Isolation Design Flow + Dynamic Function eXchange Example

XAPP1361 (v1.2) May 23, 2024

Summary

This lab application note describes combining isolation design flow (IDF) and dynamic function eXchange (DFX) within a single design. With the help of this application note, designers can develop a fail-safe single chip solution, using the AMD device IDF combined with the Dynamic Function eXchange (DFX), which allows modification of an operating FPGA design, by loading a dynamic configuration file.

This application note follows the rules defined in *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) and the DFX rules defined in *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909). Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for details on the combined flow of IDF + DFX.

An IDF design example is provided in the *Isolation Design Example for Zynq Ultrascale+ MPSoC Application Note* (XAPP1336). For a DFX example on AMD UltraScale+™ devices, refer to the *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947). The document is written with the assumption that the reader is familiar with both IDF and DFX methodologies.

Download the reference design files for this application note from the AMD website. For detailed information about the design files, see the Reference Design section.

Introduction

The AMD device isolation design flow (IDF) is a design methodology that allows for information assurance, functional safety implementations, or any other application module requiring both physical and logical isolation. This methodology is backed by significant schematic analysis and software verification, namely, AMD Vivado™ Isolation Verifier (VIV), that ensures elimination of single points of failure. Single Chip Crypto (SCC) is one specific application of IDF allowing the implementation of a multichip cryptography system, in a single FPGA or a SoC.

Dynamic Function eXchange (DFX) allows for the reconfiguration of modules within an active design. This flow requires the implementation of multiple configurations, which ultimately results in full bitstreams for the first configuration, and partial bitstreams for each reconfigurable module (RM). The number of configurations required varies by the number of modules that need to be implemented. However, all configurations use the same top-level, or static, placement and routing results. These static results are exported from the initial configuration, and imported by all subsequent configurations, using checkpoints.

AMD supports the combined flow of Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) from Vivado 2020.2 onwards. From Vivado 2021.1 onwards, block design container (BDC) flow shall be used to create an IDF+DFX design. Refer to Chapter 5 in *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) for more information about BDC.

Hardware and Software Requirements

The following hardware and software are required for this application note:

- AMD ZCU102 Evaluation Board (revision 1.0 or later, with production silicon)
- AC to DC power adapter (12 VDC)
- USB Type-A to Micro-B USB cable for UART communication
- Secure Digital (SD-MMC) card ≤ 32 GB
- AMD Vivado™ 2021.1 or later

  Note: This application note is verified against 2021.1, 2021.2, and 2022.1 versions.
- Integrating AMD LogiCORE™ SEM IP. Refer to *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices* (XAPP1298) and the associated reference design files.
- AMD Vitis™ unified software development platform 2021.1 or later.

  Note: Future versions of the Vitis unified software platform have not been verified with this application note.
- Serial communication terminal application (Tera Term or PuTTY)
- Required associated reference design files that can be downloaded from the AMD website.

IDF + DFX

The Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) are two production solutions from AMD. They have been available for Zynq UltraScale+ devices from Vivado 2018.3 and later. From Vivado 2020.2 and later, AMD supports the combined flow of IDF and DFX. The document is written with the assumption that the reader is familiar with both IDF and DFX methodologies. Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335), *Isolation Design Example for Zynq Ultrascale+ MPSoC Application Note* (XAPP1336), *Vivado Isolation Verifier User Guide* (UG1291), *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909), Block Design Containers in Chapter 5 of the

Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator (UG994) and *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947) for details on these individual methodologies. With this combined support, the user can create nested isolated modules (IM) inside of a reconfigurable partition (RP).

IDF+DFX is enabled by default for all AMD Zynq™ UltraScale+ MPSoC designs, there is no special `param` needed to enable combined flow. You need to set a `param` to enable the appropriate set of design rule checks for IDF+DFX. To enable IDF+DFX DRCs, set the following parameter in Vivado:

```
set_param hd.enableIDFDRC 1
```

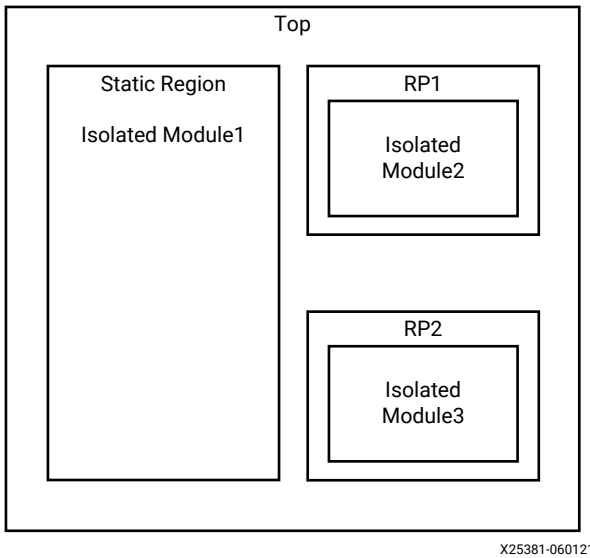
Note: From Vivado 2021.1 onwards, to enable the IDF DRCs there is no need to set `hd.enableIDFDRC` param. IDF DRCs are enabled automatically by the tool when the tool detects that `HD.ISOLATED` property is set to true.

Implementing the Example Design

This reference design contains two reconfigurable partitions (RP), and under each RP, there is an isolated module (IM). There is one more isolated module in the static region. The design uses a simple bare metal application running in PS to load partial bitstreams. The partial bitstreams are placed into a PS-DDR memory and loaded into the device using an application running on the Cortex®-A53. The application uses the xilfpga library to load the partial bitstreams through the Processor Configuration Access Port (PCAP). More information on the xilfpga library can be found in the *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137).

The design static IM contains a MicroBlaze™ and the ICAP to show isolation capabilities. This lab uses the RTL files from Lab 7 of *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947). This lab operates with the assumption that the user is familiar with the DFX design creation, covered through the labs in *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947).

Figure 1: **Design Hierarchy**



To implement this example design first create top level block design, create block design containers for the instances that needs to be converted to reconfigurable partitions, add reconfigurable modules, then using DFX wizard define configurations then run synthesis. After completion of Synthesis, enable IDF for initial config and second configuration RMs. Next step is to create a floorplan for the initial/first configuration and run VIV DRCs to ensure everything is

fine and correct the floorplan if any violations are reported by the tool and save the design. Implement the first configuration and after successful implementation of the first configuration, run VIV DRCs and verify that no violations are reported. Launch the implementation for the second configuration and verify that there are no warnings or errors from the DRC report. Run pr verify and then generate bitstreams. Complete the following steps to implement the example design.

Step 1: Extract the Tutorial Design Files

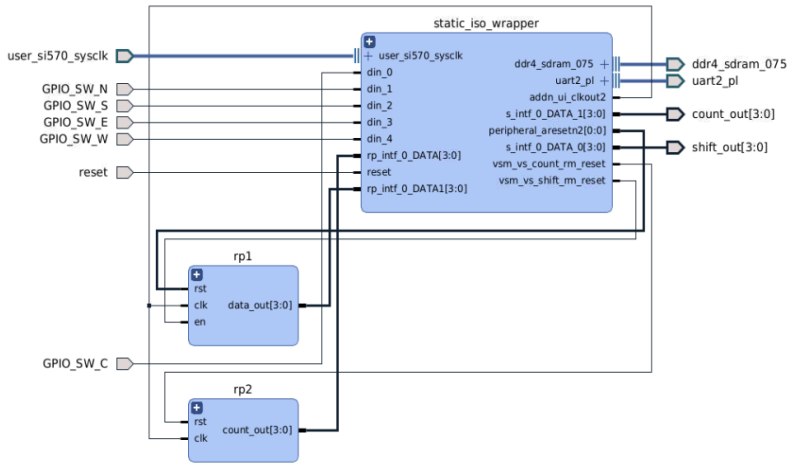
1. Download the reference design files files from the AMD website.
2. Extract the zip file contents to any write-accessible location.

Step 2: Create the Vivado Design and Run Through Synthesis

1. Launch Vivado 2021.1 or a later version from the directory where the lab design scripts are located.
2. Source the project script from the Vivado Integrated Design Environment (IDE) by running the following command from the Tcl console.

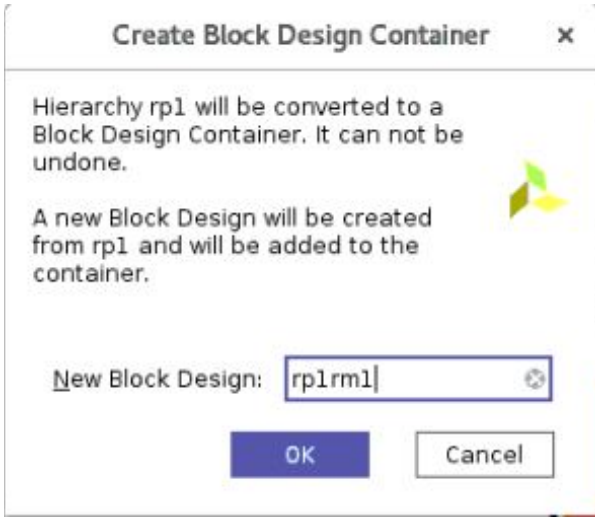
```
source ./create_top_bd.tcl
```

***Note*:** The script `create_top_bd.tcl` creates a project and constructs the top level block design. The block design consists of three hierarchical blocks: `static_iso_wrapper`, rp1 and rp2.



X26959-081622

3. Create a block design container to create the Reconfigurable Partition for rp1 instance:
   a. Right-click the collapsed rp1 instance and select **Create Block Design Container**.
   b. Name the container `rp1rm1` and click **OK**.



X26820-062222

c. Open the **rp1rm1** tab, select the **shift_0** and **shift_addr_0** modules , then right-click to create the hierarchy. Name the hierarchy `iso_2`.

d. Validate and save rp1rm1.

e. Open the **mb_dfx_controller** tab and click **Refresh Changed Modules**.

f. Click **Refresh IP Catalog** if the option appears in the mb_dfx_controller tab.

g. Validate the design.

h. Right-click the collapsed **rp2** instance and select **Create Block Design Container**.

i. Name the container `rp2rm1` and click **OK**.

j. Open the **rp2rm1** tab and select the **count_0** module of rp2rm1.

k. Right-click to create the hierarchy and name the hierarchy `iso_3`.

l. Validate and save rp2rm1.

m. Open the **mb_dfx_controller** tab and click **Refresh Changed Modules**.

n. Click **Refresh IP Catalog** if the option appears in the mb_dfx_controller tab.

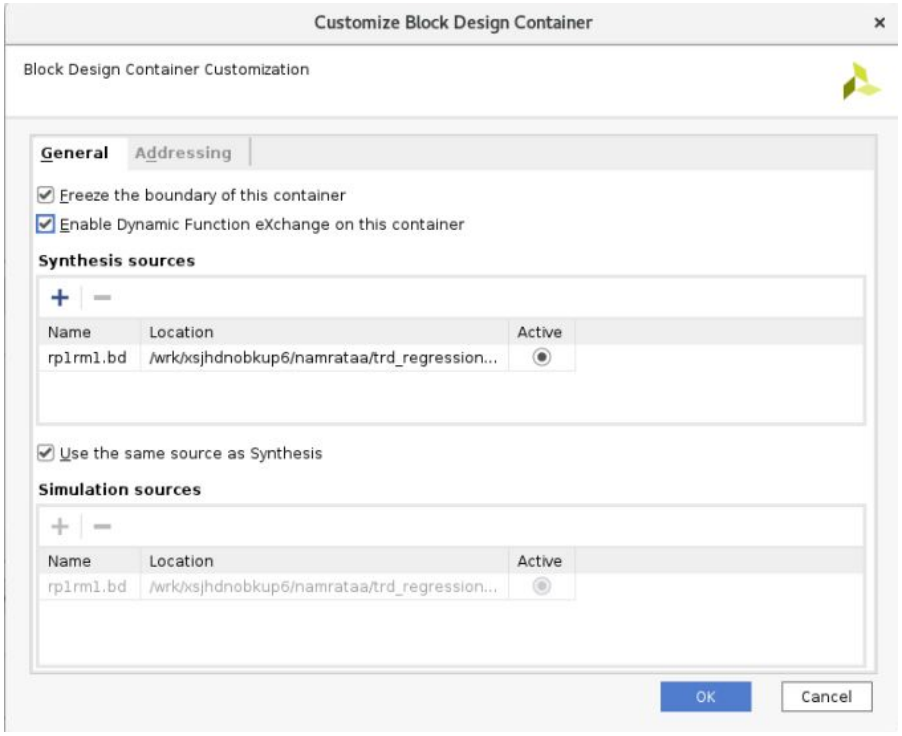
o. Validate and save the design.

4. Enable DFX

   The following instructions enable the DFX capabilities within IP integrator and add new Reconfigurable Modules in the rp1, rp2 block design container.

   a. In the mb_dfx_controller diagram, double-click the **rp1** instance to edit the block design container.

   b. Under the General tab, check both the **Enable Dynamic Function eXchange on this container** and the **Freeze the boundary of this container** options, then click **OK**.



X26821-062222

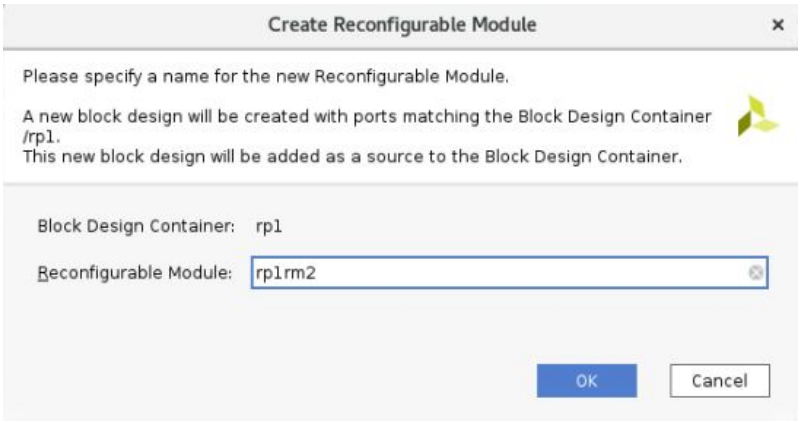
c. Validate and save the design.

d. Repeat previous steps a, b, and c for rp2 to enable DFX for rp2.

5. Add a New Reconfigurable Module.

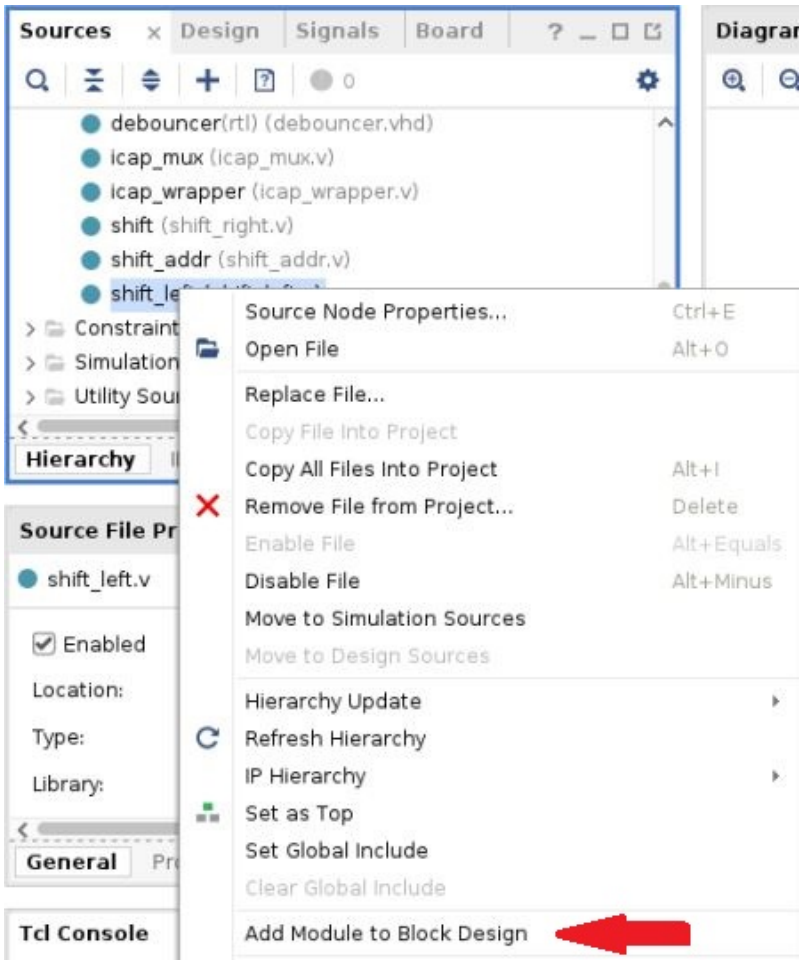
   In the following steps, a new RM is created for the RP that now exists. Follow these instructions to create RM for each RP:

   a. Right-click **rp1** and select **Create Reconfigurable Module**. In the dialog box that opens, give the RM a name of `rp1rm2` and click **OK**.



X26822-062222

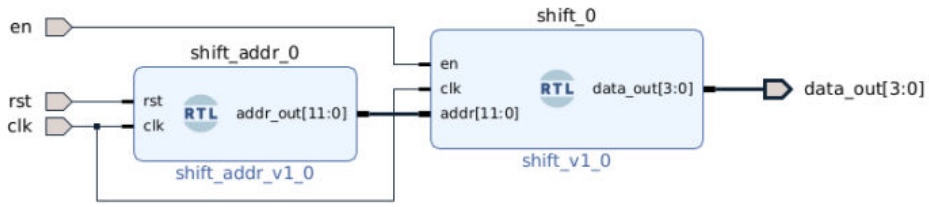
   b. Right-click **shift_left** in the Design Sources tab under the Sources and click **Add Module to Block Design**.



X26823-062222

c. In the Sources tab, select **Design Sources**, then right-click **shift_addr**, then click **Add Module to Block Design**.

d. Connect the pins to create the diagram as shown in following figure.



e. Select the **shift_left_0** and **shift_addr_0** modules, right-click to create the hierarchy, and name the hierarchy `iso_2`.

f. Validate and save **rp1rm2**.

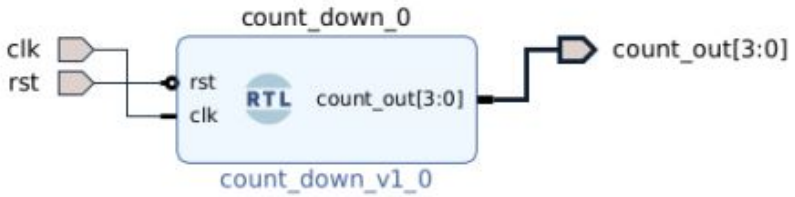
g. Open the **mb_dfx_controller** tab and validate and save the design.

h. Right-click the **rp2** instance, select **Create Reconfigurable Module**. In the dialog box that opens, and name the RM `rp2rm2`.

i. Click **OK**.

j. In the Sources tab, select **Design Sources**, right-click on **count_down**, then click **Add Module to Block Design**.

k. Connect the pins to create the diagram as shown in following figure.



X26825-062222

l. Open the **rp2rm2** tab, select the **count_down** module, right-click to create the hierarchy, and name the hierarchy `iso_3`.

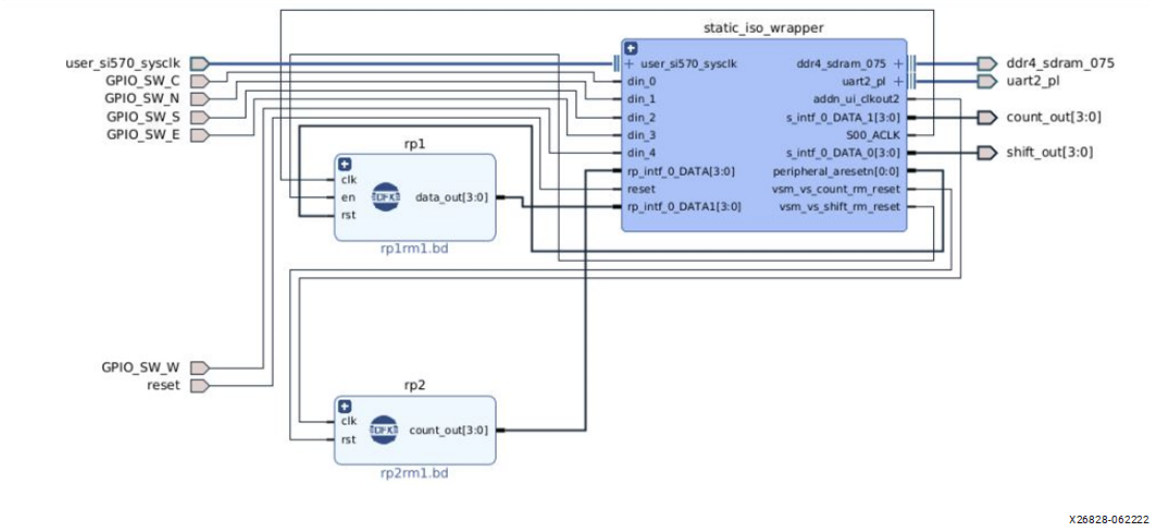
m. Validate and save **rp2rm2**.

n. Open the **mb_dfx_controller** tab.

o. Validate and save the design.

6. Click **Generate Block Design**, which is under IP INTEGRATOR in the Flow Navigator window. Ensure the **Out of context per IP** synthesis option stays selected, and then click **Generate**.

   This step creates all of the IP identified in the block design and launches them through synthesis. This process can take a while to run. For this design, the block design covers the vast majority of the static logic representing the design infrastructure. This step does not launch out-of-context synthesis for the RTL submodules, which includes the shift and count Reconfigurable Modules. The example design instantiates the Zynq UltraScale+ MPSoC devices to support partial bitstream loading, via the PCAP.

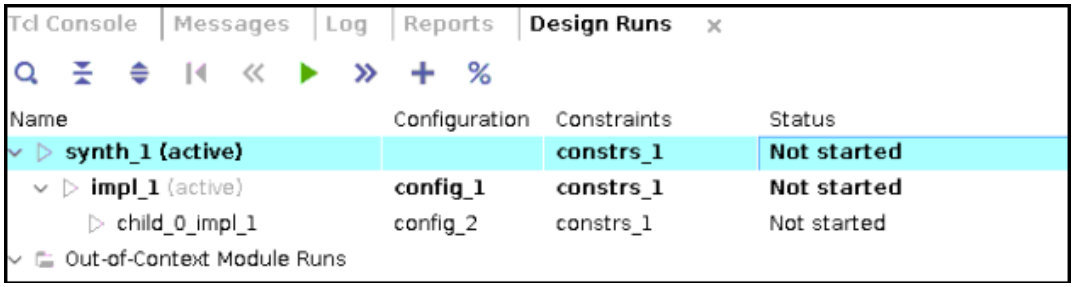


Dynamic Function eXchange Wizard is used to define relationships between the different parts of a DFX design. Within the DFX Wizard, you will define configurations, and configuration runs. A configuration is a full design image, with one RM per RP. A configuration run is a pass through the place and route tools to create a routed checkpoint for that configuration. The DFX Wizard also establishes parent-child relationships between configuration runs, helping automate required parts of the flow, including static design locking and `pr_verify`, and setting up dependencies between runs, so Vivado knows what steps to rerun when sources are modified. This example project has two configurations and their runs defined. There are two configurations created for the design, `Config1` and `Config2`.

`Config1` contains shift_right and count_up RMs. `Config2` contains shift_left and count_down RMs. `Synth_1` is the parent run for the design.

There are two runs in the example design. This first run establishes the static design, and the first pair of RMs. The second run uses the locked static design, and the second pair of RMs.

The run `impl_1` is the parent run and implements `Config1`. The child run `child_0_impl_1` implements `Config2`.

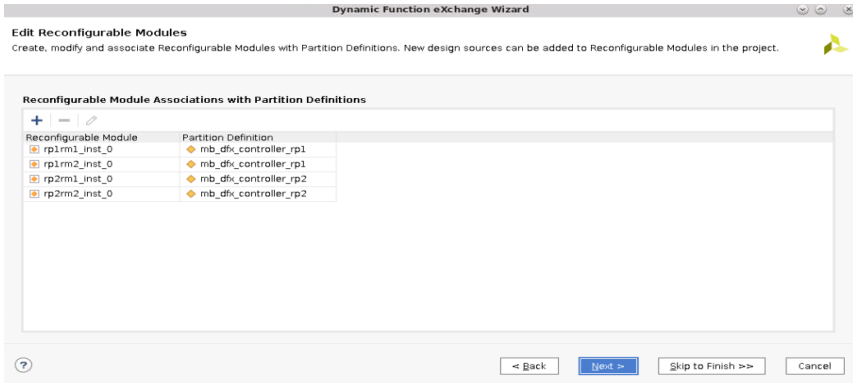


7. Use Dynamic Function eXchange Wizard to define configurations and run synthesis Within the DFX Wizard, you will define configurations and configuration runs. The DFX Wizard also establishes parent-child relationships between configuration runs. To create the configuration runs, follow the instructions:

   a. Open the DFX Wizard by clicking Dynamic Function eXchange Wizard in the Flow Navigator or by selecting that option under the Tools menu.

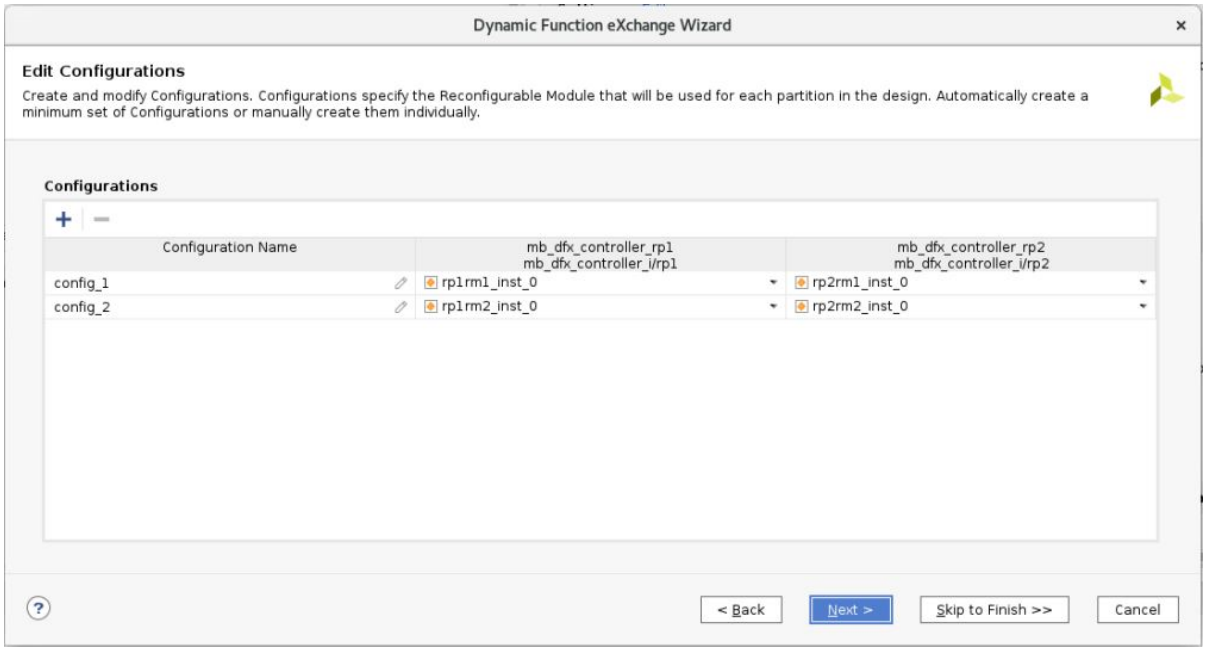
X26826-062222

b. Click **Next**. In the Edit Reconfigurable Modules step, you will see the Partition Definitions and their corresponding Reconfigurable Modules.



X26961-081622

c. Click **Next**. In the Edit Configurations step, click the **automatically create configurations** link to generate two configurations.



X26827-062222

d. Click **Next**. In the Edit Configuration Runs step, click the **automatically create configuration runs** link to create one run per configuration.

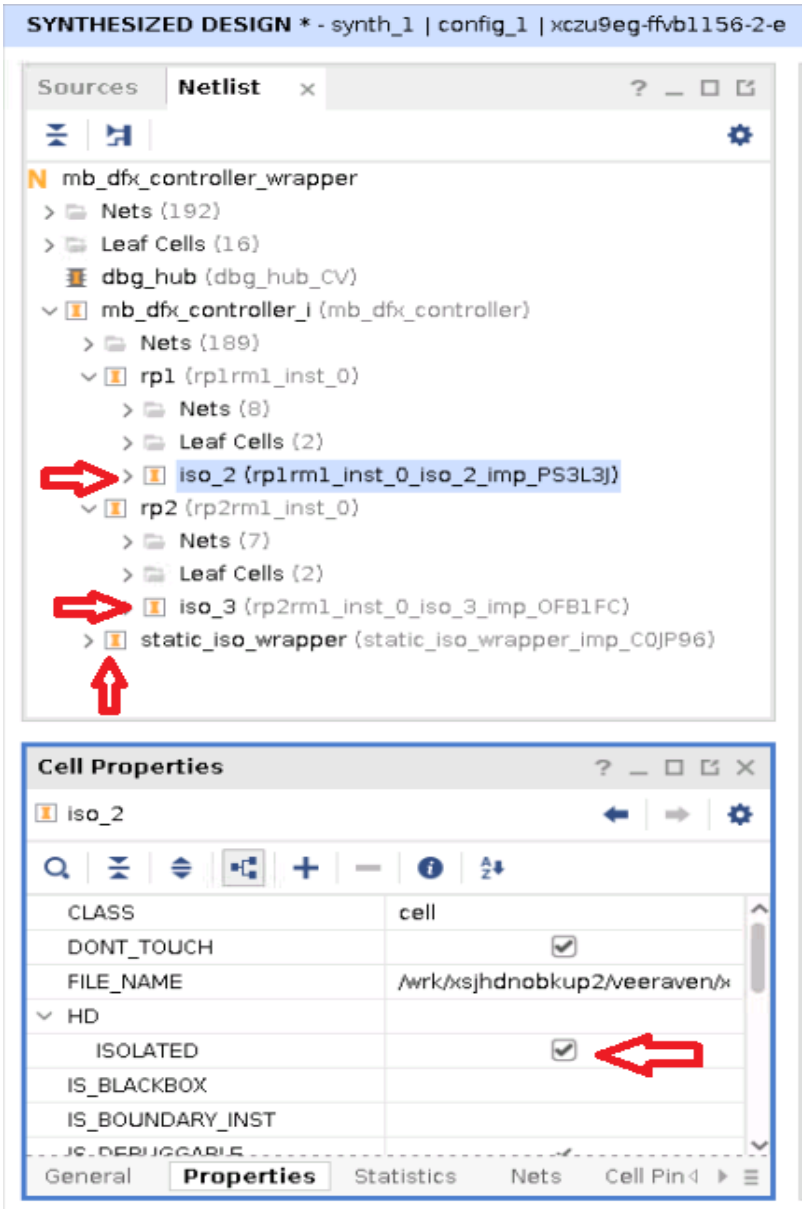
e. Click **Next**, then click **Finish** to complete this section.

f. Once the configuration runs are created, validate and save the design.

g. Click **Run Synthesis** in the Flow Navigator. This launches the synthesis of the remaining modules and the entire example design.

Step 3: Enabling IDF

1. After synthesis completion, open **Synthesized Design**. As shown in the following figure, set HD.ISOLATED to **true** for `static_iso_wrapper`, `rp1/iso_2`, and `rp2/iso_3`. Save the constraints by naming the file **top**.



X26962-081622

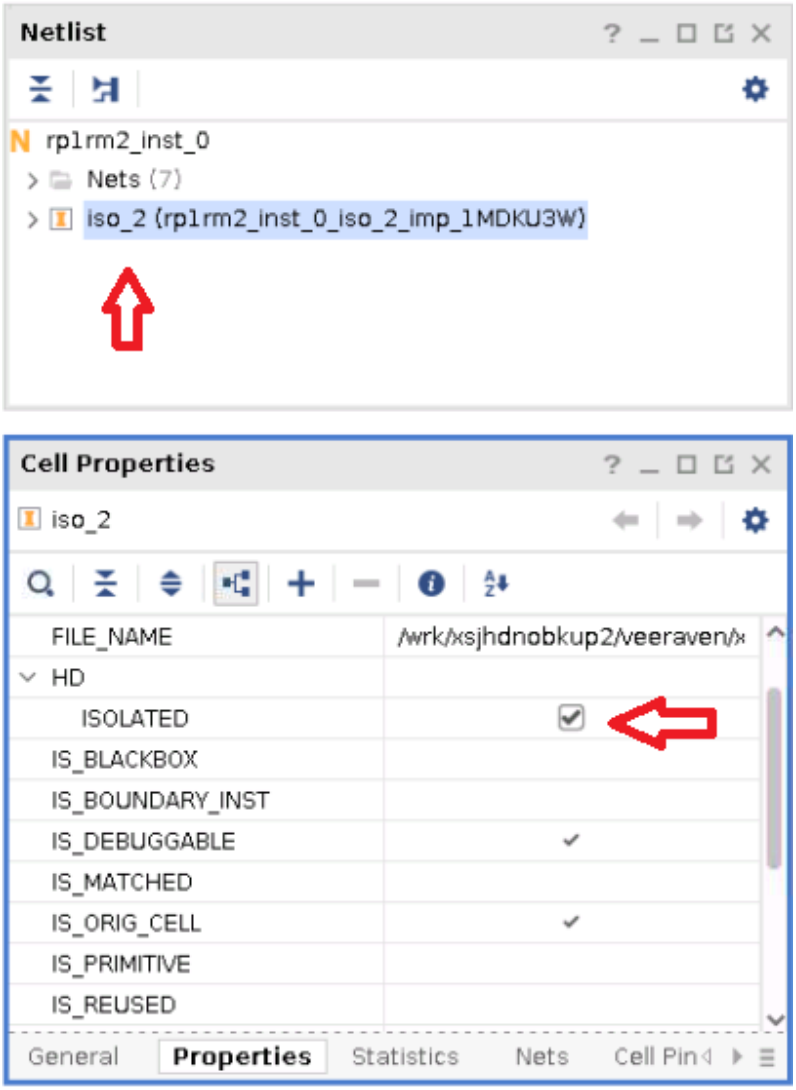
This enables IDF for the `iso wrapper` module in the static region and the two modules in the `Config1` RMs.

2. To enable IDF in `Config2` RM submodules, you must enable HD.ISOLATED on the synthesis DCPs of the Config2 RMs, and save them. IDF related optimizations take place in `link_design`. Therefore, the HD.ISOLATED property must be enabled for RM submodules before link design for subsequent configurations. To enable HD.ISOLATED on the Config2 RMs, perform the following *or* this can easily be completed using a post-synth Tcl Hook script, as mentioned in the Enable HD.ISOLATED on Config2 RMs using Post-Synth Tcl Hook Script section.

   a. Select **File → Check Point → Open**, then open **shift_left synth dcp** by selecting:

   ```
   project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/
   rp1rm2_inst_0_synth_1/rp1rm2_inst_0.dcp
   ```

   When asked to close the project, choose **No**, then click **OK** and ignore the critical warning. Set HD.ISOLATED to **true** for `iso_2`. Save and close the dcp.



X26963-081622

   b. Open **RP2RM2 synth dcp** from:

   ```
   project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/
   rp2rm2_inst_0_synth_1/rp2rm2_inst_0.dcp
   ```

Set HD.ISOLATED to **True** for `iso_3`. Save and close the dcp.

Enable HD.ISOLATED on Config2 RMs using Post-Synth Tcl Hook Script

IMPORTANT! *These steps can be executed as an alternative to the previous two steps that enabled IDF on the rp1rm2 and rp2rm2 RMs. This step must be performed before launching the Synthesis in the previous section.*

1. Create a file with the following contents and name the file `rp1_rm2_post_synth_tcl.tcl`.

```
set_property HD.ISOLATED true [get_cells shift] -quiet write_checkpoint -
force -noxdef rp_shift.dcp
```

2. Create one more file with the following contents and name the file `rp2_rm2_post_synth_tcl.tcl`.

```
set_property HD.ISOLATED true [get_cells count] -quiet write_checkpoint -
force -noxdef rp_count.dcp
```

3. Add the created files to the project by running the following commands from the Tcl console.

```
add_files -fileset utils_1 -norecurse ./rp1_rm2_post_synth_tcl.tcl
add_files -fileset utils_1 -norecurse ./rp2_rm2_post_synth_tcl.tcl
```

4. Add `rp1_rm2_post_synth_tcl.tcl` and `rp2_rm2_post_synth_tcl.tcl` files as `post-synth Tcl Hook` scripts for the corresponding synthesis runs by running the following commands from the Tcl console:

```
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files ./
rp1_rm2_post_synth_tcl.tcl -of [get_fileset utils_1] ] [get_runs
shift_left_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files ./
rp2_rm2_post_synth_tcl.tcl -of [get_fileset utils_1] ] [get_runs
count_down_synth_1]
```

5. Launch **Synthesis**.

Step 4: Enabling IDF+DFX DRCs

Prior to AMD Vivado™ 2021.1 IDF+DFX DRCs are disabled by default.

To enable the IDF DRCs, run the following command from the Tcl console:

```
set_param hd.enableIDFDRC 1
```

This parameter information is not stored in the database and must be re-enabled for every Vivado session. The parameter is most easily set in `vivado_init.tcl` which automatically sets the parameter for each of the Vivado sessions. From Vivado 2021.1 onwards, to enable the IDF DRCs there is no need to set `hd.enableIDFDRC` param. IDF DRCs are enabled automatically by the tool when the tool detects that `HD.ISOLATED` property is set to true.

Step 5: Floorplanning First Configuration

Open the `synth` dcp for floorplanning the first configuration (Config1). Ensure that the HD.ISOLATED is set to **True** for the following modules: static_iso_wrapper, rp1rm1/iso_2, rp2rm1/iso_3.

1. Assign package pins and I/O ports for the design. Run the following command from the Tcl console to assign package pins and I/O ports.

   ```
   source ./sources/pins.xdc
   ```

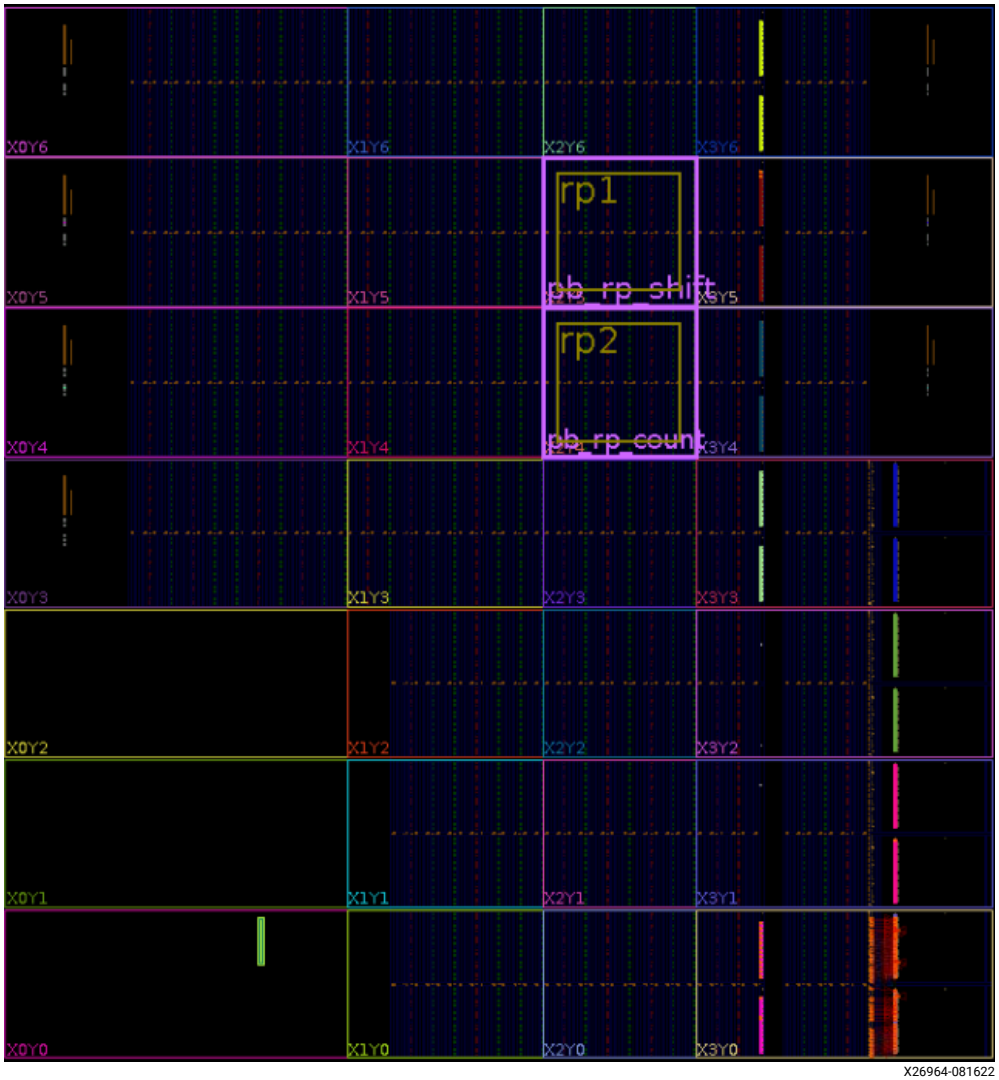
2. Create Pblocks for Reconfigurable Partitions. Create a Pblock for `rp1` and assign site ranges for the Pblock. Run the following command from the Tcl console to create a Pblock for the `rp1`:

   ```
   source ./sources/pb_rp_shift.xdc
   ```

3. Create a Pblock for `rp2` by running the following command from the Tcl console:

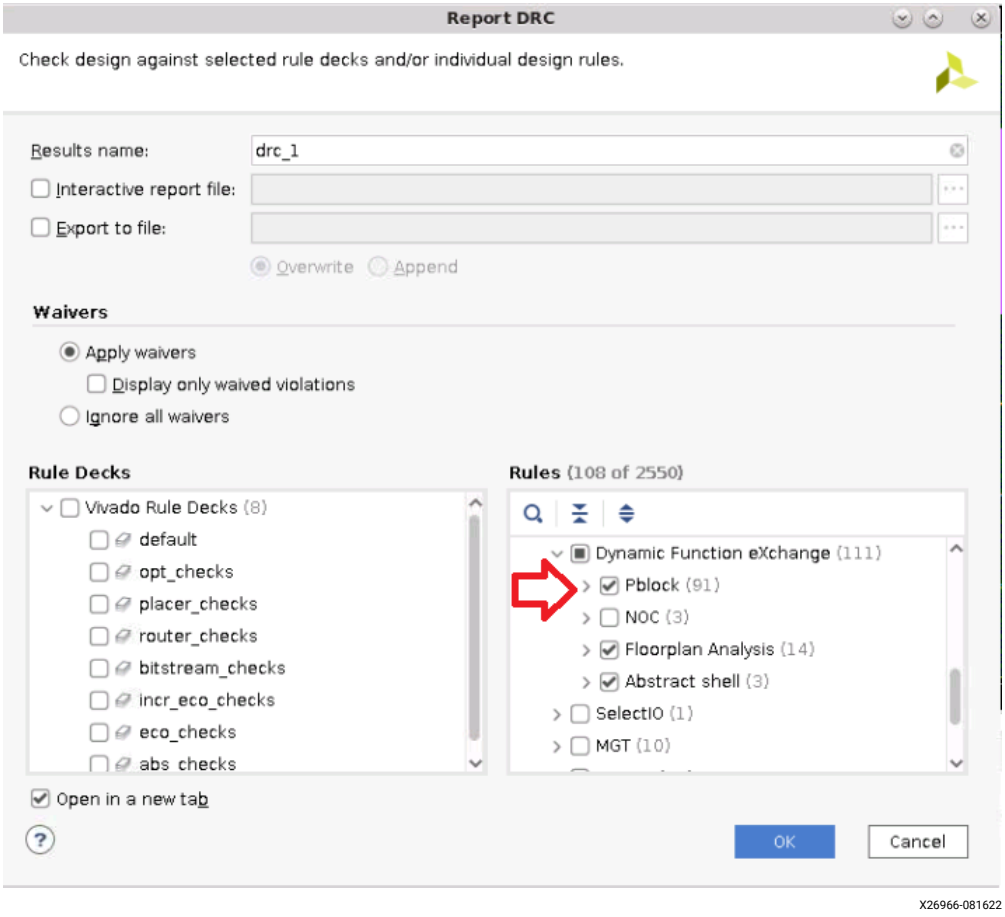
   ```
   source ./sources/pb_rp_count.xdc
   ```

   After running the floorplan, the device looks like the following figure:



X26964-081622

4. Save the design. Run the DFX DRCs by opening **Reports → Report DRC...**. Select **DFX DRCs**, and click **OK** to run the DRCs.



After running the DFX DRCs, you can see there are no warnings or errors. There are two advisory messages. You can ignore those messages, as they will resolve after creating the Pblock for the static region, which is the `static_iso_wrapper`.

5. Draw Pblocks for the Isolated Modules.
   a. Draw a Pblock for the static Isolated Module which is the `static_iso_wrapper`. Run the following command from the Tcl console:

```
source ./sources/pb_iso_wrapper.xdc
```

   b. Draw a Pblock for the isolated module shift (`iso_2`) inside the Reconfigurable Partition `rp1`. Run the following command from the Tcl console:

```
source ./sources/pb_shift_right.xdc
```

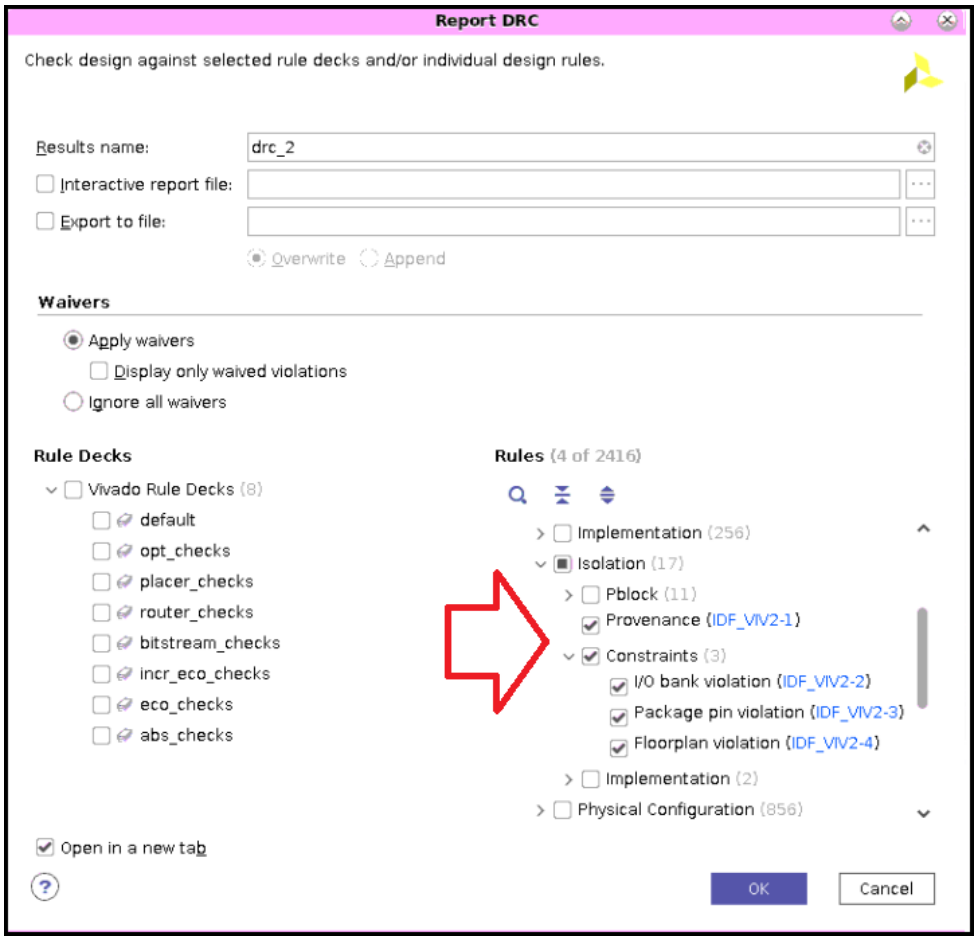
   c. Draw a Pblock for the Isolated Module count (`iso_3`) inside of the Reconfigurable Partition`rp2`. Run the following command from the Tcl console:

```
source ./sources/pb_count_up.xdc
```

   d. Save the constraints by saving the design.

      Note: Ignore the constraints order changed warning.

6. Run VIV DRCs. These are IDF DRCs which are updated for the IDF+DFX flow. Under Report DRC, select **IDF_VIV2-1**, **IDF_VIV2-2**, **IDF_VIV2-3**, and **IDF_VIV2-4**. Click **OK** to run the DRCs.



***Note*:** Do not select IDF_VIV2-5 and IDF_VIV2-6 as the design is not implemented yet. After the DRC run is completed, you can see many IDF-4 violations, as shown in the following figure.

7. Correct the floorplan violation.

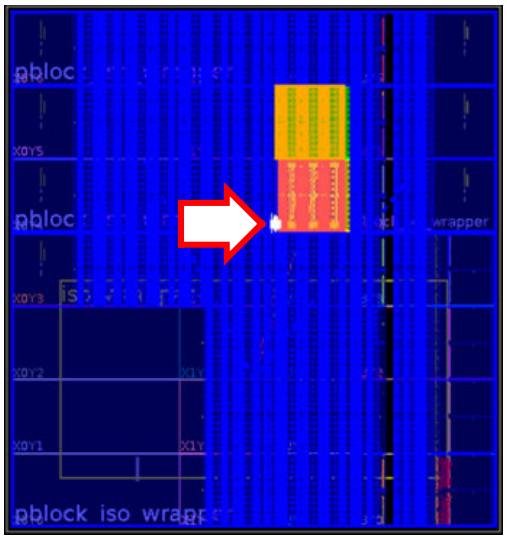
   IDF highly recommends taking advantage of the highlighting features of the Vivado tools. The following Tcl script highlights all the Pblocks in the design:

```
Set pblocks [get_pblocks *];set ci 1;foreach pblock $pblocks
{highlight_objects -color_index [expr {1 + ($ci % 19)}] [get_pblocks
$pblock]; incr ci}
```

Shading the Pblocks tells the user what resources are included in it. Although, shading is visible when the Pblock is selected, highlighting it helps for better visibility. Additionally, it helps to differentiate between different Pblocks. In a highlighted Pblock, resources that have color are added to the Pblock, and the regions that are black are not included.

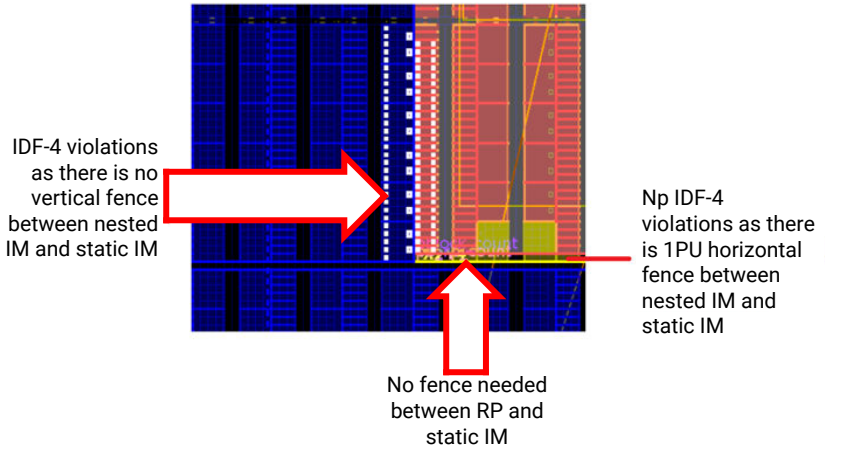
Perform the following to correct the IDF violations.

a. Click the first violation from the DRC report window. This selects the violating Programmable Units (PU) in the device view. Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for details on PU. You can select multiple violations to see the corresponding violating PUs.



X25385-060121

b. Zoom to the selected PU area. You can see there is no fence between the nested IM inside of the RP and the static IM in a vertical direction. IPU fence is needed here; the fence can either be inside of the RP or outside of the RP.

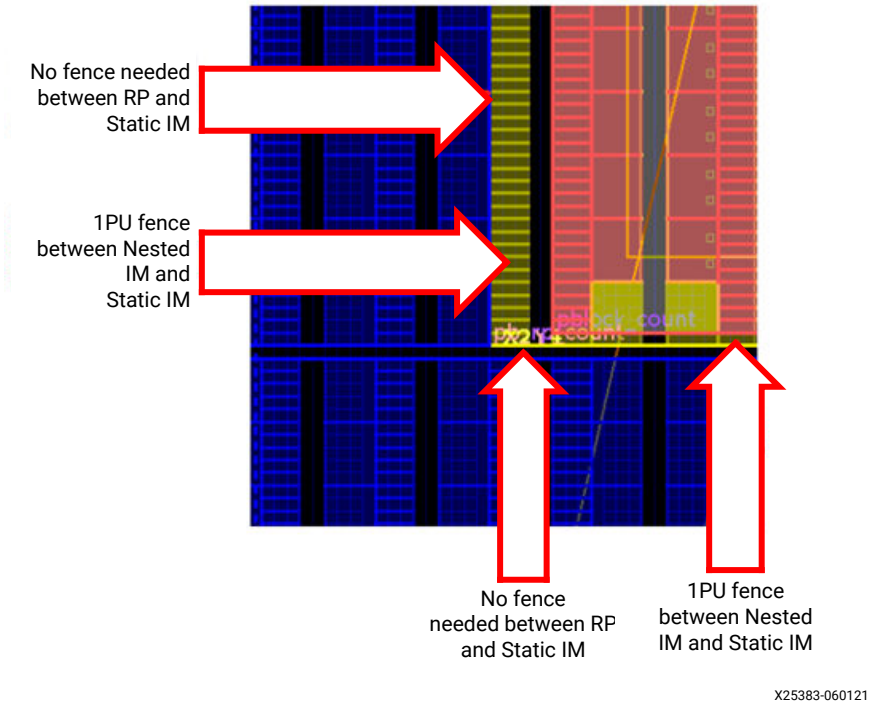


X25382-061421

c. Create a fence inside of the RP. Select the nested IM Pblock, and drag the Pblock edge towards the right side, so that the 1PU fence is created. Or, you can use the following `resize_pblock` command to change the Pblock boundaries.

```
resize_pblock pblock_count -add {SLICE_X58Y241:SLICE_X75Y298
BUFCE_LEAF_X312Y16:BUFCE_LEAF_X407Y19
BUFCE_ROW_FSR_X77Y4:BUFCE_ROW_FSR_X102Y4
DSP48E2_X12Y98:DSP48E2_X14Y117
HARD_SYNC_X14Y8:HARD_SYNC_X19Y9 RAMB18_X7Y98:RAMB18_X9Y117
RAMB36_X7Y49:RAMB36_X9Y58} -remove {SLICE_X56Y241:SLICE_X75Y298
BUFCE_LEAF_X304Y16:BUFCE_LEAF_X407Y19
BUFCE_ROW_FSR_X76Y4:BUFCE_ROW_FSR_X102Y4
DSP48E2_X12Y98:DSP48E2_X14Y117
HARD_SYNC_X14Y8:HARD_SYNC_X19Y9 RAMB18_X7Y98:RAMB18_X9Y117
RAMB36_X7Y49:RAMB36_X9Y58} -locs keep_all
```

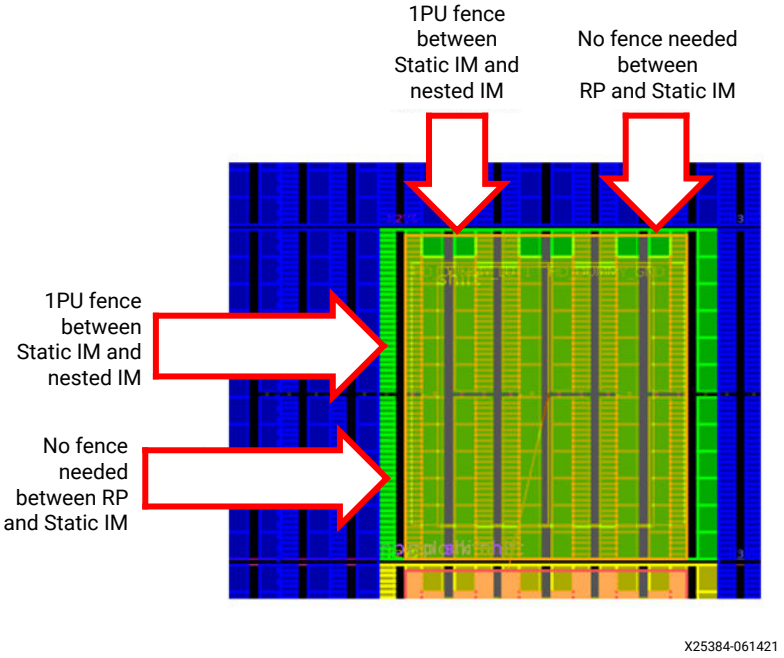
After changing the Pblock boundaries, the Pblock loses its coloring. You can rerun the command to color the Pblocks. After creating the fence, the Pblock boundary will be as shown in the following figure.



d. Similarly, create an 1PU fence between the static IM Pblock and the nested shift Pblock. Drag the nested Pblock such that the fence is inside of the RP. For the shift block, create a horizontal fence and a vertical fence. You can drag the Pblock boundaries to create the fence or use the following `resize_pblock` command.

```
resize_pblock pblock_shift -add {SLICE_X58Y300:SLICE_X75Y358
BUFCE_LEAF_X312Y20:BUFCE_LEAF_X407Y23
BUFCE_ROW_FSR_X77Y5:BUFCE_ROW_FSR_X102Y5
DSP48E2_X12Y120:DSP48E2_X14Y141
HARD_SYNC_X14Y10:HARD_SYNC_X19Y11 RAMB18_X7Y120:RAMB18_X9Y141
RAMB36_X7Y60:RAMB36_X9Y70} -remove {SLICE_X56Y300:SLICE_X75Y359
BUFCE_LEAF_X304Y20:BUFCE_LEAF_X407Y23
BUFCE_ROW_FSR_X76Y5:BUFCE_ROW_FSR_X102Y5
DSP48E2_X12Y120:DSP48E2_X14Y143
HARD_SYNC_X14Y10:HARD_SYNC_X19Y11 RAMB18_X7Y120:RAMB18_X9Y143
RAMB36_X7Y60:RAMB36_X9Y71} -locs keep_all
```

The following figure shows the Pblock boundaries after creating the fence.



e. Re-run the VIV DRCs as mentioned in Step 6.

***Note*:** Advisories are reported as *Violations* in the report summary of the DRCs. This is common to all the DRCs and has nothing to do with the VIV DRCs. This is an issue with the way the error counts are reported and print the advisories without incrementing the error count.

***Note*:** There are no warnings or errors in the DRC report.

8. Save the constraints by saving the design.

Step 6: Set HD.ISOLATED_EXEMPT for Multi-Region/Global Clock Nets

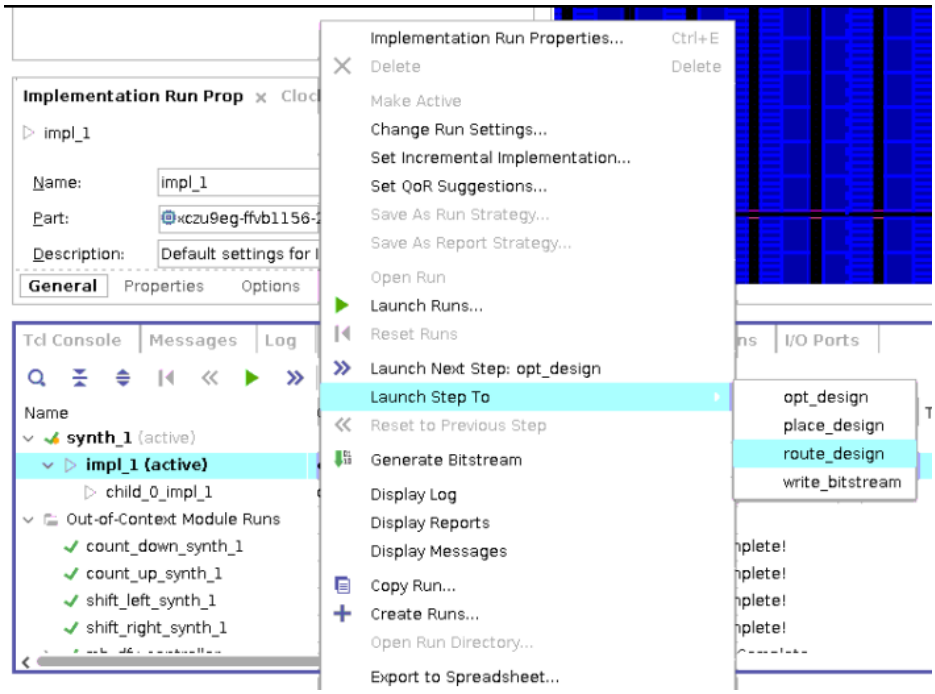
There are a couple of clock nets which are going to multiple isolated modules. Ensure to make them exempt for the IDF rules by applying HD.ISOLATED_EXEMPT on those driver cells. Run the following command from the Tcl console:

```
source ./sources/iso_exempt.xdc
```

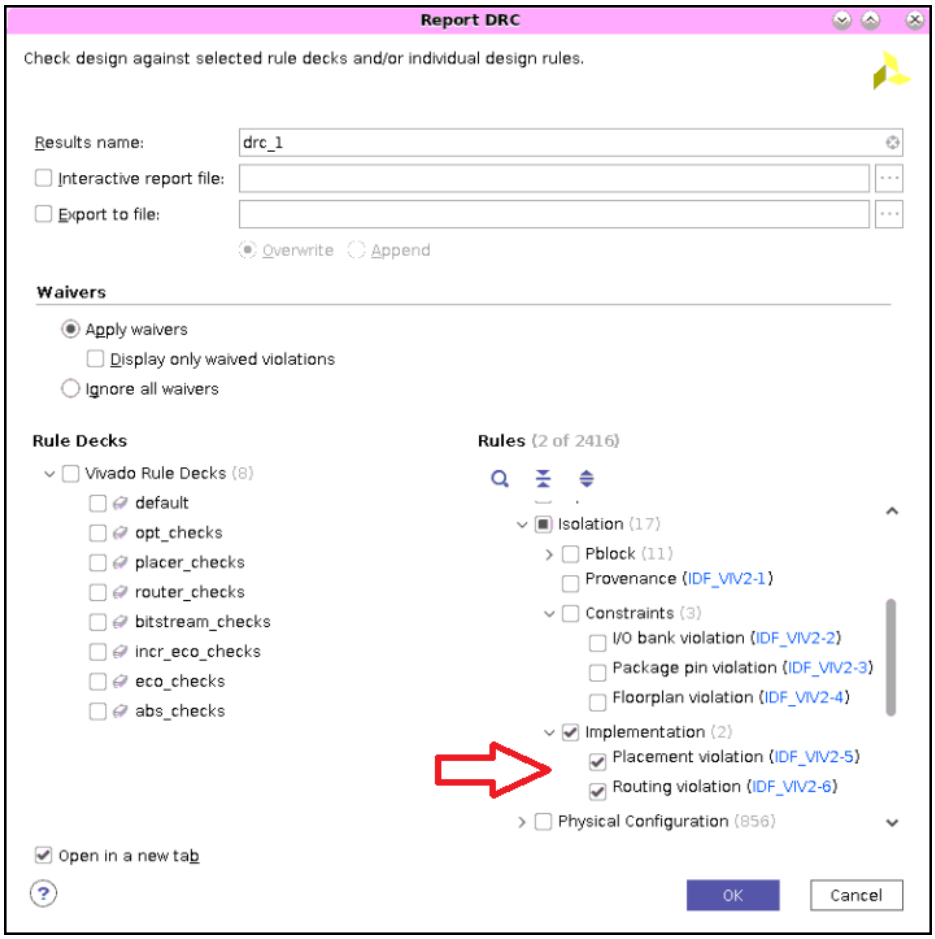
Save the constraints by saving the design.

Step 7: Implement the First Configuration

1. Launch the implementation of the first configuration. Right-click **impl_1** from the Design Runs tab, and then click **Launch Step To → route_design**.



2. Click **Open Implemented Design** after implementation is complete. You can close the Synthesized design.
3. Run the VIV DRCs for implementation. Open the Report DRC window and select **IDF_VIV2-5** and **IDF_VIV2-6** from the Isolation group. Click **OK** to run these DRCs.



You can see there are no warnings or errors in the DRC report. Advisories are reported as *Violations* in the report summary of the DRCs. This is common to all the DRCs and has nothing to do with the VIV DRCs. This is an issue with the way the error counts are reported and print the advisories without incrementing the error count.

4. Inspect the files in the `impl_1` directory under the `Project Runs` directory. You will see `mb_dfx_controller_wrapper_routed_bb.dcp`, which is a *blackbox* dcp, and RM dcps `mb_dfx_controller_i_rp1_rp1rm1_inst_0_routed.dcp` and `mb_dfx_controller_i_rp2_rp2rm1_inst_0_routed.dcp`.
5. Close the implemented design. You can now proceed to the second configuration.

Step 8: Implement the Second Configuration

`Child_0_impl_1` is the run for the second configuration (Config2). You should use a `pre-opt Tcl Hook` script for floorplanning the second configuration. Run the following commands to floorplan and implement the second configuration. Before proceeding with the implementation of the second configuration, ensure that the RMs of the second configuration has HD.ISOLATED property enabled.

1. Add the `child_0_pre_opt_tcl.tcl` script to the project by running the following command from the Tcl console:

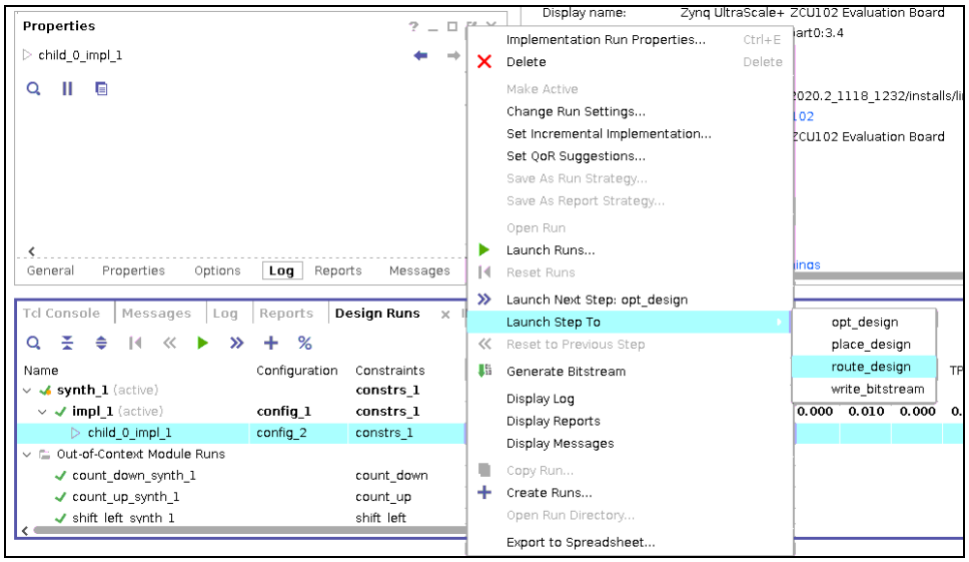
```
add_files -fileset utils_1 -norecurse ./sources/child_0_pre_opt_tcl.tcl
```

   This file has commands to load XDC constraints, which creates RM Pblocks for the second configuration.

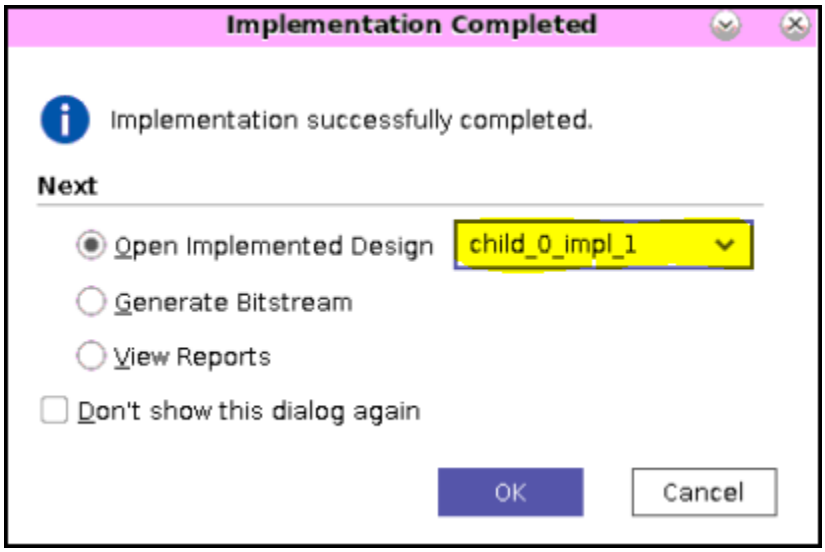
2. Add the `child_0_pre_opt_tcl.tcl` script as pre-opt Tcl Hook script for the run child_0_impl_1 by running the following command from the Tcl console:

```
set_property STEPS.OPT_DESIGN.TCL.PRE [ get_files
child_0_pre_opt_tcl.tcl -of [get_fileset utils_1] ] [get_runs
child_0_impl_1]
```

3. Launch the implementation of the second configuration. Right-click **child_0_impl_1** from the Design Runs tab, and then select **Launch Step To → route_design**.



4. After the child implementation is complete, click **Open Implemented Design**, select **child_0_impl_1** from the drop-down menu, and click **OK** to open the implemented design of Config2.



5. Run the VIV DRCs for implementation. Open the Report DRC window and select **IDF_VIV2-5** and select **IDF_VIV2-6** from the Isolation group. Click **OK** to run these DRCs.

   ***Note*:** There are no warnings or errors from the DRC report.

6. Close the implemented design.
7. Inspect the files in the `child_0_impl_1` directory under the `Project Runs` directory. You can see `mb_dfx_controller_wrapper_routed.dcp`, which is a full design dcp for the Config2 and RM dcps as follows:

```
mb_dfx_controller_i_rp1_rp1rm2_inst_0_routed.dcp,
mb_dfx_controller_i_rp2_rp2rm2_inst_0_routed.dcp
```

Step 9: Running pr_verify

Run the `pr_verify` command from the Tcl console:

```
pr_verif ./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/impl_1/
mb_dfx_controller_wrapper_routed.dcp ./project_idf_dfx_zcu102/
project_idf_dfx_zcu102.runs/child_0_impl_1/
mb_dfx_controller_wrapper_routed.dcp
```

You can see there are no errors with the following commands reported in the Tcl console.

```
INFO: [Vivado 12-3253] PR_VERIFY: check points
./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/impl_1/
mb_dfx_controller_wrapper_routed.dcp
and
./project_idf_dfx_zcu102/project_idf_dfx_zcu102.runs/child_0_impl_1/
mb_dfx_controller_wrapper_routed.dcp are compatible
```

Step 10: Generating Bitstreams

You will now create the bitstreams, including the full bitstream for Config1 and partial bitstreams for both Config1 and Config2.

Run the following command from the Tcl console to create the bitstreams:

```
source ./create_all_bitstreams_zcu102.tcl
```

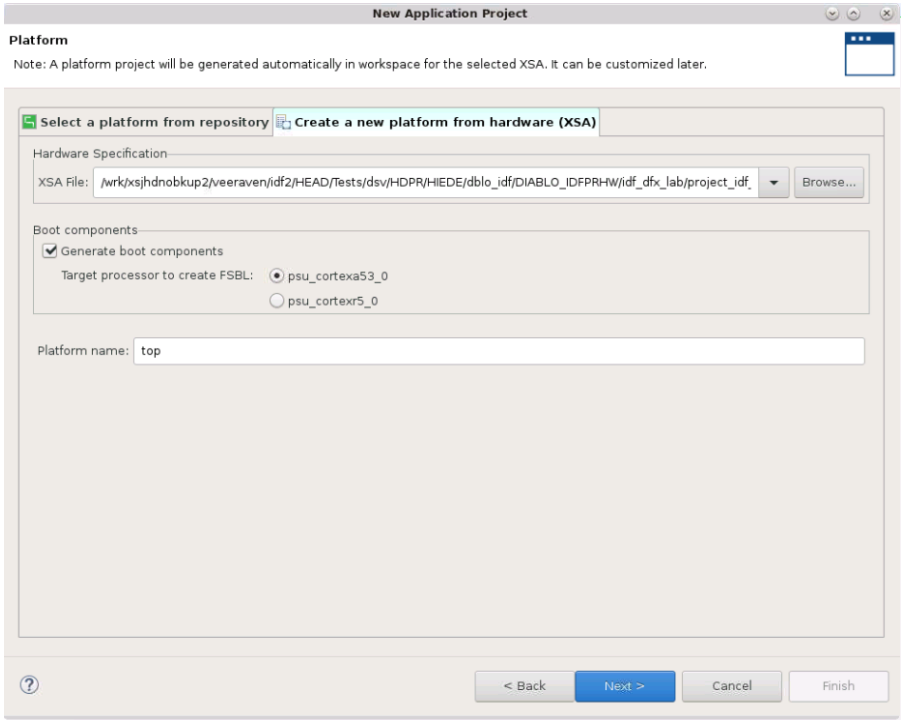
This creates a new directory called `Bitstreams` and generates the bitstreams under it.

Step 11: Creating Vitis Application to Load Partial Bitstreams

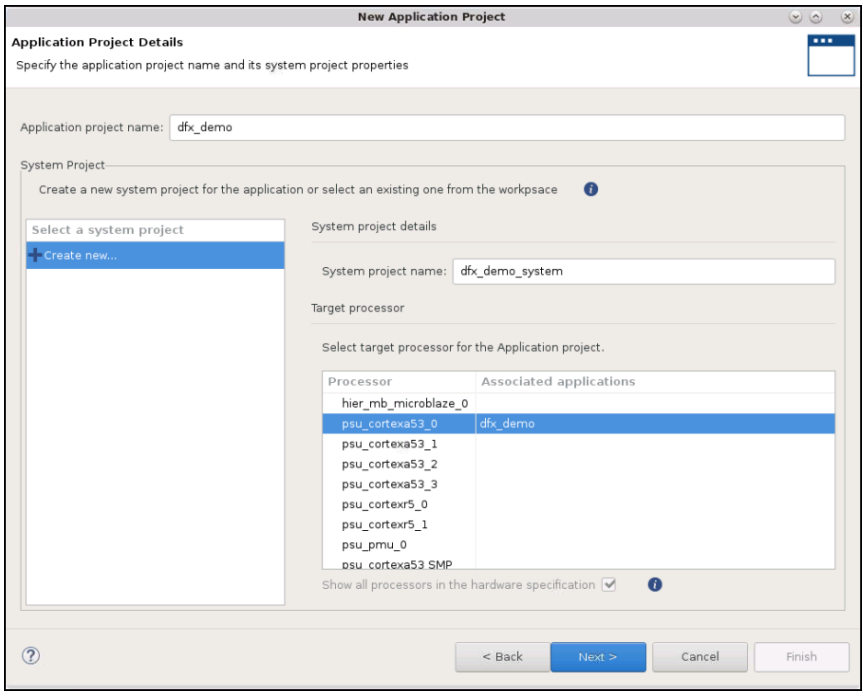
For this particular example design, partial bitstreams are placed into PS-DDR memory and loaded into the device using a baremetal application, running on the Cortex-A53. This application takes inputs from the user through a `UART` to select the RM to load. The application uses the xilfpga library to load the partial bitstreams through the PCAP. More information on the xilfpga library can be found in the*Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137).

Perform the following steps to create the software application.

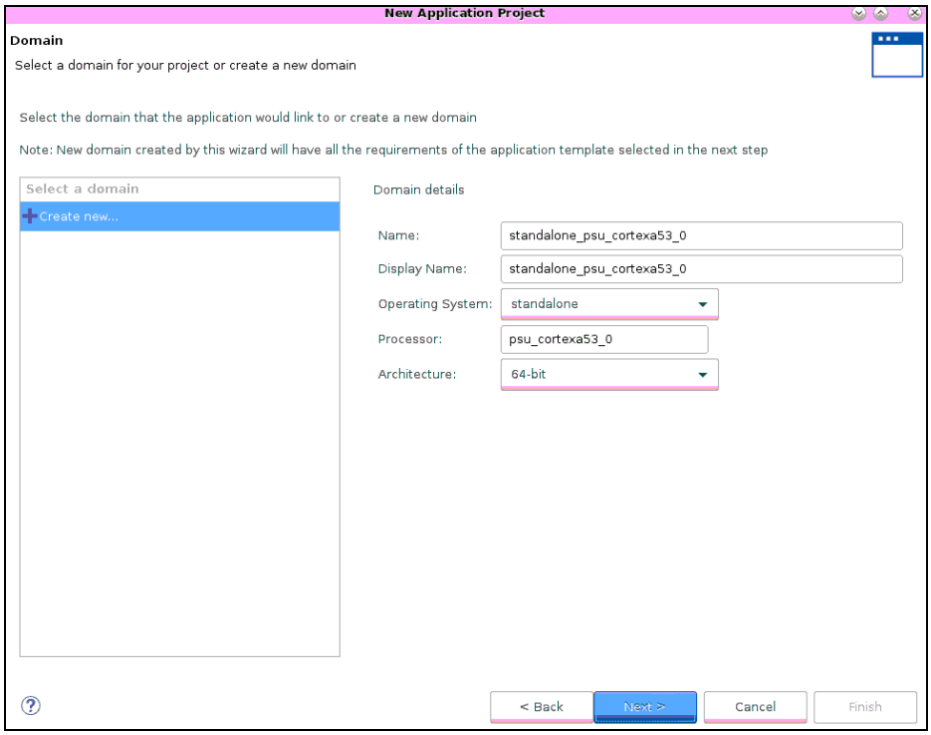
1. In Vivado, select **File** → **Export** → **Export Hardware**.
2. Click **Next** in the Export hardware platform window.
3. Leave the output set to Pre-synthesis on the output window, and click **Next**.
4. Change the XSA file name to `top` and the Export to as a location of the project directory. Click **Next** and then select **Finish** to build a design image for the Vitis integrated design environment. This creates the `top.xsa` file under the `project_idf_dfx_zcu102` directory.
5. Select **Tools** → **Launch Vitis IDE**. The Eclipse launcher dialog box appears in Vivado.
6. Ensure the workspace maps to the current project directory (`project_idf_dfx_zcu102`) in the Eclipse launcher, and then click **Launch** to open the main Vitis integrated design environment GUI.
7. Select **File** → **New** → **Application Project** in the Vitis IDE.
8. Click **Next** on the welcome screen.
9. Select the **Create a new platform from hardware (XSA)** tab, and browse to select **top.xsa** to import the file that was exported from Vivado.
10. Keep the platform name as `top`, select **Generate boot components**, and select **psu_cortexa53_0**.



11. Click **Next**.
12. Name the project `dfx_demo` in the Application Project window, and select **psu_cortexa53_0** as the target processor. Click **Next**.

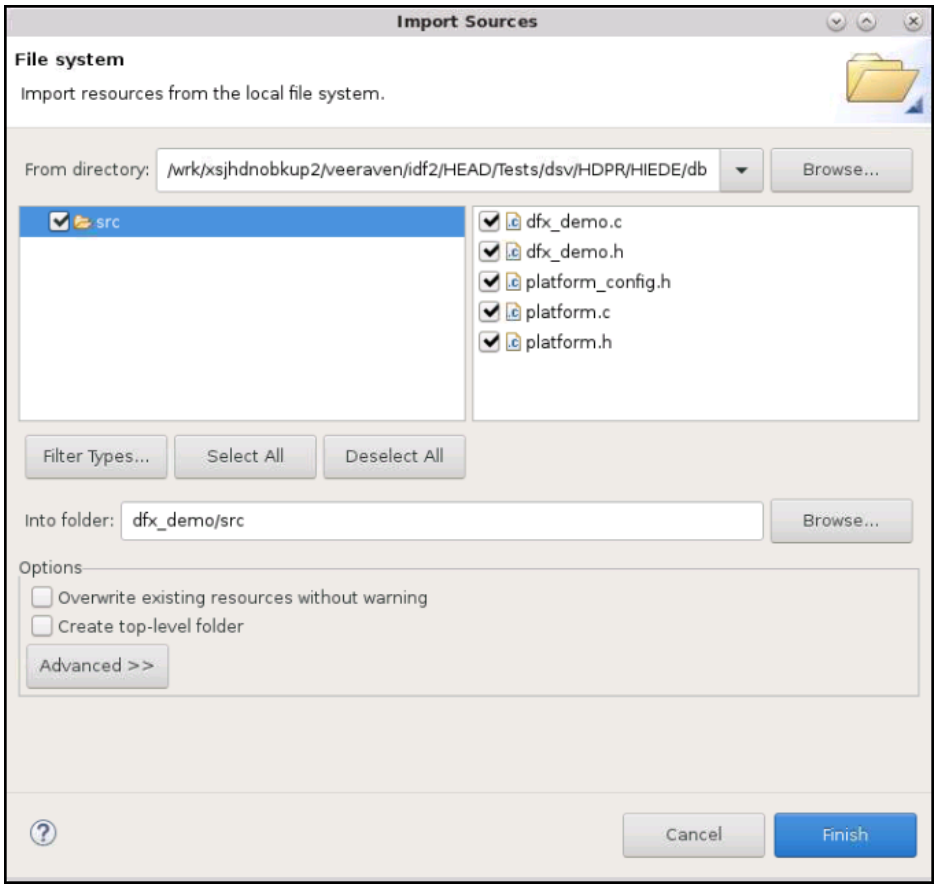


13. Keep default values for Domain, and click **Next**.

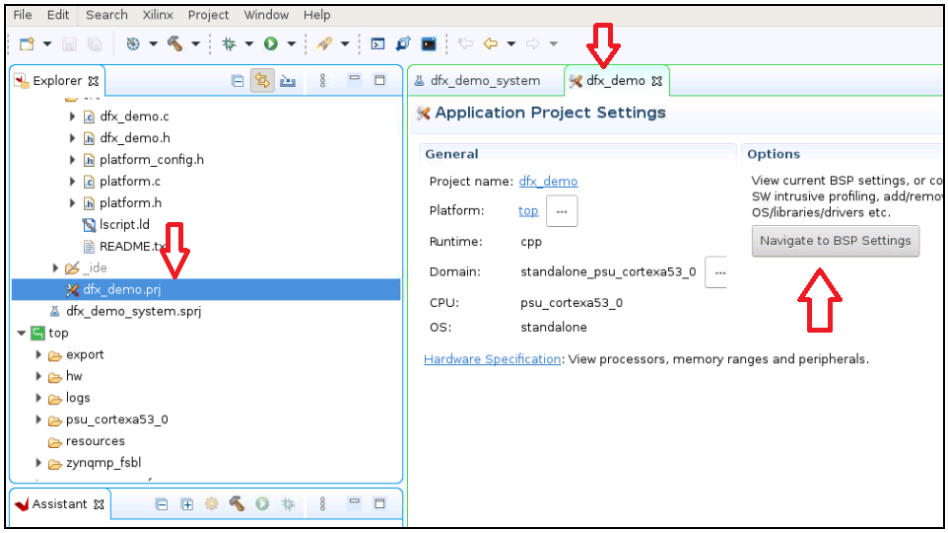


14. Select **Empty Application(C)**, and click **Finish**.
15. Expand **dfx_demo** in the Project Explorer window. Right-click **src** and select **Import Sources**. Browse to the `sources/dfx_demo/src` directory, and click **Open**. Finally, check all **.c** and **.h** sources in that folder, and click **Finish**.

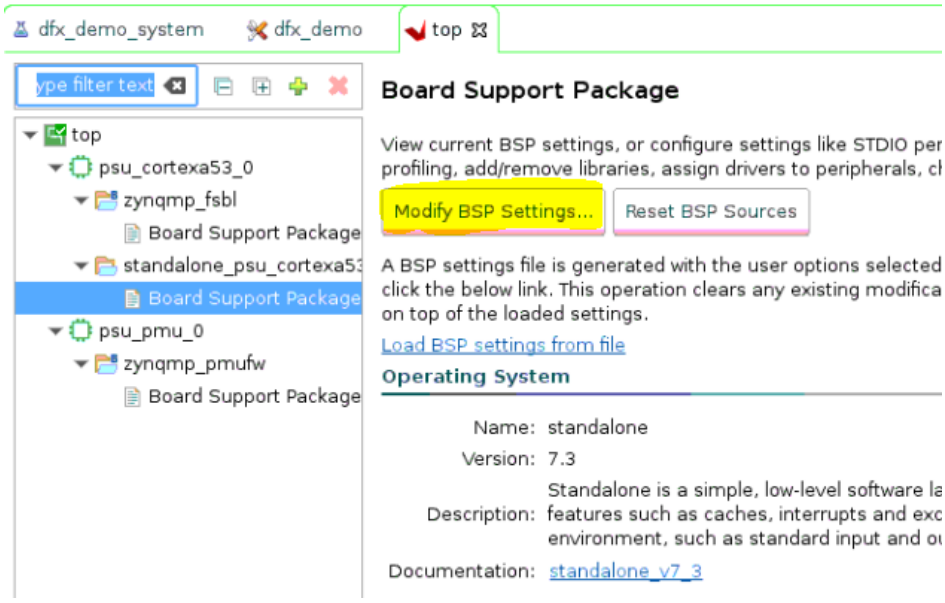
> **IMPORTANT!** *If the actual bitstream file size does not match with the size of bit file in `dfx_demo.h`, then loading the partial bitlstream on hardware might fail. Check the size of partial bitstream files (.bin) and update the new bitstream file size in `dfx_demo.h`. Update the address range in bif file accordingly.*



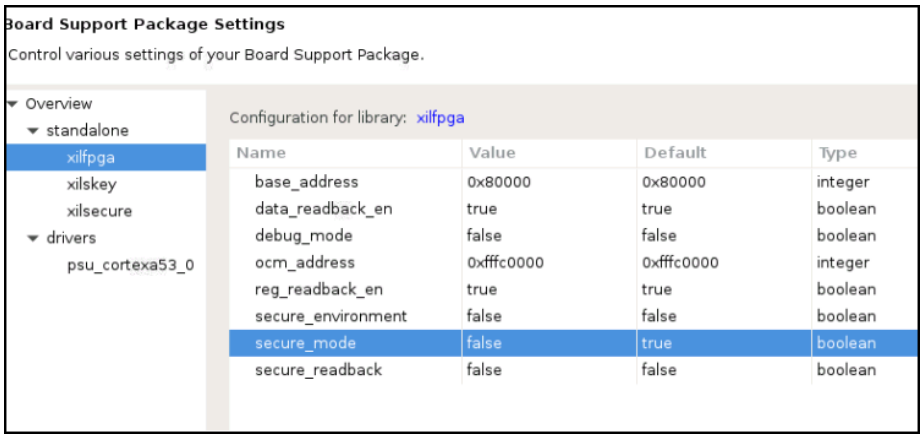
16. The application uses the xilfpga library to load the partial bitstreams via PCAP. More information on the xilfpga library can be found in the *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137). You must enable the xilfpga and dependent libraries in the BSP settings. Select **dfx_demo.prj** and click **Navigate to BSP settings**.



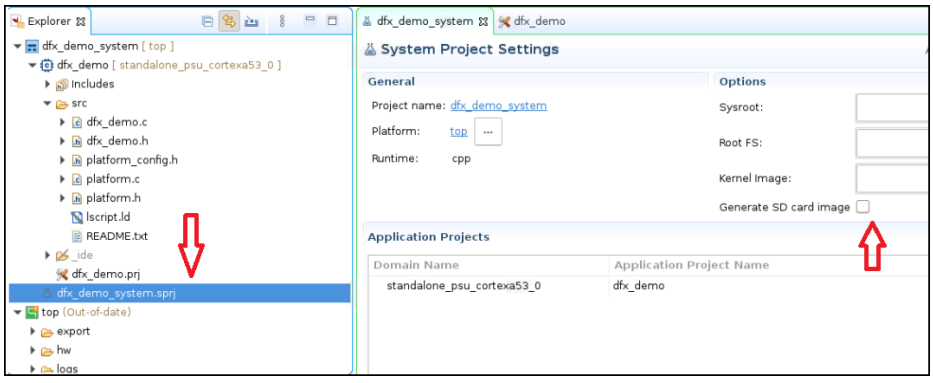
17. Select **Board Support Package** under standalone_psu_cortexa53_0 and then click **Modify BSP Settings**.



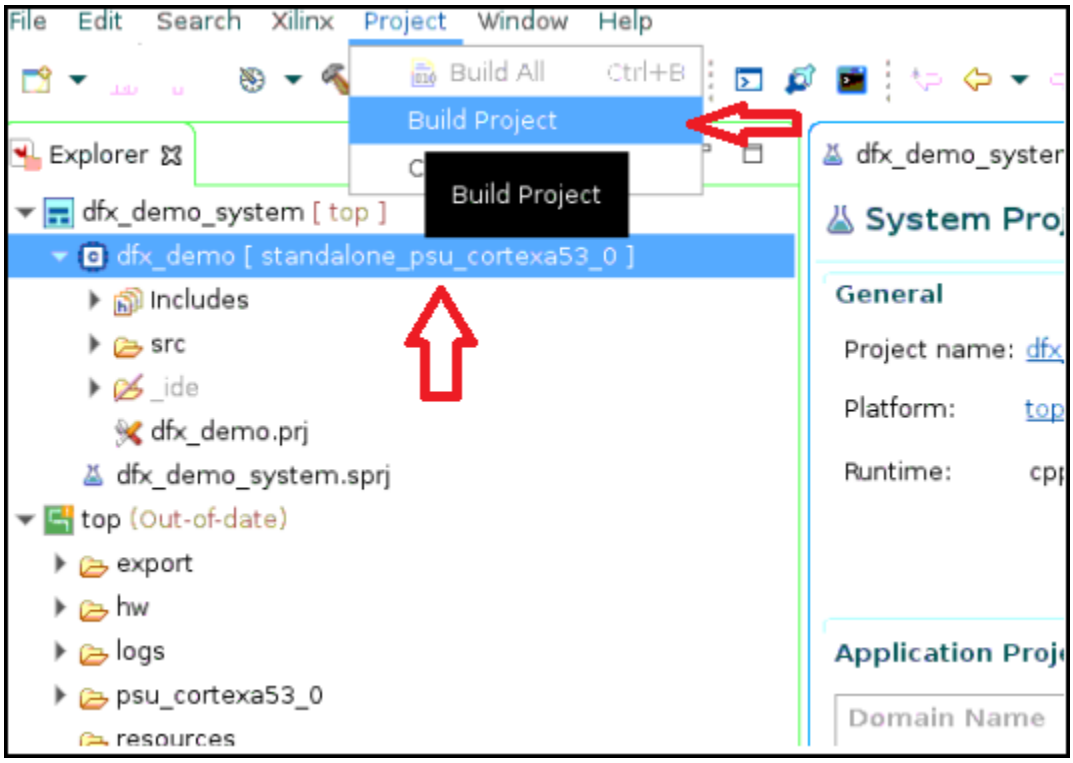
18. Select **xilfpga**, **xilsecure**, and **xilskey** from the supported libraries list.
19. Select **Overview** → **standalone**, then select **xilfpga** to open configuration for the library. Change the `secure_mode` value to **false**.



20. Click **OK**.
21. Open the dfx_demo_system settings and disable **Generate SD card image**.

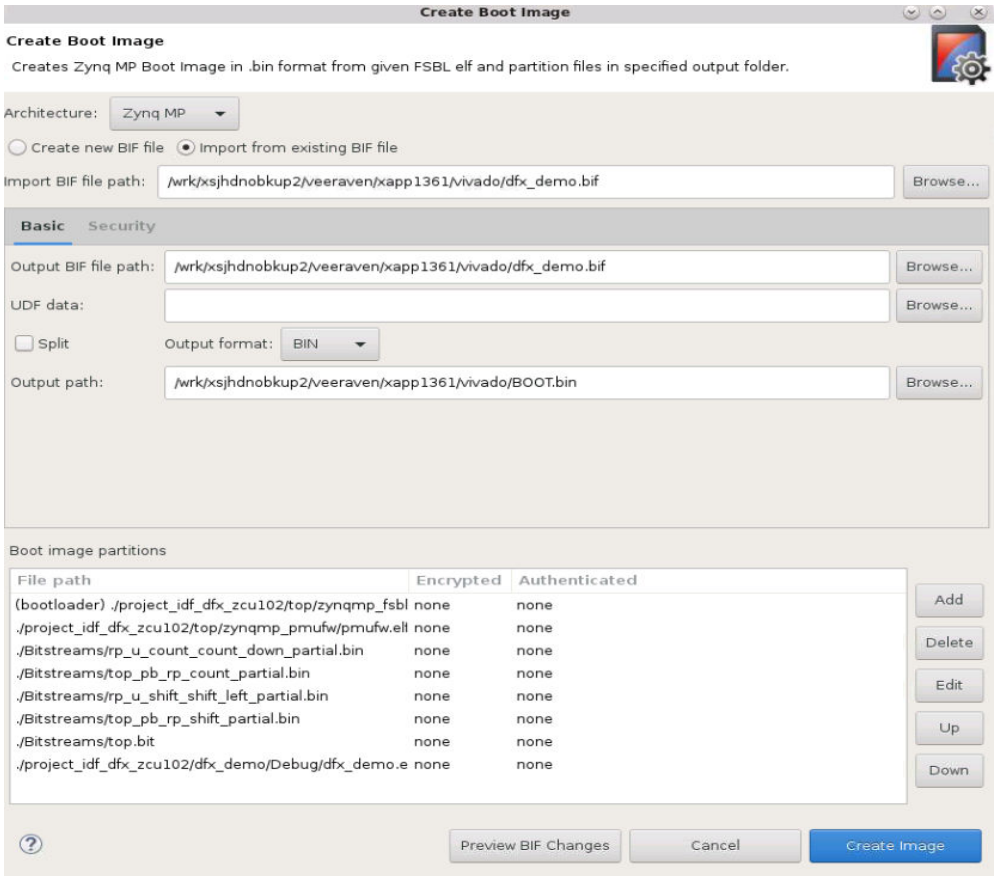


22. Build the `dfx_demo` project, and generate `dfx_demo.elf`. To build the project, select the project from the Explorer drop-down list, and click **Project** → **Build Project**.



23. Create a boot image, which loads partial bitstreams into the PS-DDR and initializes PL with the Config1 full bitstream. Click **Xilinx → Create Boot Image → Zynq and Zynq Ultrascale**. Select **Import from existing BIF file option**, browse to the directory where you placed the application note files, and then select and open **dfx_demo.bif**.

> **IMPORTANT!** *Check the load address of the partial bitstreams. The address range should be more than the actual bitstream file size.*



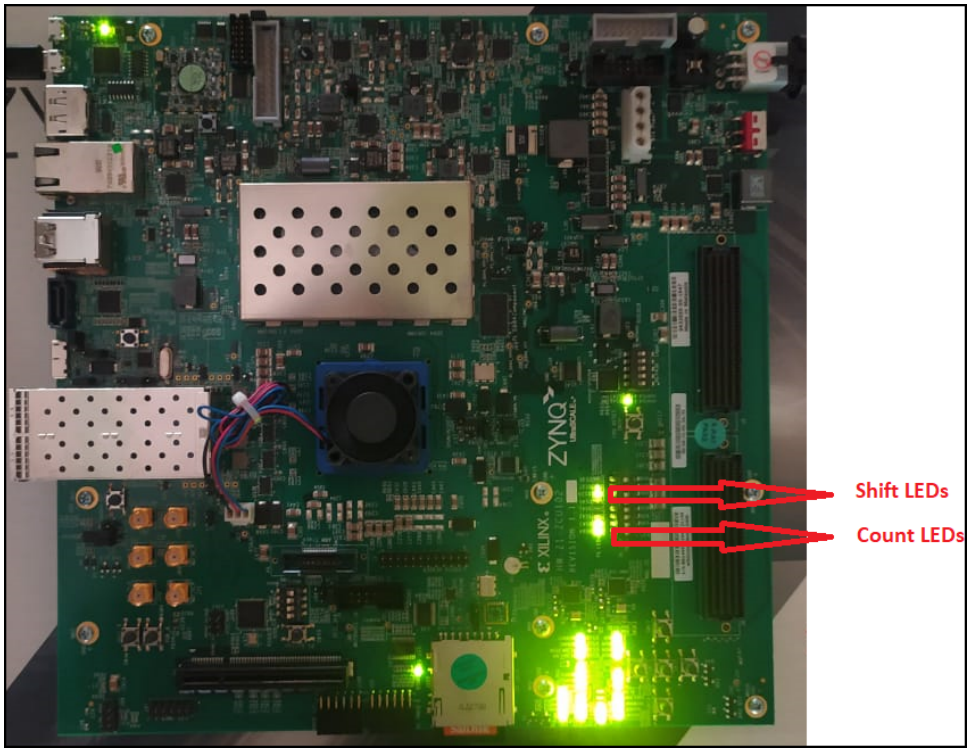
X26967-081622

24. Click **Create Image** and then click **OK** to overwrite the bif file. This creates the BOOT.bin.

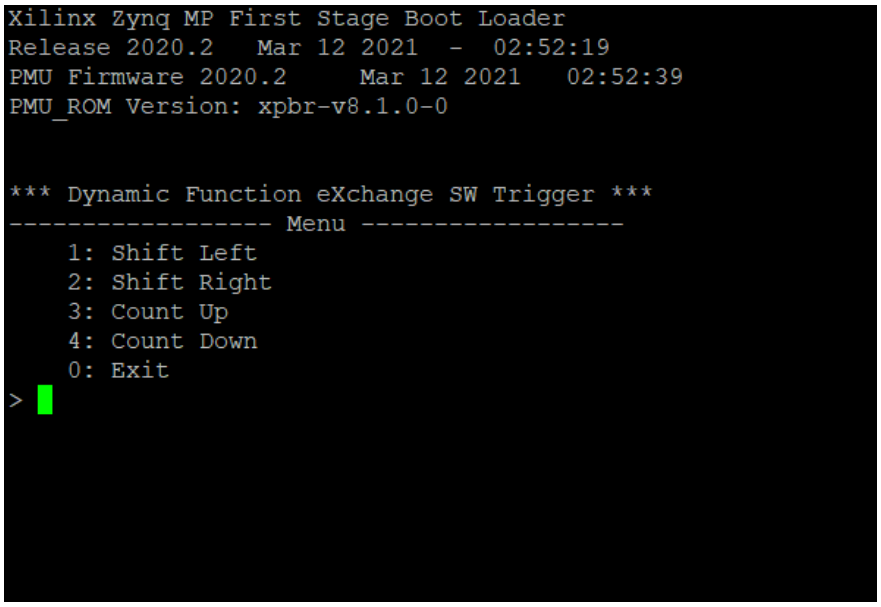
Step 12: Running Demo on ZCU102 Board

1. Copy the BOOT.bin to the ZCU102 SD card, that was created in the previous step.
2. Connect the UART cable and power cable to the board. Set the **COM port** to an appropriate value for your computer. Set the **Baud Rate** to `115200`. Open a UART terminal to communicate with the software running on the PS.
3. Configure the board to boot in SD-Boot mode by setting switch SW6 to **1-ON**, **2-OFF**, **3-OFF**, and **4-OFF** as shown in the following figure.

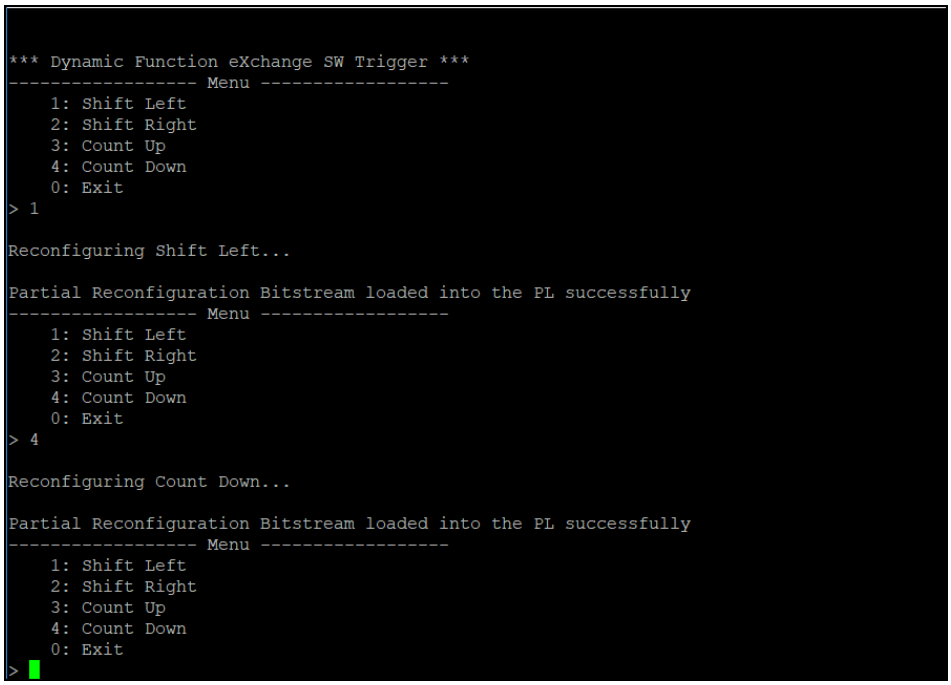
4. Load the SD card into the ZCU102 board and power up the board. You can see the boot image loaded successfully and the LEDs blink.



In the UART terminal, you can see the software menu to load the partial bitstreams as shown in the following image..



5. Enter your inputs to load different partial bitstreams. You can see the LEDs pattern change on the board after each reconfiguration. The following image shows the UART terminal after loading `shift_left` and `count_down` partials.



Reference Design

This reference design is created using the Vivado Design Suite 2021.1 and the Vitis Unified Software Development Platform 2021.1. This design is verified against 2021.1, 2021.2, and 2022.1 versions. Download the reference design files for this application note from the AMD website. The reference design files contains RTL source code for the design, and the C files for the Vitis application.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Satya Pitaka
Target devices	Zynq UltraScale+ Devices
Source code provided?	Yes
Source code format (if provided)	C, Tcl, HDL, Verilog
Design uses code or IP from existing reference design, application note, third party or Vivado software? If yes, list.	Yes, design reuse (Lab 7) from *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947)
Hardware Verification	
Hardware verified?	Yes
Platform used for verification	ZCU102 Evaluation Board

Conclusion

This application note provides a step-by-step example to combine Isolation Design Flow (IDF) and Dynamic Function eXchange (DFX) on the Zynq UltraScale+ MPSoC devices. This lab highlights the following:

- Enabling IDF on all configurations
- Floorplanning different configurations
- Running VIV DRCs and fixing floorplan violations
- Implementing and generating Bitstreams for different configurations
- Creating a SW application to load partial Bitstreams via PCAP from PS by using the Vitis unified software platform
- Running the demo on ZCU102 and triggering partial reconfiguration using UART console

References

These documents provide supplemental material useful with this guide:

1. *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335)
2. *Isolation Design Example for Zynq Ultrascale+ MPSoC Application Note* (XAPP1336)
3. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
4. *Vivado Design Suite User Guide: Hierarchical Design* (UG905)
5. *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909)
6. *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947)
7. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)

8. Isolation Design Flow available on the AMD website.
9. *Vivado Isolation Verifier User Guide* (UG1291)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/23/2024 Version 1.2	
General updates	Updated links. Updated figure titles.
Step 2: Create the Vivado Design and Run Through Synthesis	Clarified and added steps. Updated graphics.
Step 3: Enabling IDF	Updated figure reference. Clarified steps. Updated code blocks.
Step 5: Floorplanning First Configuration	Reordered steps.
Step 7: Implement the First Configuration	Changed mb_dfx_controller_wrapper_routed_bb to mb_dfx_controller_wrapper_routed_bb.dcp.
Step 11: Creating Vitis Application to Load Partial Bitstreams	Added directory name project_idf_dfx_zcu102.
08/31/2022 Version 1.1	
Updated throughout document	Updated instructions for new Block Design Container (BDC) flow; updated software version start range for Vivado and Vitis to 2021.1 for IDF+DFX with BDC, updated software terms to match latest version.
IDF + DFX	Updated Vivado code parameter.
Implementing the Example Design	Updated steps and figures to provide latest BDC instructions in sub-section Step 2: Create the Vivado Synthesis Design and Run Through Synthesis. Added new figure to Step 2(2); Moved figure from Step 2(7b) to Step 2(7c) and added new image to Step 2(7b); Replaced figures in Steps 3(1) and 3(2a); Replaced figure in Steps 5(3) and 5(4); Replaced figure in Step 11(23).
06/30/2021 Version 1.0	
Initial release.	N/A

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

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

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

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

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright