Isolation Design Example for the Zynq UltraScale+ MPSoC

XAPP1336 (v2.0) April 12, 2021

Summary

This lab application note describes the creation and implementation of a single-chip general purpose 2 channel compute system using the Low Power Domain as one channel and a triple modular redundant MicroBlaze™ processor in the programmable logic (PL) both interconnected by a Mailbox also in the PL.

Complete step-by-step instructions are given for the entire process, explaining the use of the Isolation Design Flow (IDF). This document explains how to implement isolated functions in a single Xilinx® Zynq® UltraScale+™ MPSoC device for the example solution.

With this application note, designers can develop a fail-safe single chip solution using the Xilinx IDF that meets fail-safe and physical security requirements for an example high-assurance application.

The rules for IDF defined in this application note follow the rules defined in the *Isolation Design Flow for Zynq Ultrascale+ Application Note* (XAPP1335).

Download the isolation design example reference files from the Xilinx website. For detailed information about the design files, see Reference Design.

For detailed information about the Isolation Design Flow (IDF), refer to the IDF website.

Introduction

The Xilinx® Isolation Design Flow (IDF) is the design methodology that allows for Information Assurance, Functional Safety implementations, or any other application requiring module both physical and logical isolation. This methodology is backed by significant schematic analysis and software verification—Vivado® Isolation Verifier (VIV)—to ensure elimination of single points of failure. SCC is one specific application of IDF allowing the implementation of a multichip cryptography system in a single FPGA or SoC.

Note: This lab was verified using Vivado Design Suite 2020.1. Other versions might have slightly different screen images.

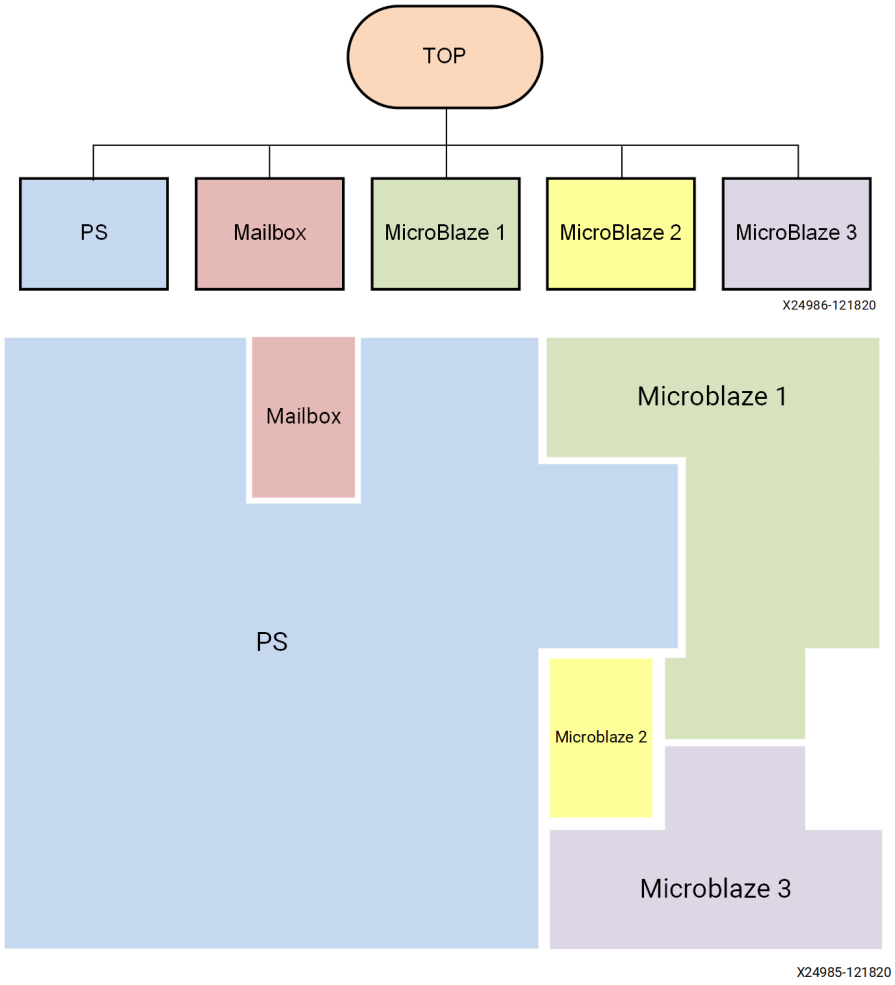
Lab Design Overview

The Zynq UltraScale+ Isolation Design Flow (IDF) rules are outlined in the *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335).

This lab gives details on how functions are to be isolated, specific differences between a normal partition flow and an IDF partition flow, information on IDF-specific hardware description language (HDL) code mnemonics, and trusted routing rules.

To illustrate the IDF and its capabilities, this design implements an isolated, triple modular redundant MicroBlaze processing subsystem, which communicates to the Arm® Cortex®-R5F core in the Low Power Domain. Refer to the *Triple Modular Redundancy (TMR) LogiCORE IP Product Guide* (PG268) for TMR guidance. The following figure shows a hierarchical diagram of the sub-blocks used in the implementation of this design.

Figure 1: **Design Hierarchy Block Diagram and Corresponding Floorplan**



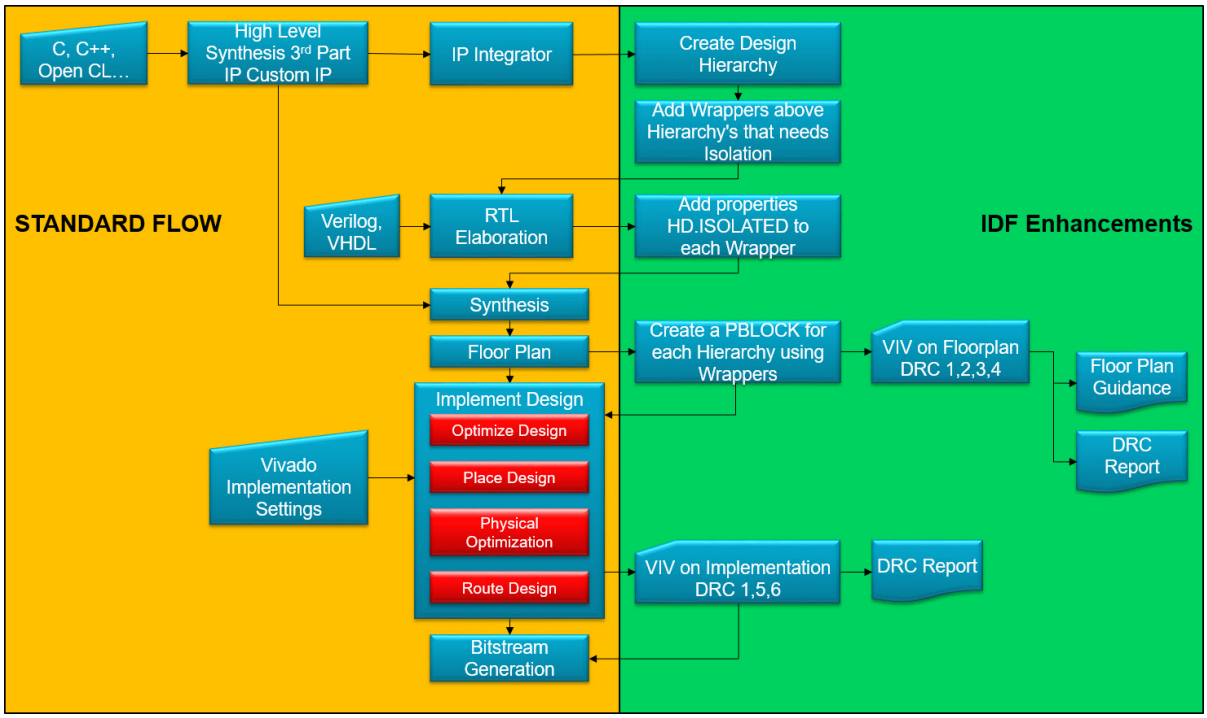
IMPORTANT! *For functional safety, to mitigate the probability of Common Cause Failures, isolate each MicroBlaze CPU with each other and the processing system (PS).*

Isolation Design Flow

The Xilinx® Isolation design flow (IDF) adds the standard design flow by mapping the design hierarchy into isolated containers called Pblocks. The reason is to stop logical optimization between Pblocks and to separate concerns (isolation from a single fault).

The standard flow is shown on the left with the IDF enhancements on the right side with reference to the following figure. The lab is divided into eight phases for reference purposes, and each phase is highlighted in this application note.

Figure 2: **Isolation Design Flow Enhancements**



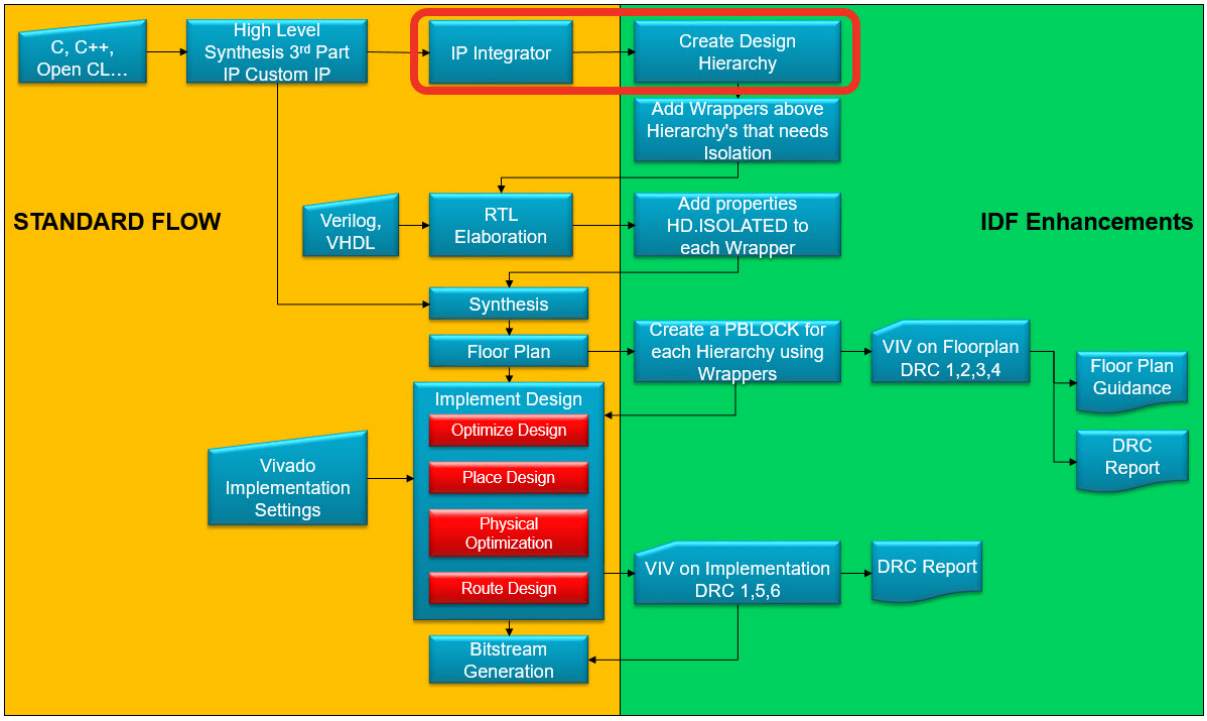
The lab is divided into eight phases. You may run each phase as explained or run Tcl script provided for that phase. Each phase needs to be completed in numerical order. You may save the design at any point and continue to run the lab at a later date.

Along with the eight phase files, there are two Tcl files `lab_workspace_setup.tcl` and `lab_server_option.tcl`. If you are running the lab on a server you can speed up the lab's execution by setting the jobs counter to a higher value (16) in `lab_server_option.tcl`, This variable is temporary, so, if you saved your lab, it needs to be sourced again at the very beginning of your next session. `lab_workspace_setup.tcl` is used to set the variables needed by eight phase files. This should be run before starting the project. If you are using step-by-step mode for some phases and running script for others, then ensure you run this script before running phase script.

Phase 1

You will create the project using IP Integrator in the first phase. A single-instance of a MicroBlaze™ processor system and the processing system (PS) are instantiated. The hierarchy is then created for the MicroBlaze processor, followed by the instantiation of the TMR Manager, which is used to create the triple mode redundancy (TMR), as shown in the following figure.

Figure 3: **Phase1 - Create Design and Generate Block Diagram**

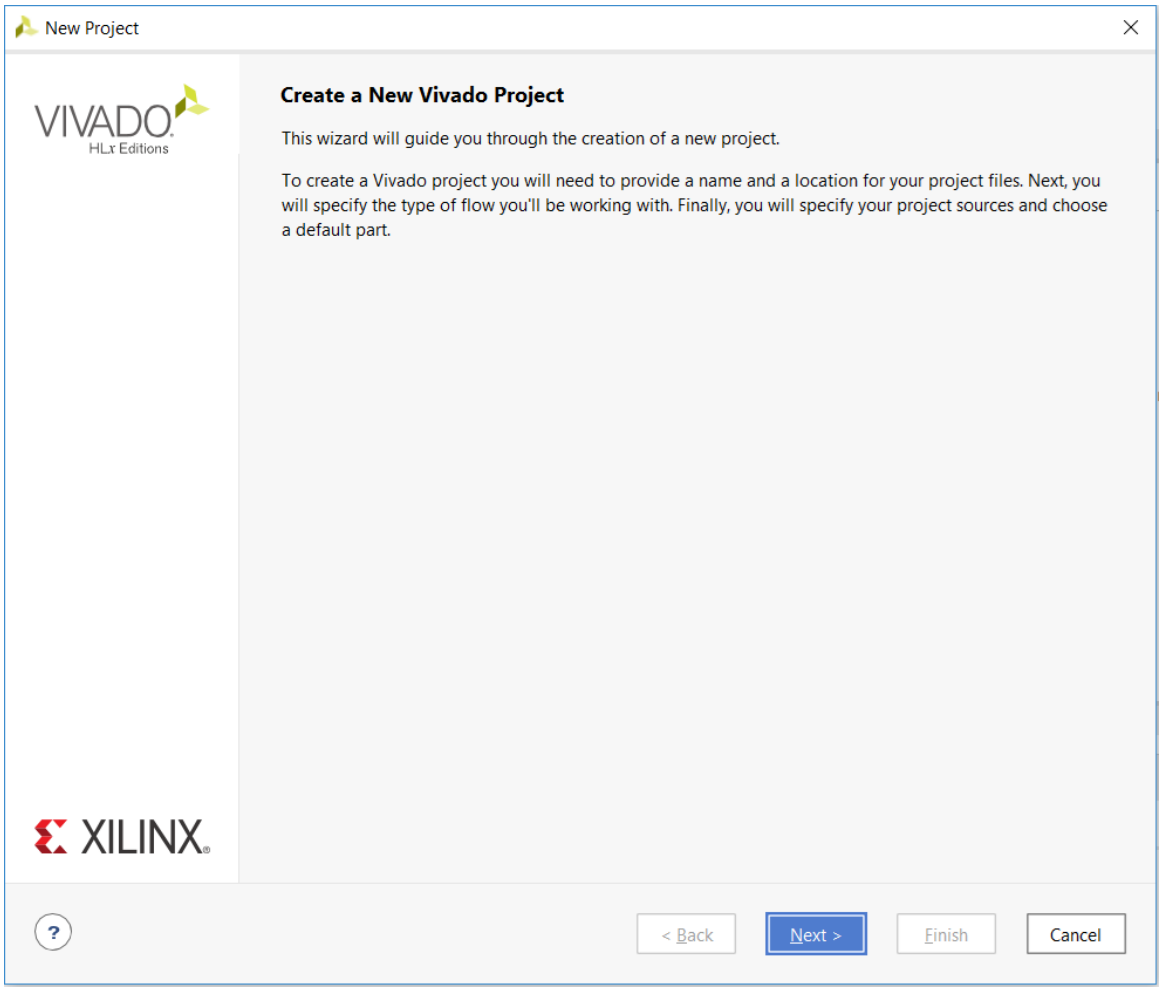


1. Create a target design folder. For this example, use `ZUP_LAB`.
2. Copy the eight lab scripts (`lab_phase1.tcl` through `lab_phase8.tcl`) into the target design folder.
3. Start Vivado® Design Suite version 2018.3 or a newer version.
4. In the Tcl Console, point the console input to the design folder.
5. Perform the following steps to generate and build the project. Alternatively, you can run phase1 Tcl script from the Tcl Console, by typing `source ./lab_phase1.tcl`.

Vivado Project Creation

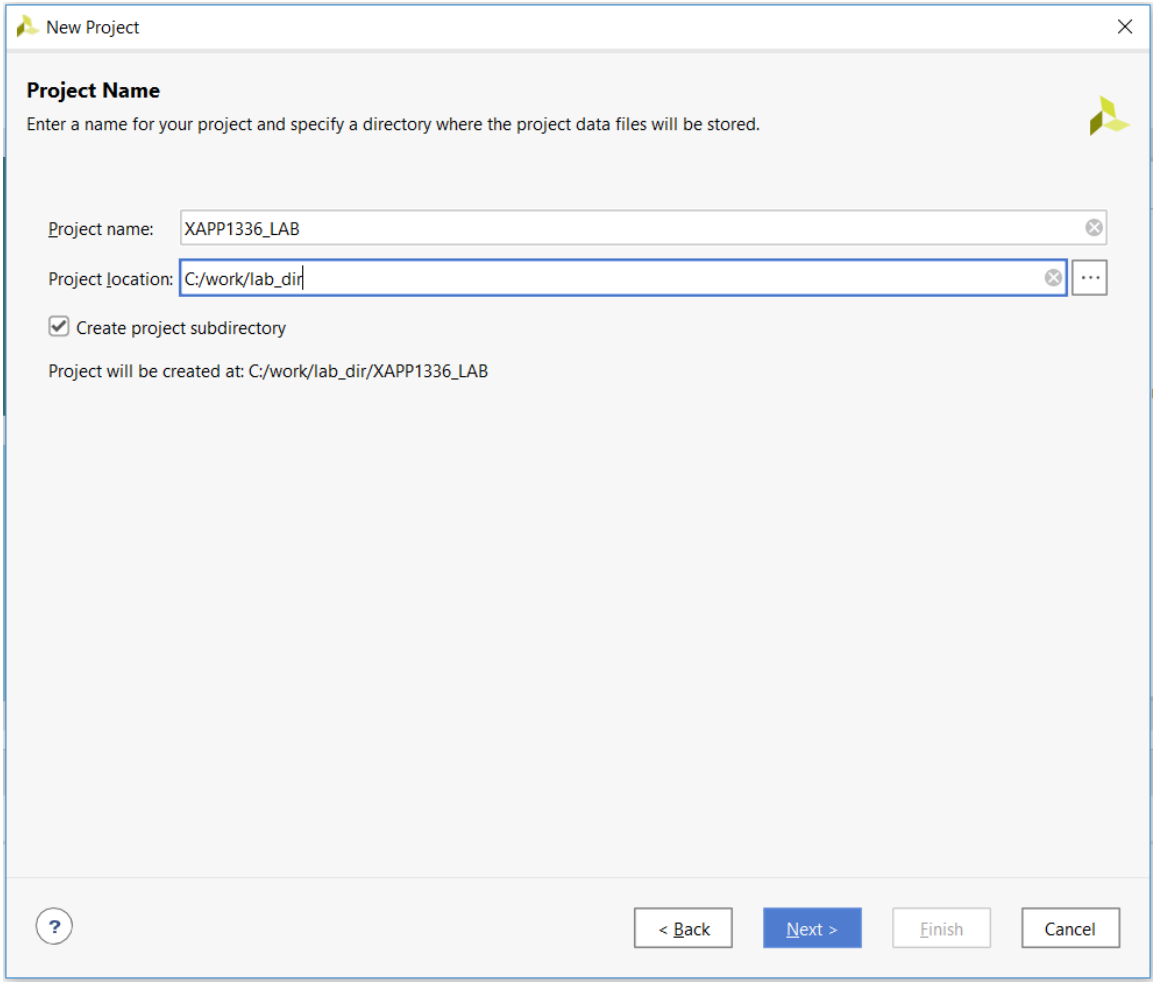
1. Set up a new Vivado project:
   - From the Quick Start menu, click **Create New Project**, and wait for the **New Project - Create a New Vivado Project** window to pop up. You may refer to the following figure. Select **Next**.

Figure 4: **Vivado New Project Window**



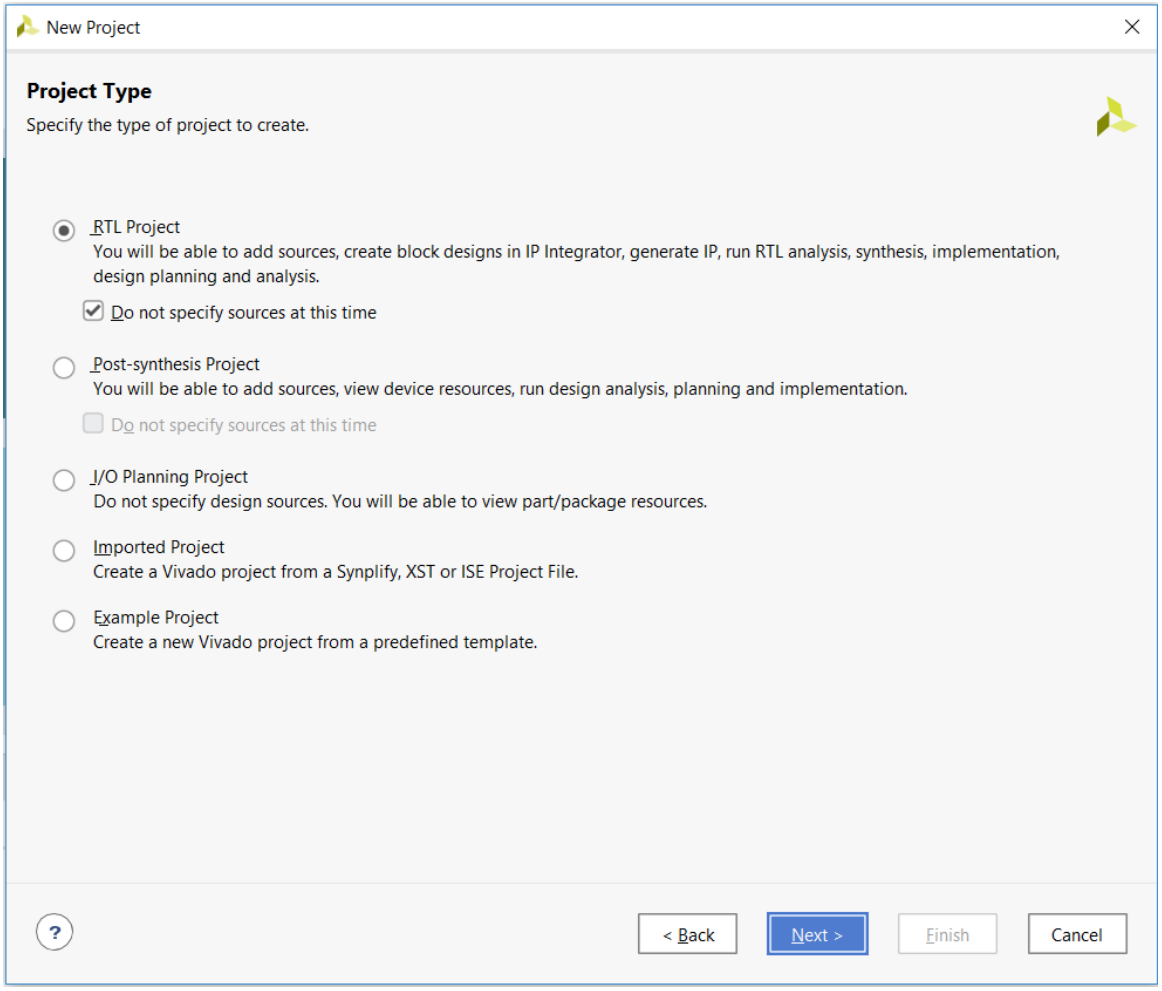
2. In the New Project **Project Name window** enter:
   - Project name: XAPP1336_LAB (project name used for this lab).
   - Project location: ZUP_LAB (path to the directory).
   - Check **Create project subdirectory.**

Figure 5: **Vivado New Project – Project Name Window**



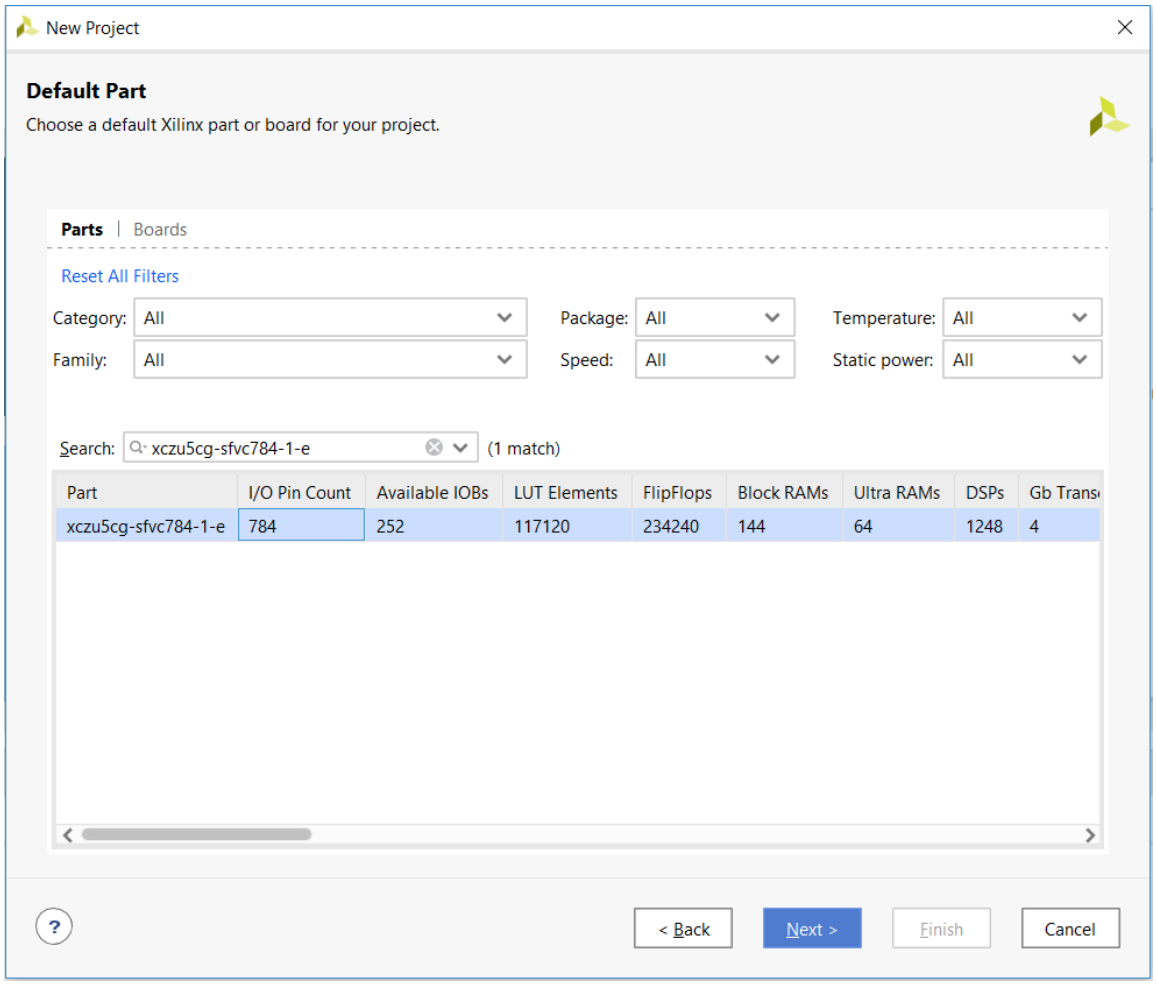
3. Click **Next**.
4. Select **RTL Project** and check **Do not specify sources at this time**.

Figure 6: **Vivado New Project - Project Type Window**



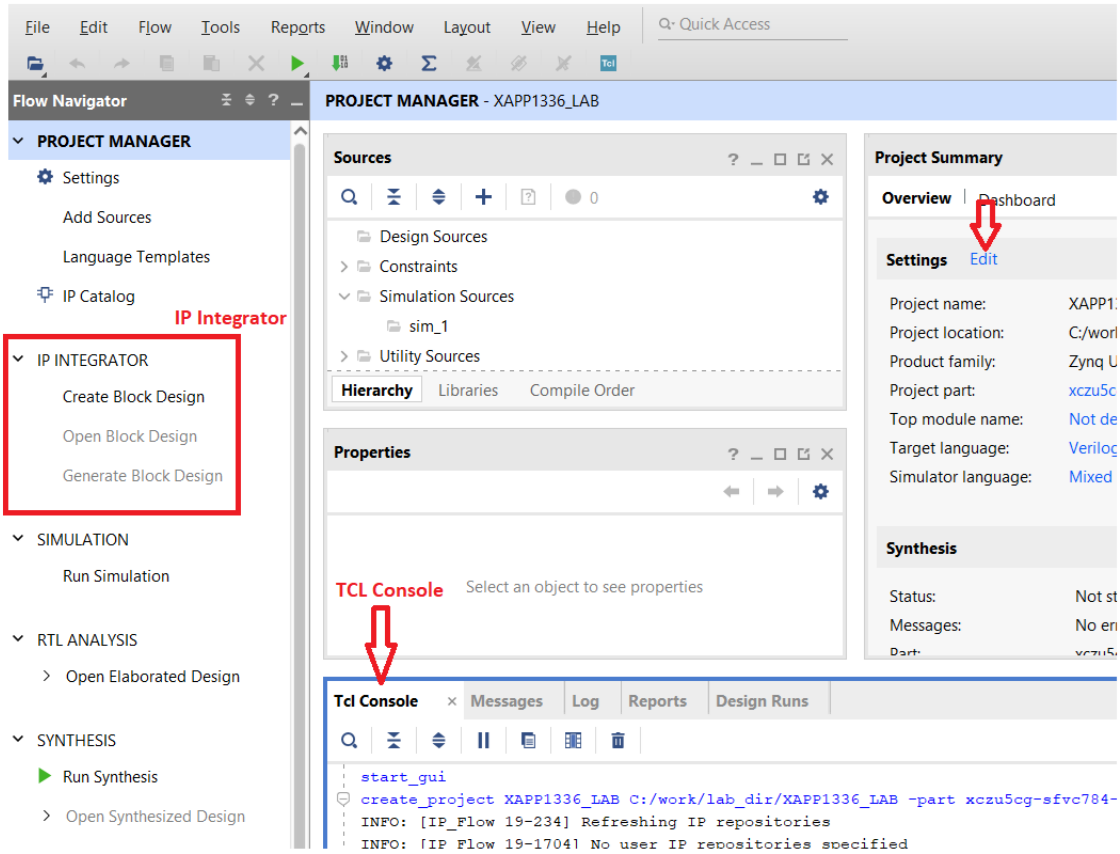
5. Click **Next**.
6. Select `xczu5cg-sfvc784-1-e` in the default window, as shown in the following figure. Type the part name as `xczu5cg-sfvc784-1-e` and then select the device from the results.

Figure 7: **Vivado New Project – Default Part Window**



7. Click **Next** and then click **Finish** on the **New Project Summary** window.
8. The Vivado project is now created. The following figure shows the project manager window for the project created.
9. Click **Edit** in the **Project Summary** window and change **Target Language** to **VHDL**.

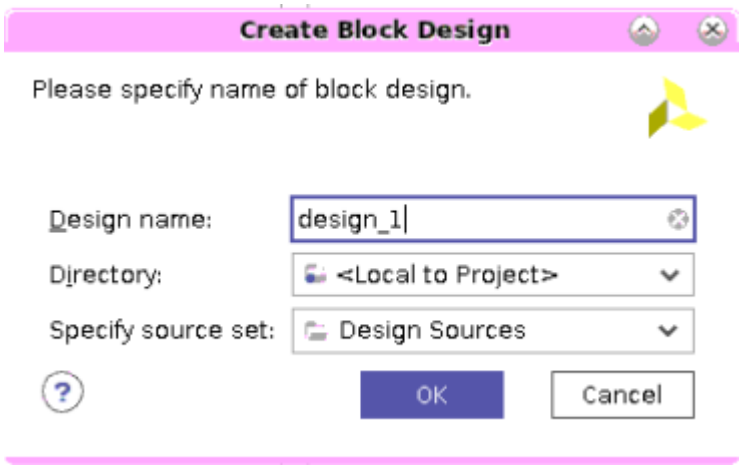
Figure 8: **VivadoProject Manager View**



Creating an IPI Block Design

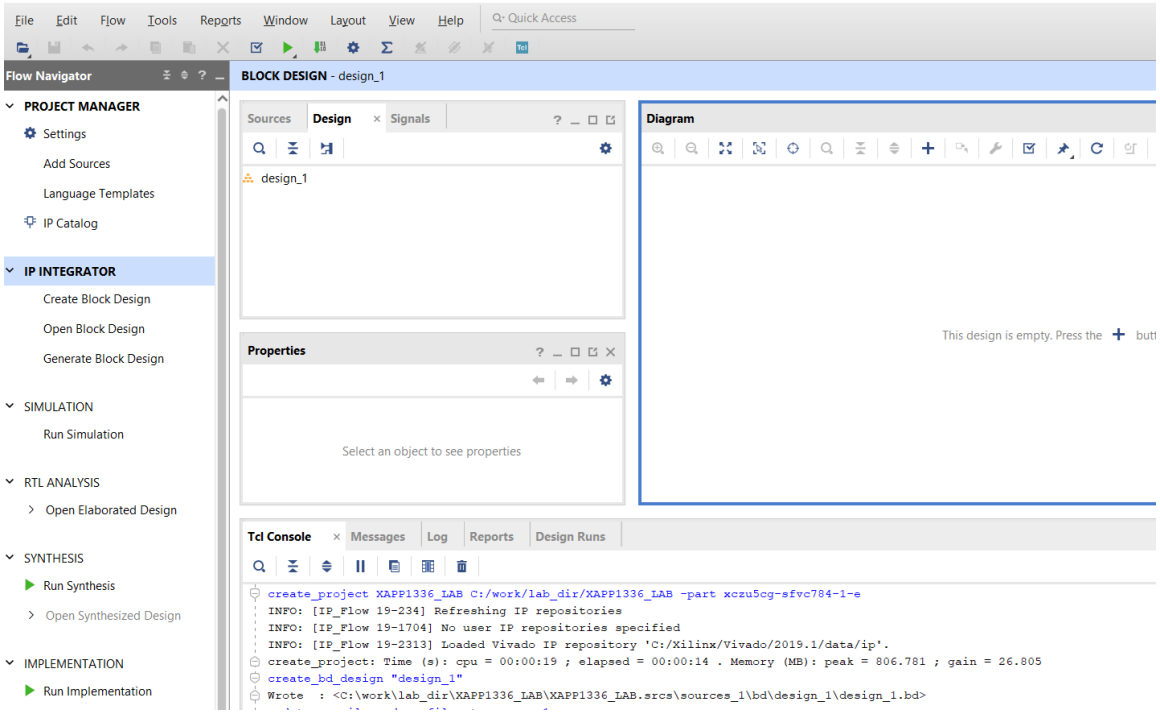
1. Click **Create Block Design** under IP INTEGRATOR menu. You may refer to the previous figure (Figure 8).
2. Keep the default design name **design_1** and select **OK**. Refer to the following figure.

Figure 9: **Create Block Design**



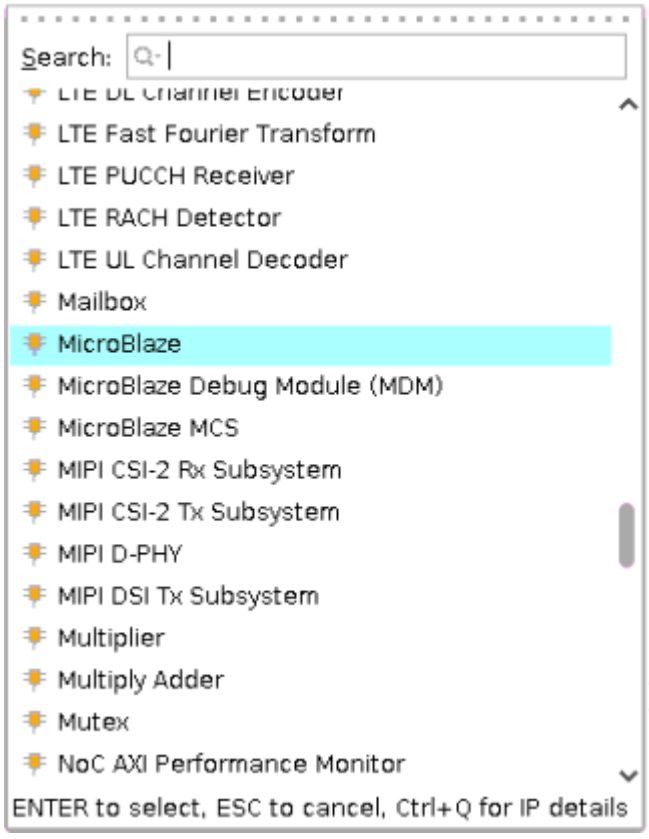
3. You should be able to see a blank diagram. Refer to the following figure.

Figure 10: **IPI Block Diagram**



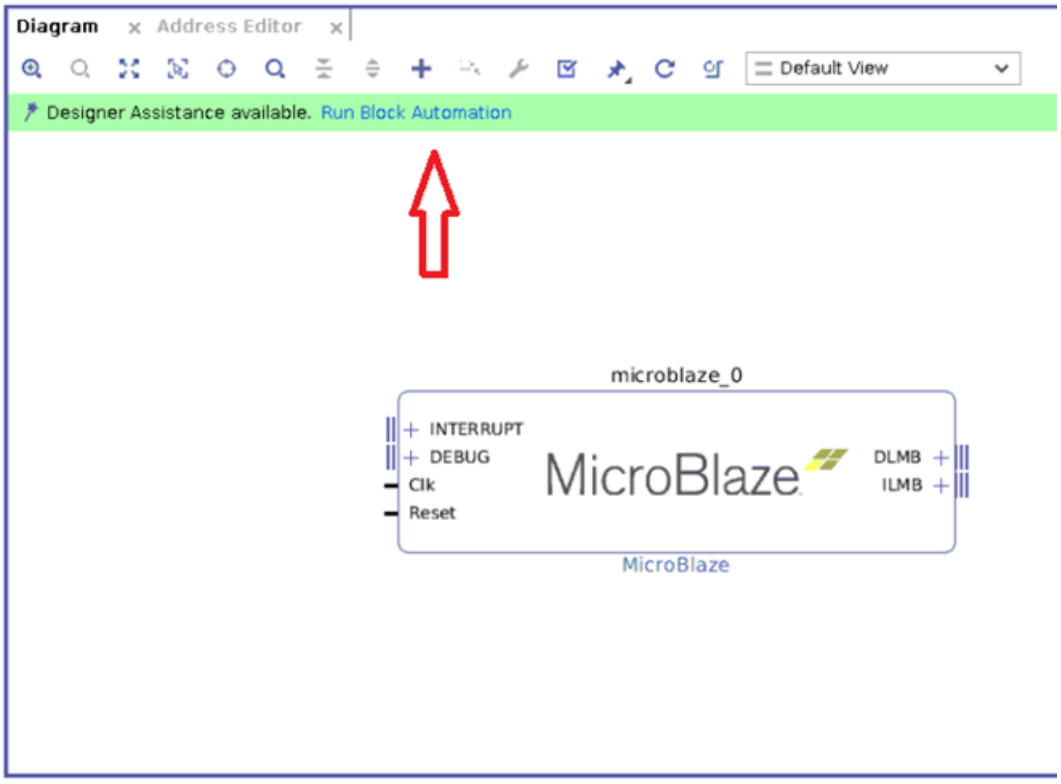
4. Right-click in the **Diagram** canvas and select **Add IP**. Alternatively, you can click the **+** symbol at the center of the blank diagram.
5. Scroll to the **MicroBlaze** IP, select it, and press **Enter**. Refer to Figure 11 and Figure 12.

Figure 11: **Add IP > MicroBlaze**



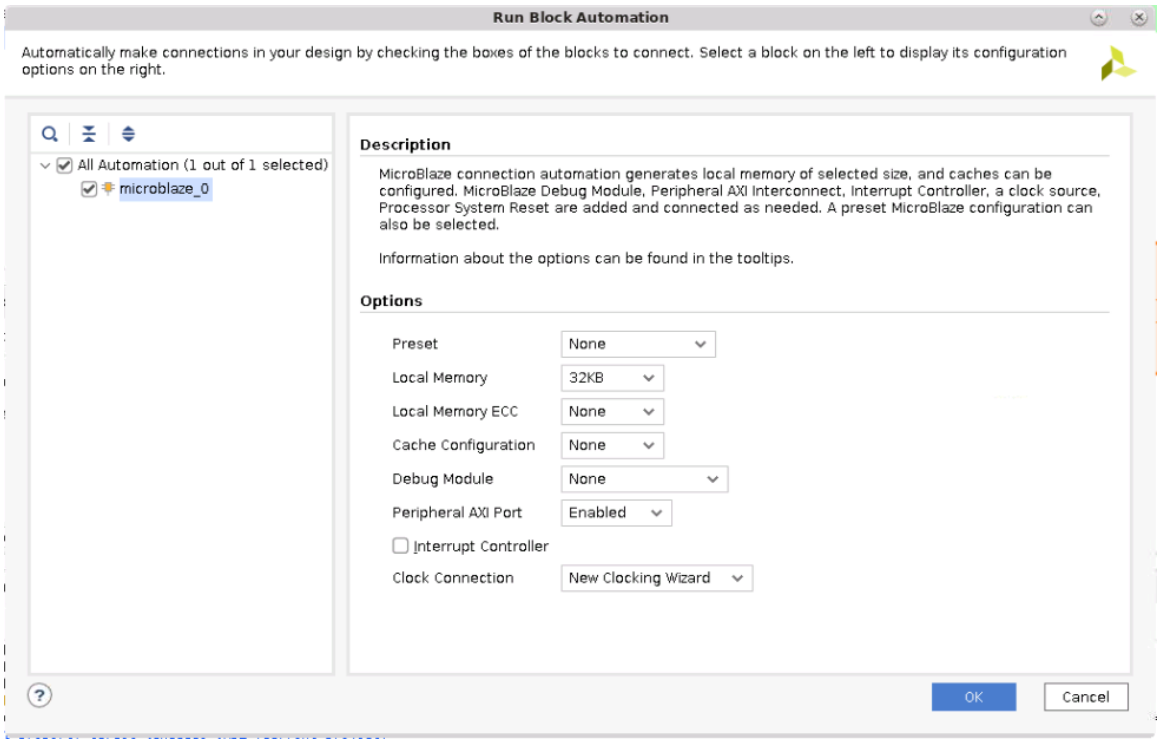
6. Click **Run Block Automation** located on top of the diagram window. Refer to the following figure.

Figure 12: **IPI Canvas after IP Selection**



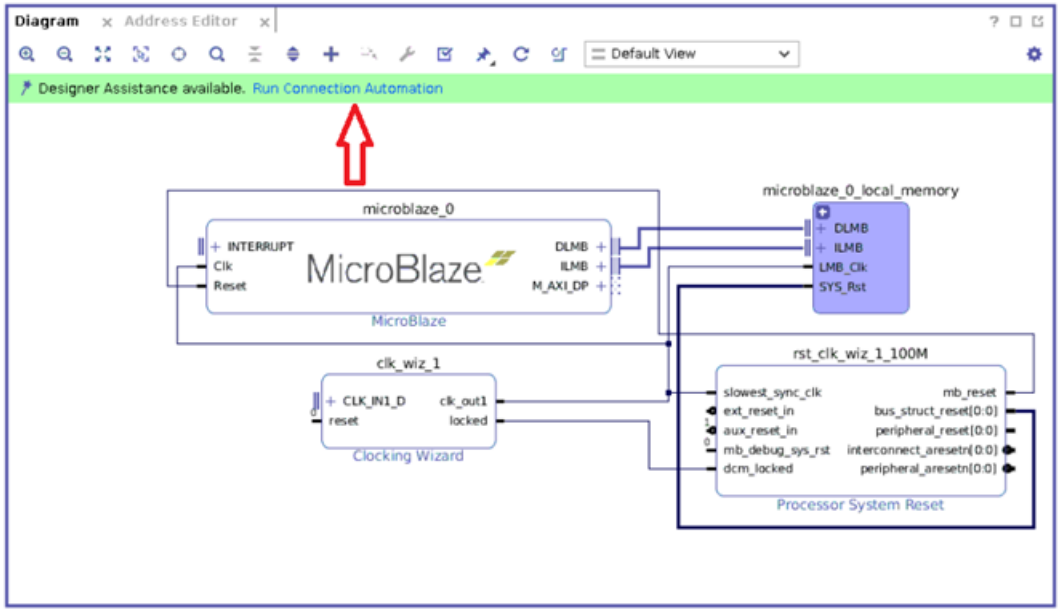
7. Select the following values in the Run Block Automation window and then click **OK**. Refer to the following figure.
   - Preset- None
   - Local memory- 32KB
   - Local memory ECC- None
   - Cache configuration- None
   - Debug module- None
   - Peripheral AXI port- Enabled
   - Clock connection- New clocking wizard

Figure 13: **Run BlockAutomation window**



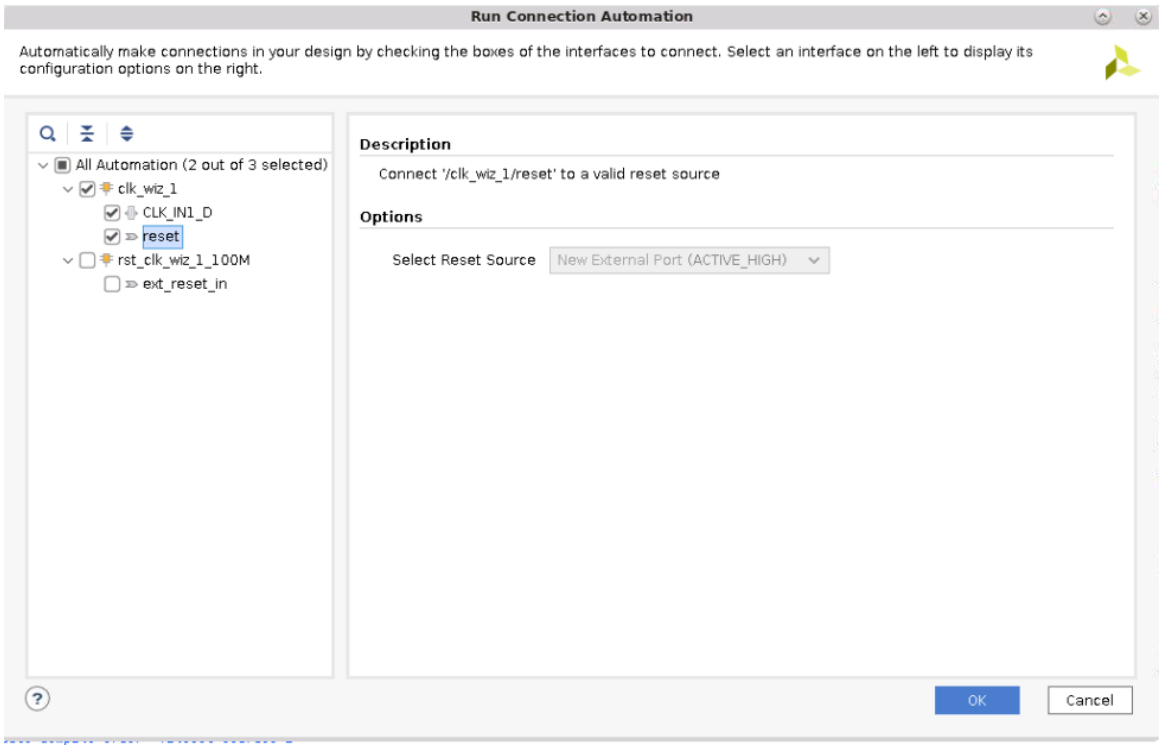
8. Once block automation finishes, **Run Connection Automation** option appears. Click **Run Connection Automation**. Refer to the following figure.

Figure 14: **Once Run Block Automation is Completed**



9. Select `clk_wiz_1` from drop-down and click **OK**. Refer to the following figure.

Figure 15: **Run Connection Automation Window**

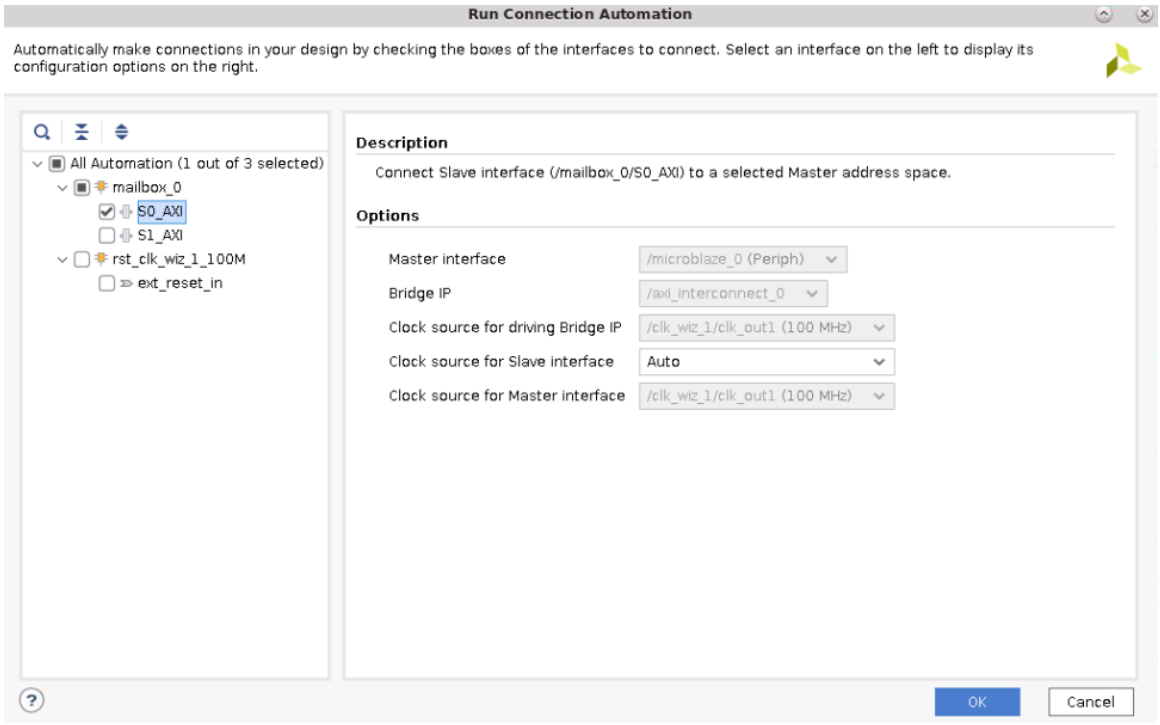


10. Add **AXI Uartlite** IP by right-clicking in diagram canvas and then select, and click **Add IP**, and then select **AXI Uartlite**.
11. Next, click **Run Connection Automation**, select **axi_uartlite_0** and click **OK**.
12. This will create an **AXI Interconnect** IP. Change the name of the IP to **axi_interconnect_0** by selecting the Interconnect IP and then changing the name on Sub-block Properties window. Alternatively you can run the following command from Tcl console.

```
 set_property name axi_interconnect_0 [get_bd_cells
microblaze_0_axi_periph]
```

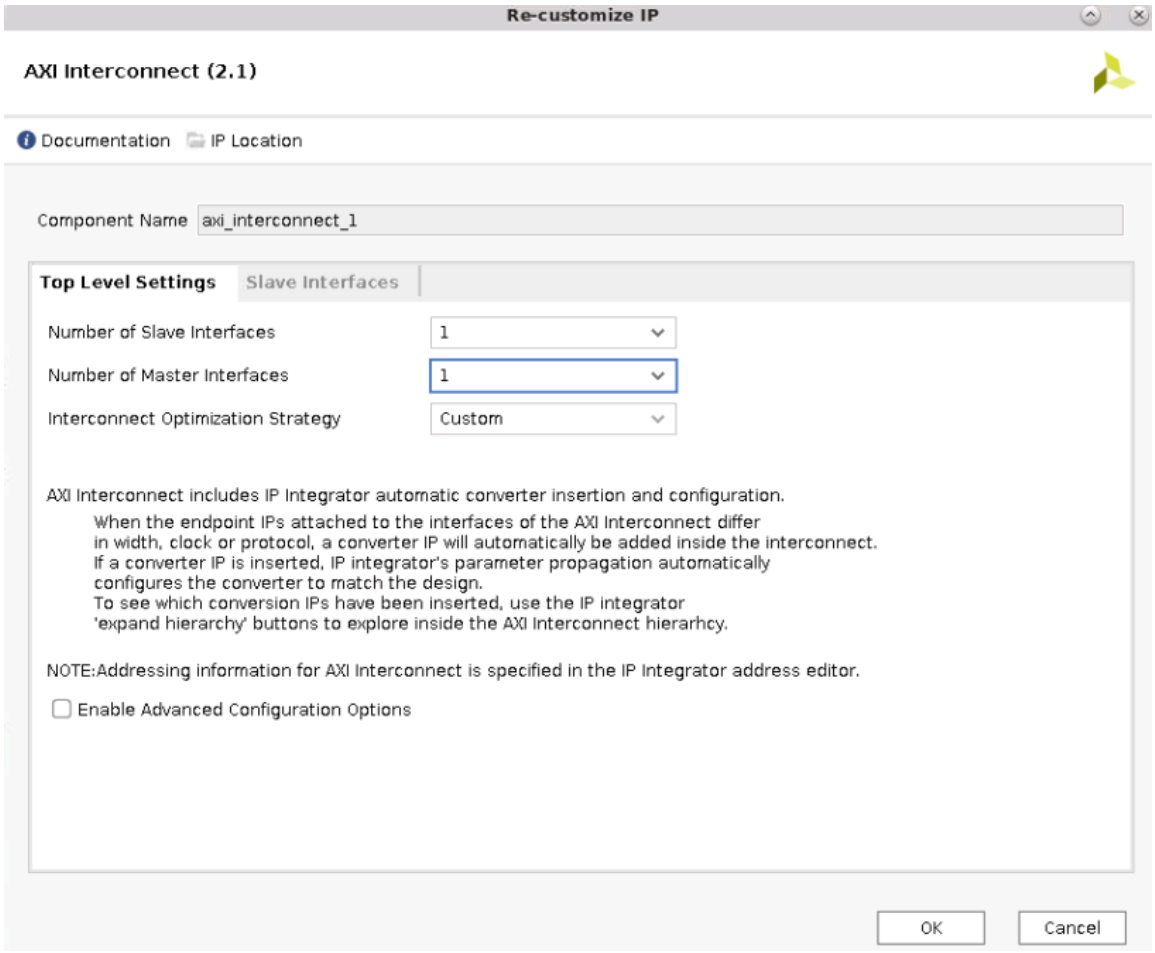
13. Add Mailbox IP by right-clicking in the Diagram canvas and clicking on **Add IP**. Then select click **Mailbox**.
14. Click on **Run Connection Automation** and then select **S0_AXI** of **mailbox_0** and click **OK**. Refer to the following figure.

Figure 16: **Run Connection Automation Window**



15. Add **Zynq UltraScale+ MPSOC** and **AXI Interconnect** IP by right-clicking diagram canvas and click **Add IP**.
16. Customize **axi_interconnect_1** IP to **Number of Master Interfaces** to **1**. To customize, double-click **axi_interconnect_1** IP. It opens the re-customize IP window. Refer to the following figure. Select **1** for the **Number of Master Interfaces** from the drop-down menu and click **OK**.

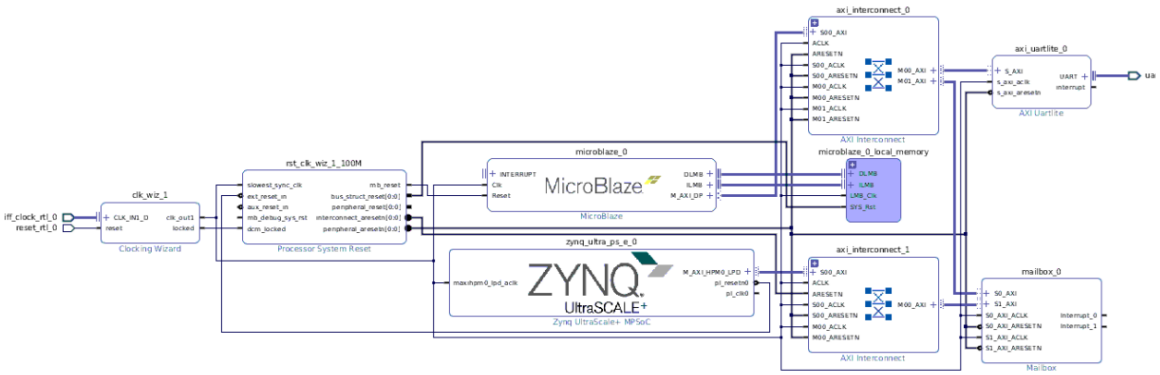
Figure 17: **Re-Customize IP Window**



17. Make the following connections:
    - S00_AXI of axi_interconnect_1 to M_AXI_HPM0_LPD of zynq_ultra_ps_e_0.
    - M00_AXI of axi_interconnect_1 to S1_AXI of mailbox_0.
    - ext_reset_in of rst_clk_wiz_1_100M to pl_resetn0 of zynq_ultra_ps_e_0.
    - maxihpm0_lpd_aclk of zynq_ultra_ps_e_0 to clk_out1 of clk_wiz_1.
    - ACLK of axi_interconnect_1 to clk_out1 of clk_wiz_1.
    - S00_ACLK of axi_interconnect_1 to clk_out1 of clk_wiz_1.
    - M00_ACLK of axi_interconnect_1 to clk_out1 of clk_wiz_1.
    - ARESETN of axi_interconnect_1 to interconnect_aresetn of rst_clk_wiz_1_100M.
    - S00_ARESETN of axi_interconnect_1 to peripheral_aresetn of rst_clk_wiz_1_100M.
    - M00_ARESETN of axi_interconnect_1 to peripheral_aresetn of rst_clk_wiz_1_100M.
    - S1_AXI_ACLK of mailbox_0 to clk_out1 of clk_wiz_1.
    - S1_AXI_ARESETN of mailbox_0 to peripheral_aresetn of rst_clk_wiz_1_100M.
18. Assign the address by running **assign_bd_address** command from Tcl Console. Refer to Figure 8 for the location of the Tcl Console.

19. Right-click an empty space in the diagram canvas and select **Regenerate Layout**. When you are finished, the IPI canvas will be re-organized to resemble the following figure.

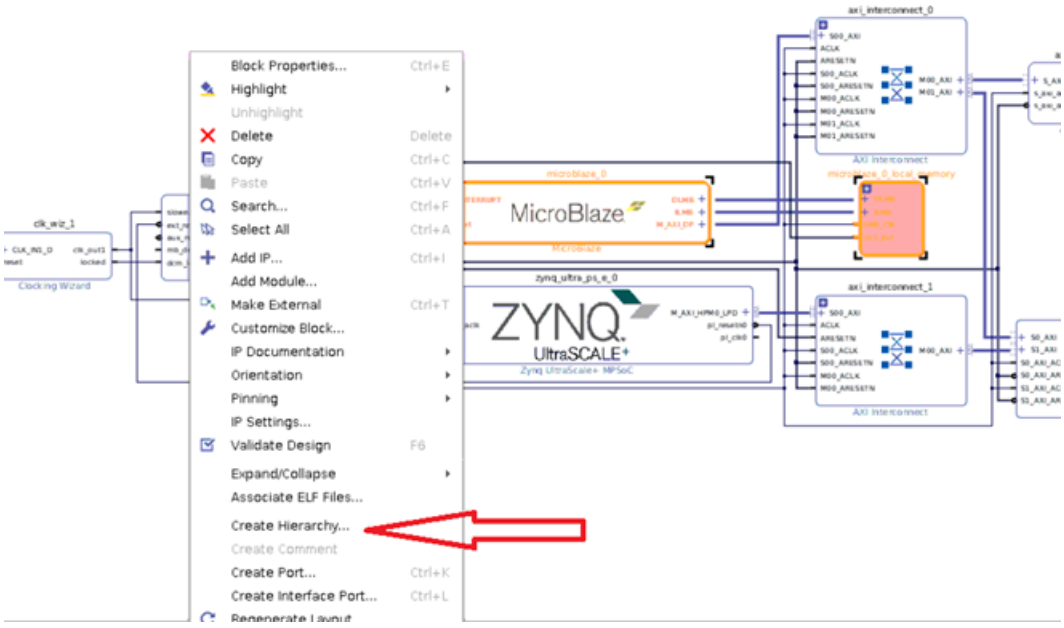
Figure 18: **IP Canvas after Regenerate Layout**



TMR MicroBlaze CPU

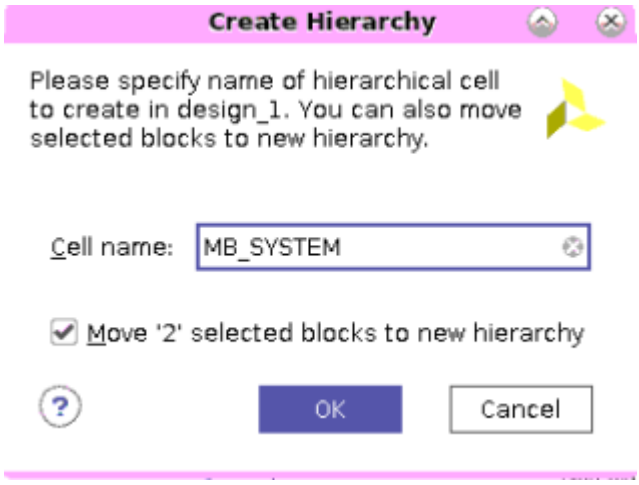
1. Create a **Hierarchy** temporarily for **microblaze_0** and **microblaze_0_local_memory**. Select both the **IPs, right-click,** and then click on **Create Hierarchy**. Refer to the following figure.

Figure 19: **IP Canvas after Selecting IPs and Right-Click**



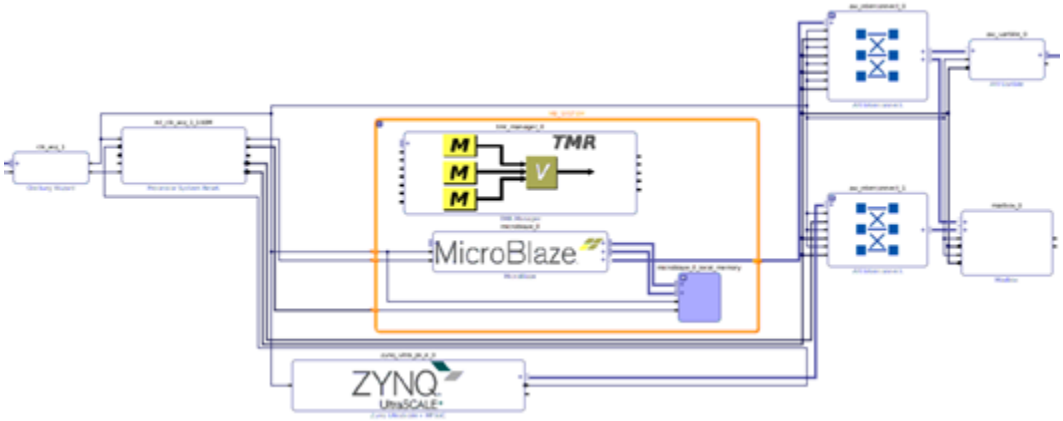
2. Name the cell as MB_SYSTEM, select **Move 2 selected blocks to new hierarchy**, and then click **OK**. Refer to the following figure.

Figure 20: **Create Hierarchy Window**



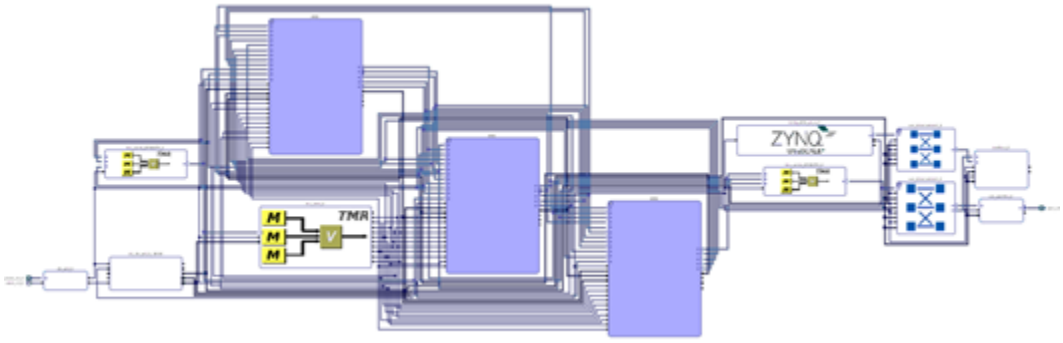
3. Add **TMR Manager** IP under **MB_SYSTEM** by expanding the **MB_SYSTEM**. Then right-click in the **MB_SYSTEM** box and click **ADD IP**. Refer to the following figure.

Figure 21: **IP Canvas after Adding TMR Manager IP**



4. Click **Run Block Automation**, *keeping the default selected values for tmr_manager*, and click **OK** on the **Run Block Automation window**.
5. Ungroup the **MB_SYSTEM** hierarchy. Select **MB_SYSTEM** hierarchy and right-click. Then click **Ungroup Hierarchy**.
6. Right-click on an empty space in the canvas and select **Regenerate Layout**. When you are finished, the IPI canvas looks like the following figure.

Figure 22: **Phase 1 Result**



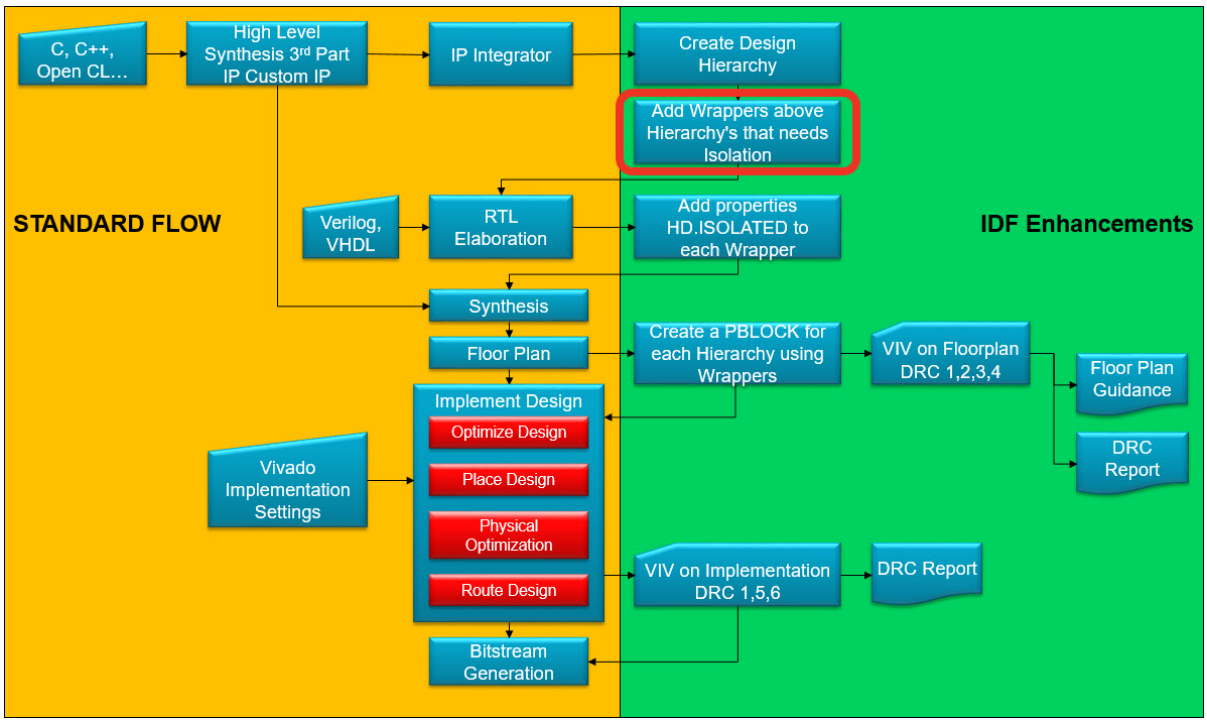
Phase 2

This phase adds wrappers to each of the three MicroBlaze™ containers, creating three hierarchies, and also creates two more hierarchies—one for the mailbox and the other for the PS, as shown in Figure 24. This allows the tools to split nets between isolated blocks which is required for a multi-region net. Such a net needs to be split so that a fault from a block affecting that net cannot fault any other block. You may refer to *Isolation Design Flow for Zynq UltraScale+ Application Note* (XAPP1335) for more information on net-splitting.

The following steps specify how to add the wrappers. Alternatively, you can run phase2 Tcl script from Tcl Console by typing `source ./lab_phase2.tcl`.

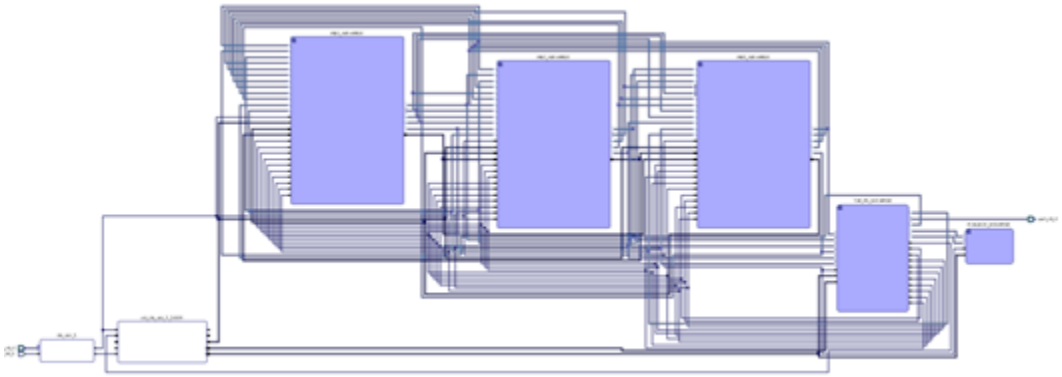
IMPORTANT! *IPI introduces some design complexities by automatically adding DONT_TOUCH properties on every design block of an IPI design. This conflicts with IDF where multi-regional nets are concerned. To split multi-regional nets to meet IDF rules, the tools must modify the design by adding LUT buffers. However, DONT_TOUCH prevents any modification by the tools. Much of this complexity can be minimized by adding a wrapper around modules intended for isolation. Ultimately, it is the wrapper that is marked as isolated. Such wrappers are not required for custom HDL designs not implemented using IPI.*

Figure 23: **Phase 2 - Adding Wrappers**



1. In the Diagram canvas, right-click **MB1** and select **Create Hierarchy...**.
   a. Name it **MB1_WRAPPER**.
   b. Ensure the check box **Move MB1 selected block to new hierarchy** is selected, and then select **OK**.
2. Repeat the previous step for **MB2** naming **MB2_WRAPPER**.
3. Repeat Step 1 for **MB3** naming **MB3_WRAPPER**.
4. Repeat Step 1 for **mailbox_0** naming **MAILBOX_WRAPPER**.
5. Repeat Step 1 for **zynq_ultra_ps_e_0**, **tmr_sem_0**, **axi_uartlite_0**, **tmr_voter_AXI4LITE_0**, **tmr_voter_AXI4LITE_2**, **axi_interconnect_0** and **axi_interconnect_1** by simultaneously selecting all of them and naming it **TOP_PS_WRAPPER**. After completing the preceding steps as mentioned, the design splits into five hierarchies, as shown in the following figure.

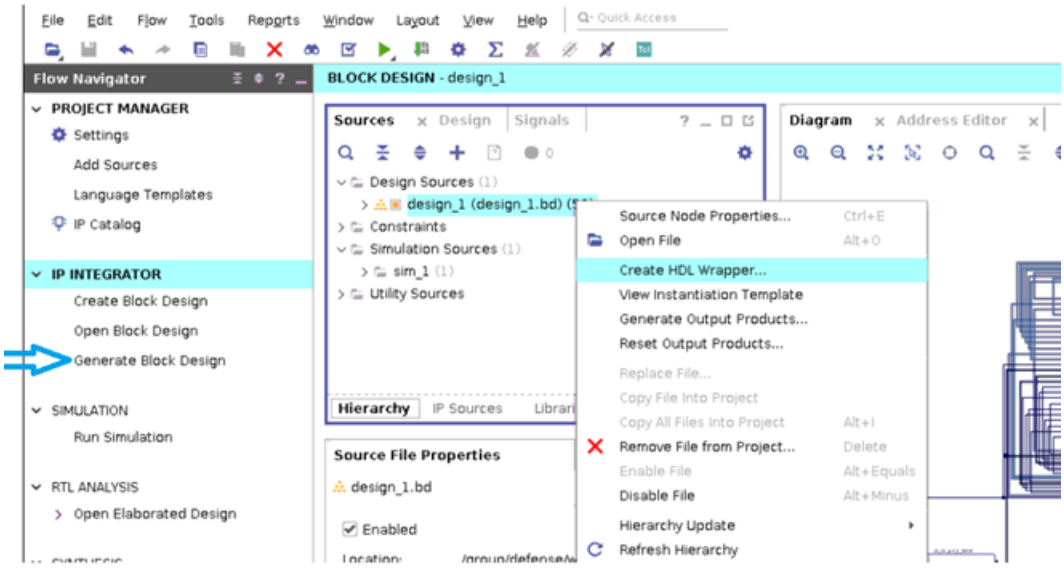
Figure 24: **Result Showing the Five Wrappers**



6. Verify all connections are valid by right-clicking the blank canvas. Select **Validate Design** and click **OK** to continue.
7. **Save** the design.
8. Generate all necessary output products for the block design you just created.
   a. In the **Sources** tab, right-click **design_1**, and select **Create HDL Wrapper**, as shown in Figure 25. On the pop-up window, select **Let Vivado manage the wrapper and auto-update** and select **OK**.
   b. Under the IP integrator menu on the left, select **Generate Block Design** as shown in the following figure. Select **Generate** on the pop-up window. Select **OK** when the process completes. Keep the default values for Systhesis Options.

      The design is now ready for the RTL elaboration phase.
9. **Save** the design.

Figure 25: **Create HDL Wrapper/Generate Block Design**

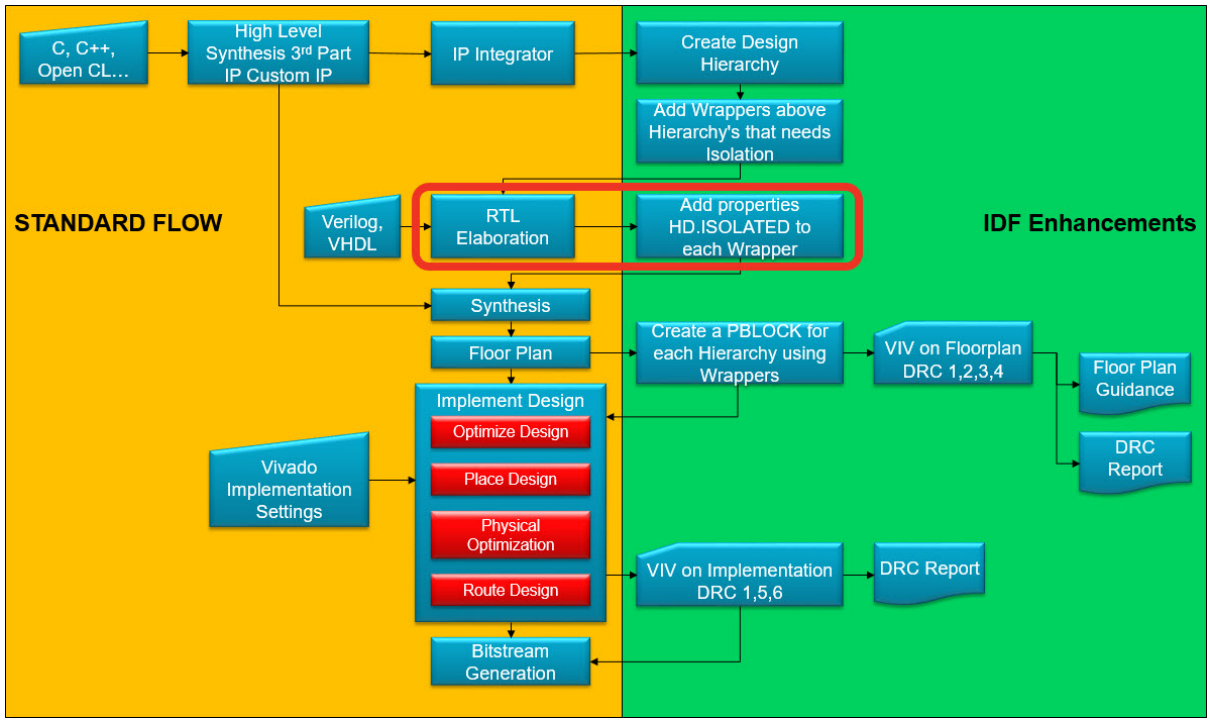


Phase 3

This phase performs the register transfer level (RTL) elaboration and then sets the HD.ISOLATED property for each hierarchical wrapper, as shown in the following figures. Setting HD.ISOLATED is the key property that changes the rule set for the Placer and Router driving an isolated design.

Follow the steps listed here for RTL elaboration, and also to set HD.ISOLATED property. Alternatively, you can run phase3 Tcl script from the Tcl Console by typing `source ./lab_phase3.tcl`.

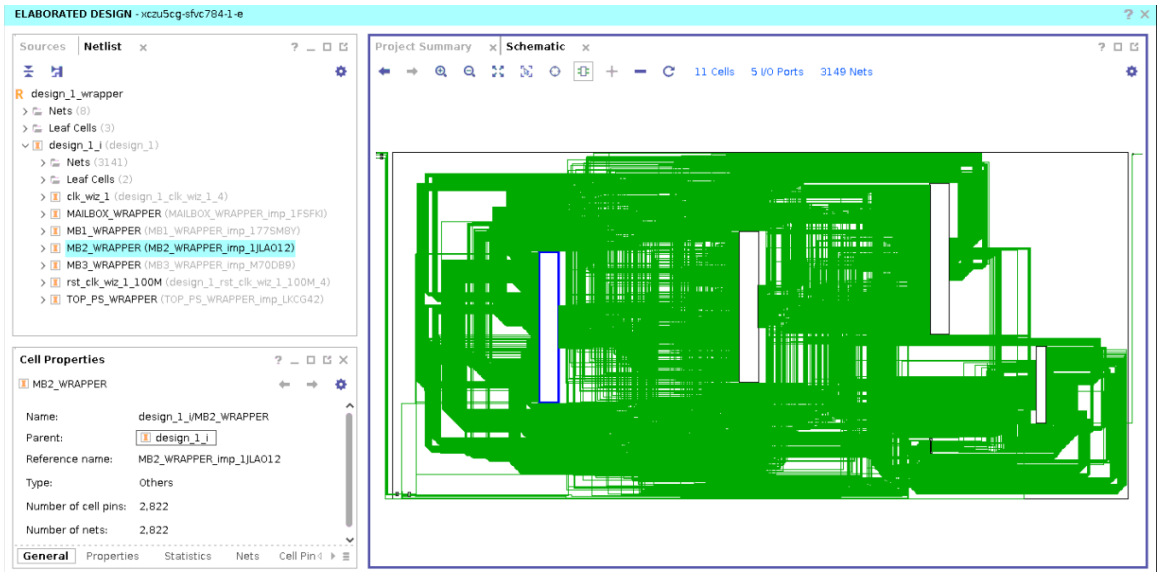
Figure 26: **Phase 3 - Elaboration and setting HD.ISOLATED Properties**



The Simulation and the RTL Analysis sections are located underneath the IP integrator section.

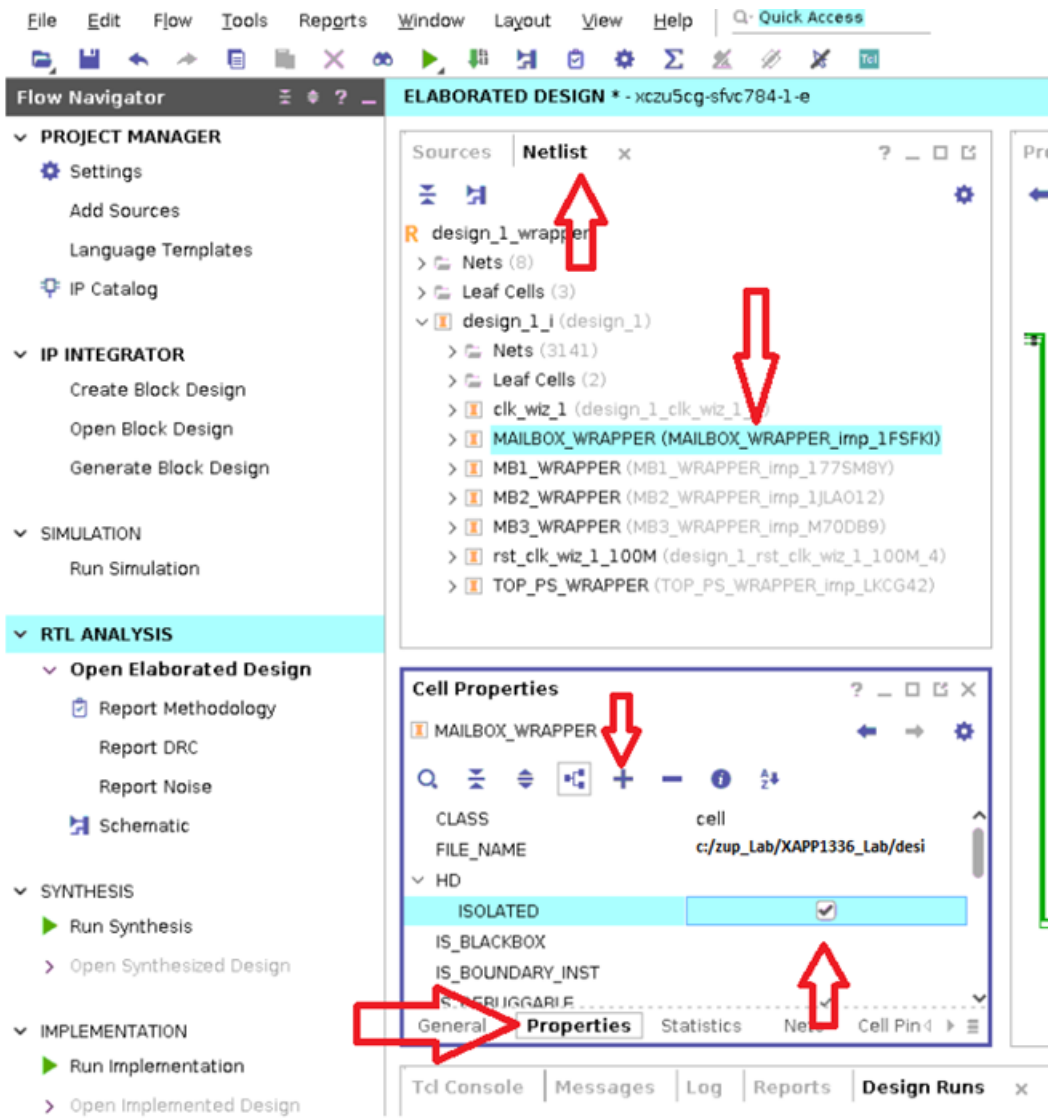
1. Click **Open Elaborated Design**. This launches the process. When this is complete, you can see a large block diagram labeled `design_1_i`. This is the default name generated by the IP integrator when you add the HDL wrapper.
2. You can navigate your design by using the mouse pointer to expand the hierarchy using the **+** button on the top left of any of the block, as seen in the following figure (with one level expanded).

Figure 27: **RTL Schematic - Expanded**



3. Next, you need to add some attributes to tell the tools which modules are going to be isolated using the IDF. This is done with the **HD.ISOLATED** attribute. This attribute not only evokes the IDF routing rules, but also protects redundant modules from undesired optimization. Synthesis optimization cannot happen across an **HD.ISOLATED** boundary. It can happen within one, but that is typically desired.
4. Expand the **design_1_i** instance in the **Netlist** tab so that you can see each of the modules you created in the IP integrator, as shown in the following figure.
5. Select **MAILBOX_WRAPPER**.
   a. Right-click and select **Cell Properties** (if this window is not already visible).
   b. Select the **Properties** tab in the cell properties window.
   c. Click the **+** symbol and add the attribute **HD.ISOLATED** from the **Add Properties** pop-up window.
   d. Expand the newly added attribute and check the unchecked box, as shown in the following figure.

Figure 28: **Setting HD.ISOLATED Attribute**

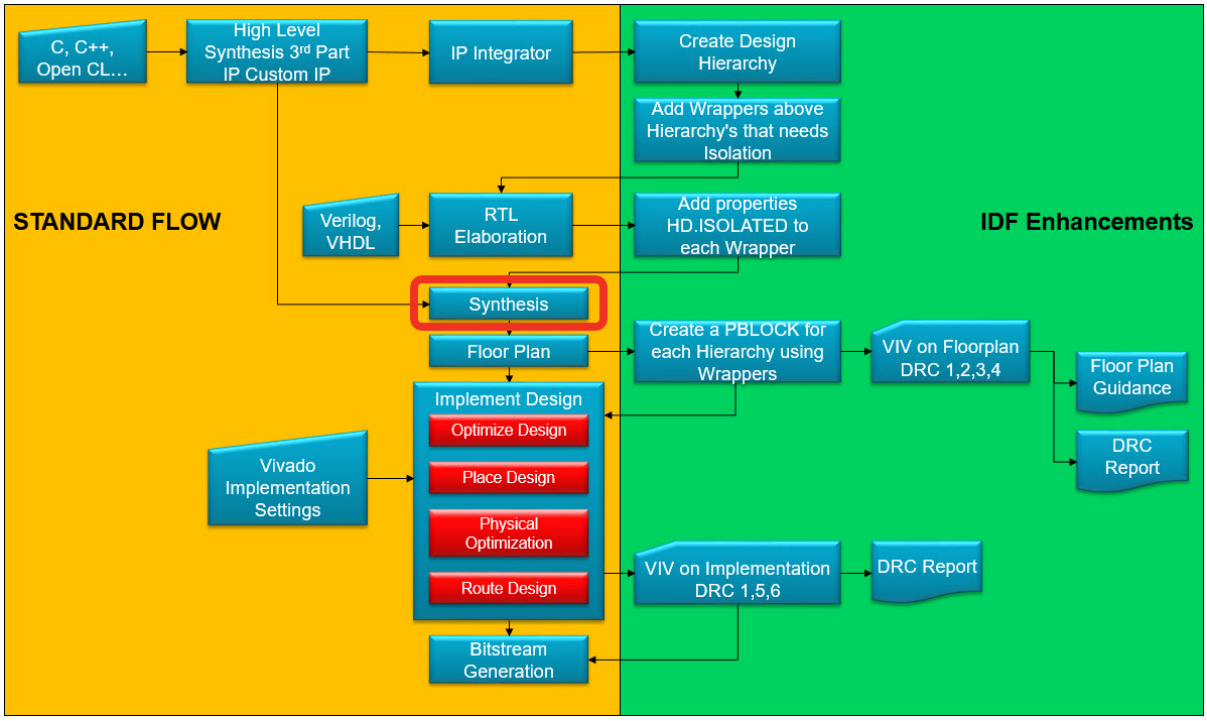


6. Repeat the previous step for the **MB1_WRAPPER**, **MB2_WRAPPER**, **MB3_WRAPPER**, and **TOP_PS_WRAPPER**.
7. Save the design and enter **Top** when requested to enter the file name of the Xilinx design constraints (XDC) file. Lastly, select **OK**.

Phase 4

This phase synthesizes the design which supports and creates the actual logical resources needed to complete the design as shown in the following figures.

Figure 29: **Phase 4 - Synthesize**

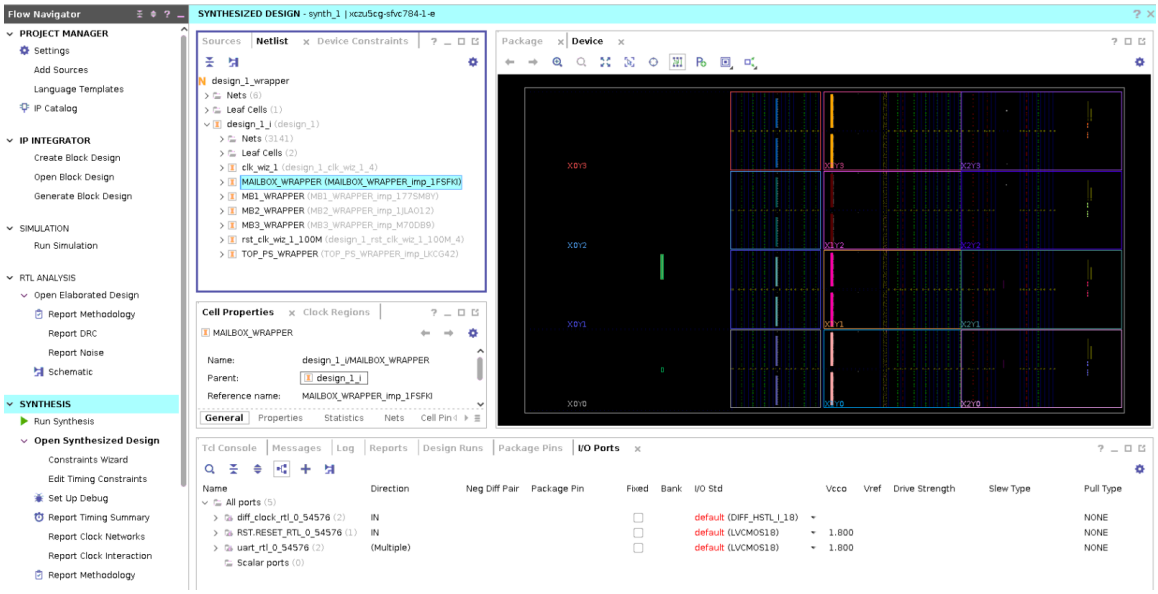


Follow the steps listed here for synthesis. Alternatively, you can run phase4 Tcl script from the Tcl Console by typing `source ./lab_phase4.tcl`.

Launch Synthesis

1. Under the Synthesis menu on the left of the Vivado GUI, select **Run Synthesis**. If prompted, save the design.
2. When it is complete, change the check box to **Open Synthesized Design** and click **OK**.
3. If asked to close the elaborated design before opening the synthesized design, you can do so by selecting **Yes**. This is a good practice because memory might be limited.

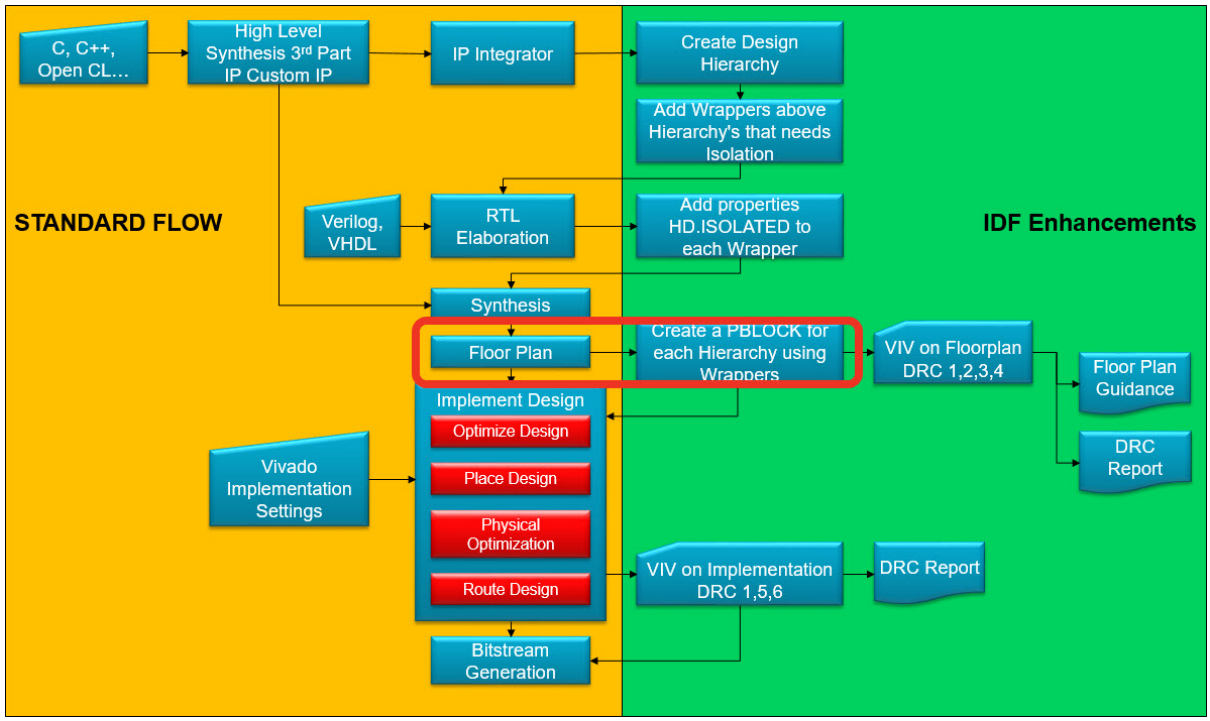
Figure 30: **Synthesized Design**



Phase 5

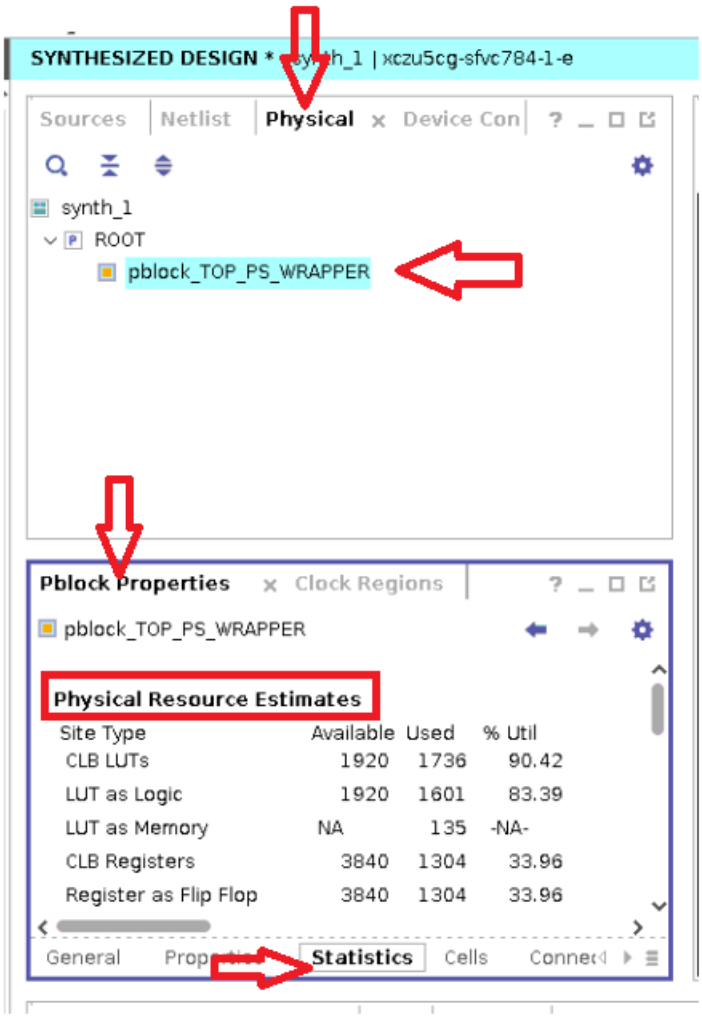
This phase is the floorplanning step where Pblocks are created and assigned to each logical hierarchy. This defines which resource the placer can use to implement the hierarchy of the design.

Figure 31: **Phase 5 - Floor Planning**



When creating a Pblock, verify that adequate resources are available, based on the synthesis of each hierarchy. This is verified by looking at the physical resources estimates table in the Pblock properties window. To view this table, select the **Statistics** tab, as shown in the following figure.

Figure 32: **Pblock Properties Window**

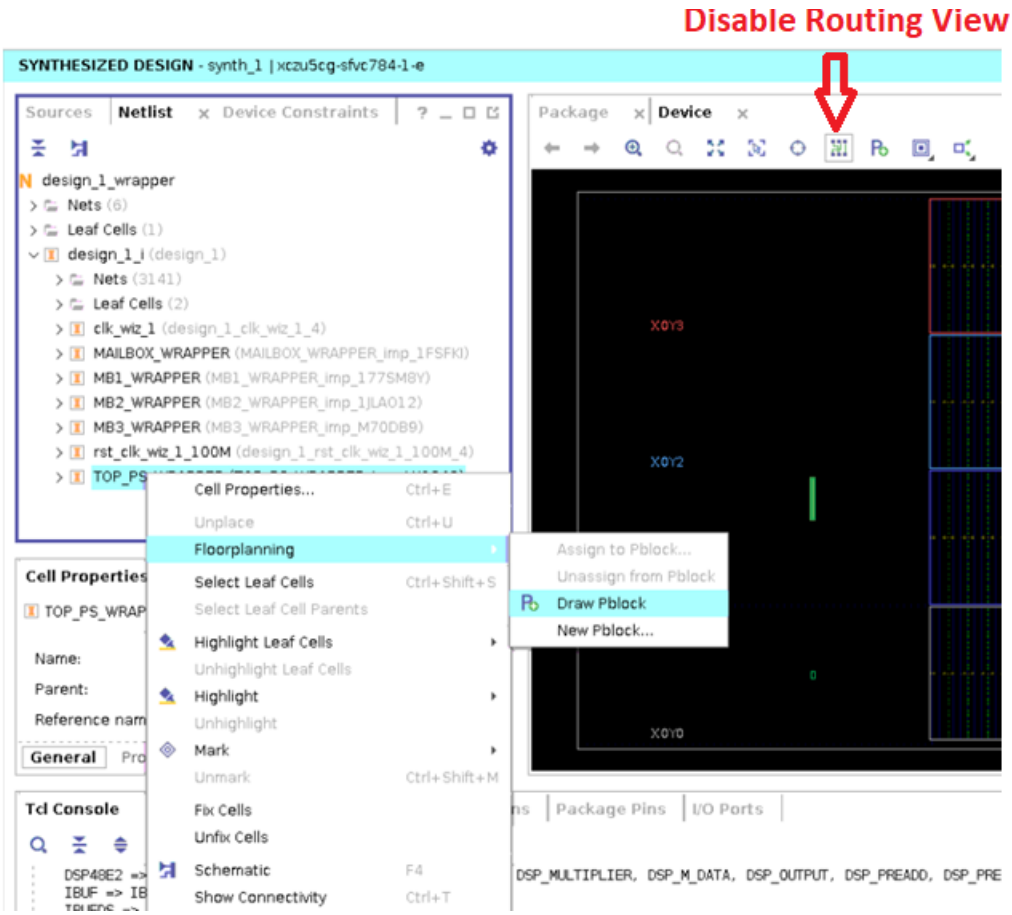


If there are any entries in red (which indicates that there are no resources for a desired site type), the design will not route. If the %Utilization is high, the design might not route. Floorplanning a design is the most time-consuming part of the isolation design flow. This lab helps you to familiarize with the floorplanning from the GUI. It is very important to understand the floorplanning rules and complexities that are associated with floorplanning any design. You can refer to the following documents for design guidance, *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909) and *Vivado Design Suite User Guide: Hierarchical Design* (UG905). You can find more details on the IDF rules in *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335). Execute the following steps. Alternatively, you can run phase5 Tcl script from Tcl Console by typing `source ./lab_phase5.tcl`.

Floorplanning must be done in a non-routing view. Enter non-routing view by de-selecting the routing view icon from the device tab menu, as shown in the following figure.

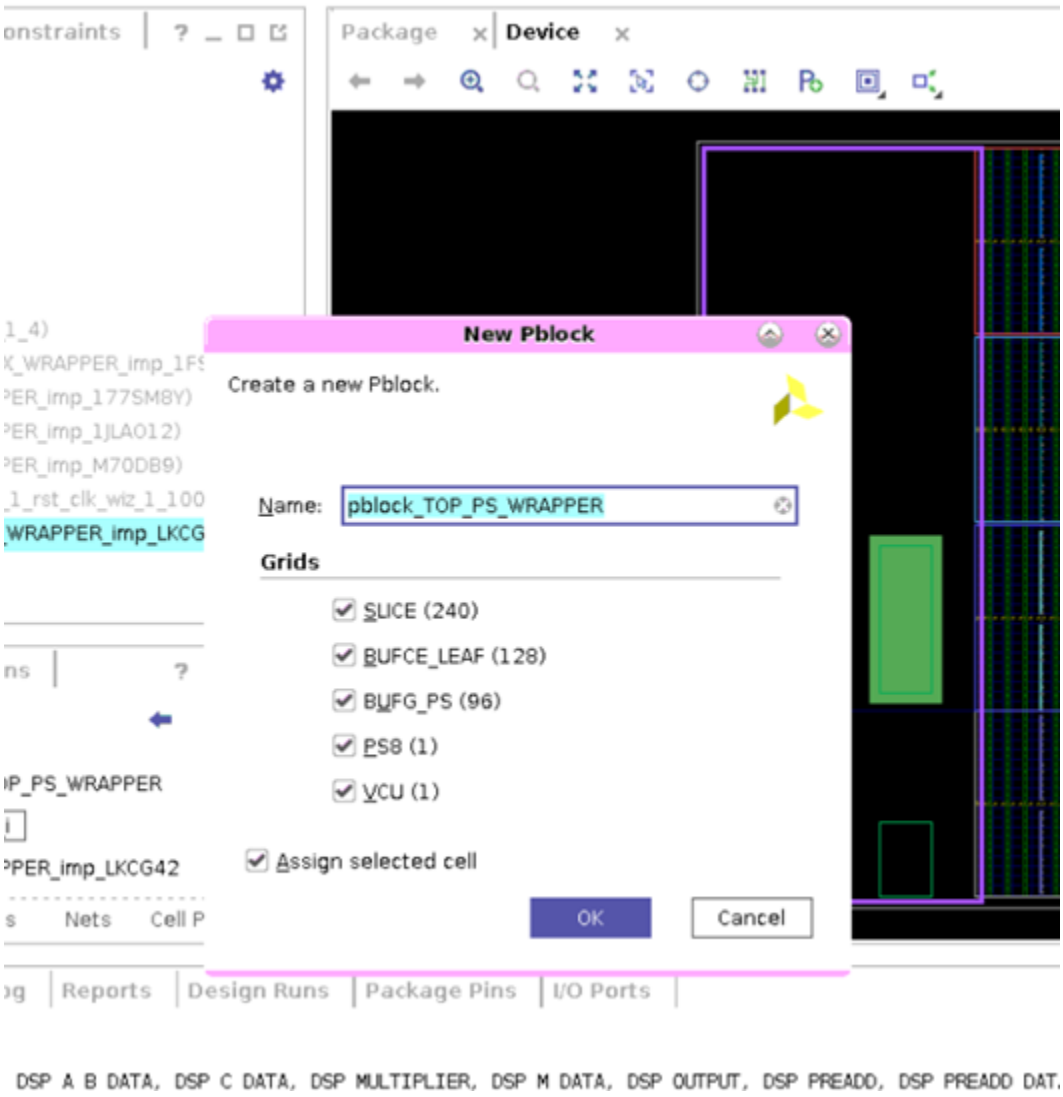
1. Right-click **TOP_PS_WRAPPER** from the **Netlist** window, click **Floorplanning,** and then click **Draw Pblock** as shown in the following figure. You get the option to select sites for the Pblock when you move your mouse pointer to the device window.

Figure 33: **Netlist – Floorplannig > Draw Pblock**



2. Move your mouse pointer to the top left corner of the PS and hold the mouse left key and select **PS PU** and **VCU PU**. **Release** the mouse. You may refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for more knowledge on Programmable Units (PUs). Next, the **New Pblock** window opens. You can assign a name to the Pblock; for this lab, the default name is being used. Ensure **Assign selected cells** check box is checked, as shown in the following figure. Click **OK**.

Figure 34: **New Pblock Window**



3. Open the **Physical Constraints** window from the **Window** menu, as shown in the following figure. Now open the **Pblock Properties** window by selecting the Pblock you just created from the **Physical Constraints** window. Select **the Statistics** tab from the **Pblock Properties** window. You will see the **Physical Resource Estimates** as shown in Figure 36. From the **Physical Resource Estimates**, you will get %Util. Based on this, more resources to the Pblock will be included.

   ***Note*:** Xilinx recommends including more resources to the Pblock than is required by design. Higher utilization (%Util) can lead to routing difficulty or an inability to route the design. The following are guidelines to help ensure the design will successfully route.

```
Slice < 50%
BRAM/DSP < 75%
Other bigger tiles like GT, PS8, IOs, PCIE < 100%
```

Figure 35: **Window > Physical Constraints**

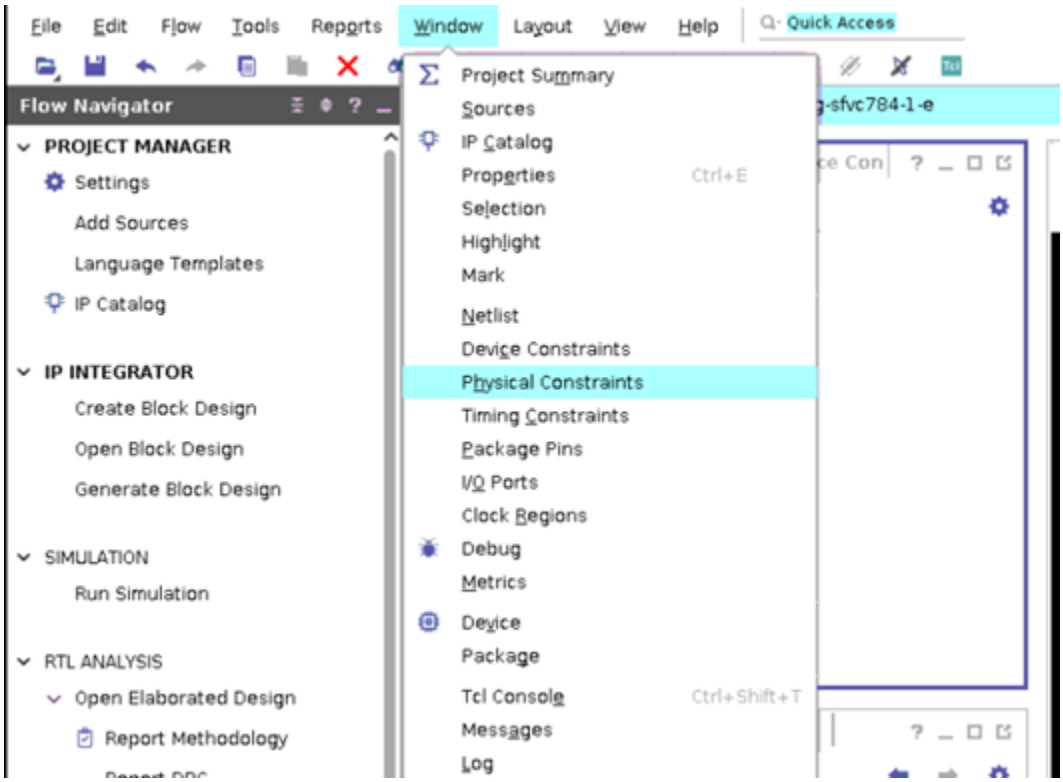
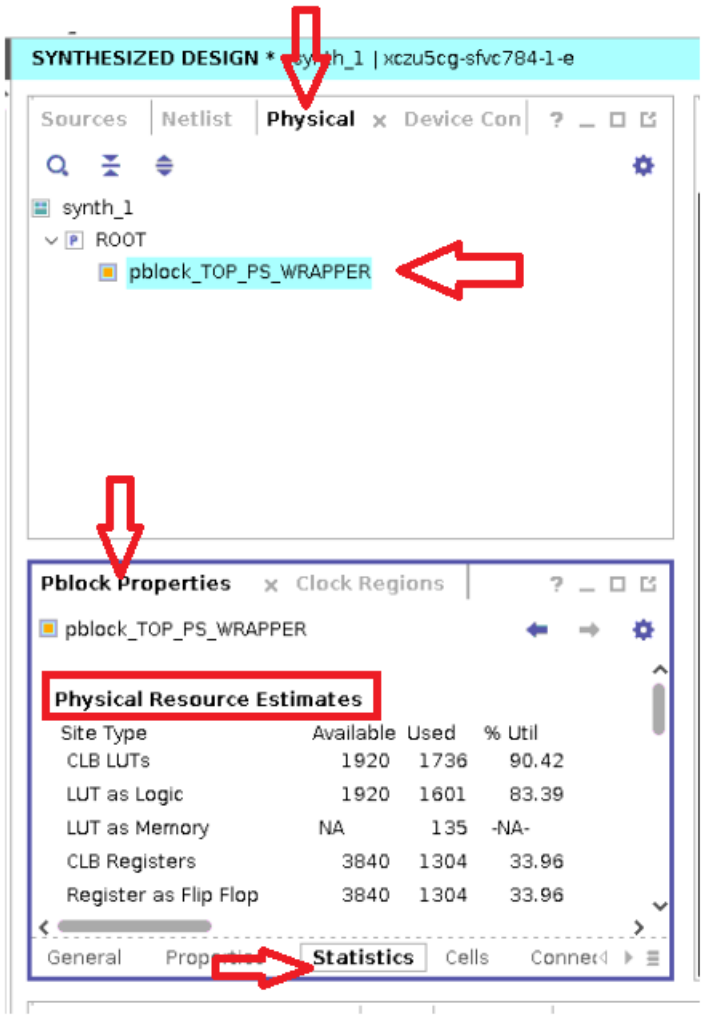
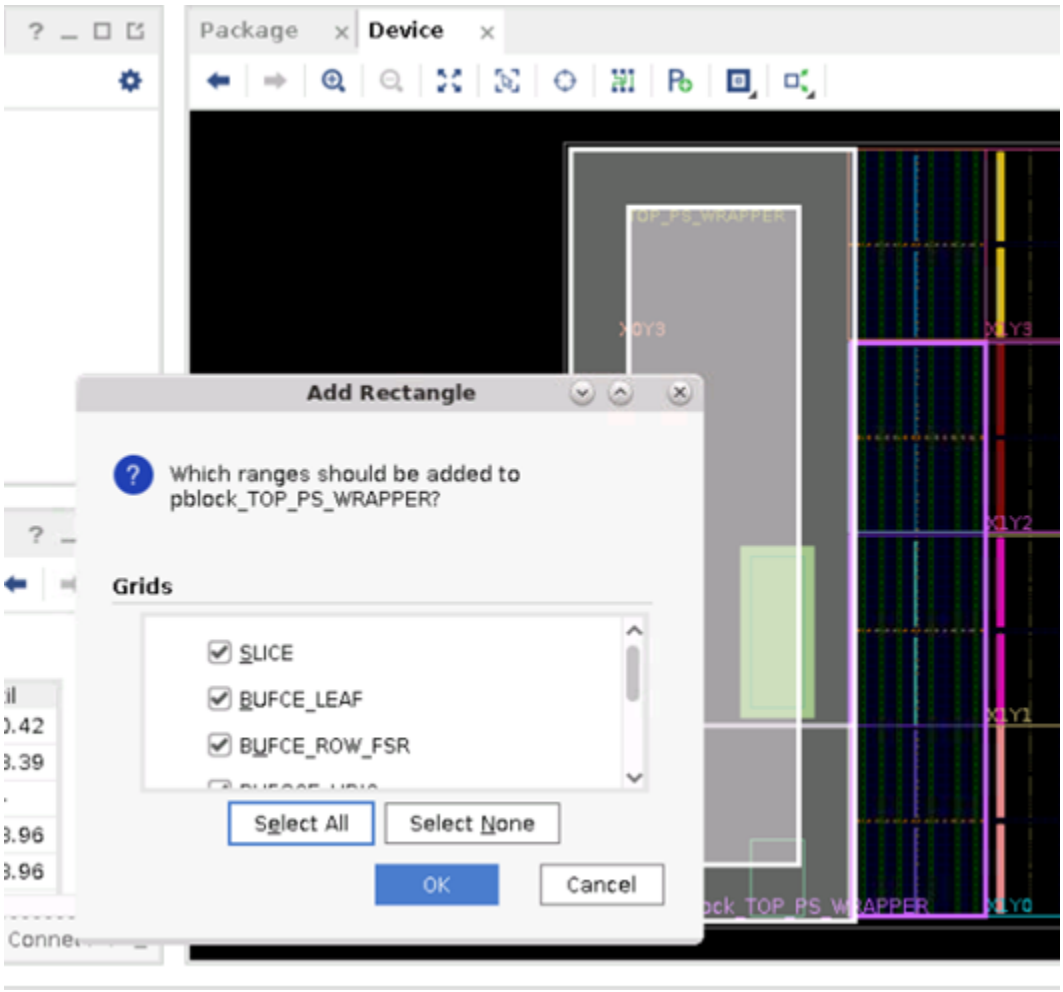


Figure 36: **Pblock Properties Window**



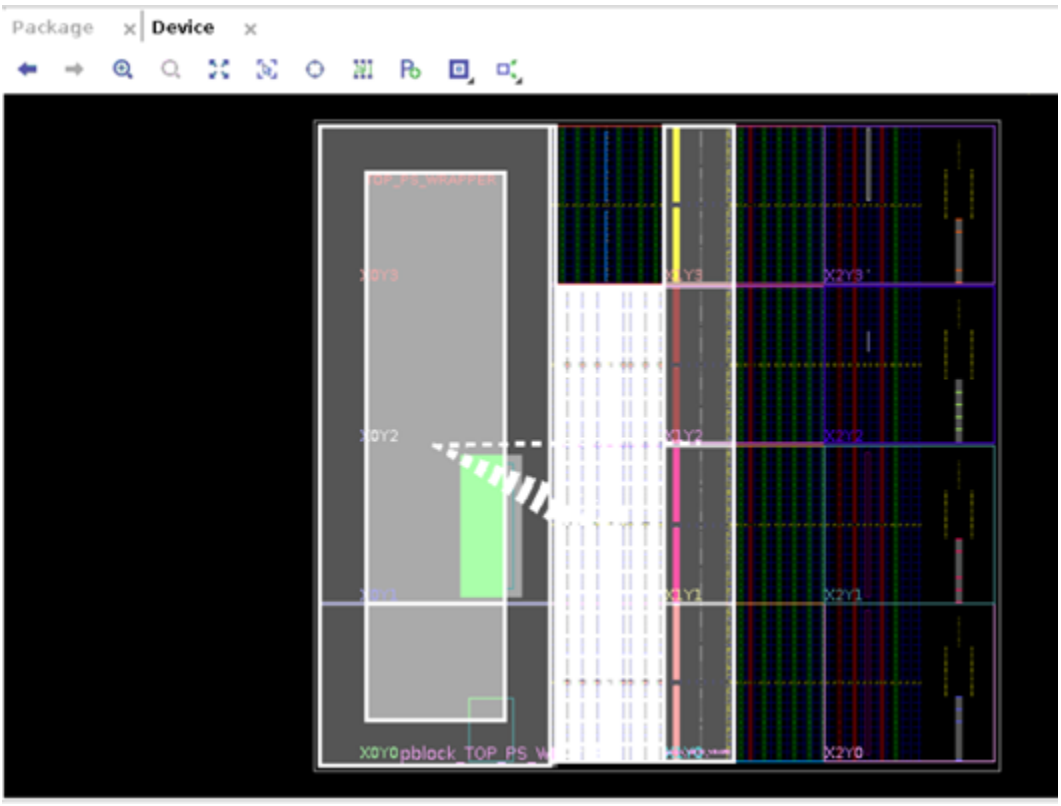
4. Go to the **Properties** tab in **Pblock Properties** window and ensure **SNAPPING_MODE** of the Pblock is **FINE_GRAINED**. You may refer to the *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for more details on **SNAPPING_MODE**.
5. Right-click the Pblock you just created and click **Add Pblock Rectangle**. You will get the option to select more resources on the device window. Move the mouse pointer to the bottom right corner of the **X0Y0 Clock-Region** and hold the mouse left key to select till the top left corner of the **X0Y0 Clock-Region** and then release the mouse. Next, **Add Rectangle** window opens. Click **Select All** and then click **OK**.

Figure 37: **Add Rectangle Window**



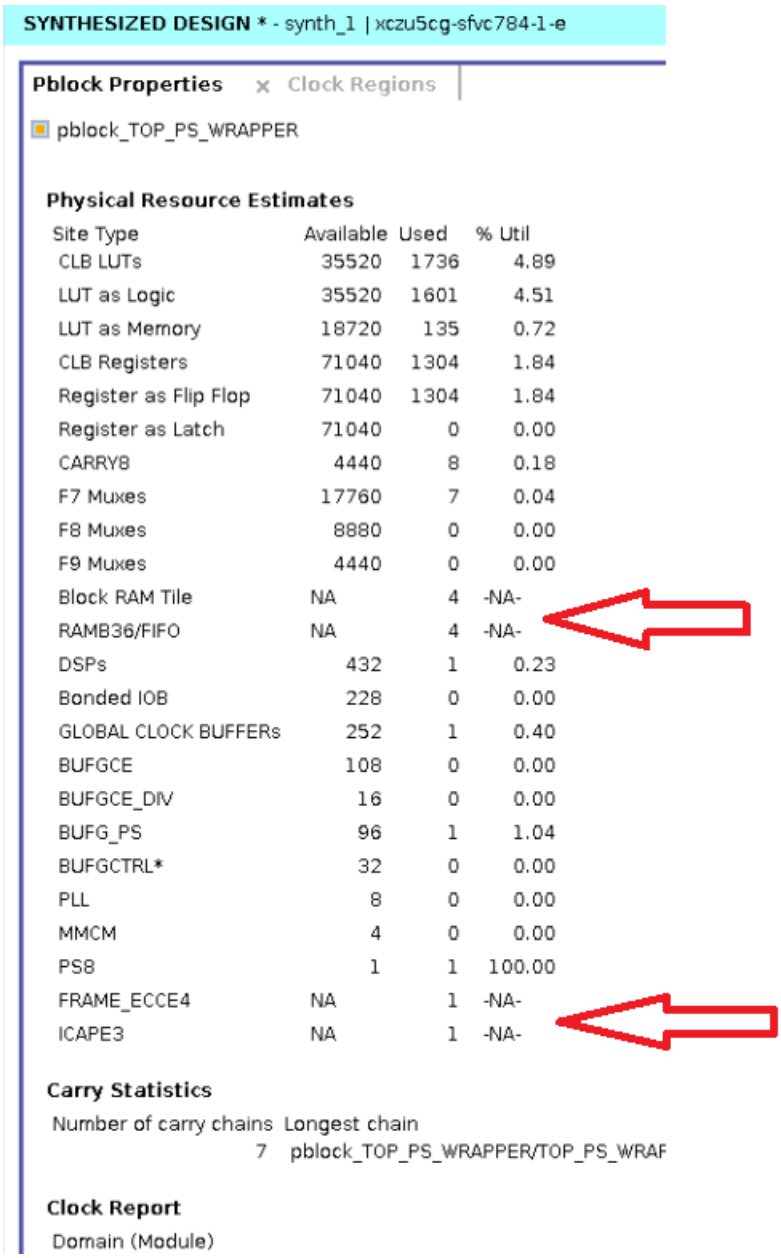
6. Add IOBs to the pblock by repeating step 5 and selecting HPIO PUs, as shown in the following figure.

Figure 38: **After Adding HPIOs to the Pblock**



7. Now look at the **Physical Resource Estimates** section from the **Pblock Properties > Statistics** tab. BRAMs and ICAP need to be added to the Pblock, as shown in the following figure. Repeat Step 4 to add **Config Center** to the Pblock for ICAP, as seen in Figure 40.

Figure 39: **Physical Resource Estimates Section**



SYNTHESIZED DESIGN * - synth_1 | xczu5cg-sfvc784-1-e

Pblock Properties × Clock Regions

pblock_TOP_PS_WRAPPER

Physical Resource Estimates

Site Type	Available	Used	% Util
CLB LUTs	35520	1736	4.89
LUT as Logic	35520	1601	4.51
LUT as Memory	18720	135	0.72
CLB Registers	71040	1304	1.84
Register as Flip Flop	71040	1304	1.84
Register as Latch	71040	0	0.00
CARRY8	4440	8	0.18
F7 Muxes	17760	7	0.04
F8 Muxes	8880	0	0.00
F9 Muxes	4440	0	0.00
Block RAM Tile	NA	4	-NA-
RAMB36/FIFO	NA	4	-NA-
DSPs	432	1	0.23
Bonded IOB	228	0	0.00
GLOBAL CLOCK BUFFERs	252	1	0.40
BUFGCE	108	0	0.00
BUFGCE_DIV	16	0	0.00
BUFG_PS	96	1	1.04
BUFGCTRL*	32	0	0.00
PLL	8	0	0.00
MMCM	4	0	0.00
PS8	1	1	100.00
FRAME_ECCE4	NA	1	-NA-
ICAPE3	NA	1	-NA-

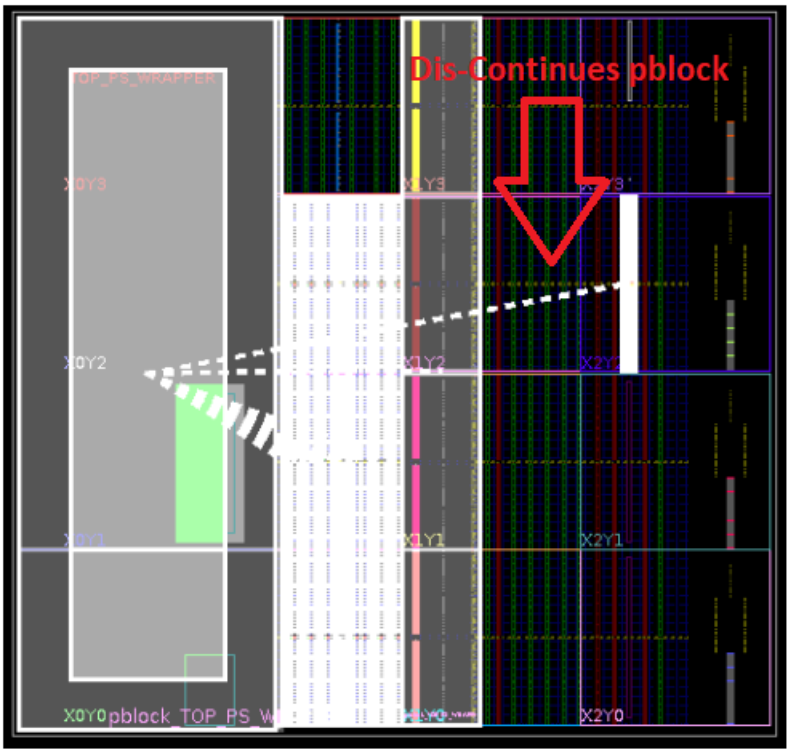
Carry Statistics

Number of carry chains	Longest chain
7	pblock_TOP_PS_WRAPPER/TOP_PS_WRAF

Clock Report

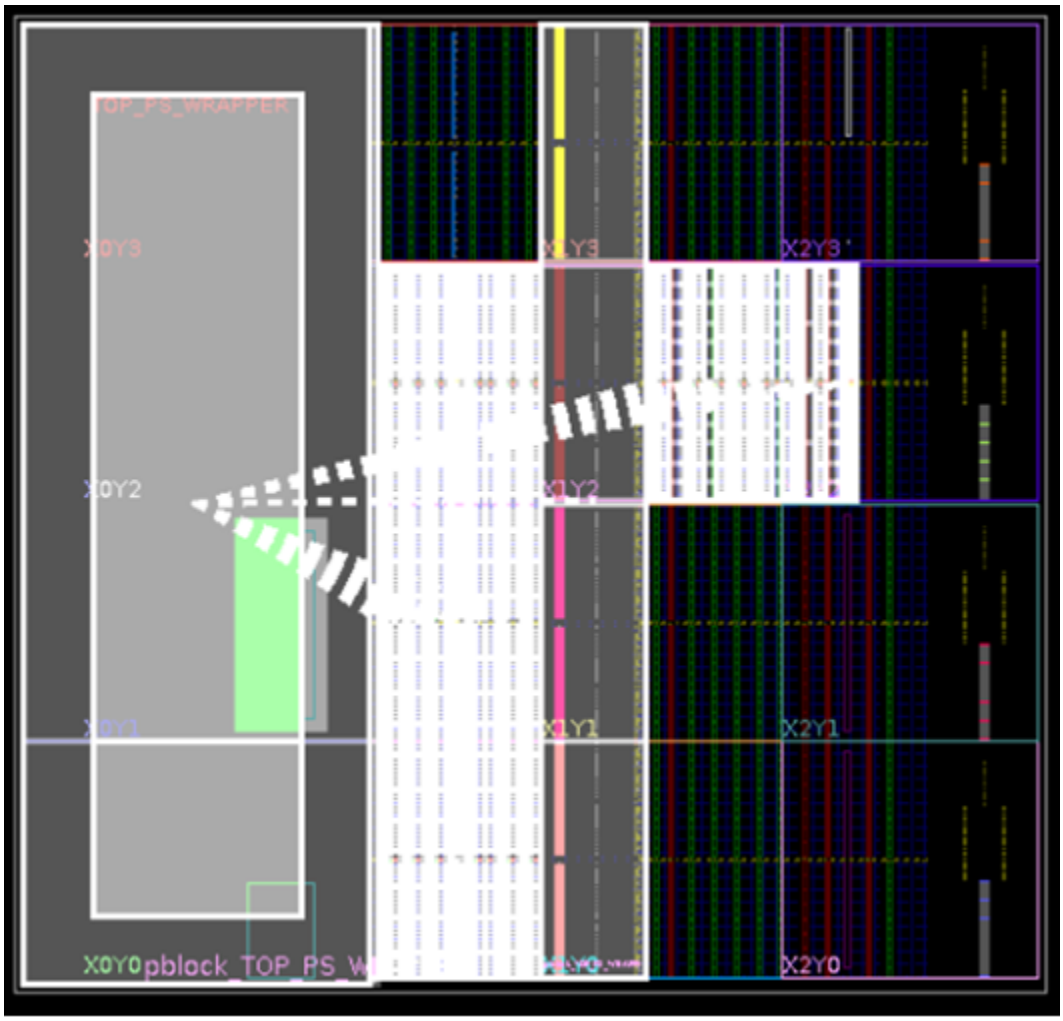
Domain (Module)

Figure 40: **After Adding Config Center**



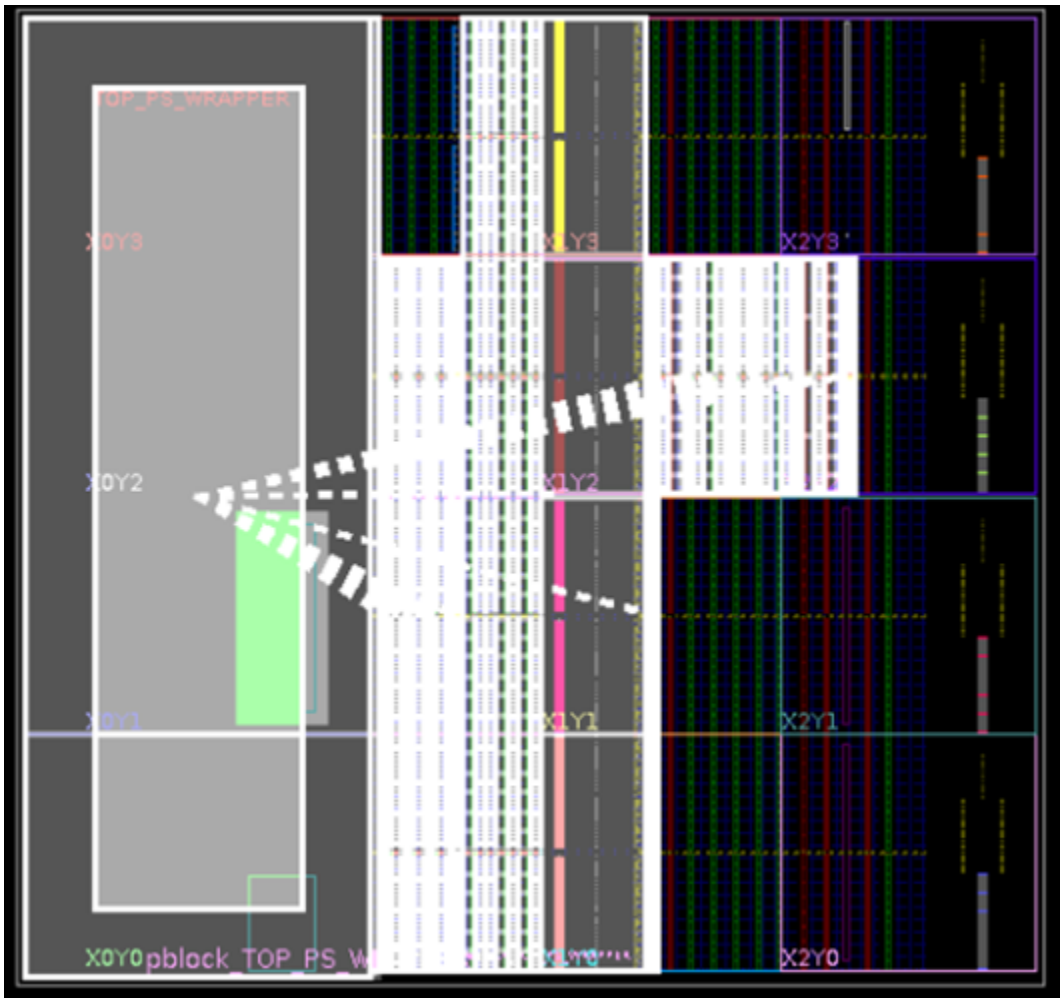
8. You will end up with **dis-continues pblock** after the preceding step. As there is a gap between Pblock rectangles, there are no routes from one rectangle to the other rectangle without having a **touch-down (PIP)** in non-Pblock interconnects. Hence routing tends to fail with the current floorplan. Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for more details on IDF rules for routing. Pblock must be made continuous. Include the resources between the two triangles by following Step 5. After completion, it looks like the following figure.

Figure 41: **After Making Pblock Continuous**



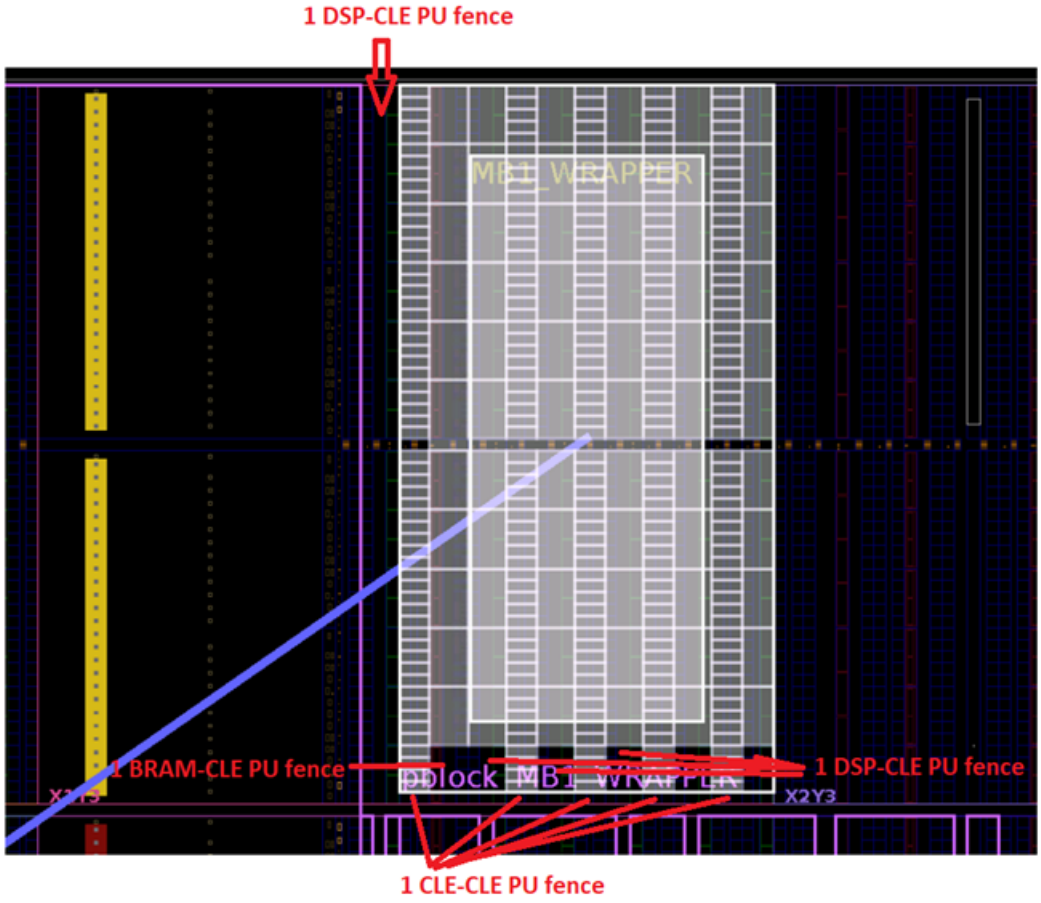
9. Include **DSP PUs** which are adjacent to **HPIO PUs** in clock-regions **X1Y0** and **X1Y1** by repeating Step 5. Include resources in clock region XOY3 which falls between HDIO PU and HPIO PU. The completed Pblock looks like the following figure.

Figure 42: **Pblock_TOP_PS_WRAPPER**



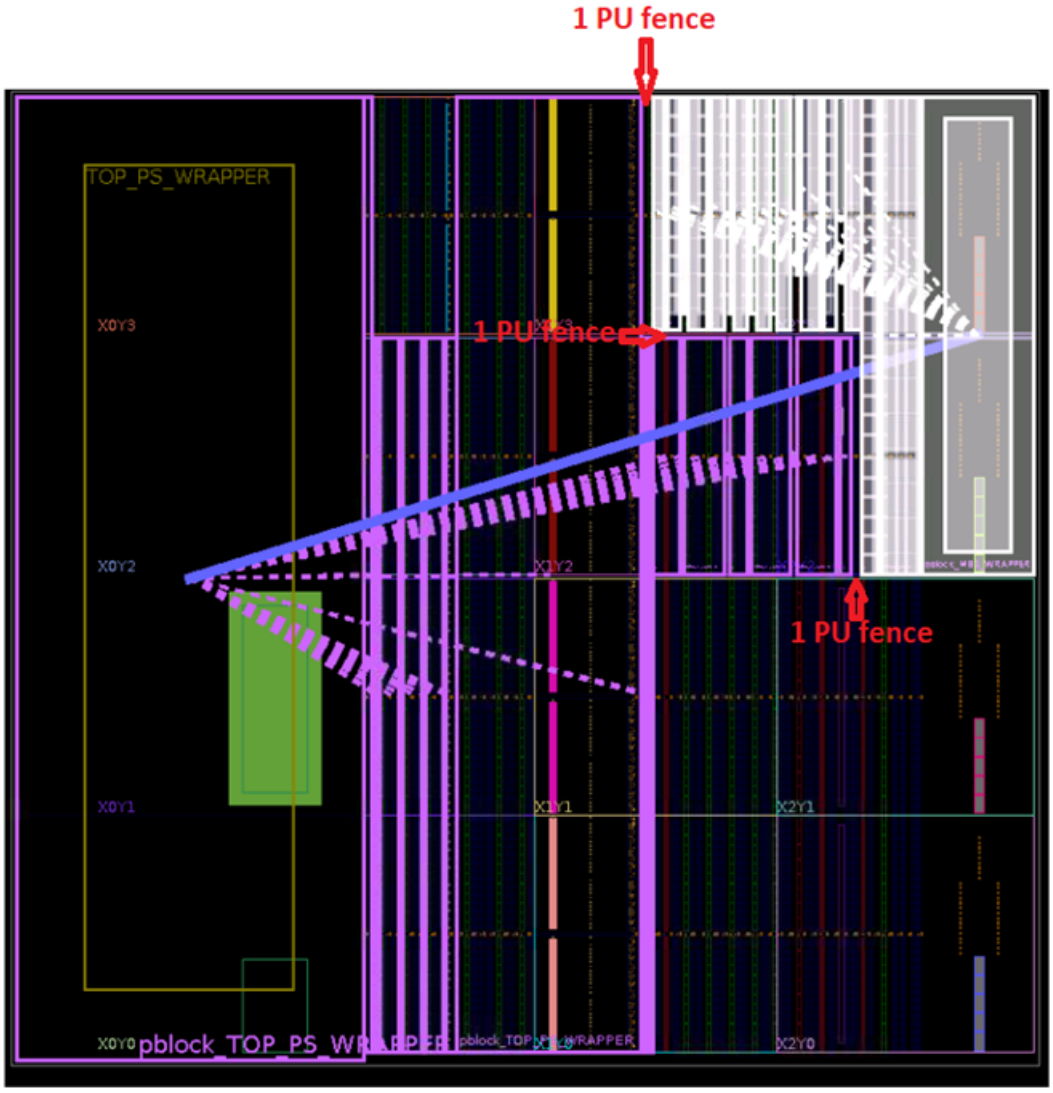
10. Create Pblock for MB1_WRAPPER by leaving valid fence with `Pblock_TOP_PS_WRAPPER`. Refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for more details on fence rules. Repeat step 1 and step 2 for MB1_WRAPPER by starting at the top side of the clock region X1Y3 after the HPIO PU leaves one PU for fence. Leave DSP-CLE PU column beside HPIO PU for fence and continue to select remaining sites in clock region X1Y3. Leave one PU fence at the bottom. There are two different types of PUs as fence, as shown in the following figure.

Figure 43: **Pblock_MB1_WRAPPER with Valid Fence**



11. Ensure to check **SNAPPING_MODE** of Pblock is **FINE_GRAINED** from the **Properties** tab in the **Pblock Properties** window.
12. Add more resources to the Pblock. Follow step 5 to perform this step. Add GT PUs in clock regions X2Y2 and X2Y3. Then add remaining resources in X2Y2 and X2Y3. Ensure to leave one PU fence with `pblock_TOP_PS_WRAPPER`. The following figure shows how the Pblock looks like after the completion of this step.

Figure 44: **pblock_MB1_WRAPPER with 1 PU Fence**



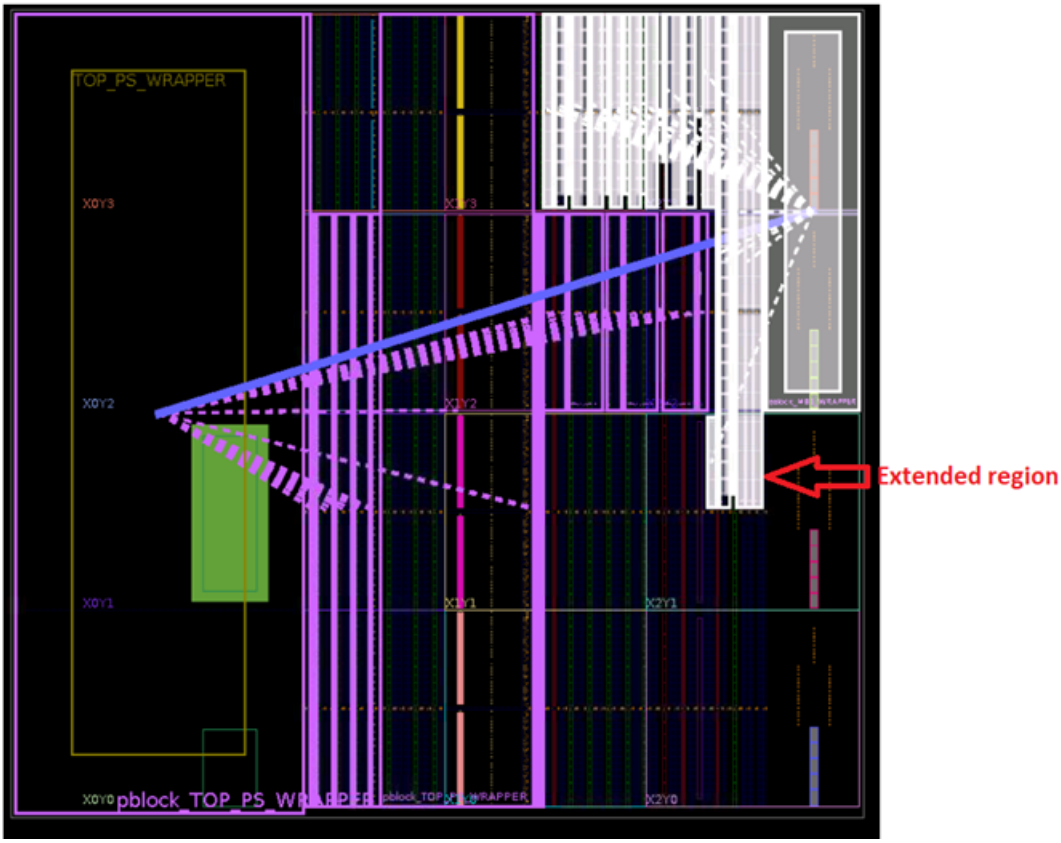
13. You will have to draw Pblocks for MB2_WRAPPER, MB3_WRAPPER, and MAILBOX_WRAPPER.MB1_WRAPPER has routes with both MB2_WRAPPER and MB3_WRAPPER. So Pblocks of MB2_WRAPPER and MB3_WRAPPER must be adjacent to pblock_MB1_WRAPPER otherwise routing may fail. With the current floorplan, it is tough to create two Pblocks adjacent to pblock_MB1_WRAPPER with valid fence between each Pblock. To solve this issue, we need to add more resources to pblock_MB1_WRAPPER. By following step 5, you can add the following ranges in the clock region X2Y1 to `pblock_MB1_WRAPPER`, as shown in the following figure. Alternatively, you can use the resize_pblock command to add these ranges (using the command on the next page).

```
{SLICE_X50Y91:SLICE_X51Y118
SLICE_X52Y91:SLICE_X59Y119
RAMB18_X2Y38:RAMB18_X2Y47
DSP48E2_X12Y38:DSP48E2_X12Y47}
```

To add resources using resize_Pblock, run the following command from TCL Console. You can refer to Figure 9.

```
resize_pblock pblock_MB1_WRAPPER -add {SLICE_X50Y91:SLICE_X51Y118
SLICE_X52Y91:SLICE_X59Y119 RAMB18_X2Y38:RAMB18_X2Y47
DSP48E2_X12Y38:DSP48E2_X12Y47}
```

Figure 45: **Complete pblock_MB1_WRAPPER**



14. Create pblock_MB2_WRAPPER for MB2_WRAPPER by following step 10 and including the following ranges.

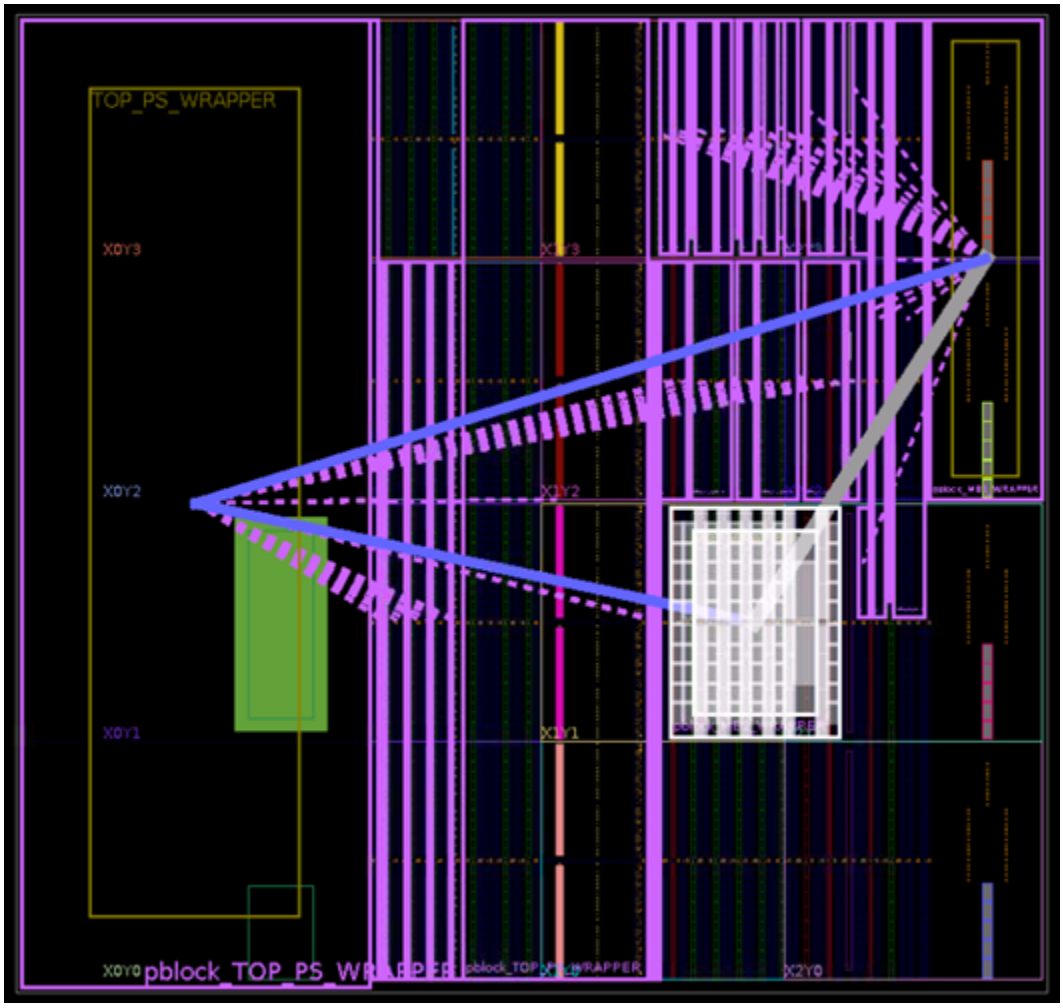
```
{ SLICE_X27Y61:SLICE_X47Y118
  BUFCE_LEAF_X152Y4:BUFCE_LEAF_X271Y7
  BUFCE_ROW_FSR_X29Y1:BUFCE_ROW_FSR_X59Y1
  DSP48E2_X7Y26:DSP48E2_X11Y45
  HARD_SYNC_X0Y2:HARD_SYNC_X3Y3
  RAMB18_X0Y26:RAMB18_X1Y45
  RAMB36_X0Y13:RAMB36_X1Y22
  URAM288_X0Y20:URAM288_X0Y27}
```

Alternatively, you can do the same from the TCL Console by running the following commands.

```
create_pblock pblock_MB2_WRAPPER
add_cells_to_pblock pblock_MB2_WRAPPER [get_cells [list design_1_i/
MB2_WRAPPER]]
resize_pblock pblock_MB2_WRAPPER -add {SLICE_X27Y61:SLICE_X47Y118
BUFCE_LEAF_X152Y4:BUFCE_LEAF_X271Y7
BUFCE_ROW_FSR_X29Y1:BUFCE_ROW_FSR_X59Y1
DSP48E2_X7Y26:DSP48E2_X11Y45 HARD_SYNC_X0Y2:HARD_SYNC_X3Y3
RAMB18_X0Y26:RAMB18_X1Y45 RAMB36_X0Y13:RAMB36_X1Y22
URAM288_X0Y20:URAM288_X0Y27}
```

***Note*:** Ensure **SNAPPING_MODE** of Pblock is **FINE_GRAINED** from the **Properties** tab of the **Pblock Properties** window.

Figure 46: **Complete pblock_MB2_WRAPPER**



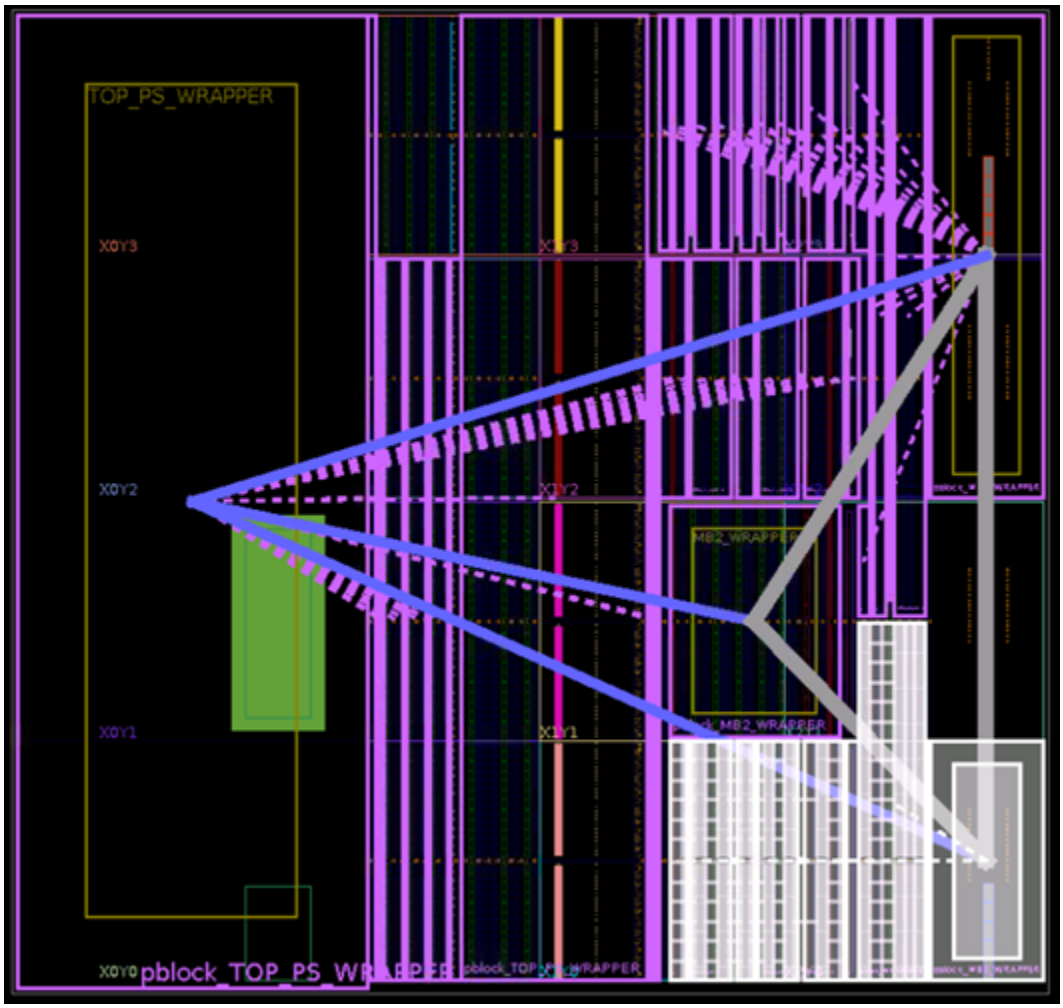
15. Create pblock_MB3_WRAPPER for MB3_WRAPPER by following step 10 and including following ranges. Ensure **SNAPPING_MODE** of Pblock is **FINE_GRAINED** from the **Properties** tab of the **Pblock Properties** window.

```
{ SLICE_X27Y0:SLICE_X59Y59
BUFCE_LEAF_X152Y0:BUFCE_LEAF_X335Y3
BUFCE_ROW_FSR_X29Y0:BUFCE_ROW_FSR_X71Y0
DSP48E2_X7Y0:DSP48E2_X12Y23
HARD_SYNC_X0Y0:HARD_SYNC_X5Y1
PCIE40E4_X0Y0:PCIE40E4_X0Y0
RAMB18_X0Y0:RAMB18_X2Y23
RAMB36_X0Y0:RAMB36_X2Y11
URAM288_X0Y0:URAM288_X0Y15
SLICE_X50Y60:SLICE_X59Y89
DSP48E2_X12Y24:DSP48E2_X12Y35
RAMB18_X2Y24:RAMB18_X2Y35
RAMB36_X2Y12:RAMB36_X2Y17
SLICE_X60Y0:SLICE_X60Y59
BUFCE_LEAF_X336Y0:BUFCE_LEAF_X343Y3
BUFCE_ROW_FSR_X72Y0:BUFCE_ROW_FSR_X72Y0
BUFG_GT_X0Y0:BUFG_GT_X0Y23
BUFG_GT_SYNC_X0Y0:BUFG_GT_SYNC_X0Y14
GTHE4_CHANNEL_X0Y0:GTHE4_CHANNEL_X0Y3
GTHE4_COMMON_X0Y0:GTHE4_COMMON_X0Y0}
```

Alternatively, you can do the same from the TCL Console by running the following commands.

```
create_pblock pblock_MB3_WRAPPER
add_cells_to_pblock pblock_MB3_WRAPPER [get_cells [list design_1_i/
MB3_WRAPPER]]
resize_pblock pblock_MB3_WRAPPER -add {SLICE_X27Y0:SLICE_X59Y59
BUFCE_LEAF_X152Y0:BUFCE_LEAF_X335Y3
BUFCE_ROW_FSR_X29Y0:BUFCE_ROW_FSR_X71Y0
DSP48E2_X7Y0:DSP48E2_X12Y23 HARD_SYNC_X0Y0:HARD_SYNC_X5Y1
PCIE40E4_X0Y0:PCIE40E4_X0Y0 RAMB18_X0Y0:RAMB18_X2Y23
RAMB36_X0Y0:RAMB36_X2Y11
URAM288_X0Y0:URAM288_X0Y15} -locs keep_all
resize_pblock pblock_MB3_WRAPPER -add {SLICE_X50Y60:SLICE_X59Y89
DSP48E2_X12Y24:DSP48E2_X12Y35 RAMB18_X2Y24:RAMB18_X2Y35
RAMB36_X2Y12:RAMB36_X2Y17} -locs keep_all
resize_pblock pblock_MB3_WRAPPER -add {SLICE_X60Y0:SLICE_X60Y59
BUFCE_LEAF_X336Y0:BUFCE_LEAF_X343Y3
BUFCE_ROW_FSR_X72Y0:BUFCE_ROW_FSR_X72Y0
BUFG_GT_X0Y0:BUFG_GT_X0Y23 BUFG_GT_SYNC_X0Y0:BUFG_GT_SYNC_X0Y14
GTHE4_CHANNEL_X0Y0:GTHE4_CHANNEL_X0Y3
GTHE4_COMMON_X0Y0:GTHE4_COMMON_X0Y0} -locs keep_all
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks pblock_MB3_WRAPPER]
```

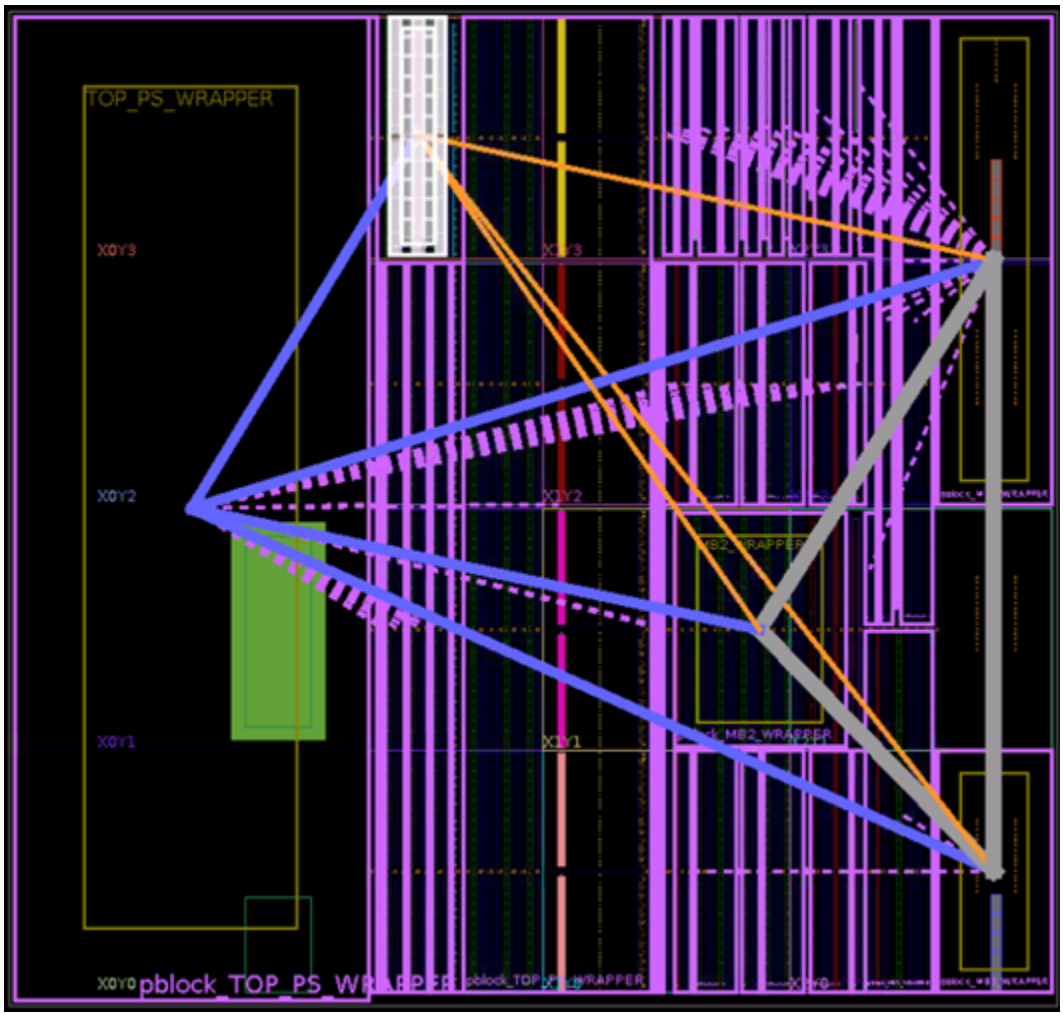
Figure 47: **Complete pblock_MB3_WRAPPER**



16. Create pblock_MAILBOX_WRAPPER for MAILBOX_WRAPPER by running the following commands from TCL Console.

```
create_pblock pblock_MAILBOX_WRAPPER
add_cells_to_pblock pblock_MAILBOX_WRAPPER [get_cells [list
design_1_i/MAILBOX_WRAPPER]] -clear_locs
resize_pblock pblock_MAILBOX_WRAPPER -add {SLICE_X2Y181:SLICE_X9Y239
BUFCE_LEAF_X16Y12:BUFCE_LEAF_X55Y15
BUFCE_ROW_FSR_X3Y3:BUFCE_ROW_FSR_X11Y3
DSP48E2_X1Y74:DSP48E2_X2Y95}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks
pblock_MAILBOX_WRAPPER]
```

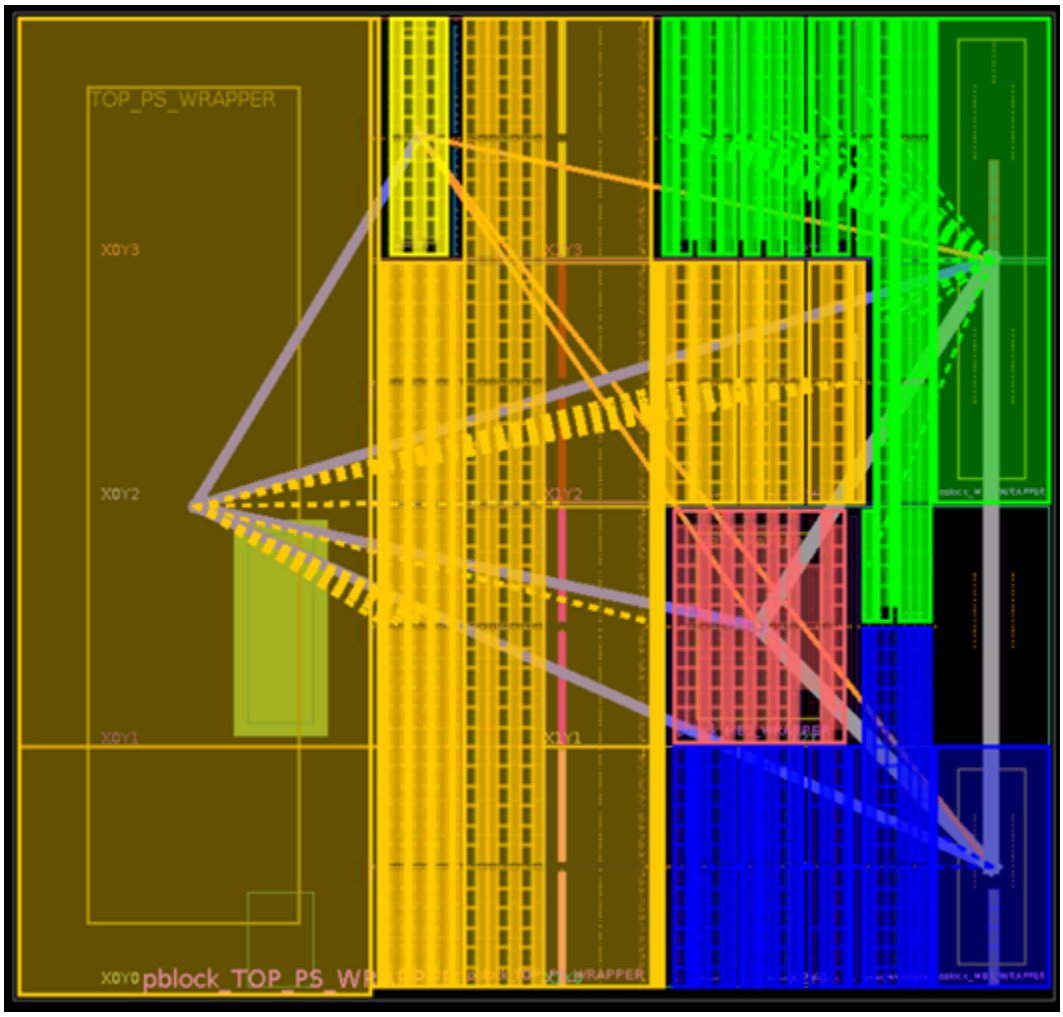
Figure 48: **Complete pblock_MAILBOX_WRAPPER**



17. Your floorplanning is completed here. Run the following command in the TCL Console to highlight Pblocks, as shown in the following figure.

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

Figure 49: **Device after Floorplan**

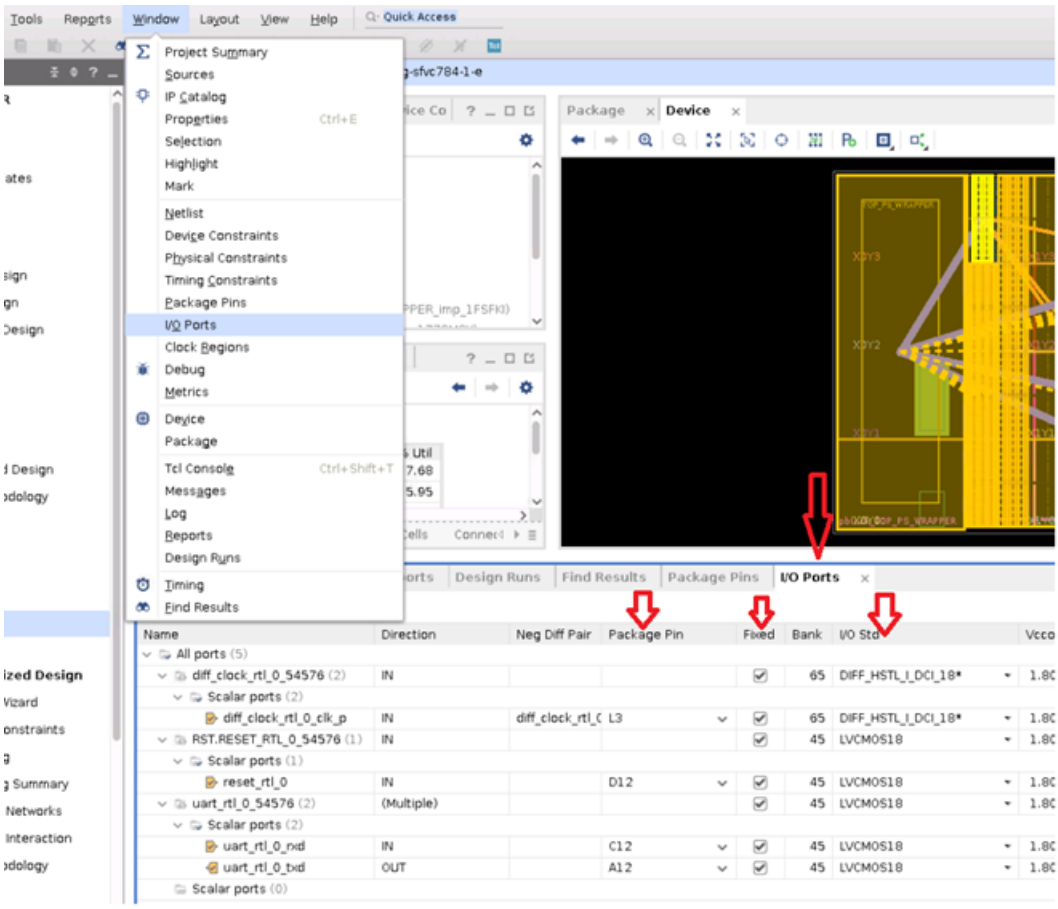


18. Assign **Package Pins**. Open **I/O Ports** tab from **Windows** menu, as shown in the following figure. Ensure **Fixed check box** is enabled. You may refer to *Isolation Design Flow for UltraScale + FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for more details on IDF rules for package pins assignment.

Table 1: **Package Pin Assignments**

Ports	I/O Std	Package Pin
reset_rtl_0	LVCMOS18	D12
diff_clock_rtl_0_clk_p	DIFF_HSTL_I_DCI_18	L3
uart_rtl_0_rxd	LVCMOS18	C12
uart_rtl_0_txd	LVCMOS18	A12

Figure 50: **I/O Ports and Package Pins**

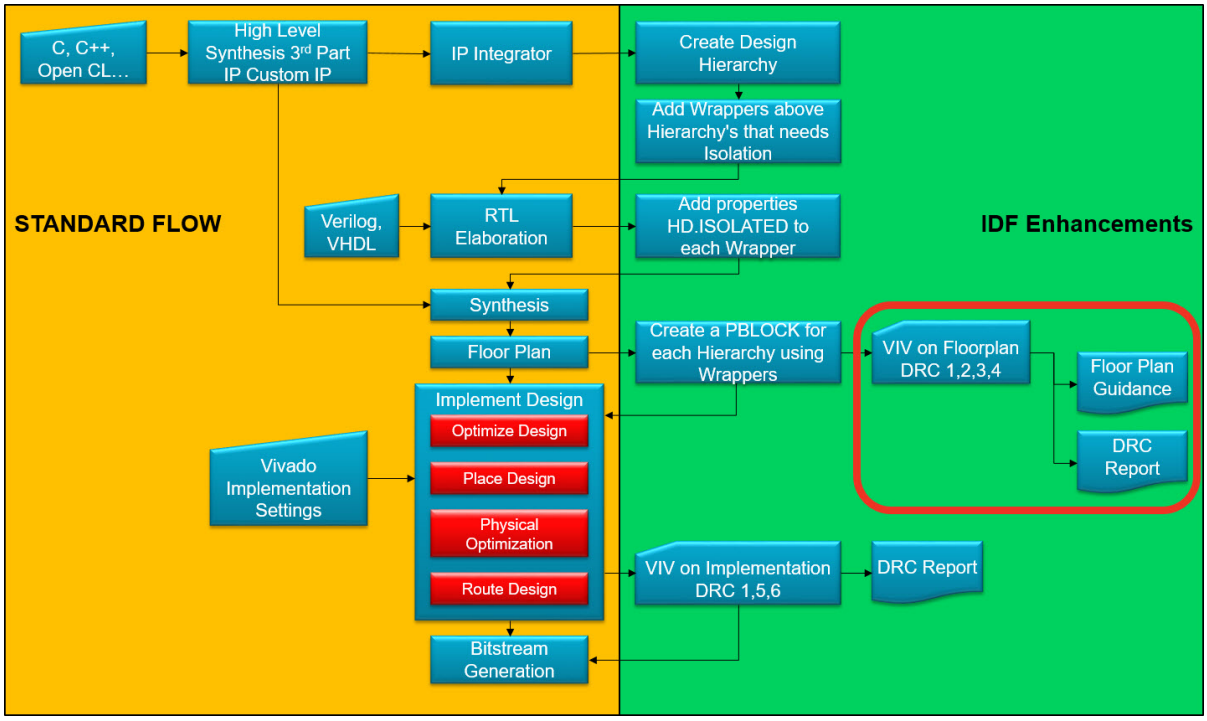


19. **Save** the design.

Phase 6

This phase, as depicted in the following figure, verifies the creation of the fence by checking the Pblock placement, and I/O bank violations caused by incorrect pin assignments.

Figure 51: **Phase 6 - VIV 2.0 Floor Planning DRC Run**



Vivado Isolation Verifier (VIV) 2.0 is used to run four design rule checks (DRCs) (DRC #1, #2, #3, and #4), as shown in the following figures. Refer to the*Vivado Isolation Verifier User Guide* (UG1291) for details on the DRCs.

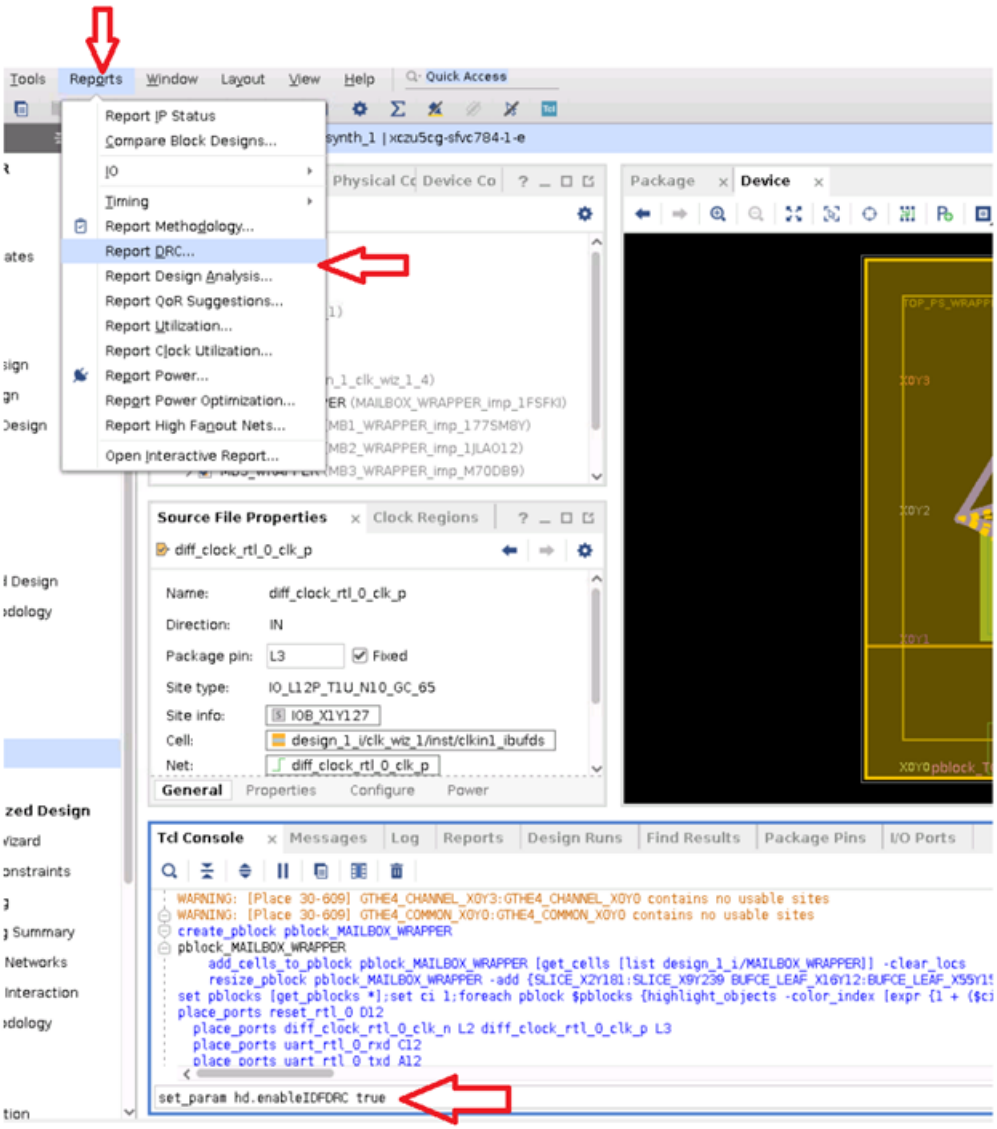
Follow the steps listed here for running the VIV DRCs. Alternatively, you can run phase6 Tcl script from Tcl Console by typing `source ./lab_phase6.tcl`.

1. Enable **VIV DRCs** by running the following command from the TCL Console.

```
set_param hd.enableIDFDRC true
```

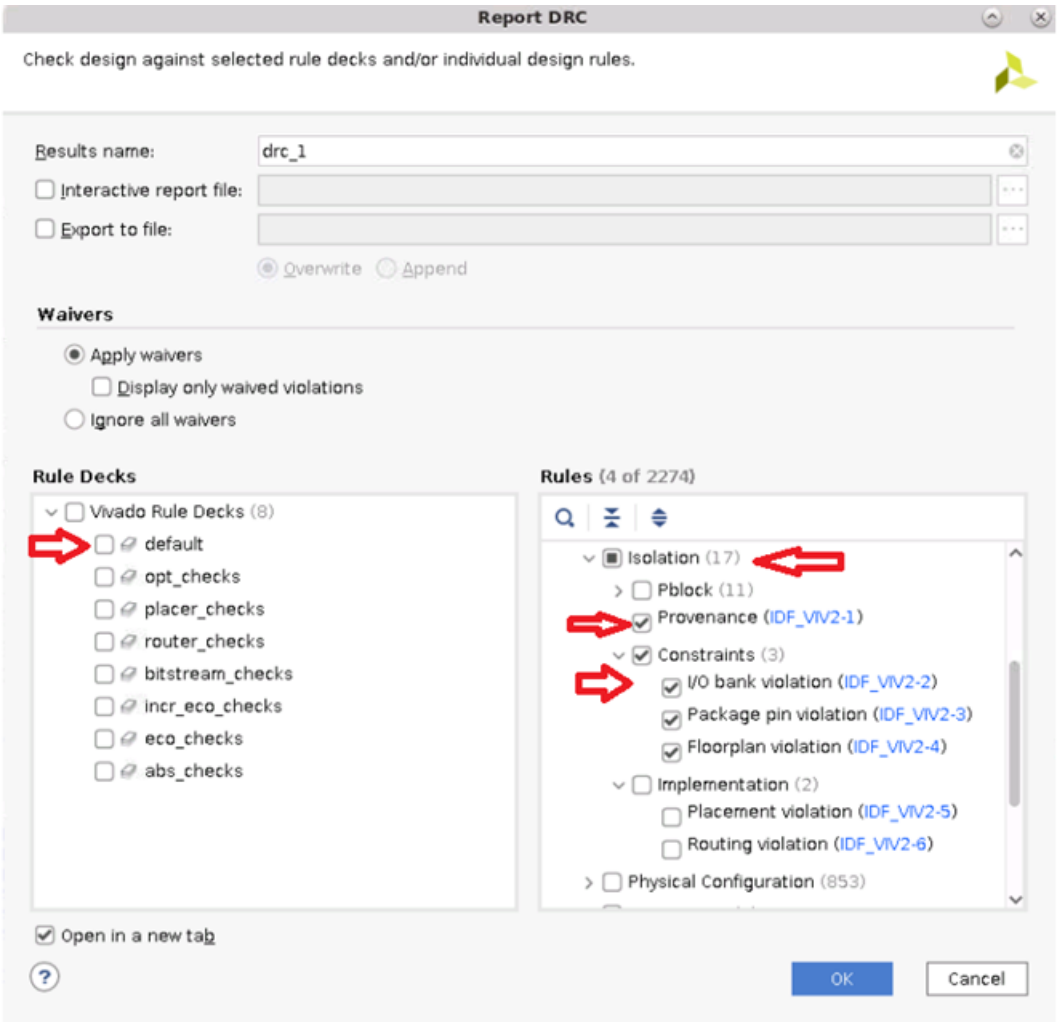
2. Open **Report DRC...** from the **Reports** window, as shown in the following figure.

Figure 52: **Reports > Report DRC...**



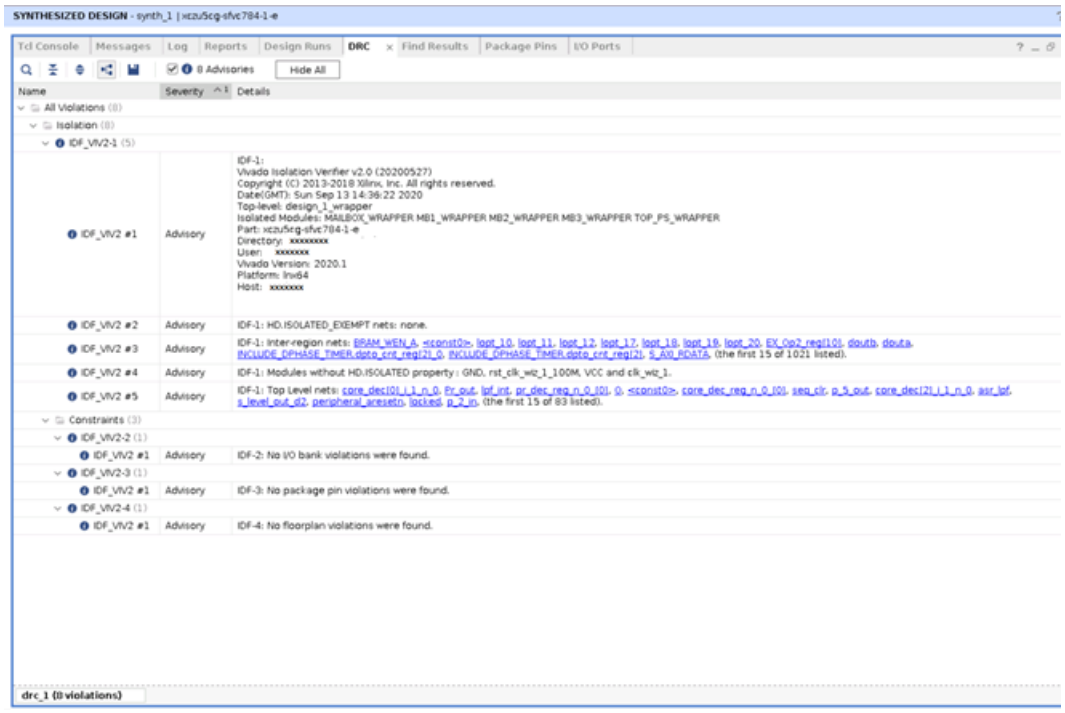
3. Uncheck **default** under **Rule Decks** in the **Report DRC** window. Under **Rules**, expand **Isolation**, and then check **Provenance** (IDF_VIV2-1) and **Constraints** (IDF_VIV2-2, IDF_VIV2-3, IDF_VIV2-4). Click **OK** to run the DRC.

Figure 53: **Report DRC Window**



4. DRC tab opens after completion. Maximized DRC view looks like the following figure.

Figure 54: **VIV 2.0 DRC 1, 2, 3, and 4 Report**



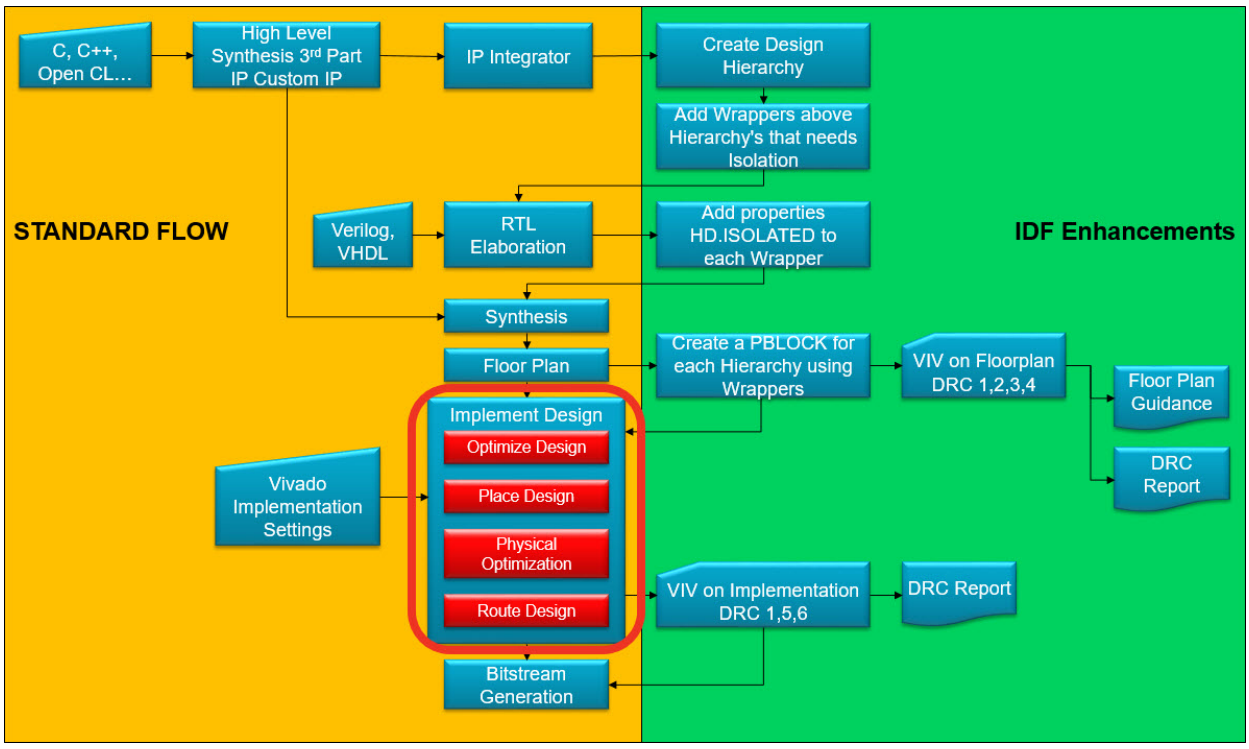
The report shows information on the device provenance (IDF_VIV2-1) and the negative results for constraints testing IDF_VIV2-2 I/O bank violation, IDF_VIV2-3 package pin violation, and IDF_VIV2-4 floorplan violation. No IDF violations will be seen in the DRC report of this design. Refer to the *Vivado Isolation Verifier User Guide* (UG1291) for DRC details.

Phase 7

This phase implements the design by:

1. Optimizing the design within the constraints of the Xilinx isolation design flow (IDF).
2. Placing the design based on the Pblocks.
3. Routing the design.

Figure 55: **Phase 7 - Implementation**



Run the following command from the Tcl Console to disable buffer insertion for GND and VCC, before running the implementation. The setting of the following parameter is not mandatory for the IDF flow, but it is needed specifically for this lab design.

```
set_param hd.BufferInsertion false
```

You need to disable buffer insertion `param` when **ISO** module **boundary net** is connecting to VCC/GND nets. When this `param` is set to false, then no buffers will be inserted for VCC/GND nets. If there is a ISO module **boundary net** which is connected to VCC/GND and the buffer insertion `param` is enabled, then the tools will insert buffers and connect VCC/GND nets across the different Isolated boundaries and thus routing will fail with Multi Region Net DRC error.

Click **Run Implementation** under the **Implementation** menu, on the left of the Vivado GUI. Alternatively, you can run phase7 Tcl script from the Tcl Console by typing `source ./lab_phase.7tcl`.

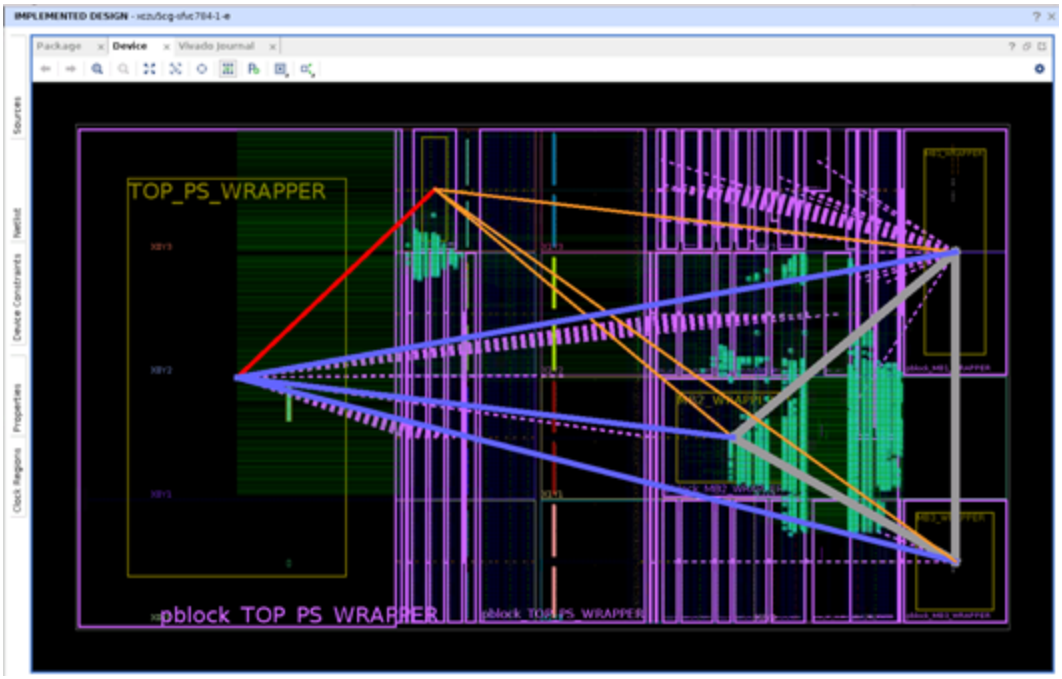
Save the project or constraints if asked. If any constraints were modified, the Vivado tool requests starting from Synthesis. When implementation is complete, select the **Open Implemented Design** option and click **OK**. The device view pops up and looks like the following figure. If you are asked to close the Synthesized Design before opening the Implemented Design, select **Yes**. This is a good practice to follow because memory might be limited.

Routing resources view is the default view when opening a routed design. However, it can be turned on and off by selecting the button in the top of the Device View. You may refer to Figure 33 for further details. Gaps in the floorplan appear in this view, because the tools do not consider the interconnected tiles associated with all user tiles as part of the Pblock. This is visual only; the gaps do not exist. In this view, you can track a schematic net to the routed net. This is a very powerful view when tracing timing issues.

Also, from this view you can manually route any component or net as you wish. This mode is the Vivado replacement to the ISE FPGA editor. The replacement is significantly more user friendly.

It is possible for routes, not touchdowns, from one isolated region to cross over into another isolated region, if the region in question does not have any routing in that area. This allows for maximum flexibility to the router while still obeying IDF rules for isolation. This only happens in designs where sparsely populated isolated regions are adjacent to regions that are densely populated. No placement or touchdowns are ever allowed outside the intended isolated region. You may refer to *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) for details on IDF routing.

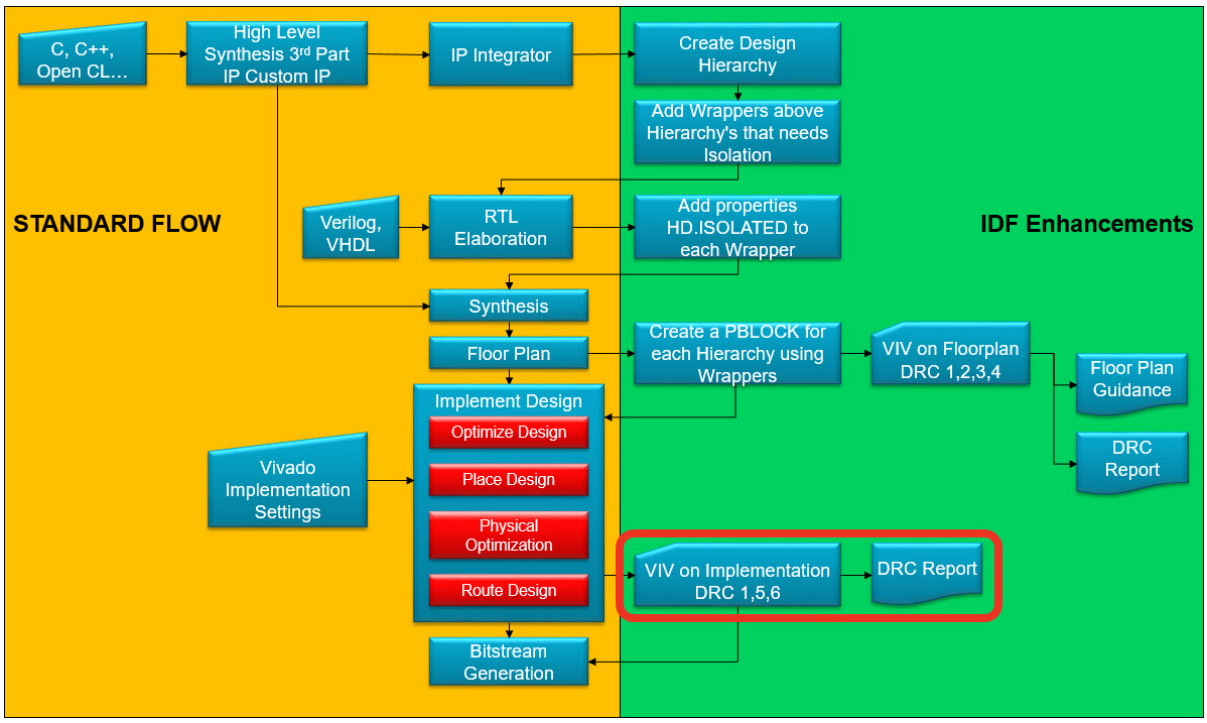
Figure 56: **Implemented Design - Device View**



Phase 8

After implementation completes, use the Vivado® Isolation Verifier (VIV) 2.0 tool to run a final set of design rule checks to verify both the placement and routing complies with the IDF rules.

Figure 57: **Phase 8 - VIV Placement and Routing DRC Run**

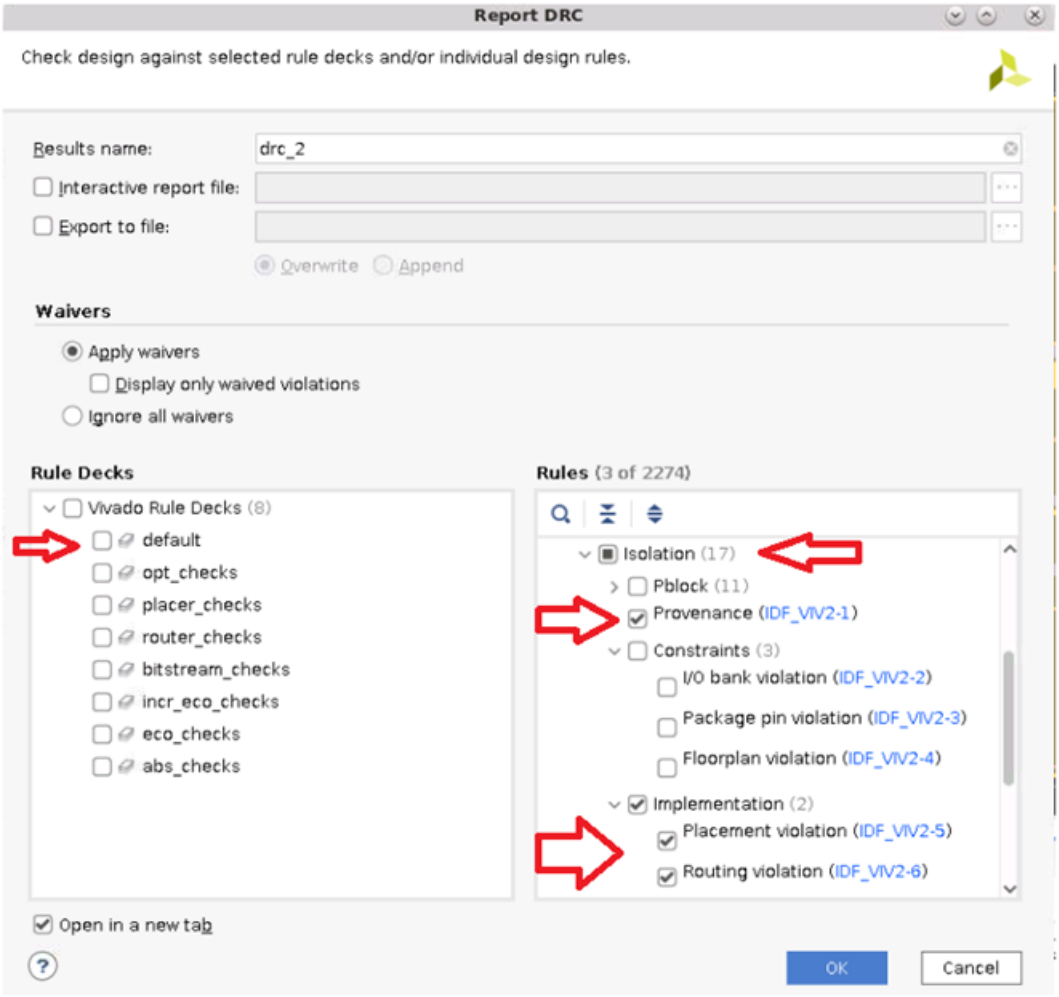


Re-run DRC #1 to maintain the provenance in the report, followed by DRC #5 and DRC #6, which are run on the finished design. Refer to *Vivado Isolation Verifier User Guide* (UG1291) for DRC details.

Follow the steps listed here to execute the VIV DRCs. Alternatively, you can run phase8 Tcl script from Tcl Console by typing `source ./lab_phase8.tcl`.

1. Open the **Report DRC** window by following steps mentioned in Phase 6 and Select **Provenance** (IDF_VIV2-1) and **Implementation** (IDF_VIV2-5, IDF_VIV2-6) and click **OK**.

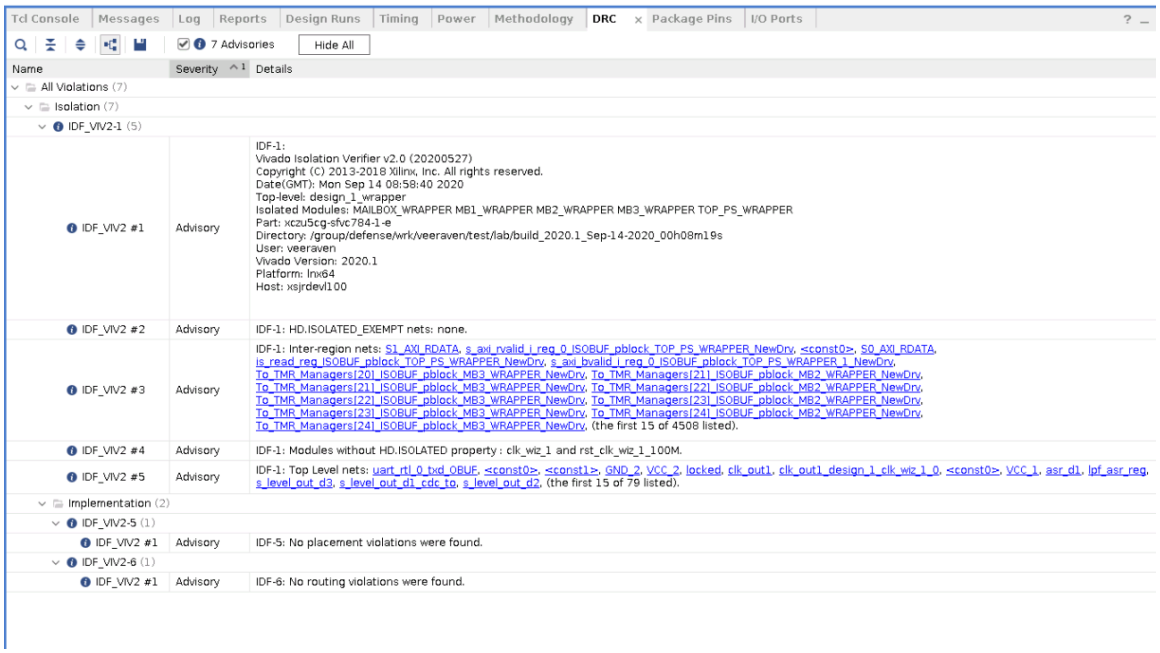
Figure 58: **Report DRC Window for IDF_VIV2-1 IDF_VIV2-5 IDF_VIV2-6**



The result after VIV 2.0 is run is shown in the following figure.

***Note*:** No IDF violations will be seen in the DRC Report for this design. Refer to *Vivado Isolation Verifier User Guide* (UG1291) for DRC detail.

Figure 59: **VIV 2.0 DRC 1, 5, and 6 Report**



Reference Design

The reference design file contains example Tcl scripts and can be downloaded from the Xilinx website. The following table shows the reference design matrix.

Table 2: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target devices	Zynq UltraScale+ devices
Source code provided?	Yes
Source code format (if provided)	Tcl
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	N/A
Simulation	
Functional simu v2018.3lation performed	N/A
Timing simulation performed?	N/A
Test bench provided for functional and timing simulation?	N/A
Test bench format	N/A
Simulator software and version	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Isolation Design Flow (IDF) and Vivado Design Suite v2018.3
Implementation software tool(s) and version	Isolation Design Flow (IDF) and Vivado Design Suite v2018.3
Static timing analysis performed?	N/A

Table 2: **Reference Design Matrix** *(cont'd)*

Parameter	Description
Hardware Verification	
Hardware verified?	N/A
Platform used for verification	N/A

Conclusion

This application note provides a step-by-step example to implement a complete Zynq UltraScale+ MPSoC isolated design. All of the necessary IDF steps are shown, and the rules and guidelines detailed in the *Isolation Design Flow for UltraScale+ FPGAs and Zynq UltraScale+ MPSoCs* (XAPP1335) are highlighted. This lab provides Tcl scripts for each phase. You can run each phase step-by-step or source each phase script as outlined in this application note.

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
04/12/2021 Version 2.0	
Figure 1	Added a new figure.
Phase 1	Added steps to generate and build the project along with graphics.
Phase 2	Added more details on net splitting along with graphics.
Phase 3	Added steps for RTL elaboration and to set HD.ISOLATED property along with graphics.
Phase 4	Added steps for demonstrating synthesis along with graphics.
Phase 5	Added steps for further clarity on floorplanning along with graphics.
Phase 6	Added steps for running VIV DRCs along with graphics.
Phase 7	Added steps for implementation along with graphics.
Phase 8	Added steps for VIV placement and routing DRC Run along with graphics.
Conclusion	Provided Tcl scripts that allows the user to single-step through the design process.
02/15/2019 Version 1.0	
Initial release.	N/A

Documentation Navigator and Design Hubs

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

- From the Vivado® IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.

- At the Linux command prompt, enter `docnav`.

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

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

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

References

1. *Isolation Design Flow for Zynq UltraScale+ Application Note* (XAPP1335)
2. *Vivado Isolation Verifier User Guide* (UG1291)
3. Isolation Design Flow website
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909)
6. *Vivado Design Suite User Guide: Hierarchical Design* (UG905)
7. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
8. *Triple Modular Redundancy (TMR) LogiCORE IP Product Guide* (PG268)
9. *Aerospace and Defense Security Monitor IP Core Product Marketing Brief*

Please Read: Important Legal Notices

www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright