Vivado Design Suite Tutorial

Partial Reconfiguration

UG947 (v2015.4) November 18, 2015

Revision History

11/18/2015: Released with Vivado Design Suite 2015.4 without changes from the previous version.

Date	Version	Changes
09/30/2015	2015.3	Updated Figure 3 with current PR DRCs. Changed the location where the license is checked in the PR flow. Updated Tcl scripting in Generate Bitstreams.
04/01/2015	2015.1	Validated with release. Updated figures in manual to reflect displays in 2015.1 release.

Table of Contents

Overview

This tutorial covers the Partial Reconfiguration (PR) software support in Vivado® Design Suite release 2015.3. 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 Vivado Design Suite QuickTake Video Tutorial: Partial Reconfiguration in Vivado provides an overview of the Vivado Partial Reconfiguration solution for 7 series devices.*

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*

Hardware and Software Requirements

This tutorial requires that the 2015.3 Vivado Design Suite software release or later is installed.

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.

This tutorial targets the Xilinx KC705 demonstration board, Rev 1.0 or 1.1.

Tutorial Design Description

The sample design used throughout this tutorial is called `led_shift_count`. The design targets an xc7k325t device for use on the KC705 demonstration board. This design is very small, which (1) helps minimize data size and (2) allows you to run the tutorial quickly, with minimal hardware requirements.

TIP: *The software flow shown here applies to all supported devices, but the floorplanning techniques and bit file details are specific to 7 series devices. For more details on UltraScale device requirements, see the Vivado Design Suite User Guide: Partial Reconfiguration* (UG909).

Step 1: Extract 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. The unzipped `led_shift_count` 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 via the use of `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` and `Config_shift_left_count_down`. 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 2015.3 > Vivado 2015.3 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 that 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.

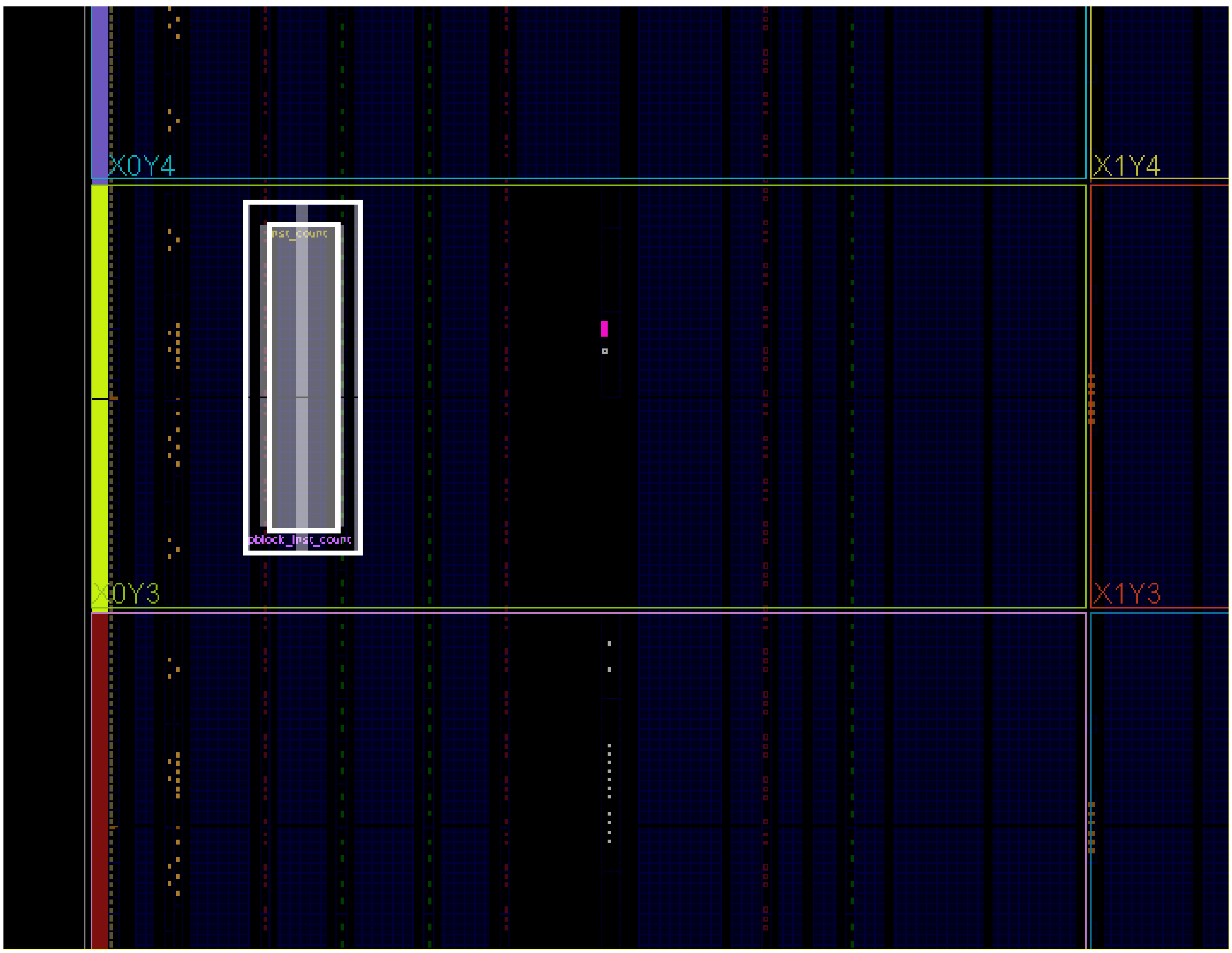

Figure 1: Pblock for the inst_count Reconfigurable Partition

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**. This will 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 will be shown in red.

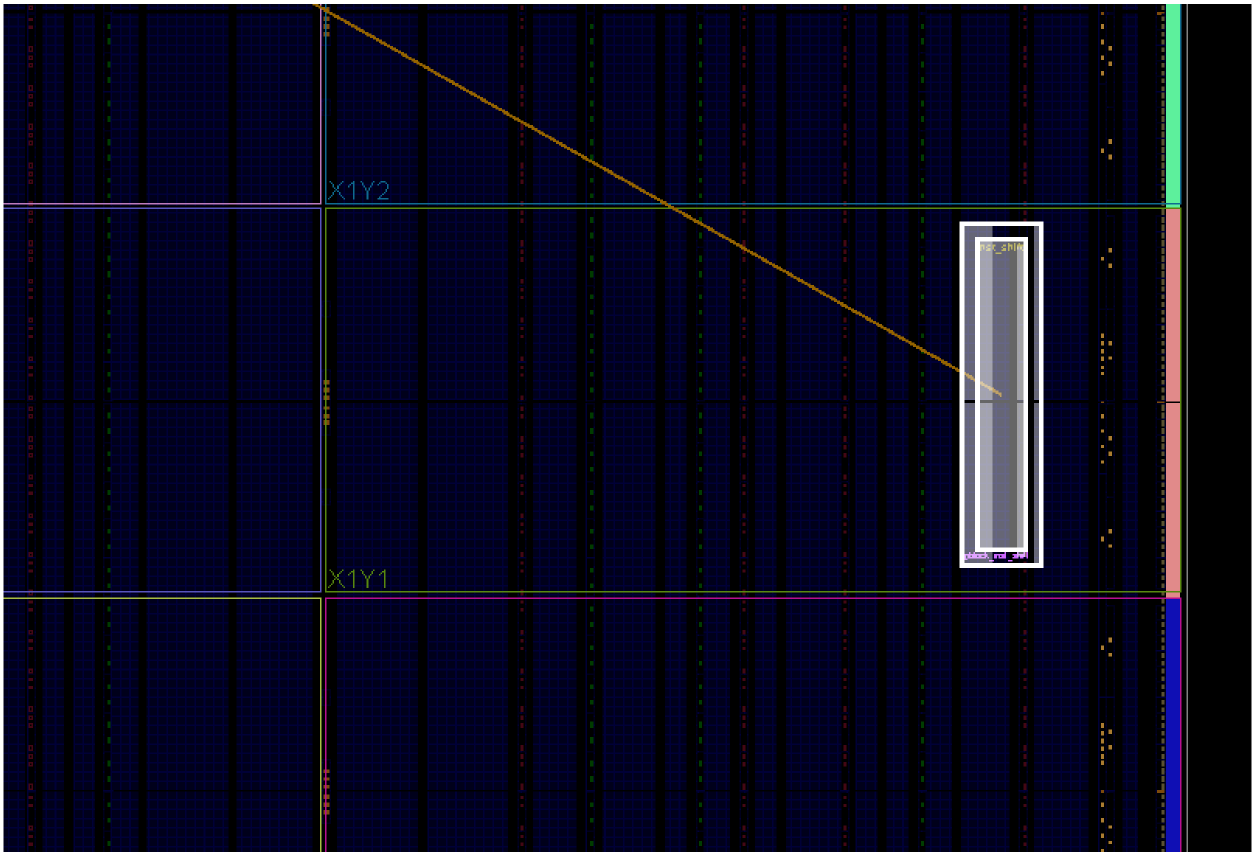

Figure 2: Pblock for the inst_shift Reconfigurable Partition

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.

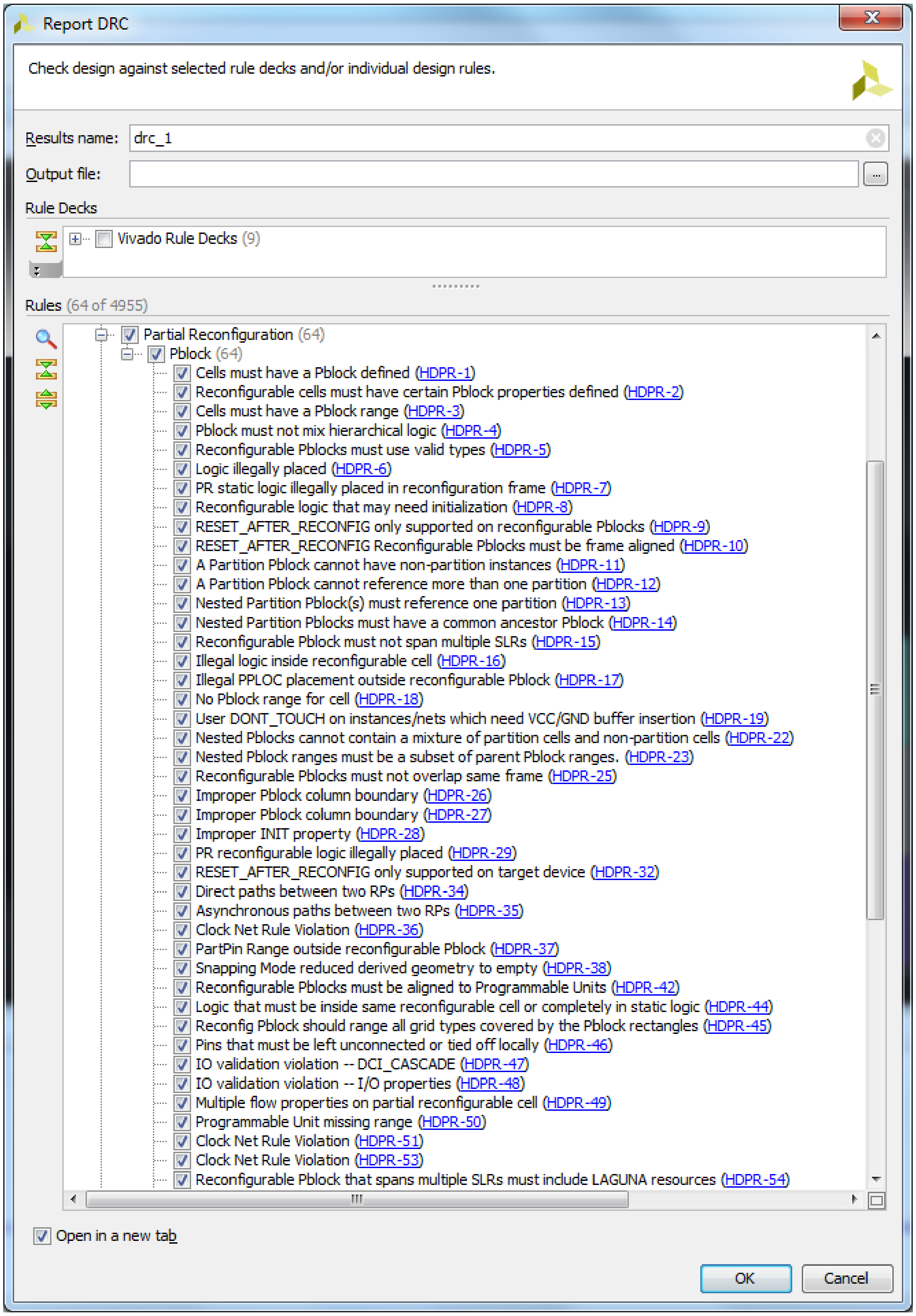

Figure 3: Partial Reconfiguration Design Rule Checks (DRCs)

One or two DRCs will be 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 will be 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. Note that the shading of the Pblock is now more uniform.

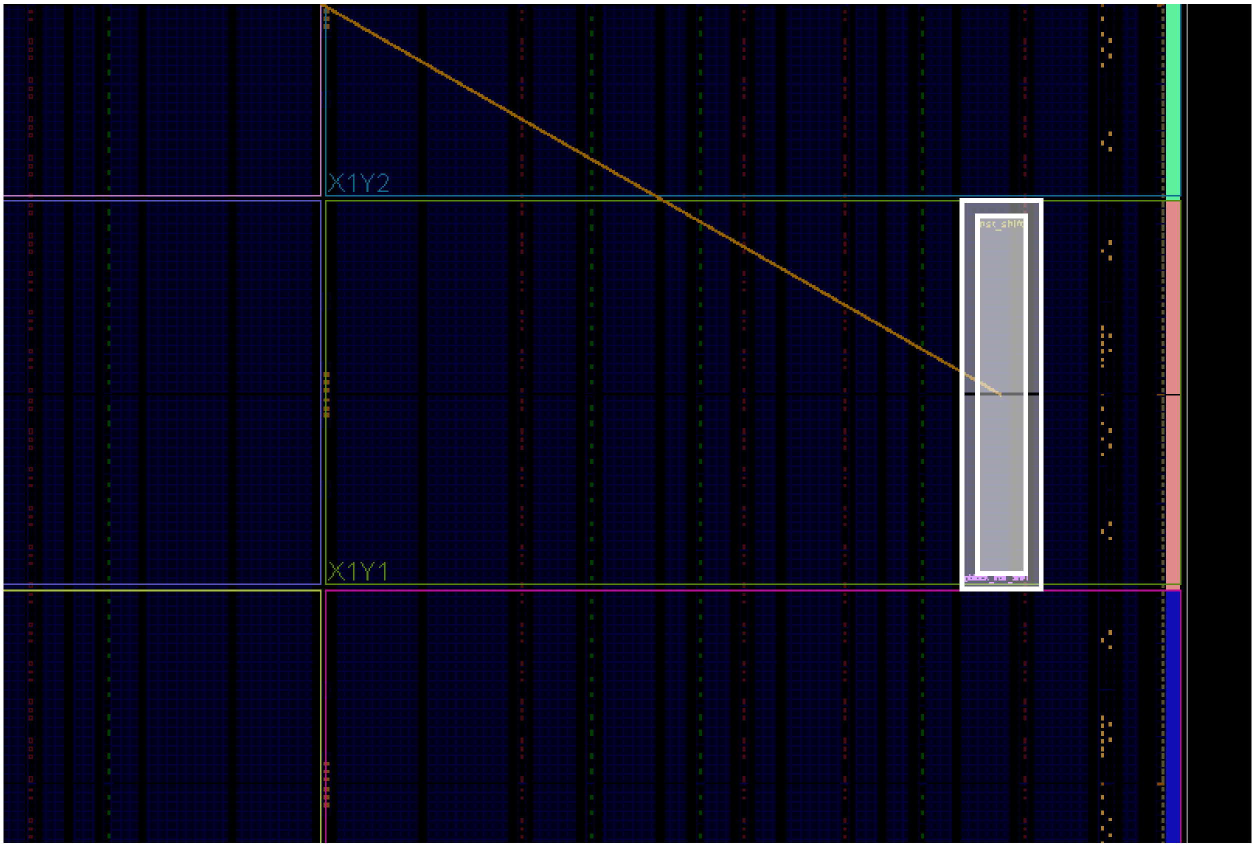

Figure 4: Pblock for the aligned `inst_shift` Reconfigurable Partition

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.

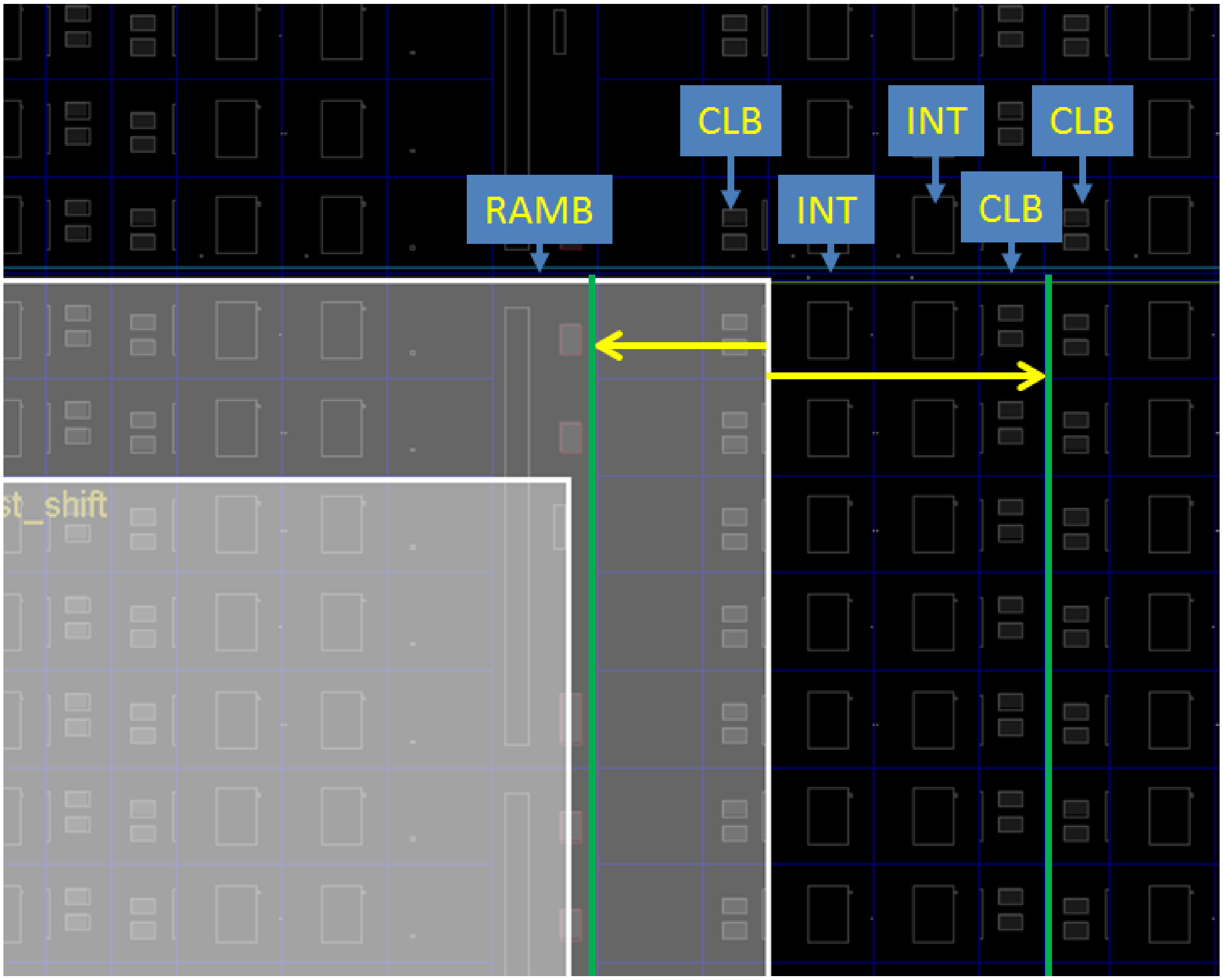

Figure 5: Adjusting the Edges of a Reconfigurable Pblock

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 will make the Pblock narrower, adjusting left and/or right edges as needed. Note that the number and type of resources available will be 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 will export 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, you will implement 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 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 and are the replacement in Vivado for what was Proxy Logic in ISE. 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.

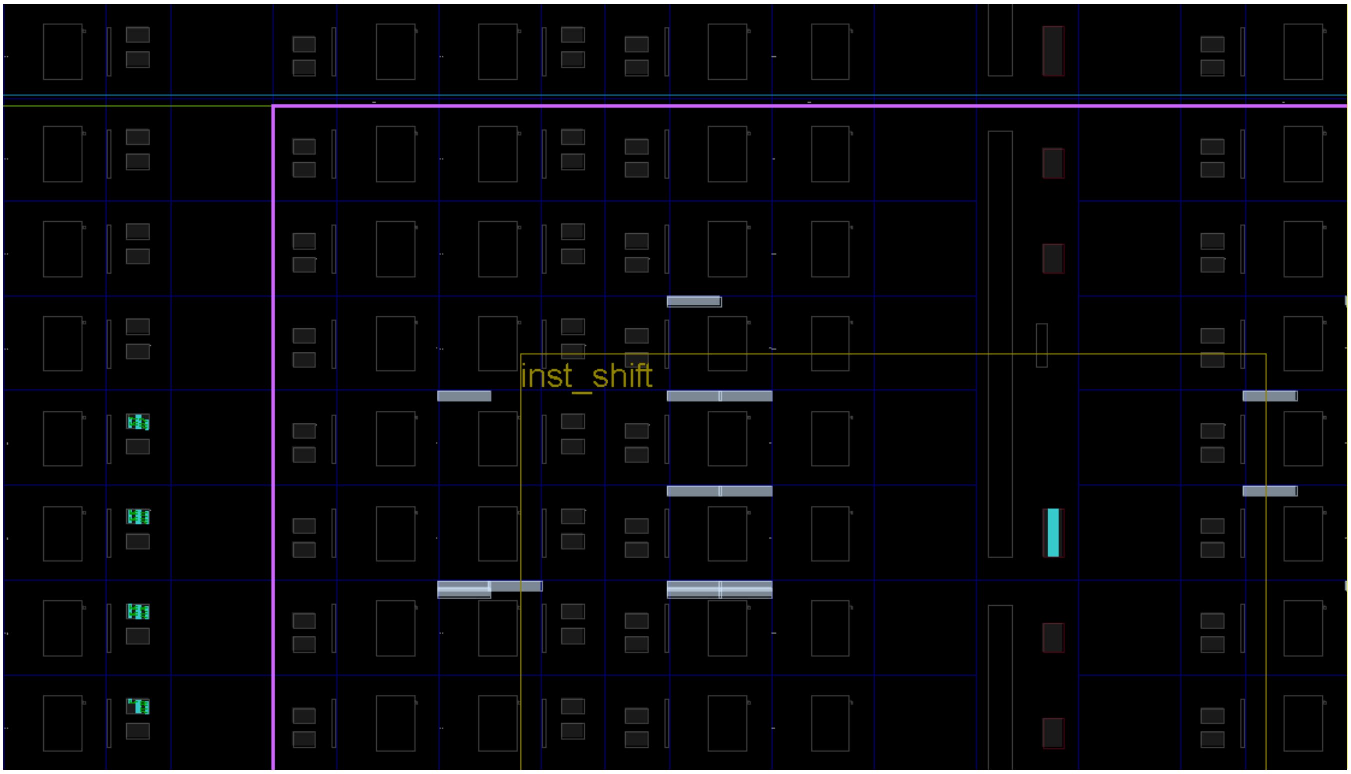

Figure 6: Partition Pins within Placed Design

3. To find these partition pins in the GUI easily:
   a. Select the Reconfigurable Module (e.g., `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.

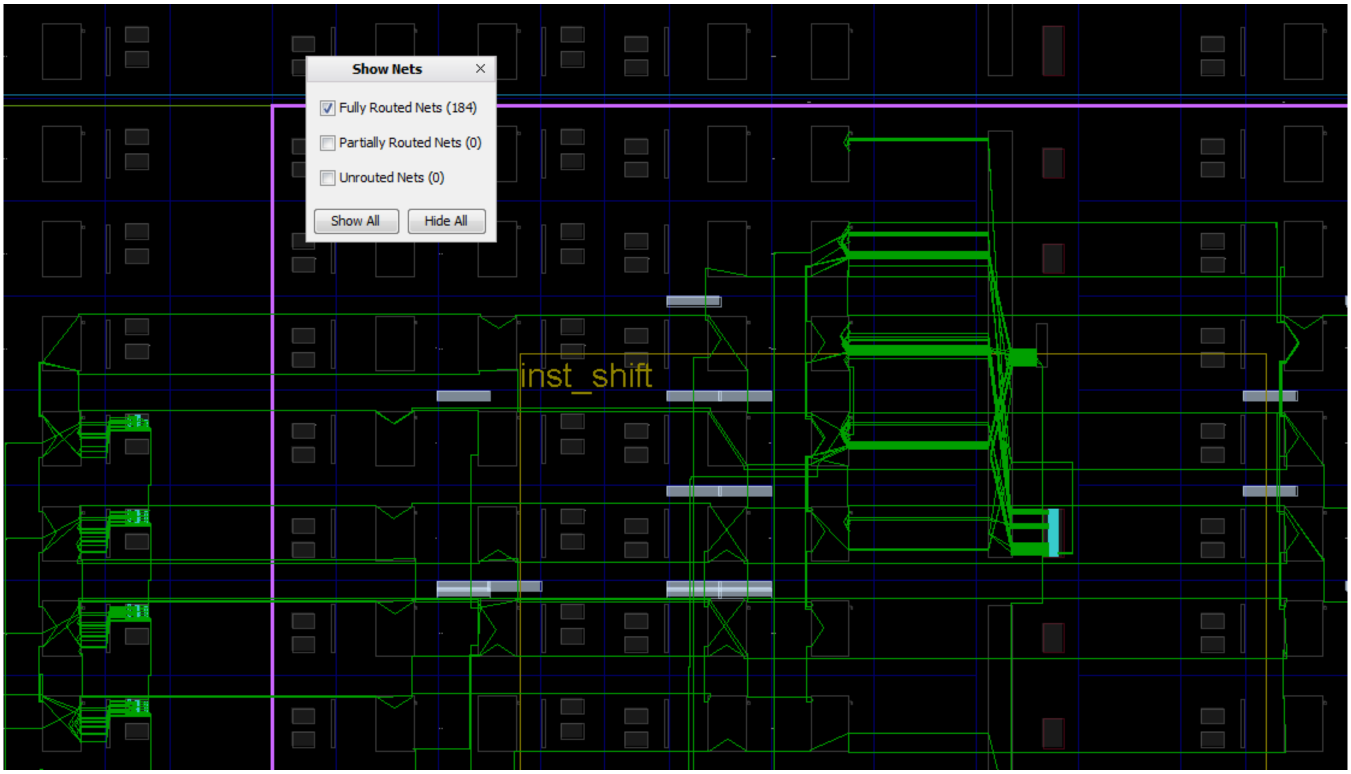

Figure 7: Close-up of First Configuration Routed

Save the Results

6. 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
```

7. [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.

8. 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`

9. 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.

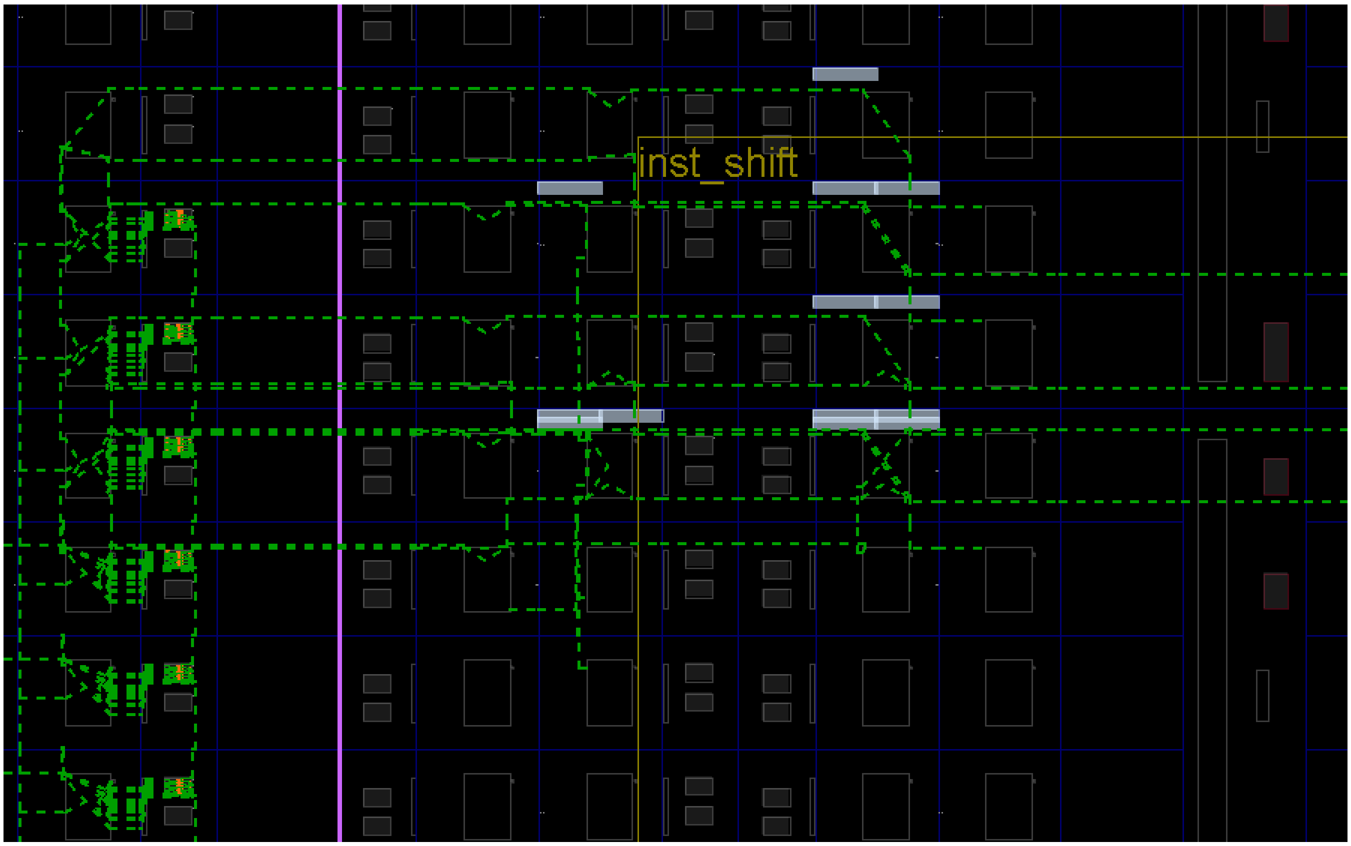

Figure 9: Close-up of Static-Only Design with Locked Routing

10. 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.

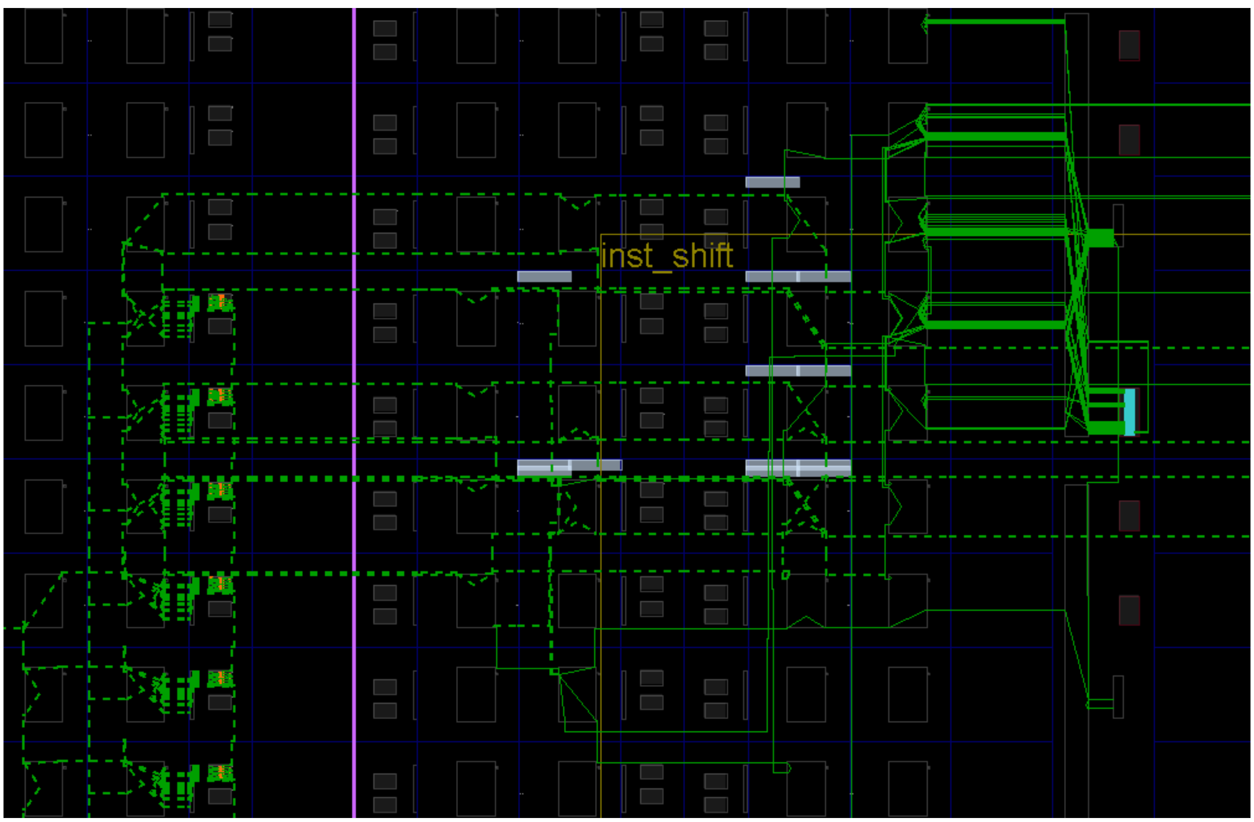

Figure 10: Second Configuration Routed, Showing Locked and New Routes

Save Results

3. 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
```

4. [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 below.

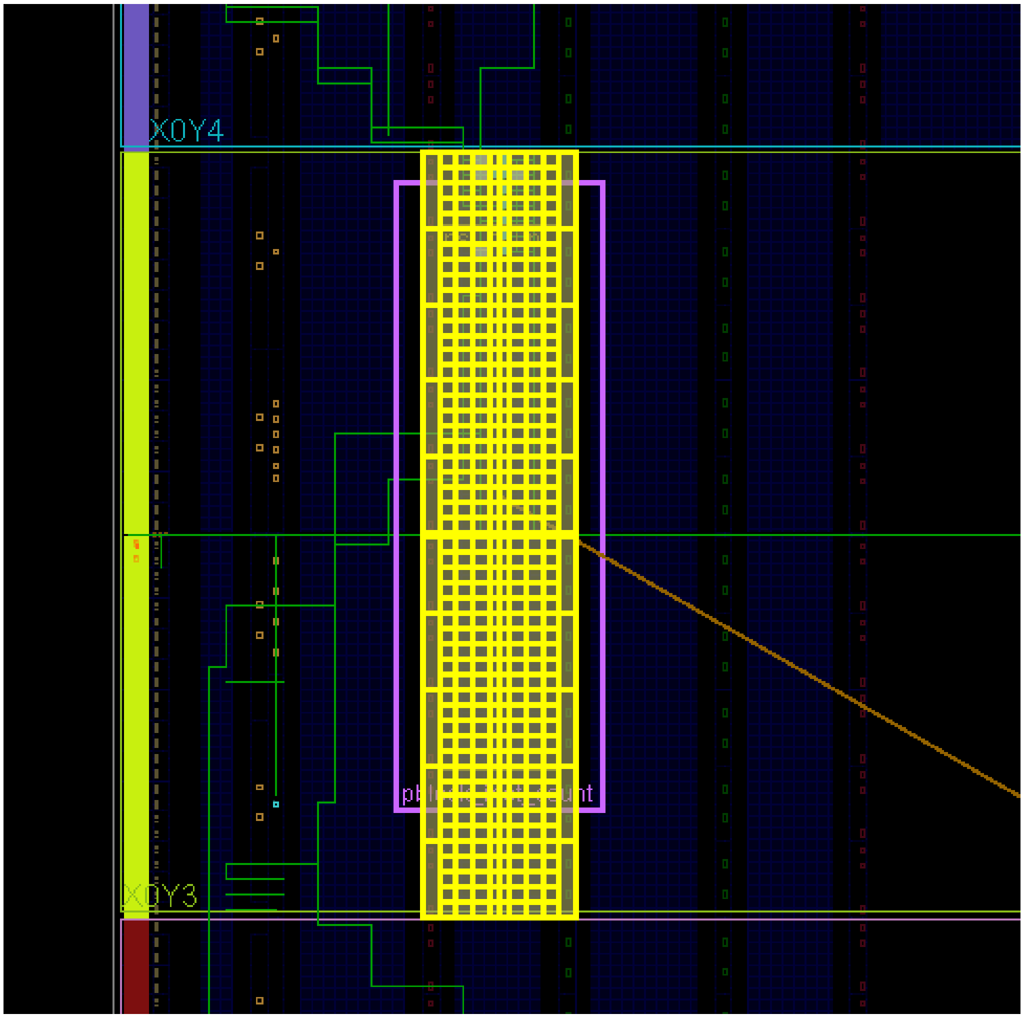

Figure 11: Reconfigurable Partition Frames Highlighted

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 (e.g., 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.

2. First, read the first configuration into memory:

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

3. 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 the 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.*

4. 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.

5. 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 will have 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 will 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 pick 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.

5. 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.

6. 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).

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

Please Read: Important Legal Notices