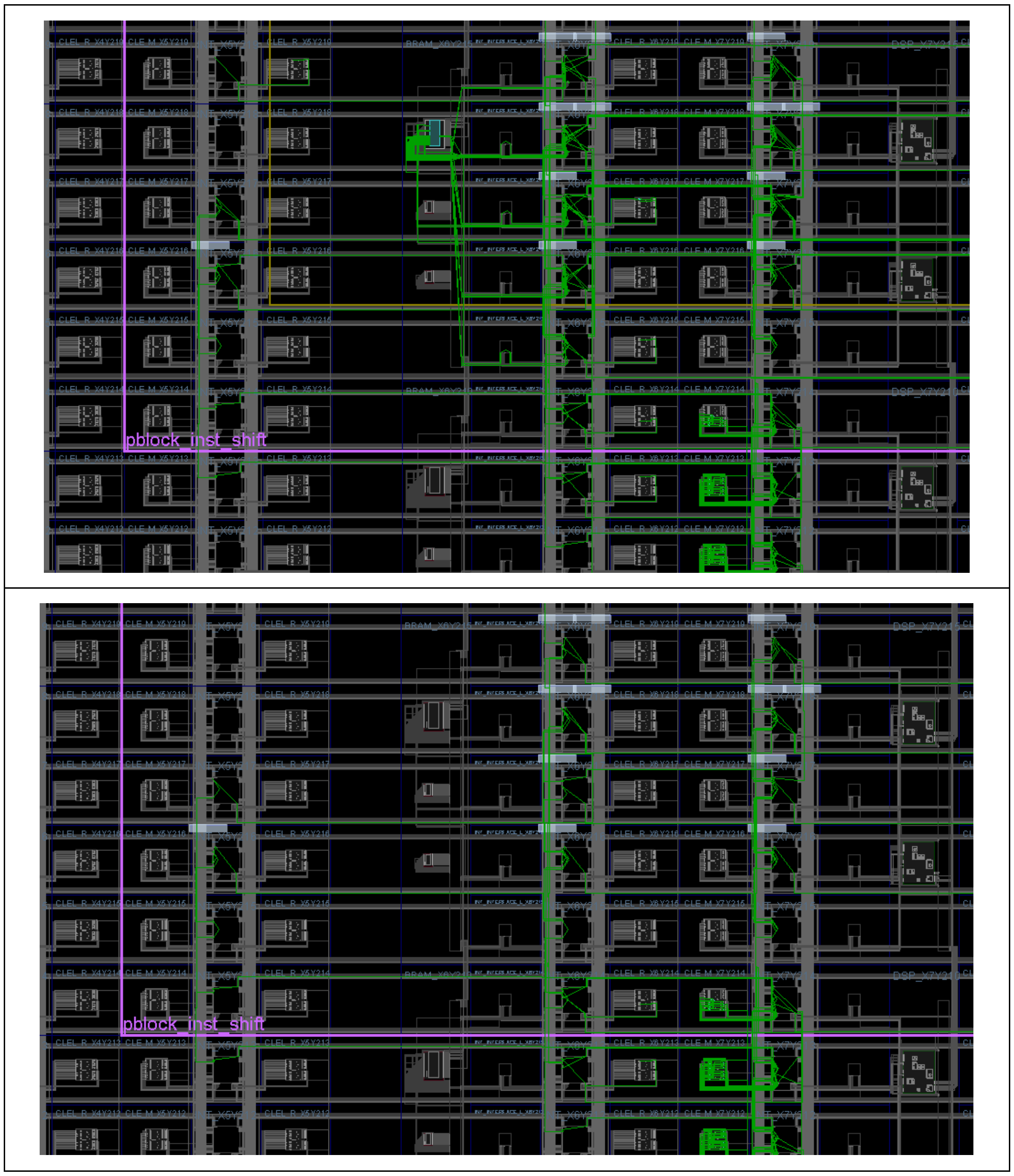


Figure 17: The inst_shift module before (top) and after (bottom) `update_design -black_box`

4. Issue the following command to lock down all placement and routing:

   ```
   lock_design -level routing
   ```

   Because no cell was identified in the `lock_design` command, the entire design in memory (currently consisting of the static design with black boxes) is affected. All routed nets now display as locked, as indicated by dashed lines in Figure 18. All placed components changed from blue to orange to show they are also locked.

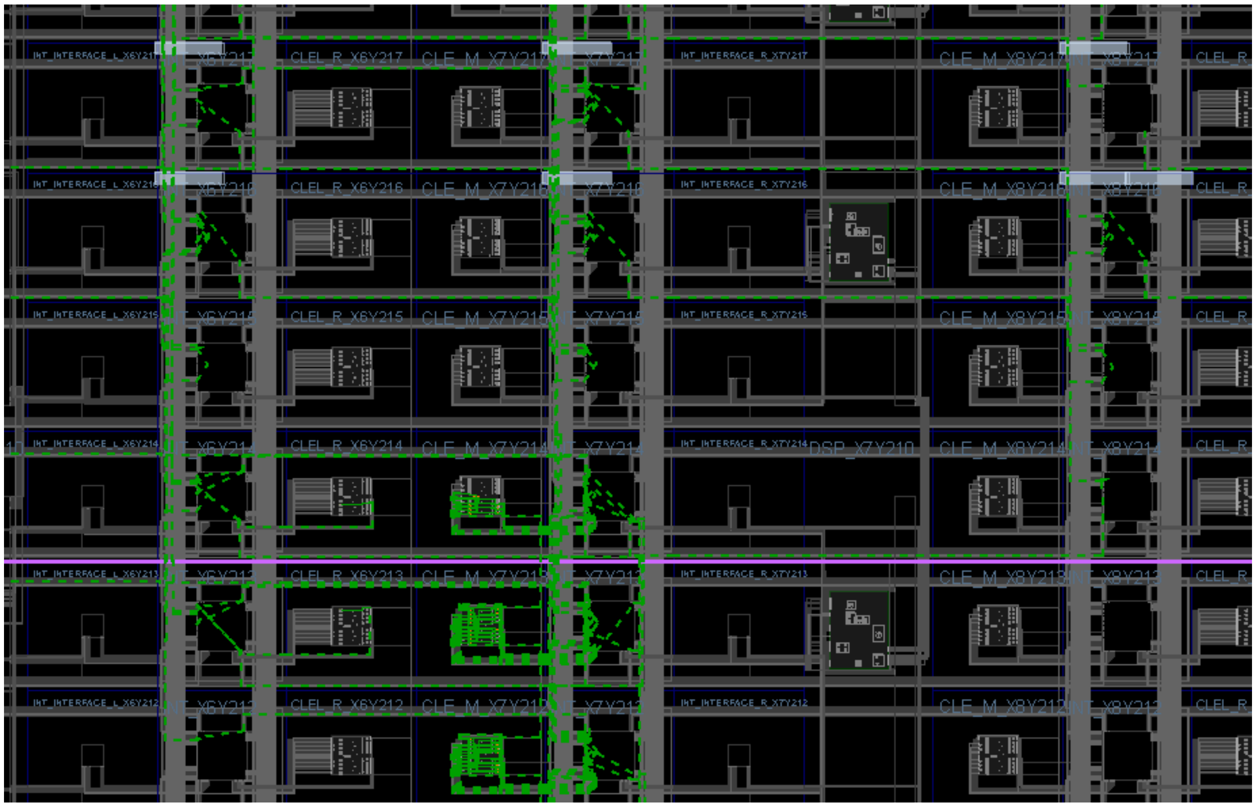


Figure 18: Close-up of Static-Only Design with Locked Routing

5. Issue the following command to write out the remaining static-only checkpoint:

```
write_checkpoint -force Checkpoint/static_route_design.dcp
```

   This static-only checkpoint would be used for any future configurations. In this tutorial, however, you simply keep this design open in memory.

Step 7: Implement the Second Configuration

The static design result is now established and locked, and you will use it as context for implementing further Reconfigurable Modules.

Implement the Design

1. With the locked static design open in memory, read in post-synthesis checkpoints for the other two Reconfigurable Modules.

```
read_checkpoint -cell inst_shift Synth/shift_left/shift_synth.dcp
```

```
read_checkpoint -cell inst_count Synth/count_down/count_synth.dcp
```

2. Optimize, place and route the new RMs in the context of static by issuing these commands:

```
opt_design
```

```
place_design
```

```
route_design
```

The design is again fully implemented, now with the new Reconfigurable Module variants. The routing is a mix of dashed (locked) and solid (new) routing segments, as shown below.

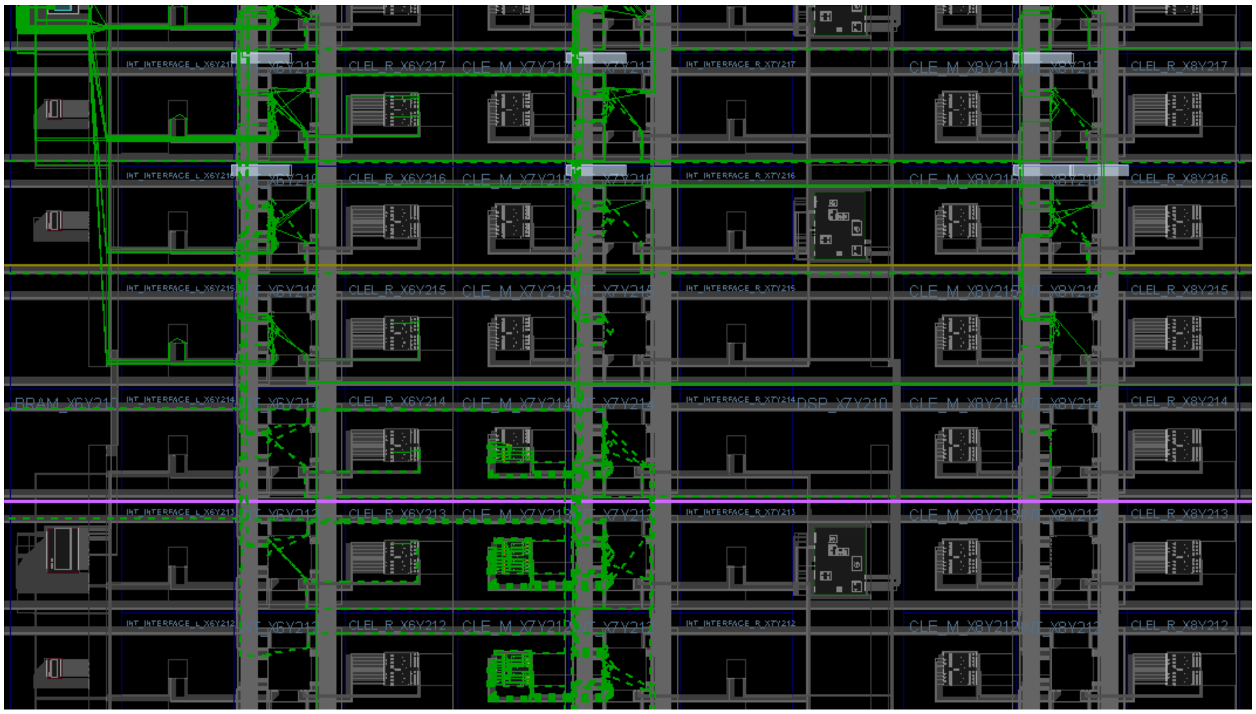


Figure 19: Second Configuration Routed, Showing Locked and New Routes

Save the Results

1. Save the full design checkpoint and report files by issuing these commands:

   ```
   write_checkpoint -force
   Implement/Config_shift_left_count_down_import/top_route_design.dcp

   report_utilization -file
   Implement/Config_shift_left_count_down_import/top_utilization.rpt

   report_timing_summary -file
   Implement/Config_shift_left_count_down_import/top_timing_summary.rpt
   ```

2. [Optional] Save checkpoints for each of the Reconfigurable Modules by issuing these two commands:

   ```
   write_checkpoint –force\
   -cell inst_shift Checkpoint/shift_left_route_design.dcp

   write_checkpoint –force\
   -cell inst_count Checkpoint/count_down_route_design.dcp
   ```

   At this point, you have implemented the static design and all Reconfigurable Module variants. This process would be repeated for designs that have more than two Reconfigurable Modules per Reconfigurable Partition.

Step 8: Examine Results

Use Highlighting Scripts

With the routed configuration open in the IDE, run some visualization scripts to highlight tiles and nets. These scripts identify the resources allocated for partial reconfiguration, and are automatically generated when the `hd.visual` parameter is enabled.

1. In the Tcl Console, issue the following commands from the <`Extract_Dir`> directory:

   ```
   source hd_visual/pblock_inst_shift_AllTiles.tcl

   highlight_objects -color blue [get_selected_objects]
   ```

2. Click somewhere in the Device view to deselect the frames (or enter `unselect_objects`), then issue the following commands:

   ```
   source hd_visual/pblock_inst_count_AllTiles.tcl

   highlight_objects -color yellow [get_selected_objects]
   ```

   The partition frames appear highlighted in the Device view, as shown in Figure 20.

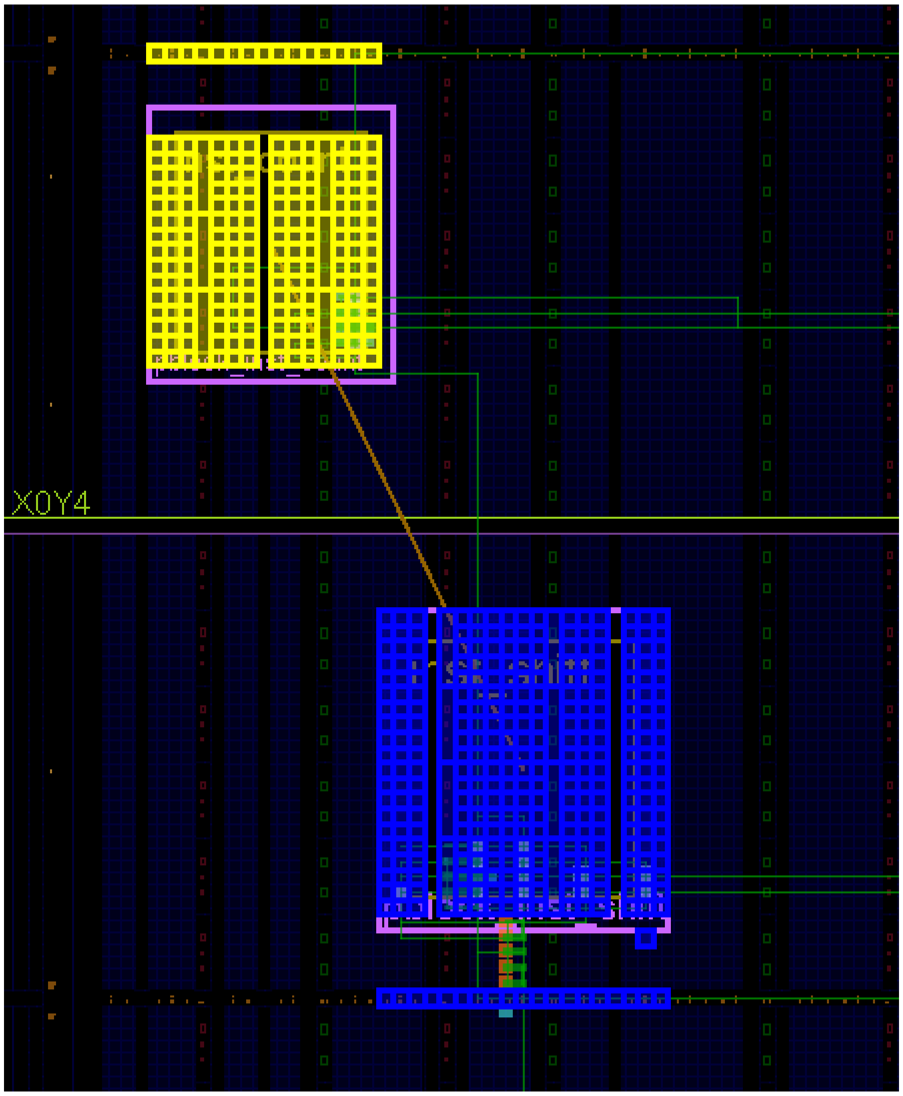


Figure 20: Reconfigurable Partition Frames Highlighted

These highlighted tiles represent the configuration frames that are sent to bitstream generation to create the partial bitstreams. As shown above, the `SNAPPING_MODE` feature has adjusted three of four edges of `pblock_count` and one edge of `pblock_shift` to account for alignment to programmable unit boundaries. This snapping behavior explains why it appears that static logic may appear to have been placed inside Reconfigurable Partitions, as seen in Figure 17, Figure 18, and Figure 19. In actuality, the effective boundary is one CLB row higher than the user-defined Pblock boundary indicates, so this static logic is placed correctly. This effective boundary can also be seen in the shading of the Pblock during creation, as shown in Figure 12 and Figure 13.

Also note that RCLK rows matching the width of the Pblocks are included. Global clocks driving logic in these Reconfigurable Partitions are connected to the spines running through these rows and are enabled or disabled during partial reconfiguration.

The other "tile" scripts are variations on these. If you had not created Pblocks that vertically aligned to the clock region boundaries, the FrameTiles script would highlight the explicit Pblock tiles, while the AllTiles script extends those tiles to the full reconfigurable frame height. Note that these leave gaps where unselected frame types (for example: global clocks) exist.

The GlitchTiles script is a subset of frame sites, avoiding dedicated silicon resources; the other scripts are more informative than this one.

Finally, the Nets scripts are created for Tandem Configuration and do not apply to PR.

3. Close the current design:

```
close_project
```

Step 9: Generate Bitstreams

Verify Configurations

RECOMMENDED: *Before generating bitstreams, verify all configurations to ensure that the static portion of each configuration match identically, so the resulting bitstreams are safe to use in silicon. The PR Verify feature examines the complete static design up to and including the partition pins, confirming that they are identical. Placement and routing within the Reconfigurable Modules is not checked, as different module results are expected here.*

1. Run the `pr_verify` command from the Tcl Console:

```
pr_verify\
Implement/Config_shift_right_count_up_implement/top_route_design.dcp\
Implement/Config_shift_left_count_down_import/top_route_design.dcp
```

If successful, this command returns the following message.

```
INFO: [Vivado 12-3253] PR_VERIFY: check points
Implement/Config_shift_right_count_up/top_route_design.dcp and
Implement/Config_shift_left_count_down/top_route_design.dcp are compatible
```

By default, only the first mismatch (if any) is reported. To see all mismatches, use the `-full_check` option.

Generate Bitstreams

Now that the configurations have been verified, you can generate bitstreams and use them to target the KCU105 demonstration board.

1. First, read the first configuration into memory:

```
open_checkpoint\
Implement/Config_shift_right_count_up_implement/top_route_design.dcp
```

2. Generate full and partial bitstreams for this design. Be sure to keep the bit files in a unique directory related to the full design checkpoint from which they were created.

```
write_bitstream –force -file Bitstreams/Config_RightUp.bit
close_project
```

Notice that five bitstreams have been created:

- `Config_RightUp.bit`
  This is the power-up, full design bitstream.
- `Config_RightUp_pblock_inst_shift_partial.bit`
  This is the partial bit file for the `shift_right` module.
- `Config_RightUp_pblock_inst_count_partial.bit`
  This is the partial bit file for the `count_up` module.
- `Config_RightUp_pblock_inst_shift_partial_clear.bit`
  This is the clearing bit file for the `shift_right` module.
- `Config_RightUp_pblock_inst_count_partial_clear.bit`
  This is the clearing bit file for the `count_up` module.

IMPORTANT: *The names of the bit files currently do not reflect the name of the Reconfigurable Module variant to clarify which image is loaded. The current solution uses the base name given by the* `-file` *option and appends the Pblock name of the reconfigurable cell. It is critical to provide enough description in the base name to be able to identify the reconfigurable bit files clearly. All partial bit files have the* `_partial` *postfix, and all clearing bit files have the* `_partial_clear` *postfix.*

3. Generate full and partial bitstreams for the second configuration, again keeping the resulting bit files in the appropriate folder.

```
open_checkpoint\
Implement/Config_shift_left_count_down_import/top_route_design.dcp

write_bitstream –force –file Bitstreams/Config_LeftDown.bit

close_project
```

Similarly, you see five bitstreams created, this time with a different base name.

4. Generate a full bitstream with black boxes, plus blanking bitstreams for the Reconfigurable Modules. Blanking bitstreams can be used to "erase" an existing configuration to reduce power consumption.

```
open_checkpoint Checkpoint/static_route_design.dcp

update_design -cell inst_count -buffer_ports

update_design -cell inst_shift -buffer_ports

place_design

route_design

write_checkpoint –force Checkpoint/Config_black_box.dcp

write_bitstream –force -file Bitstreams/config_black_box.bit

close_project
```

   The base configuration bitstream has no logic for either reconfigurable partition. The `update_design` commands here insert constant drivers (ground) for all outputs of the Reconfigurable Partitions, so these outputs do not float. The `place_design` and `route_design` commands ensure they are completely implemented. Note that as valid Reconfigurable Modules, these instances also have clearing bitstreams.

Step 10: Partially Reconfigure the FPGA

The `count_shift_led` design targets the KCU105 demonstration board. The current design supports board revisions Rev 1.0 and newer, with production silicon required.

Configure the Device with a Full Image

1. Connect the KCU105 to your computer using the Platform Cable USB and power on the board.
2. From the main Vivado IDE, select **Flow > Open Hardware Manager**.
3. Select **Open target** on the green banner. Follow the steps in the wizard to establish communication with the board.
4. Select **Program device** on the green banner and pick the **xcku040_0**.
5. Navigate to the Bitstreams folder to select `Config_RightUp.bit`, then click **OK** to program the device.

   You should now see the bank of GPIO LEDs performing two tasks. Four LEDs are performing a counting-up function (MSB is on the left), and the other four are shifting to the right. Note the amount of time it took to configure the full device.

Partially Reconfigure the Device

At this point, you can partially reconfigure the active device with any of the partial bitstreams that you have created, starting first with the appropriate clearing bitstream.

1. Select **Program device** on the green banner again. Navigate to the `Bitstreams` folder to select `Config_RightUp_pblock_inst_shift_partial_clear.bit`, then click **OK** to program the device.

   The shift portion of the LEDs has stopped, but the counter kept counting up, unaffected by the reconfiguration. Note the much shorter configuration time, as well as the fact that the DONE LED has turned off.

2. Select **Program device** on the green banner again. Navigate to the `Bitstreams` folder to select `Config_LeftDown_pblock_inst_shift_partial.bit`, then click **OK** to program the device.

   The shift portion of the LEDs has now restarted in the opposite direction, and the DONE LED is now back on.

3. Select **Program device** on the green banner again. Navigate to the `Bitstreams` folder to select `Config_RightUp_pblock_inst_count_partial_clear.bit`, then click **OK** to program the device.

   The count portion of the LEDs has stopped, but the shifter kept shifting, unaffected by the reconfiguration.

4. Select **Program device** on the green banner again. Navigate to the `Bitstreams` folder to select `Config_LeftDown_pblock_inst_count_partial.bit`, then click **OK** to program the device.

   The counter is now counting down, and the shifting LEDs were unaffected by the reconfiguration. This process can be repeated with the `Config_RightUp` partial bit files to return to the original configuration, or with the blanking partial bit files to stop activity on the LEDs (they will stay on). Keep track of the currently loaded module for each partition to ensure the correct clearing bitstream is loaded before the next partial bitstream.

Conclusion

In this tutorial, you:

- Synthesized a design bottom-up to prepare for partial reconfiguration implementation
- Created a valid floorplan for a partial reconfiguration design
- Created two configurations with common static results
- Implemented these two configurations, saving the static design to be used in each
- Created checkpoints for static and reconfigurable modules for later reuse
- Examined framesets and verified the two configurations
- Created full, partial, and clearing bitstreams
- Configured and partially reconfigured an FPGA

Lab 3: Partial Reconfiguration Controller IP

Overview

Partial Reconfiguration (PR) in Xilinx® FPGAs and SoCs introduces new design requirements compared to traditional solutions. One such requirement is the management of partial bitstreams. Designers must plan for when partial bitstreams are required, where these bitstreams are stored, how these bitstreams are delivered to the configuration engine, and how the static design must behave before, during and after the delivery of a new partial bitstream. The Partial Reconfiguration Controller (PRC) IP has been designed to help users solve these challenges.

This lab has been designed to show the fundamental details and capabilities of the Partial Reconfiguration Controller IP in the Vivado® Design Suite. The design used is based on the tutorial documented in Lab 1: 7 Series Basic Flow. It takes the basic LED shift count design and adds the PR Controller IP, an AXI embedded memory controller to access the BPI flash, and instruments the design with debug cores (not used in this tutorial) in the Vivado Design Suite.

Additional Resources

For additional information, please refer to these documents:

- *Partial Reconfiguration Controller Project Guide* (PG193)
- *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909)
- DocNav includes a Partial Reconfiguration Design Hub that links these documents and other PR-specific resources. It is also available through the Xilinx support site.

Be sure to always use the latest version of these documents and the corresponding Vivado Design Suite software.

Tutorial Design Files

1. To obtain the tutorial design file, see Tutorial Design Description.
2. Navigate to `\prc_k7` in the extracted files. The `prc_k7` data directory is referred to throughout the tutorial as the `<Extract_Dir>`.

Section 1: Customizing the Partial Reconfiguration Controller IP

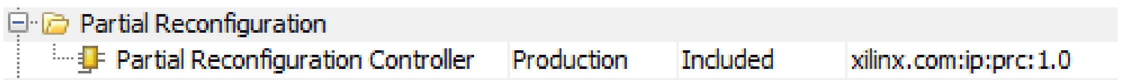
This section highlights the key features and settings that must be understood when creating a PR Controller module for your target application. It shows both the customization of the IP within the GUI, as well as post-route customization using the Tcl API.

Background

The Partial Reconfiguration Controller IP requires a few details to be entered during the customization process. Identify all information regarding each Reconfigurable Partition (RP) and Reconfigurable Module (RM) here to create a fully populated controller that understands the entirety of the reconfiguration needs of the target FPGA. Within this IP, reconfigurable portions of the design are referred to as Virtual Sockets, which encompass both the Reconfigurable Partition along with all associated static logic used to manage the RP, such as decoupling or handshaking logic. While the core parameters are customizable during operation, the more that can be entered at this stage the better, as the front-end design description will more accurately match the final implemented design.

Tutorial Instructions

1. Open the Vivado IDE and click the **Manage IP** task, and select **New IP Location**. Enter the following details before clicking **Finish**:
   - Part: Click on **Boards** to select the **KC705**
   - IP Location: `<Extract_Dir>/Sources/lab`
2. In the **IP Catalog**, expand the Partial Reconfiguration category to double-click on the Partial Reconfiguration Controller IP.



Partial Reconfiguration
Partial Reconfiguration Controller | Production | Included | xilinx.com:ip:prc:1.0

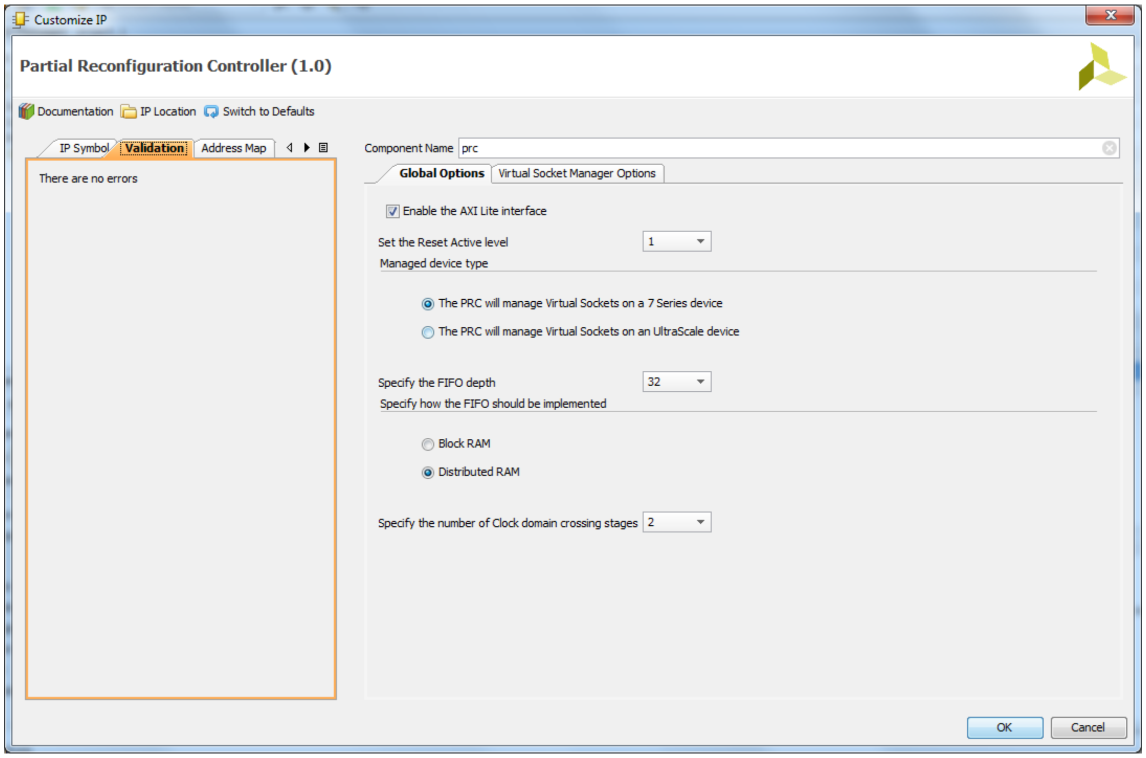
Figure 21: The Partial Reconfiguration Controller as shown in the IP Catalog

The PR Controller IP GUI has four tabs on the left side, providing feedback on the current configuration of the IP. The default pane is labeled Validation, and shows any errors that might arise as the core parameters are entered. The core does not compile if errors exist.

There are two tabs on the right side of the GUI where all customization is done. Most of the information is entered on the **Virtual Socket Manager Options** tab.

3. Change the component name from **prc_0** to **prc**, to match the component name in the supplied VHDL top level.
4. On the Global Options tab, make two changes:
   a. Set the Polarity of reset and `icap_reset` = **1**
   b. Set the number of Clock domain crossing stages = **2**

   The PRC GUI should now look like this:



There are no errors

Component Name: prc

[✓] Enable the AXI Lite interface

Set the Reset Active level: 1

Managed device type
- [✓ The PRC will manage Virtual Sockets on a 7 Series device]
- The PRC will manage Virtual Sockets on an UltraScale device

Specify the FIFO depth: 32

Specify how the FIFO should be implemented
- Block RAM
- [✓ Distributed RAM]

Specify the number of Clock domain crossing stages: 2

Figure 22: Component Name and Global Options completed

Note that this PR Controller can manage Virtual Sockets on either 7 series or UltraScale devices. This IP is not limited to managing PR on the same device on which it resides. It can connect to an ICAP on another device to manage its reconfiguration.

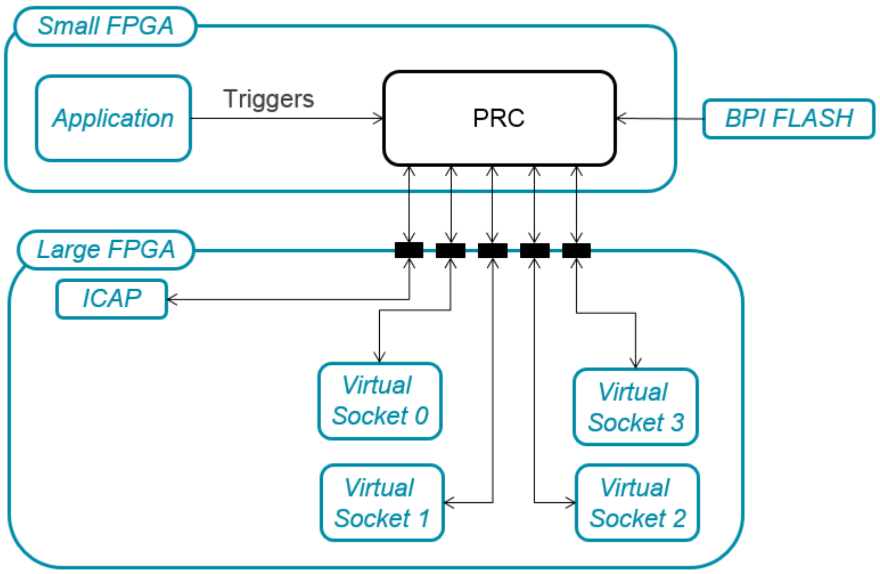


Figure 23: Example of a multi-chip solution using the PRC

5. Next, switch to the Virtual Socket Manager Options tab to define information about the Virtual Sockets and their Reconfigurable Modules.

   The PRC IP is preloaded with one Virtual Socket with one Reconfigurable Module to get you started.

   First, define the Virtual Socket Manager (VSM) for the Shift functionality.

6. Rename the current VSM from **VS_0** to **vs_shift**.
7. Rename the current RM from **RM_0** to **rm_shift_left**.

CAUTION!

- Underscores are not visible in the Virtual Socket Manager and Reconfigurable Module pull-down dialogs. The **Name** (ID) label below the pull-down shows this more accurately.
- To accept a new value in any field in the GUI, simply click in any other field in the GUI or press the Tab key. Do NOT press Enter, as this will trigger compilation of the IP.

8. Click the **New Reconfigurable Module** button to create a new RM for this VSM. Name it **rm_shift_right**.

TIP: *Up to 128 Reconfigurable Modules can be managed by a single Virtual Socket Manger.*

9. Configure the vs_shift VSM to have the following properties:

   - Has Status Channel = **checked**
   - Has PoR RM = **rm_shift_right**
   - Number of RMs allocated = **4**

   The PoR RM indicates which RM is contained within the initial full-design configuration file, so the VSM knows which triggers and events are appropriate upon startup of the FPGA. The VSM tracks the current active Reconfigurable Module in its socket.

   Even through you have only defined two RMs for this Virtual Socket, you have set aside space for four in total. This allows for expansion later on. Additional Reconfigurable Modules can be identified using the AXI4-Lite interface, but only if spaces have been reserved for them.

10. For each of these RMs, enter the following values. Use the **Reconfigurable Module to configure** pull-down to switch between the two RMs.

    - For `rm_shift_left`:
      - Reset type = **Active High**
      - Duration of Reset = **3**
    - For `rm_shift_right`:
      - Reset type = **Active High**
      - Duration of Reset = **10**

      Note: *The different reset durations are given to show that these can be independently assigned, as each RM may have different requirements.*

11. For each RM, assign a bitstream size and location to identify where it will reside in the BPI flash device.

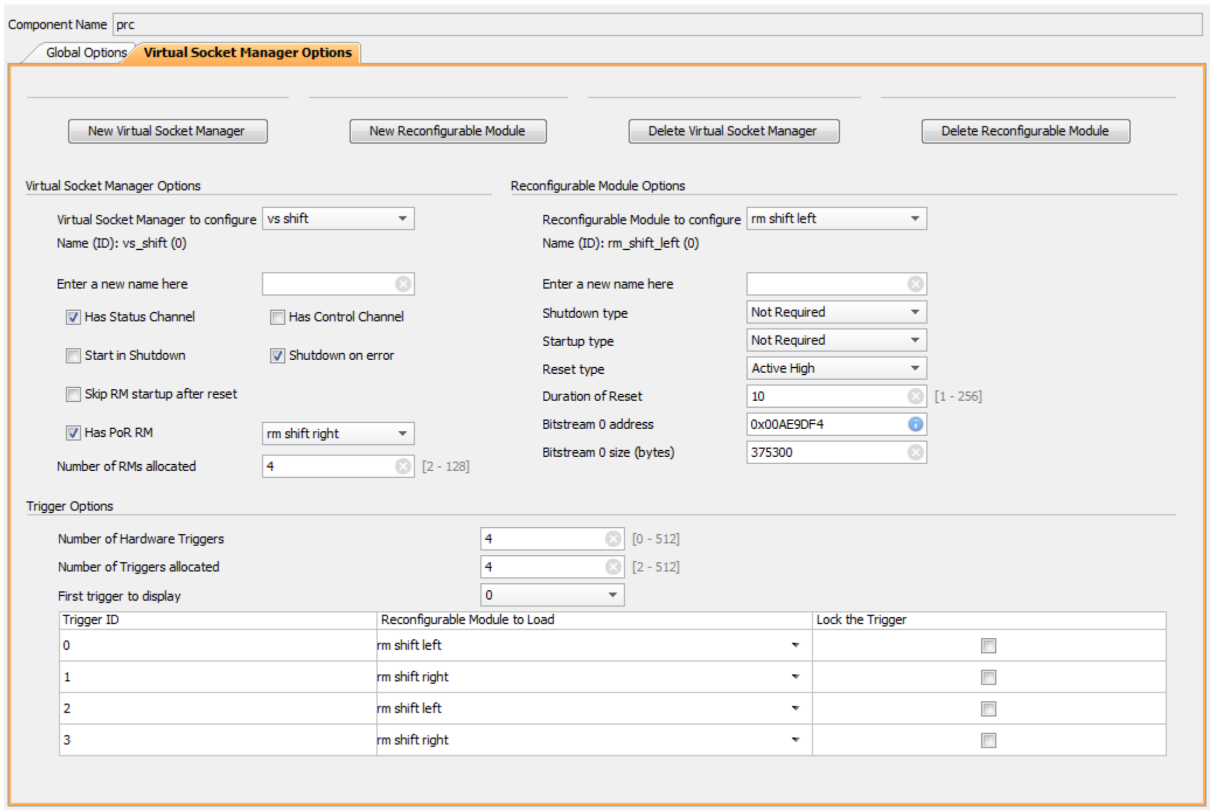
    - For `rm_shift_left`:
      - Bitstream 0 address = **0x00AEA000**
      - Bitstream 0 size (bytes) = **482828**
    - For `rm_shift_right`:
      - Bitstream 0 address = **0x00B60000**
      - Bitstream 0 size (bytes) = **482828**

This information is typically not known early in design cycles, as bitstream size is based on the size and composition of the Reconfigurable Partition Pblock, and bitstream address is based on storage details. Until the design is to be tested on silicon, these can be set to `0`. As the design settles and hardware testing with the PRC is set to begin, this information can be added. The bitstream address information must match the information passed during PROM file generation. Certain bitstream generation options, most notably bitstream compression, can lead to variations in the final bitstream size for different configurations, even for the same Reconfigurable Partition.

12. Define the Trigger Options for the Shift functionality:
    - Number of Hardware Triggers = **4**
    - Number of Triggers allocated = **4**

    The four trigger assignments are done automatically. These can be modified during device operation using AXI4-Lite, which is especially useful when you have added a new RM through the same mechanism during a field system upgrade.

    At this point, the IP GUI should look like this (showing `rm_shift_left` here):


Global Options
Virtual Socket Manager Options
New Virtual Socket Manager
New Reconfigurable Module
Delete Virtual Socket Manager
Delete Reconfigurable Module

Component Name: prc

Virtual Socket Manager Options

- Virtual Socket Manager to configure: vs shift
- Name (ID): vs_shift (0)
- Enter a new name here:
- [✓] Has Status Channel
- [ ] Has Control Channel
- [ ] Start in Shutdown
- [✓] Shutdown on error
- [ ] Skip RM startup after reset
- [✓] Has PoR RM: rm shift right
- Number of RMs allocated: 4 [2 - 128]

Reconfigurable Module Options

- Reconfigurable Module to configure: rm shift left
- Name (ID): rm_shift_left (0)
- Enter a new name here:
- Shutdown type: Not Required
- Startup type: Not Required
- Reset type: Active High
- Duration of Reset: 10 [1 - 256]
- Bitstream 0 address: 0x00AE9DF4
- Bitstream 0 size (bytes): 375300

Trigger Options

- Number of Hardware Triggers: 4 [0 - 512]
- Number of Triggers allocated: 4 [2 - 512]
- First trigger to display: 0

Trigger ID	Reconfigurable Module to Load	Lock the Trigger
0	rm shift left	☐
1	rm shift right	☐
2	rm shift left	☐
3	rm shift right	☐

Figure 24: VSM vs_shift completed

Next, you will create and populate the Count Virtual Socket following the same basic steps, with slightly different options here and there.

13. Click the **New Virtual Socket Manager** button to create a new VSM.
14. Add two RMs with these names and properties:
    - RM Name = `rm_count_up`
        - Reset type = **Active High**
        - Duration of Reset = **12**
    - RM Name = `rm_count_down`
        - Reset type = **Active High**
        - Duration of Reset = **16**

    For this Virtual Socket, leave the bitstream address and size information at the default of 0. In addition to being defined here, bitstream size information can be added to a routed configuration checkpoint via the PRC Tcl API, or can be added in an active design using the AXI4-Lite interface. For the Count Virtual Socket, the bitstream address and size information is added using the Tcl commands after place and route, but before bitstream generation.

    TIP: *For more information about how to use the PRC Tcl API, see this link in the Partial Reconfiguration Controller Product Guide (PG193).*

    Examine the Tcl scripts in the `<Extract_Dir>/Sources/scripts` directory and find the calls to the PRC Tcl API. The Tcl calls (which have been referenced by `design.tcl`) can be found in one script (`update_prc.tcl`) and the sizes themselves are stored in another (`pr_info.tcl`).
15. Modify these VSM settings from their default values:
    - Virtual Socket Manager name = **vs_count**
    - Start in Shutdown = **checked**
    - Shutdown on error = **unchecked**
    - Has PoR RM = **checked**, **rm_count_up**
16. Define the Trigger Options for the Count functionality:
    - Number of Hardware Triggers = **4**
    - Number of Triggers allocated = **4**

    This completes the planned customization of the PRC IP for this tutorial.
17. Click **OK** and then **Generate** to begin core compilation and out-of-context synthesis.

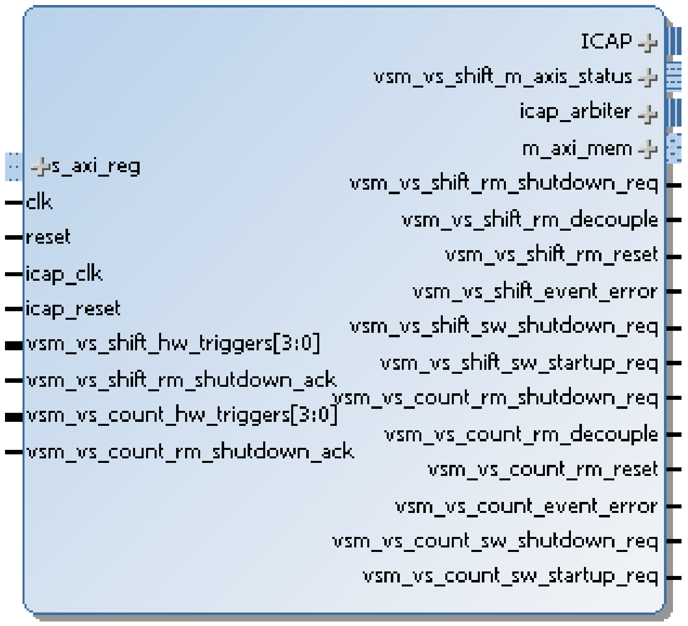

ICAP
vsm_vs_shift_m_axis_status
icap_arbiter
m_axi_mem
s_axi_reg
clk
reset
icap_clk
icap_reset
vsm_vs_shift_hw_triggers[3:0]
vsm_vs_shift_rm_shutdown_ack
vsm_vs_count_hw_triggers[3:0]
vsm_vs_count_rm_shutdown_ack
vsm_vs_shift_rm_shutdown_req
vsm_vs_shift_rm_decouple
vsm_vs_shift_rm_reset
vsm_vs_shift_event_error
vsm_vs_shift_sw_shutdown_req
vsm_vs_shift_sw_startup_req
vsm_vs_count_rm_shutdown_req
vsm_vs_count_rm_decouple
vsm_vs_count_rm_reset
vsm_vs_count_event_error
vsm_vs_count_sw_shutdown_req
vsm_vs_count_sw_startup_req

Figure 25: Final PRC symbol

Compiling the Design

The PR Controller IP is created, but the design is not yet compiled, which could take approximately an hour. In order to create the PROM image with all the necessary full and partial images, source the following scripts in Tcl mode using the commands below.

1. `vivado -mode tcl -source design.tcl`

   Sourcing `design.tcl` generates all the necessary IP (including the PRC), synthesizes and implements the entire design (three configurations), updates the `vs_count` VSM using the PRC Tcl API, and generates bitstreams.

   Note that the customization of the IP has been scripted. Examine the `gen_ip.tcl` script (in `<Extract_Dir>/Sources/scripts`) to see all these parameters defined for automated IP creation, the PRC, and others. The PRC instance you create using the IP GUI is not actually used for the full design processing, so you do not have to complete Section 1 to compile the entire design.

2. `vivado -mode tcl -source create_prom_file.tcl`

   Sourcing `create_prom_file.tcl` creates the PROM image for the KC705 target.

   The bitstream addresses information as requested in the calls to `write_cfgmem` in this script. The PRC works in byte addresses because the data is stored in bytes in AXI. This PROM uses half word addresses because it stores data in half words (16 bits). Divide the ROM address by 2 to get the AXI address. For example, the `shift_left` address is given as `00AEA000` in during PRC customization and `00575000` (half that value) for `write_cfgmem`. Note that the starting addresses are always multiples of 1024 (0x0400) to ensure that each bitstream starts on a byte address boundary.

Conclusion

The Partial Reconfiguration Controller is a hardware-based IP that manages all the critical details of partial reconfiguration events. Connect this core to your AXI bus, hardware triggers, and the ICAP to manage reconfiguration events. Note there are handshaking ports to manage decoupling tasks, request/acknowledge ports to understand if the Reconfigurable Partition is ready to be reconfigured, error flags to account for recovery from illegal conditions, and reset ports if manual reset of the IP is necessary.

Section 2: Interacting with the Partial Reconfiguration Controller IP

This part of the tutorial shows the PRC in action using hardware and software triggers, as well as communication with the core via the AXI4-Lite interface.

Once the partially reconfigurable design is in operation, you can communicate with the core to check status, deliver triggers and make modifications. This section shows these capabilities in an interactive mode using a set of procedures to help simplify the commands.

Tutorial Instructions

1. Prepare the KC705 board for programming.
   a. Connect the JTAG port (U59) to your computer via the micro-USB connection.
   b. Set the configuration mode to 010 (BPI) by setting the Address DIP Switch (SW13) to 00010 (bit 4 is high).
   c. Turn on the power to the board.
2. Open the Vivado IDE.
3. Select **Flow** > **Open Hardware Manager**
4. Click on the **Open Target** link and select **Auto Connect**. The Kintex-7 325T device will be recognized.

5. To program the BPI configuration flash, right-click the device (**xc7k325t_0**) and select **Add Configuration Memory Device**.
6. From the list shown, select the Micron flash **28f00ap30t** and click **OK** twice.
7. In the **Configuration file** field, search the tutorial directory for **pr_prom.mcs** found in the bitstreams subdirectory. Click **OK** to select this file, and then click **OK** to program the flash.

   At this point, the board is ready to operate with the tutorial design. Any power-cycle or hard reset automatically programs the Kintex-7 FPGA with this sample design.

Operating the Sample Design

Position the board so that the text is readable. The LCD screen is on the side closest to you, with the power connection on the right and the JTAG connection on the left. The buttons of interest are the five user push buttons in the lower right corner, plus the **PROG** push button in the middle right. Their functions are as follows:

- PROG (SW14) – program the device from the BPI flash
- North (SW2) – load the Count Up partial bit file
- South (SW4) – load the Count Down partial bit file
- East (SW3) – load the Shift Right partial bit file
- West (SW6) – load the Shift Left partial bit file
- Center (SW5) – reset the design

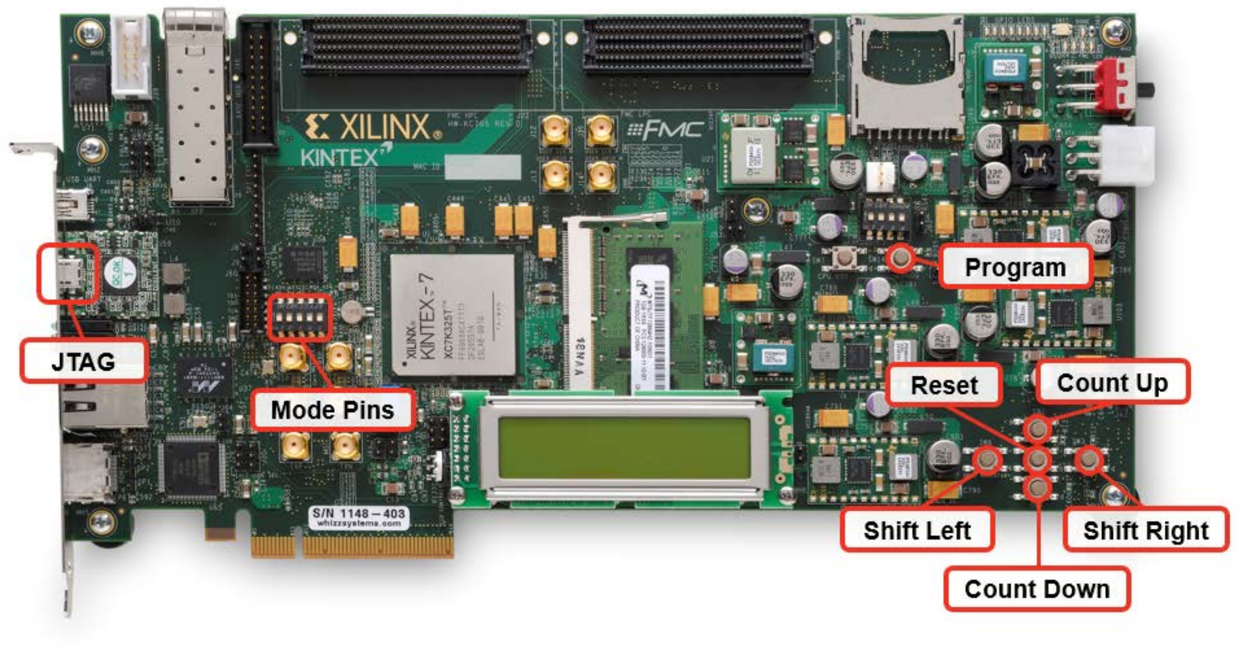


Figure 26: Push buttons, switches and connections on the KC705 demonstration board

8. Program the FPGA by pressing the **PROG** pushbutton. The 8 GPIO LEDs in the upper-right corner will start operation after the DONE LED goes high.

   At this point, the four bits on the left of the GPIO bank are counting up, and the four bits on the right are shifting to the right.

9. Press the **Shift Left** and **Shift Right** buttons alternately.

   With each push, a partial bit file is pulled from the BPI flash by the PRC and delivered to the ICAP, changing the functionality in that Reconfigurable Partition. When this happens, the LED shift direction changes, depending on the button pushed.

10. Press the **Count Down** and **Count Up** buttons alternately.

    With each push, nothing happens. When configuring the PR Controller, the Counter Virtual Socket was programmed to begin in Shutdown mode. It does not respond to any hardware or software triggers until it is moved to Active mode.

Querying the PRC in the FPGA

In this section, you interact with the core via JTAG from the Hardware Manager to understand the status of the core and issue software triggers.

In the Vivado Hardware Manager, you might need to select **Refresh Device** to establish the link to the device over the JTAG connection. Notice the XADC as well as 6 ILA cores and the `hw_axi` link shown under the device in the Hardware view.

1. In the Tcl Console, cd into the PRC tutorial directory then source the AXI4-Lite command Tcl script.

   ```
   source ./Sources/scripts/axi_lite_procs.tcl
   ```

   This enables a set of procedures that make the subsequent interaction with the PRC easier. Examine this file to see how these procedures are defined. Note that these are written explicitly (hard-coded) for this design, the references to Virtual Sockets in any other design will need to be modified. For more information on this topic, consult the *Partial Reconfiguration Controller Product Guide* (PG193).

2. Source the procedure to establish communication with the PRC.

   ```
   prc_jtag_setup
   ```

3. Check the state of each Virtual Socket to see if they are in Shutdown or not.

   ```
   is_vsm_in_shutdown vs_shift
   is_vsm_in_shutdown vs_count
   ```

   You should see that the Shift Virtual Socket is in Active mode (value = `0`), and the Count Virtual Socket is in Shutdown mode (value = `1`).

4. Examine the status of each Virtual Socket.

   ```
   prc_decode_status vs_shift
   prc_decode_status vs_count
   ```

Before examining the data returned, reference Table 2-4 in this link of the *Partial Reconfiguration Controller Product Guide* (PG193). The table in that section defines the values in the STATUS register. While this is a 32-bit register, we will only worry about the lowest 24 bits, as the upper 8 bits are used for Virtual Socket Managers (VSM) in UltraScale devices.

The status of `vs_shift` is `263`, which is `0000_0000_0000_0001_0000_0111` in binary. The status for `vs_shift` may also be `7`, where the only difference is that `RM_ID` is now `0`.

- `RM_ID` (bits 23:8) = `1`. This means RM 1 is loaded (`rm_shift_right`). It may also appear as `RM_ID` (bits 24:8) = `0`. This means RM 0 is loaded (`rm_shift_left`).
- `SHUTDOWN` (bit 7) = `0`. This VSM is not in the shutdown state.
- `ERROR` (bits 6:3) = `0000`. There are no errors.
- `STATE` (bits 2:0) = `111`. The Virtual Socket is full.

The status of `vs_count` is `129`, which is `0000_0000_0000_0000_1000_0001` in binary.

- `RM_ID` (bits 23:8) = `0`. This means RM 0 is loaded (`rm_count_up`).
- `SHUTDOWN` (bit 7) = `1`. This VSM is in the shutdown state.
- `ERROR` (bits 6:3) = `0000`. There are no errors.
- `STATE` (bits 2:0) = 001. `RM_SHUTDOWN_ACK` is `1`, as this VSM is executing the hardware shutdown step.

These explicit details are reported in the breakdown of the status register in the return value from this Tcl proc.

5. Send a software trigger to the Shift Virtual Socket.

```
prc_send_sw_trigger vs_shift 0
prc_send_sw_trigger vs_shift 1
```

Remember that values of 0 and 2 correspond to shift left, and values of 1 and 3 correspond to shift right, as defined during PRC customization.

6. Check the configurations of the RMs for the Count Virtual Socket.

```
prc_show_rm_configuration vs_count 0
prc_show_rm_configuration vs_count 1
```

The values for the bitstream sizes and address are reported here. These values could then be modified to account for necessary adjustments to the size or location of the bitstream. Different indices can be added to insert new RMs. Note that this query cannot be done for `vs_shift`, as the `vs_shift` VSM is not in the shutdown state.

7. Move the Count Virtual Socket Manager into active mode.

```
 prc_restart_vsm_no_status vs_count
```

The Count Up and Count Down pushbuttons can now be used to load these partial bitstreams using the PR Controller.

Modify the PRC in the FPGA

In the final section, you add a new Reconfigurable Module to the Shifter VSM. In the `create_prom.tcl` script, you can see that two black box modules have already been generated. These represent two new RMs that may have been created after the static design was deployed to the field. You modify the PRC settings to access one of these RMs by assigning the size, address, properties and trigger conditions.

1. Shut down the Shift VSM so it can be modified.

   ```
   prc_shutdown_vsm vs_shift
   ```

2. Check the status of the first three RM IDs to see their register bank assignments.

   ```
   prc_show_rm_configuration vs_shift 0
   prc_show_rm_configuration vs_shift 1
   prc_show_rm_configuration vs_shift 2
   ```

   Currently, RM ID 2 has the same mapping as RM ID 0, so the same shift left partial bitstream would be loaded. This is the behavior as requested when the initial trigger mapping was done during core customization.

3. When the MCS file is created for the prom, it adds additional blanking RMs that are already loaded into the BPI flash. Use this sequence of commands to reassign the trigger mapping for slot 2 to point to the blanking Reconfigurable Module for `vs_shift`.

   ```
   prc_write_register vs_shift_rm_control2 0
   ```

   This defines the settings for the `RM_CONTROL` register for slot 2. No shutdown, startup, or reset are required. Note how for the other two slots, the differing reset durations lead to different control values.

   ```
   prc_write_register vs_shift_rm_bs_index2 2
   ```

   This assigns a new bitstream reference for this RM ID.

   ```
   prc_write_register vs_shift_trigger2 2
   ```

   This assigns the trigger mapping such that trigger index 2 retrieves RM 2.

   ```
   prc_show_rm_configuration vs_shift 2
   ```

   This shows the current state of RM ID 2. Note the changes from the prior call to this command.

4. Complete the RM ID 2 customization by setting the bitstream details and then restart the VSM.

   ```
   prc_write_register vs_shift_bs_size2 482828
   prc_write_register vs_shift_bs_address2 13496320
   prc_restart_vsm_no_status vs_shift
   ```

5. Issue trigger events to this VSM using software, as there is no pushbutton assigned for slot 2.

   ```
   prc_send_sw_trigger vs_shift 2
   ```

   Switch between values of 0, 1 and 2 to reload different partial bitstreams.

Note that this same sequence of events could not be performed for the Count VSM as it is currently configured, even knowing that the PROM image has a Count black box partial bitstream sitting at address 13979648 with a size of 541812. During PRC customization, this VSM was selected to have only 2 RMs allocated, so expansion is not permitted.

Conclusion

In this lab, you only scratched the surface of what is possible with the Partial Reconfiguration Controller IP.

- The core is customizable during runtime to meet the needs of deployed systems needing remote upgrades.
- Status and Control registers provide a mechanism for reading and writing information to/from the Virtual Socket Managers.
- VSMs can be shut down and restarted by the user to allow external controllers to partially reconfigure the device.
- Connect the PRC to decoupling and handshaking routines to create a safe and reliable reconfigurable environment on any of your Xilinx FPGAs and SoCs.

Please Read: Important Legal Notices