XAPP1256 (v1.1.1) March 21, 2016

Zynq-7000 AP SoCs or 7 Series FPGAs Isolation Design Flow Lab (Vivado Design Suite)

Author: Ed Hallett

Summary

This lab application note describes the creation and implementation of a single chip cryptography (SCC) system using redundant Keccak hash modules with compare logic. 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®-7000 All Programmable SoC device for the example SCC solution. Even though this application note explains how to implement a design using the IDF for a Zynq-7000 device, the same process can be used to implement an IDF design using any 7 series FPGA device.

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.

This application note is similar to the application note *7 Series Isolation Design Flow Lab Using ISE Design Suite 14.4* (XAPP1085) [Ref 1] with the primary difference being this document is specific to using the Xilinx Vivado® Design Suite for Zynq-7000 AP SoC devices, whereas XAPP1085 is specific to using the Xilinx ISE® Design Suite for developing IDF designs for 7 series FPGA devices. The rules for IDF defined in this application note do not differ from those defined in XAPP1085, but the methodology for implementation using Vivado tools does.

This application note is accessible from the Xilinx Isolation Design Flow website [Ref 2].

You can download the Reference Design Files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design Files, page 43.

Introduction

The Isolation Design Flow is the software methodology that allows for SCC 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: Procedures will work with Vivado versions 2015.2 and later. However, this lab targets 2015.2 and versions beyond this might have slightly different screen images.

Lab Design Overview

The 7 series and Zynq-7000 IDF rules are outlined in *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)* (XAPP1222) [Ref 3]. Though the rules for IDF do not differ between ISE and Vivado, the methodology for implementation using Vivado tools does differ.

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 isolated, redundant Keccak hash modules with a compare block. Figure 1 is a hierarchical diagram of the various VHDL sub-blocks used in the implementation of this design.

IMPORTANT: *Use Vivado Integrated Design Environment (IDE) 2015.2 or later for this lab.*

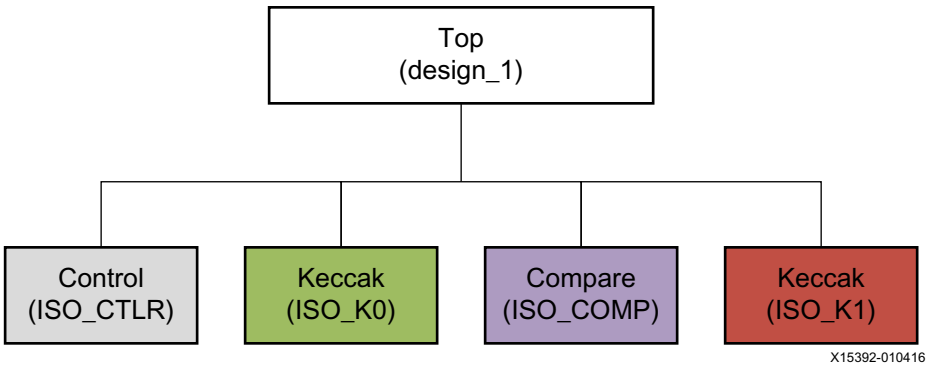


Figure 1: **Design Hierarchy Block Diagram**

Figure 2 shows the flow used during the course of this lab. The fundamental goal is to give you an idea of what the methodology looks like in Vivado tools and how tools such as Vivado IP integrator (IPI) can be of significant help (see *Vivado Design Suite User Guide - Designing IP Subsystems Using IP Integrator* (UG994) [Ref 4]).

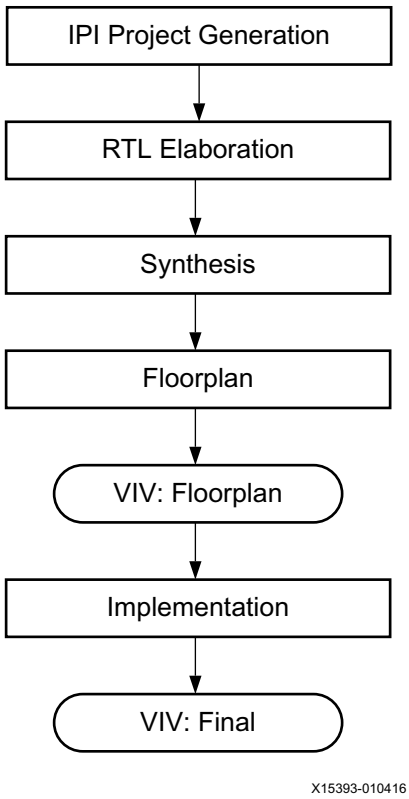


Figure 2: **Vivado IDF Lab Flow Using IP Integrator as the Primary Source Generator**

Figure 3 shows the floorplan for the lab design as implemented in an xc7z020clg484-1 device. It consists of four area groups. The first is an area group that contains the PS7 site (ARM® Dual Core A9 Processing System) and some additional space to route as needed, such as Advanced eXtensible Interface (AXI) signals. The other three represent a typical redundant system with compare module whose clocks and resets come from the processor.

IMPORTANT: *When putting the PS7 in an area group, at the very minimum, create a fence on the full right side of the PS7 block and include the first CLB tile on the lower right corner under the PS7 block. See Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools) (XAPP1222), PS Fence Tile section, for more details [Ref 3].*

Signals from the PS7 can only get to the FPGA fabric on the right side of the PS7 site. If you need to communicate to an area group below the PS7, you need to have added some of the fabric on the right and bottom to allow communication to and from the PS7.

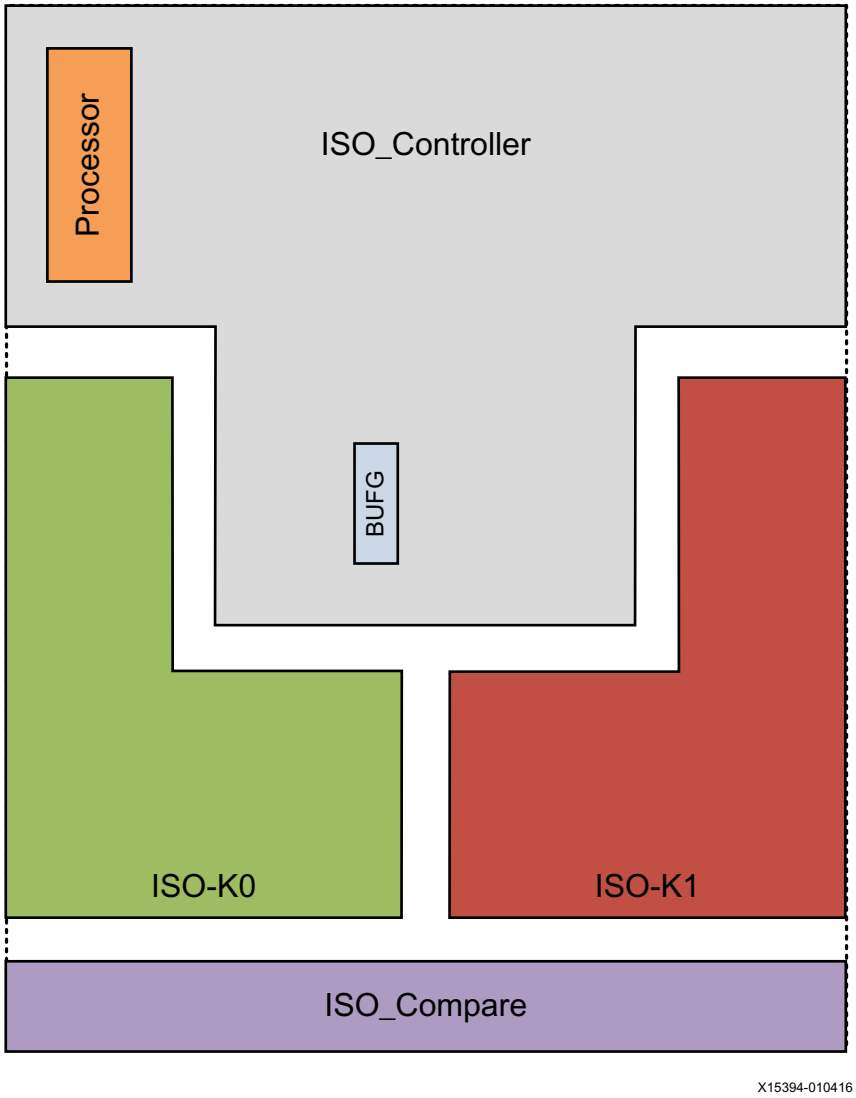


Figure 3: **Die View: IDF Lab Floorplan in an xc7z020clg484-1 Device**

Install Reference Design Files into Target Directories

These steps describe the process for installing the reference design files:

Note: For this lab, it is important that the design files are in a known location, in this case, `C:\xilinx_design`.

1. Extract the `xapp1256-idf-for-zynq-vivado.zip` file to your home drive letter (that is, `C:`).
2. The project files are placed in the `C:\xilinx_design\sources\` directory.

- `xilinx_design`
  - `sources`
    - `ip`

- viv
- xdc

When the ZIP file is extracted, a directory called `xilinx_design` holds the lab project design files. The directory structure shown above applies to that project.

Creating a Vivado IPI Project

This section describes the steps that take you through a bottom-up synthesis flow, using the Vivado Design Suite, which is the flow used for IDF designs to maintain isolation. Vivado release 2015.2 or later allows you either to create their designs manually or to import register-transfer level (RTL) source code directly into the project so the project can be done using just the Vivado tool. Refer to the Lab Flow Progression flow chart in Figure 4.

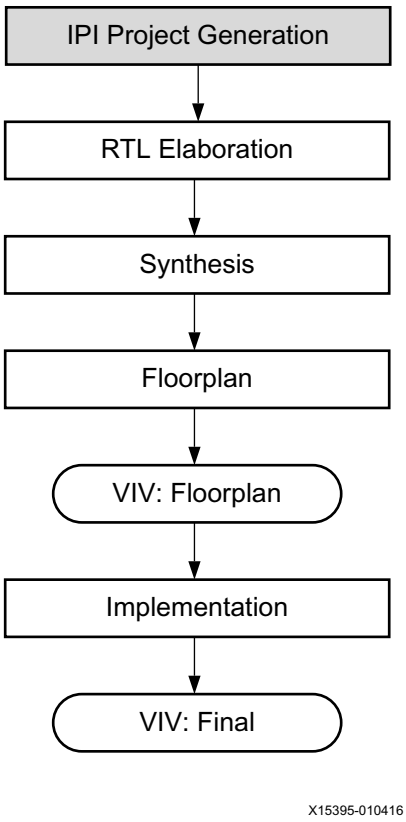


Figure 4: **Lab Flow Progression – Vivado IPI Project**

Project Entry in Vivado

Launch the Vivado IDE

To launch the Vivado tool, select **Start > All Programs > Xilinx Design Tools > Vivado 2015.2 > Vivado 2015.2.**

Vivado Project Creation

The Vivado tool works with any standard RTL source files. The regular guidelines are followed to generate a new project and import the RTL source files into the Vivado tool to create a floorplan for the design.

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 (see Figure 5). Select **Next**.

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

2. In the New Project - Project Name window enter:
   - Project name: For this lab, the **idfLab** project name is used.
   - Project location: `C:/xilinx_design`

This directory does not exist and is created by the Vivado tool.

Note: The Vivado tool automatically changes the Windows directory path separator from \ to /.

- Check **Create project subdirectory**.

3. Click **Next**.
4. Select **RTL Project** and check **Do not specify sources at this time** (Figure 6).

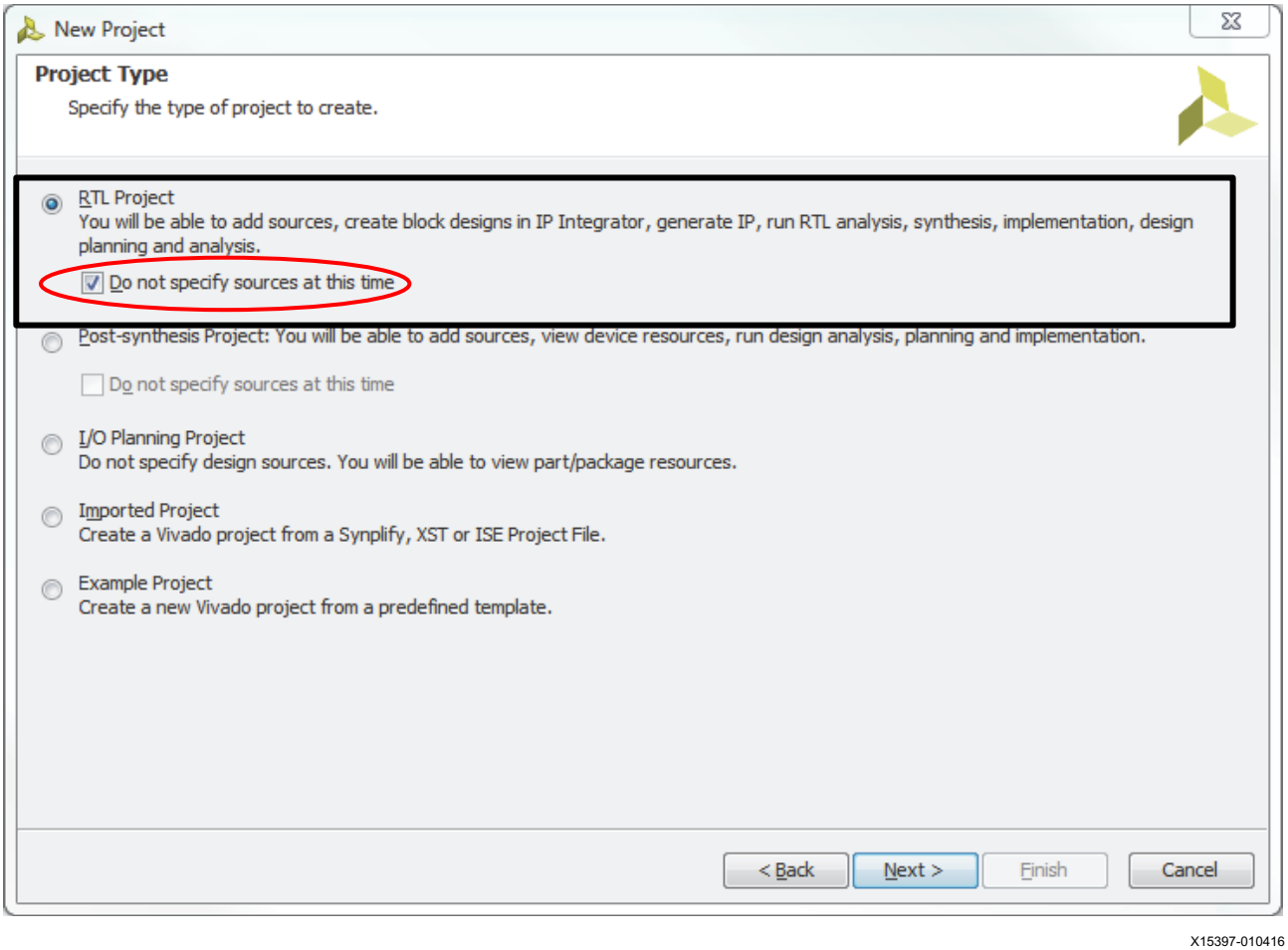


Figure 6: **Vivado New Project - Project Type Window**

Note: RTL sources are added later.

5. Click **Next**.

6. In the Default Part window, select the appropriate product filters for this lab as listed here and shown in Figure 7.

Product category	**General Purpose**
Family	**Zynq-7000**
Sub-Family	**Zynq-7000**
Package	**clg484**
Speed grade	**-1**
Temp grade	**C**
Si Revision	**All Remaining**

7. Select the **xc7z020clg484-1** device.
8. Click **Next** and **Finish**.

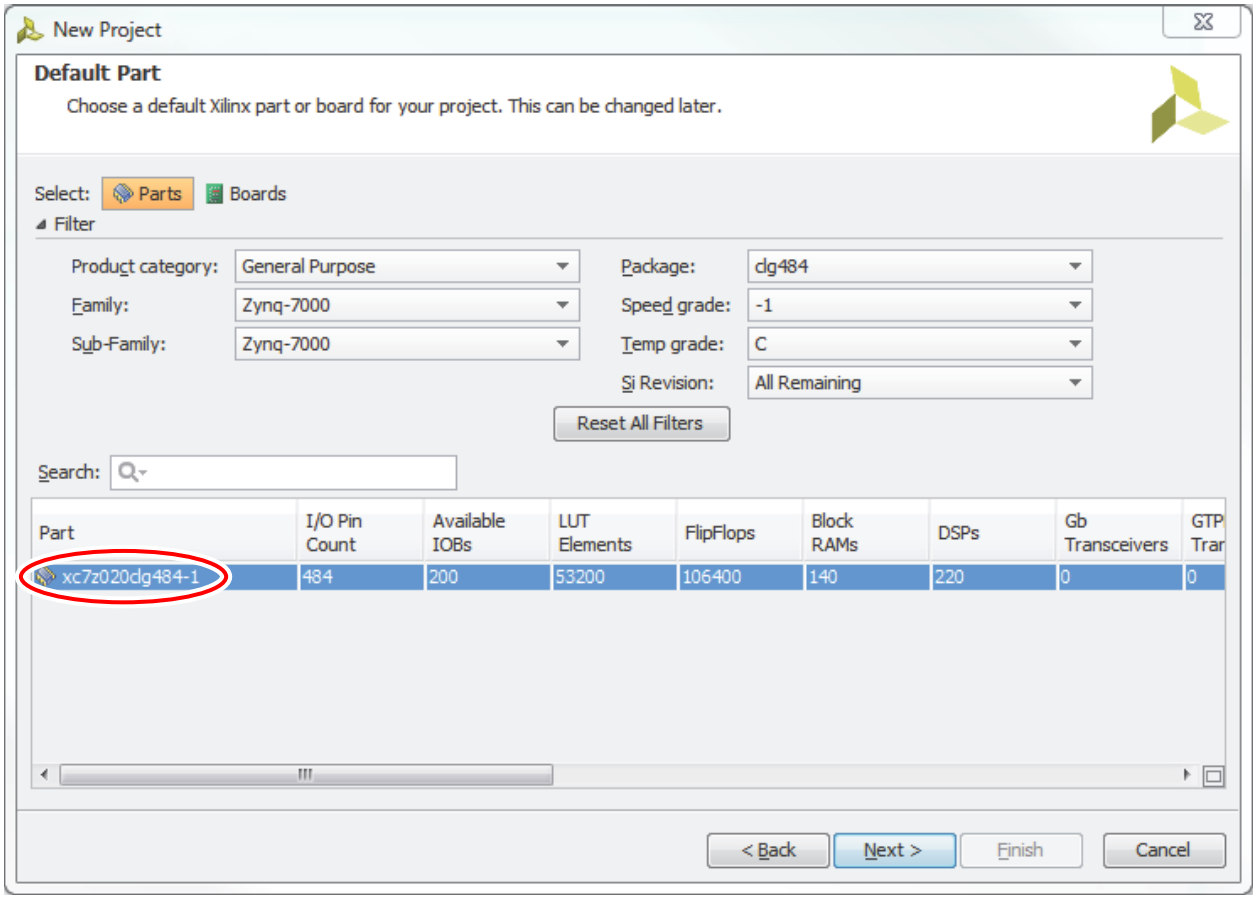
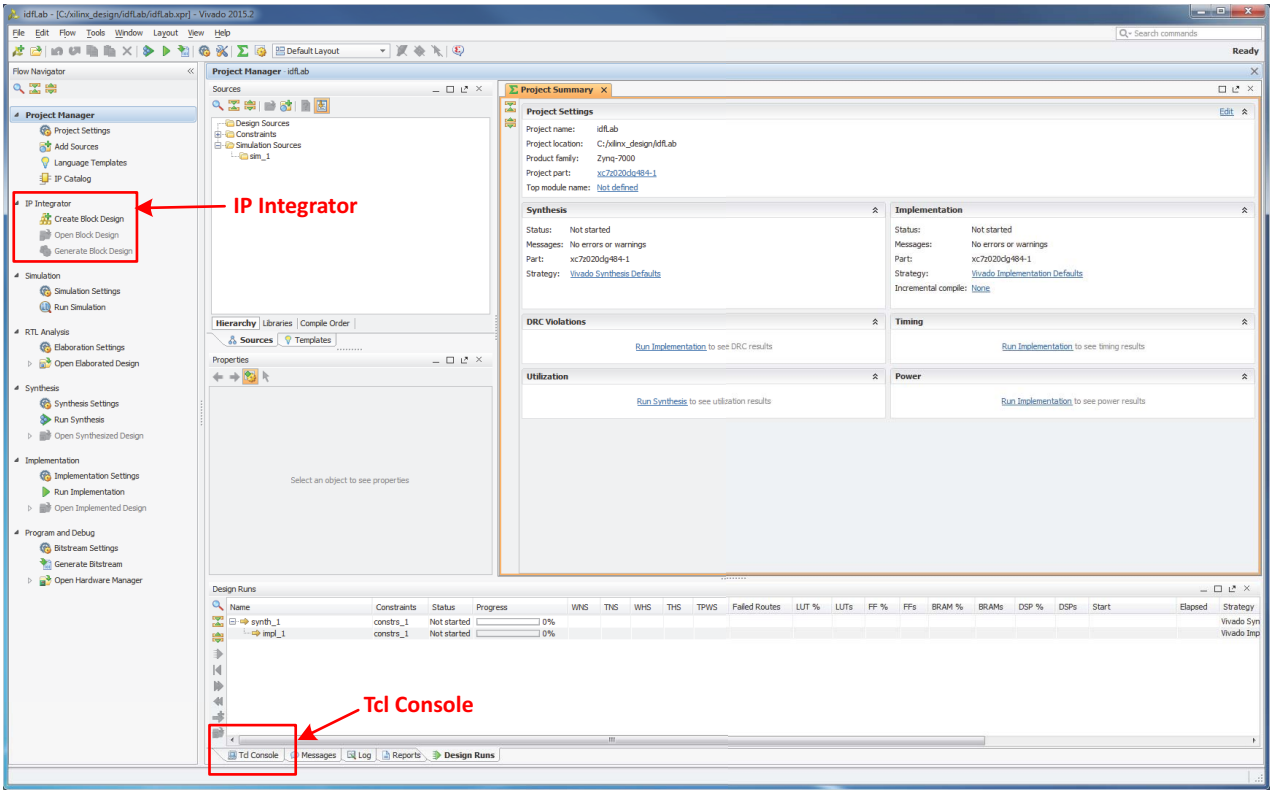


Figure 7: **Vivado New Project – Default Part Window**

9. The Vivado project now is created. Figure 8 shows the Project Manager window for the idfLab project.



X15399-010816

Figure 8: **PlanAhead Project Manager View**

Building the IP Integrator (IPI) Project

In this lab, you build a system using existing IP. Because some of this IP is custom, it is necessary to tell Vivado IPI where the custom IP is located so it can import that IP into the IP library.

1. Select the **Tcl Console** tab at the bottom left of the Vivado GUI (Figure 8).
   a. Type **cd c:/xilinx_design** on the Tcl console line. This is important because future commands in this lab rely on this being the active directory.
   b. Type **set_property ip_repo_paths ./sources/ip [*current_fileset*]** on the Tcl console line. This command sets the path to point to the location of the custom IP.
   c. Type **update_ip_catalog** on the Tcl console line. This command refreshes the IP repositories with the user IP repository.

      Note: These steps can also be accomplished in the GUI by going to IP settings in the Project Settings found in the Project Manager section on the top left of the Vivado GUI.

The next step is to create an IPI block design.

2. To create the block design in the IPI tool, navigate to the Project Manager pane on the left and select **IP Integrator > Create Block Design** (Figure 8).
3. Keep the default design name **design_1** and select **OK** (Figure 9).

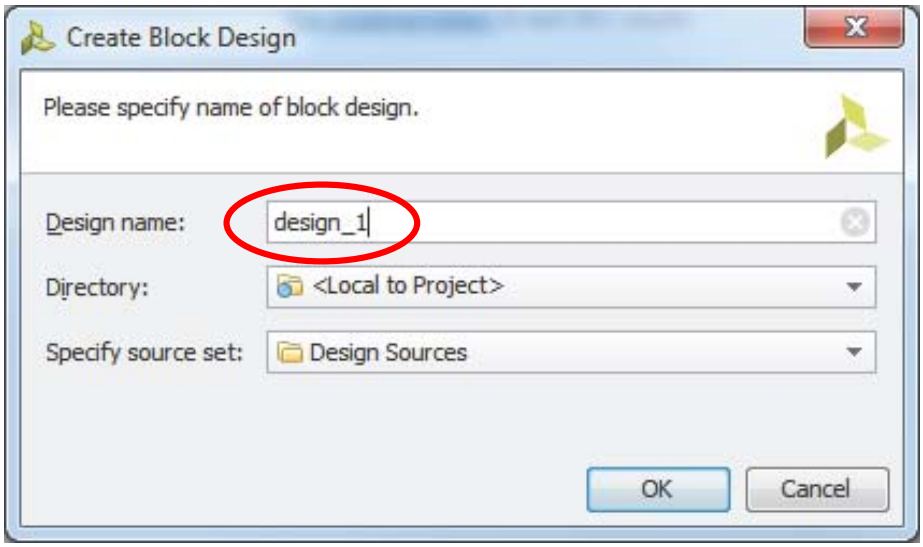


Figure 9: **Create Block Design > Set Design Name**

4. You should see a blank diagram (Figure 10).

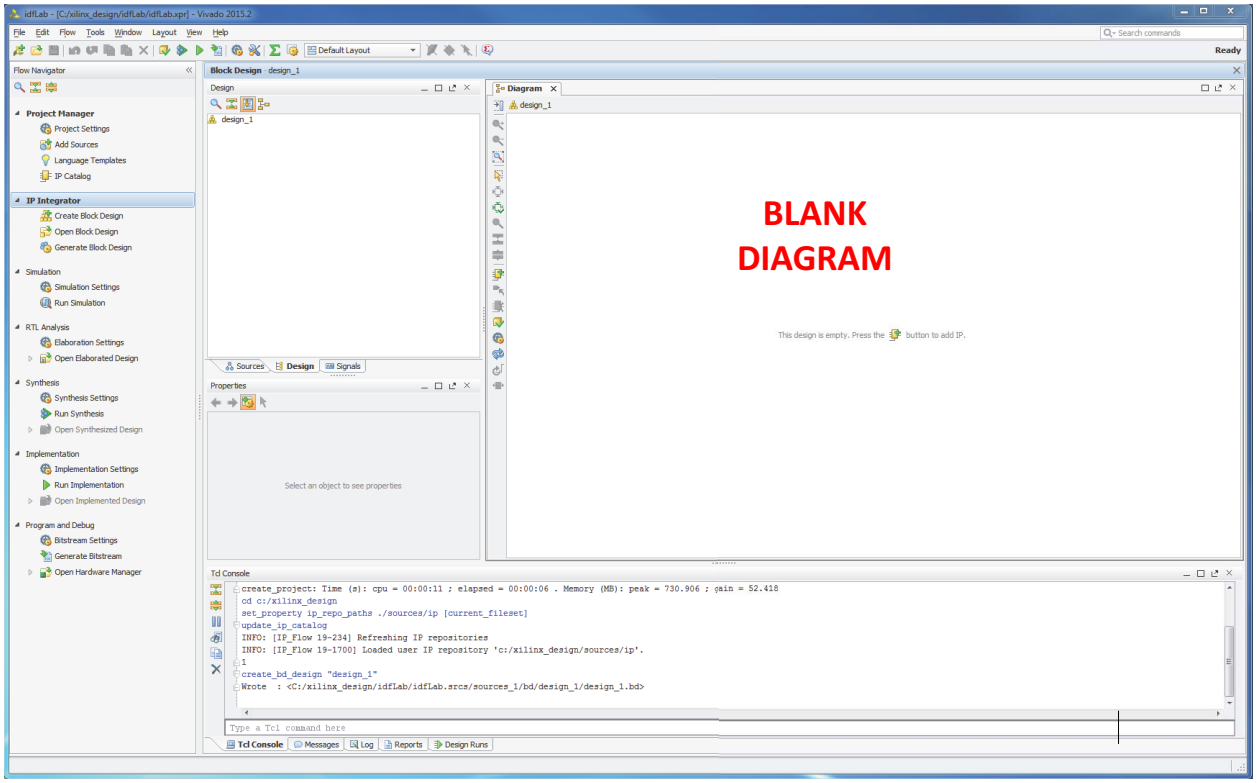


Figure 10: **IPI Block Diagram**

Creating an IPI Block Design

The purpose of using IPI in this lab is to demonstrate the ease with which a hierarchy can be added to a flat design (one not originally intended to be implemented using IDF). Additionally, it turns creating a redundant design into a copy/paste operation.

1. Right-click the **Diagram** canvas and select **Add IP**.
2. Scroll to the **ZYNQ7 Processing System** IP, select it, and press **Enter** (Figure 11 and Figure 12).

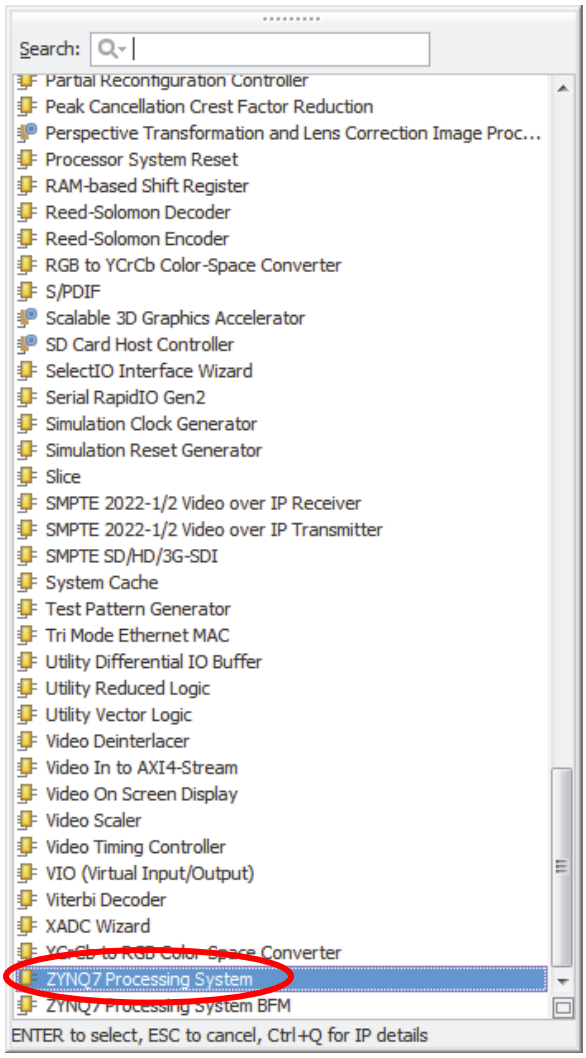


Figure 11: **Add IP > ZYNQ7 Processing System**

3. Repeat step 1 for the **keccakCompare_v1_0** IP (Figure 12).
4. Repeat step 1 for the **keccak_v1_0** IP (Figure 12).

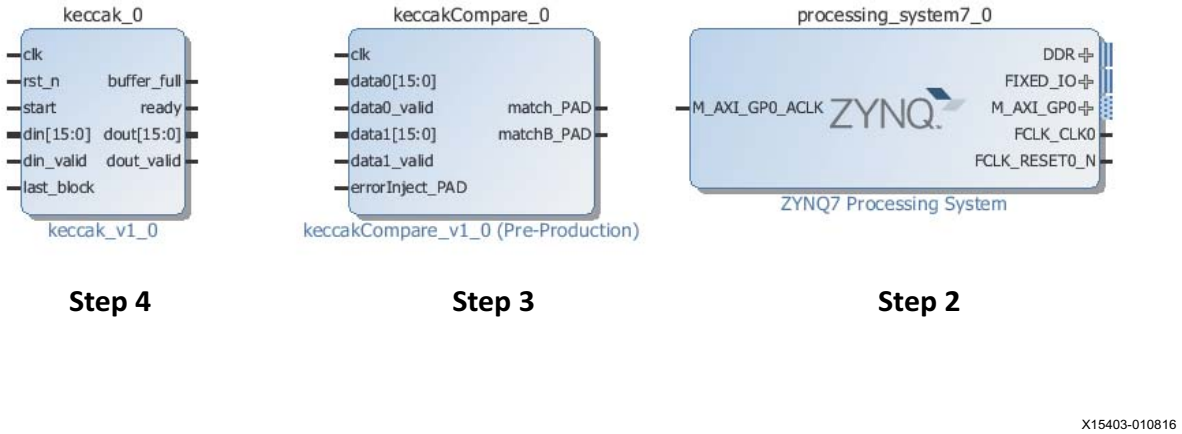


Figure 12: **IPI Canvas after IP Selection**

Now that all the necessary blocks have been added (Figure 12), it is time to start constructing the design. First, the processor is configured. In this lab, the processor's sole function is to source the clocks and resets to your design.

5. **Save** the design.
6. Double-click the **processing_system7_0** IP instance (to re-customize the IP).
   a. Under **PS-PL Configuration**, expand the **AXI Non Secure Enablement > GP Master AXI Interface** item and deselect the **M AXI GP0 interface**.
   b. In the Search box at the top, enter **reset** and select **FCLK_RESET0_N** (default) and **FCLK_RESET1_N**.
   c. Under **Clock Configuration**, expand the **PL Fabric Clocks** item and select **FCLK_CLK0** (default), **FCLK_CLK1**, and **FCLK_CLK2**. Keep their default parameters for these clocks.
   d. Select **OK**.
   e. With the processor block still selected, select **Run Block Automation** at the top of the IPI canvas (navigating the mouse to select **processing_system7_0/**).

f. Select **OK** to run block automation. The canvas should look like Figure 13.

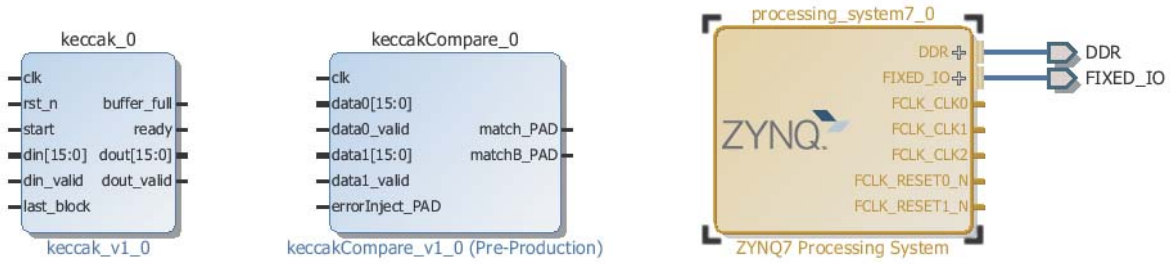


Figure 13: **IPI Canvas after Running Block Automation**

7. Wire up the first instance of the Keccak hash block.
   a. Point at **dout[15:0]** of the **keccak_0** instance until the mouse arrow turns into a pencil. Click and drag the wire to the **data0[15:0]** port of the **keccakCompare_0** instance.
   b. Repeat step a connecting **dout_valid** to **data0_valid**.
   c. Right-click **buffer_full** and select **make external**.
   d. Repeat step c for **ready**.
   e. Repeat step c for **start**.
   f. Repeat step c for **din[15:0]**.
   g. Repeat step c for **din_valid.**
   h. Repeat step c for **last_block**.
8. Wire up the second instance of the Keccak hash block. In this case, you must first create it.
   a. Right-click **keccak_0** and select **Copy**. Select elsewhere on the canvas, right-click, and select **Paste**.
   b. Point at **dout[15:0]** of the **keccak_1** instance until the mouse arrow turns into a pencil. Click and drag the wire to the **data1[15:0]** port of the **keccakCompare_0** instance.
   c. Repeat step b connecting **dout_valid** to **data0_valid**.
   d. Right-click **buffer_full** and select **make external**.
   e. Repeat step d for **ready**.
   f. Repeat step d for **start**.
   g. Repeat step d for **din[15:0]**.
   h. Repeat step d for **din_valid.**
   i. Repeat step d for **last_block**.

9. Wire up the compare block.
   a. Right-click **match_PAD** and select **make external**.
   b. Repeat step a with **matchB_PAD**.
   c. Repeat step a with **errorInject_PAD**.
10. **Save** the design.
11. Right-click an empty space in the canvas and select **Regenerate Layout**. When done, the IPI canvas should look like Figure 14.

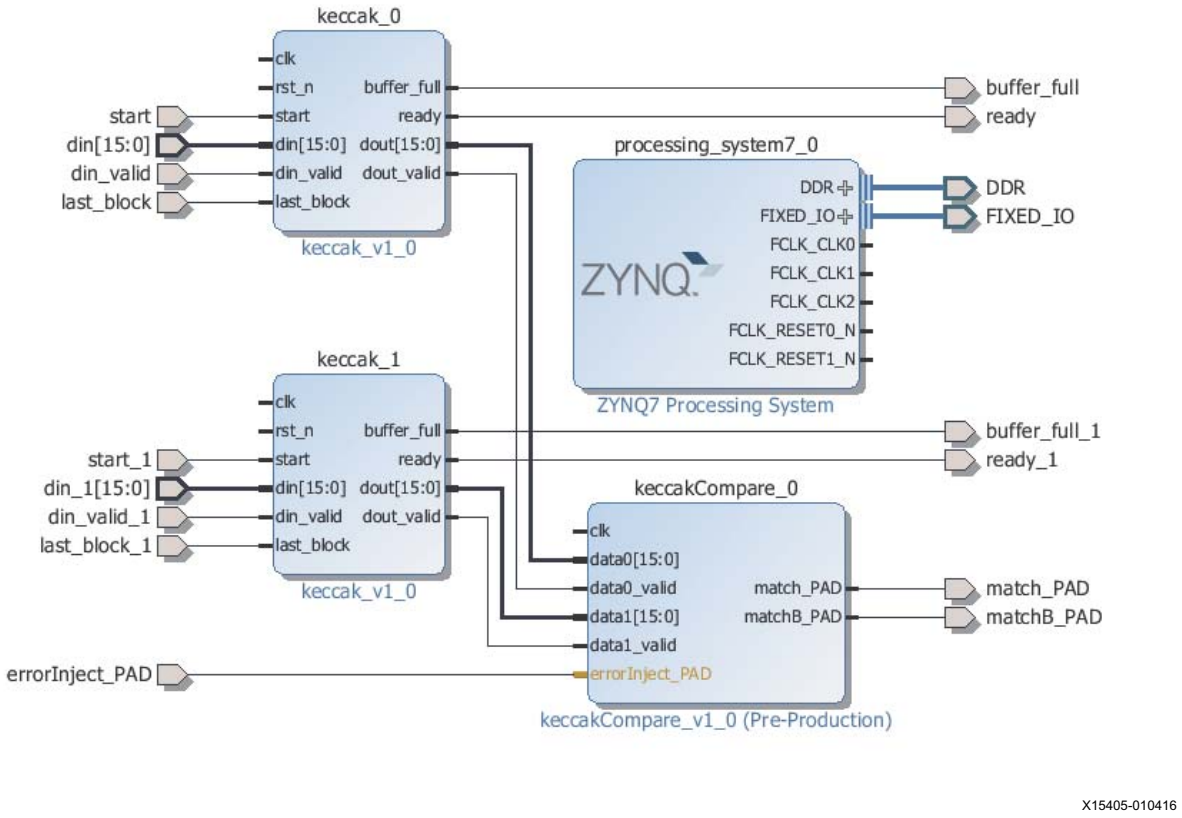


Figure 14: **IPI Canvas after Initial Connections**

12. Connect the clocks and resets.
   a. Draw a wire from **FCLK_CLK0** of the **processing_system7_0** IP to the **clk** input of the **keccak_0** instance.
   b. Draw a wire from **FCLK_RESET0_N** of the **processing_system7_0** IP to the **rst_n** input of the **keccak_0** instance.
   c. Repeat step a and step b for the **keccak_1** instance using **FCLK_CLK1** and **FCLK_RESET1_N**, respectively.
   d. Draw a wire from **FCLK_CLK2** of the **processing_system7_0** IP to the **clk** input of the **keccakCompare_0** instance.

e. **Save** the design.

f. Select **Regenerate Layout**. This is for ease of viewing. Your canvas should now look like Figure 15.

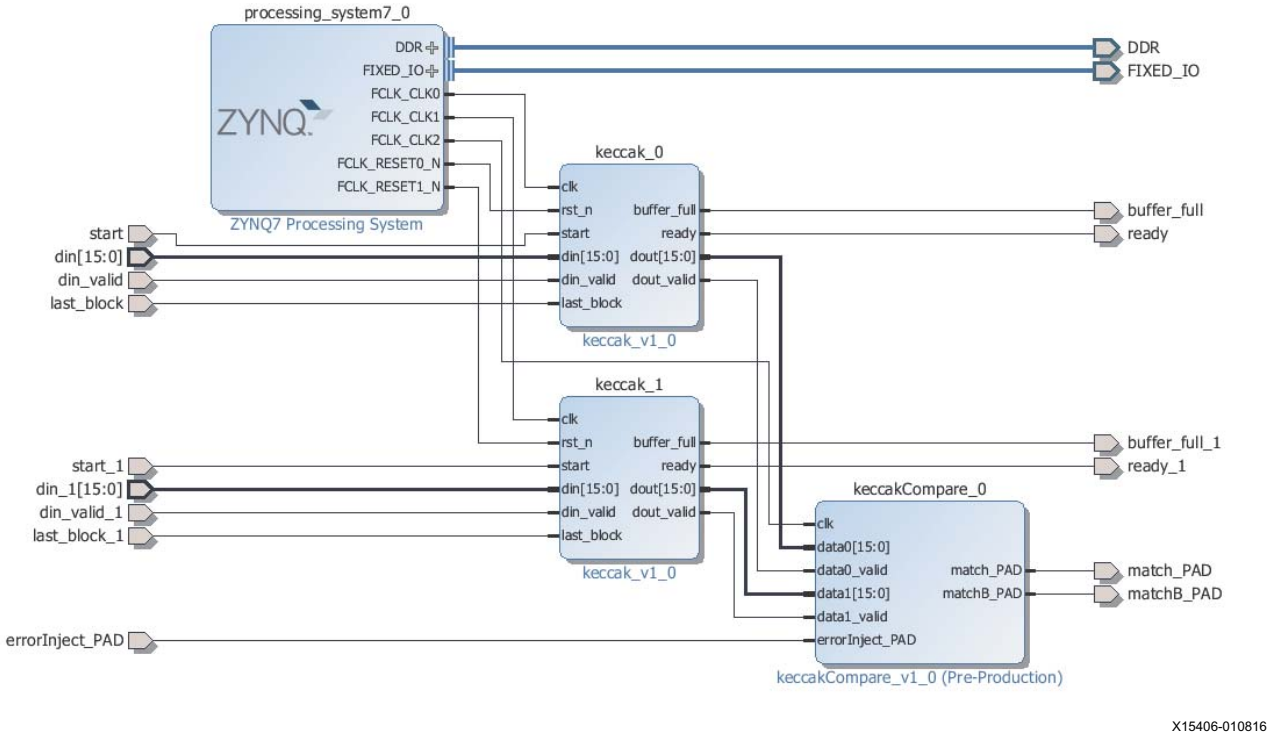


Figure 15: **IPI Canvas after Final Connections**

13. Each of the four blocks in Figure 15 are modules that need to be isolated.

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, DON'T_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.*

14. Right-click **keccakCompare_0** and select **Create Hierarchy ...**

a. Name it **keccakCompare_0_ISO_Wrapper**.

b. Make sure the check box to add the selected instance is selected and select **OK**.

15. Repeat step 14 for **keccak_0** naming it **keccak_0_ISO_Wrapper**.
16. Repeat step 14 for **keccak_1** naming it **keccak_1_ISO_Wrapper**.
17. Repeat step 14 for **processing_system7_0** naming it **ps7_ISO_Wrapper**.

    Note: A critical message pops up noting that it is necessary to associate ELF files. Select **OK**. There are no such files in this project and they would get regenerated anyway in future steps.

18. Verify all connections are valid by right-clicking on the blank canvas and selecting **Validate Design**. Select **OK** to continue.
19. **Save** the design.
20. Select **Regenerate Layout**. This is for ease of viewing. Your canvas should now look like Figure 16.

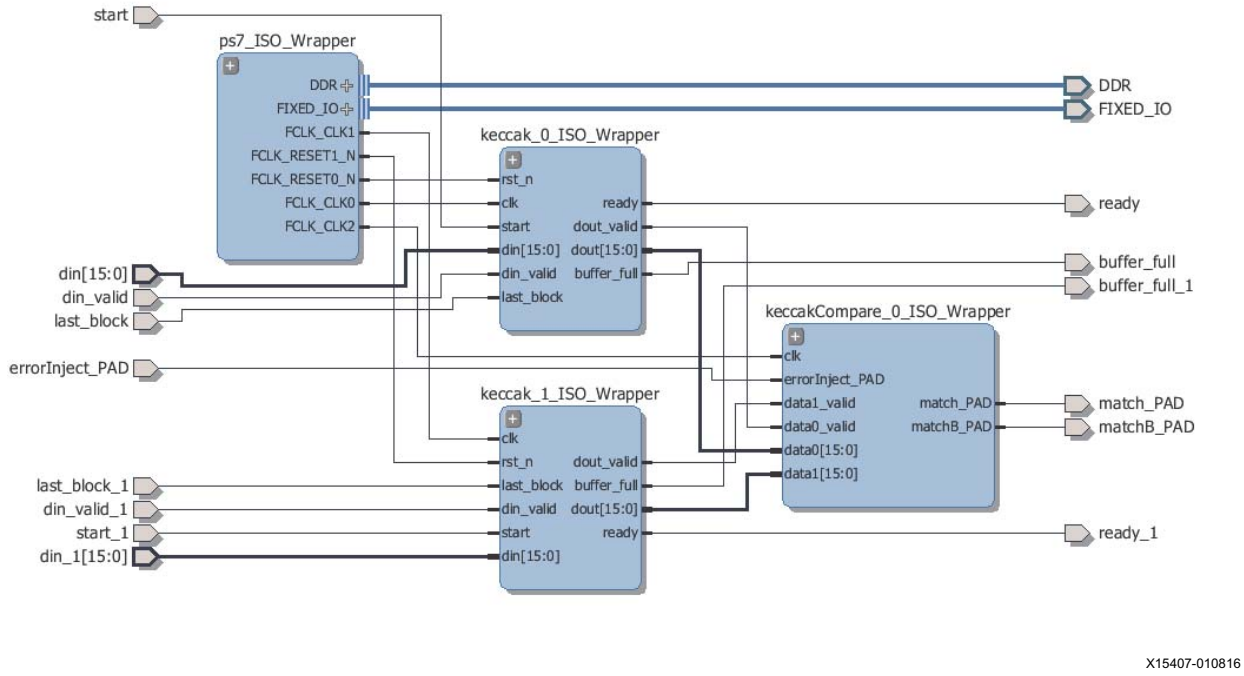


Figure 16: **Final IPI Canvas**

21. 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** (Figure 17). 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** (Figure 17). Select **Generate** on the pop-up window. Select **OK** when the process completes.

Note: Recall the critical warning with respect to ELF files. This warning is addressed in this step.

The design is now ready for the RTL elaboration phase.

22. **Save** the design.

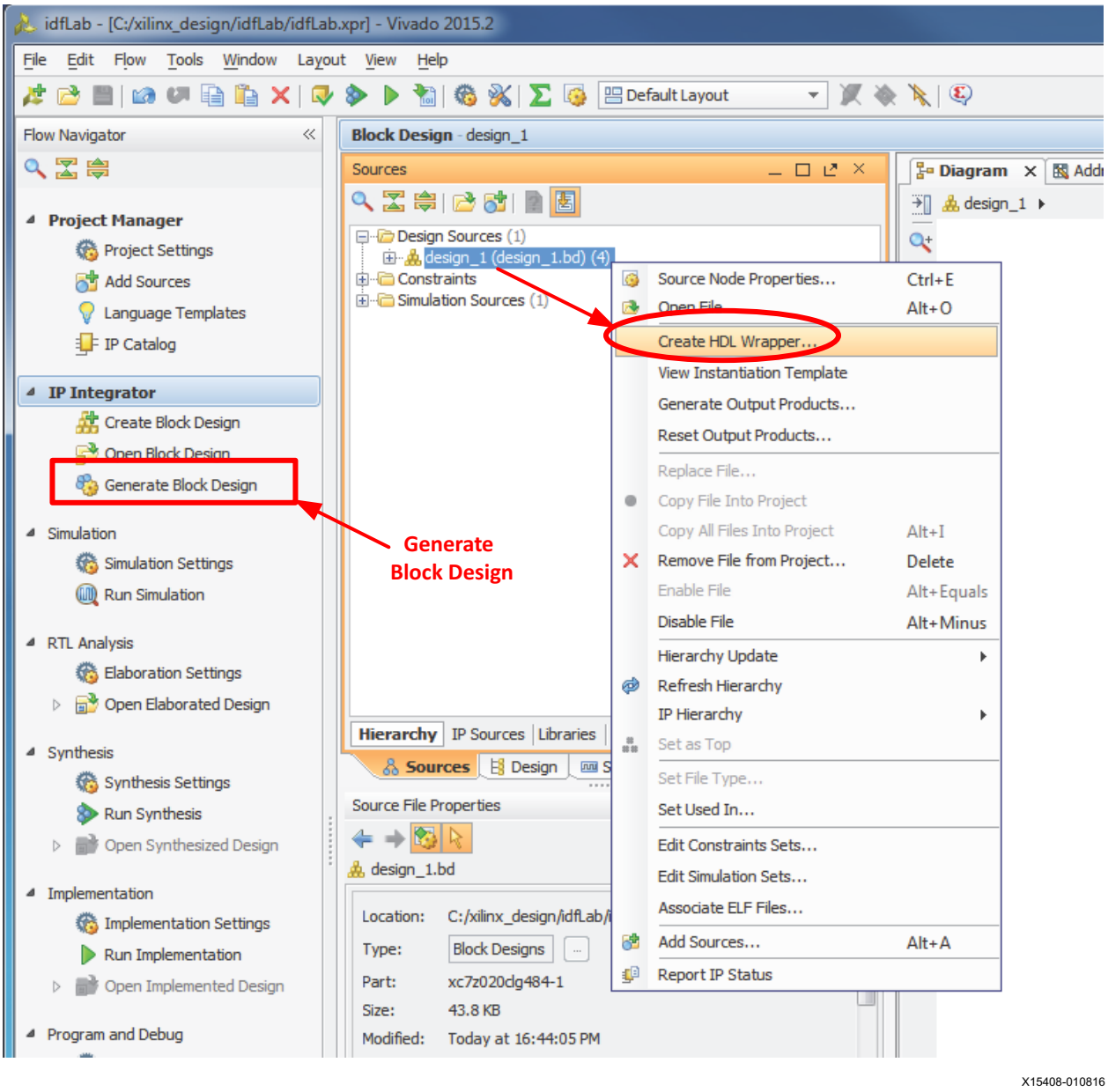


Figure 17: **Create HDL Wrapper/Generate Block Design**

Design Elaboration

Register-transfer level (RTL) design elaboration is a small step in this lab. Its primary function is to debug the HDL code of the design (Figure 18).

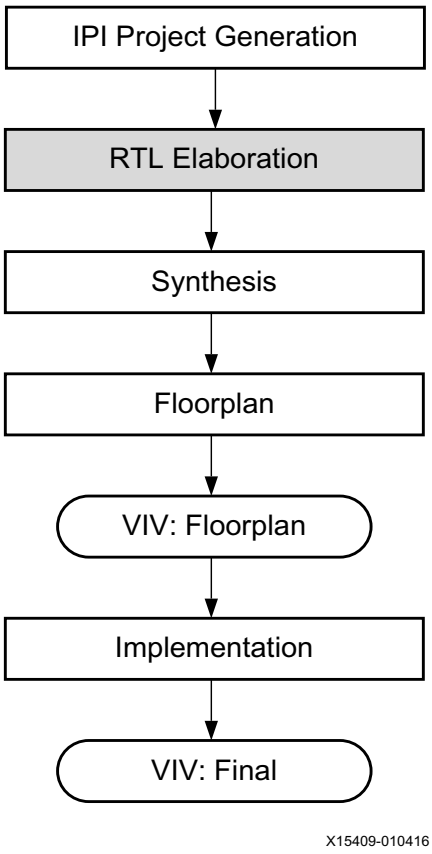


Figure 18: **Lab Flow Progression - RTL Elaboration**

Open the Elaborated Design

Just under the IP Integrator section is the Simulation section and then the RTL Analysis section.

1. Click the **Open Elaborated Design** menu item. This launches the process. When done, you see a large block diagram labeled **design_1_i**. This is the default name that was generated by IPI when you added the HDL wrapper.

2. You can navigate your design by using the mouse to expand the hierarchy using the + button on the top left of any block as seen in Figure 19 (with one level expanded).

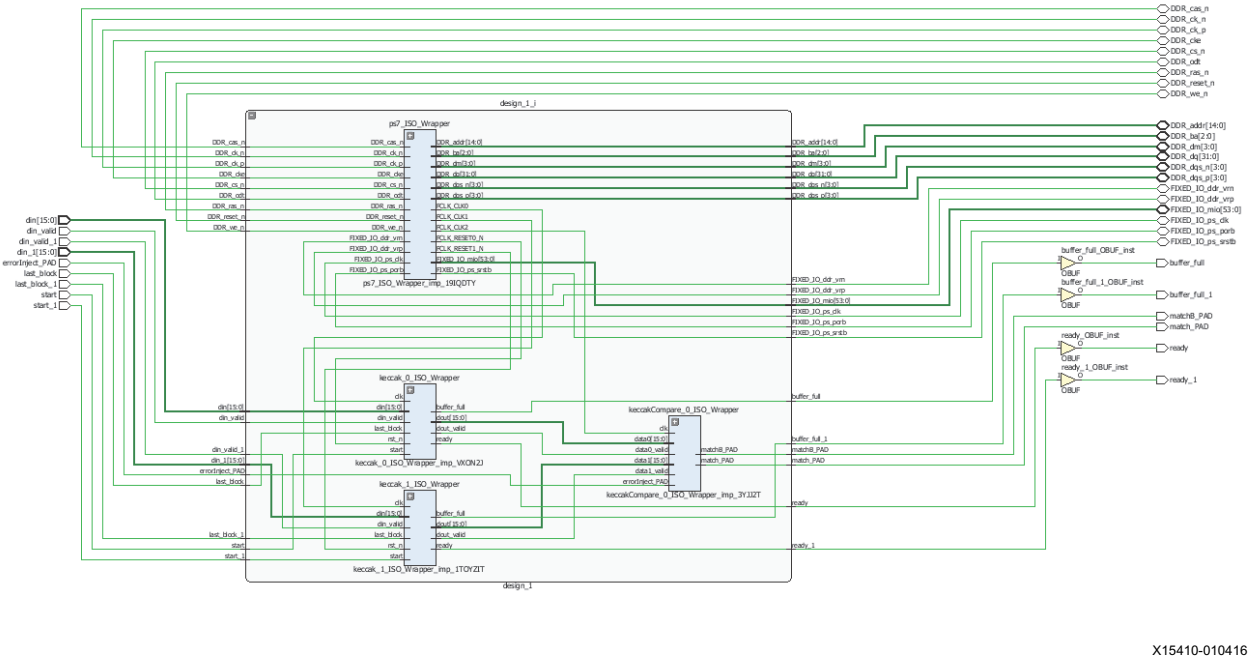
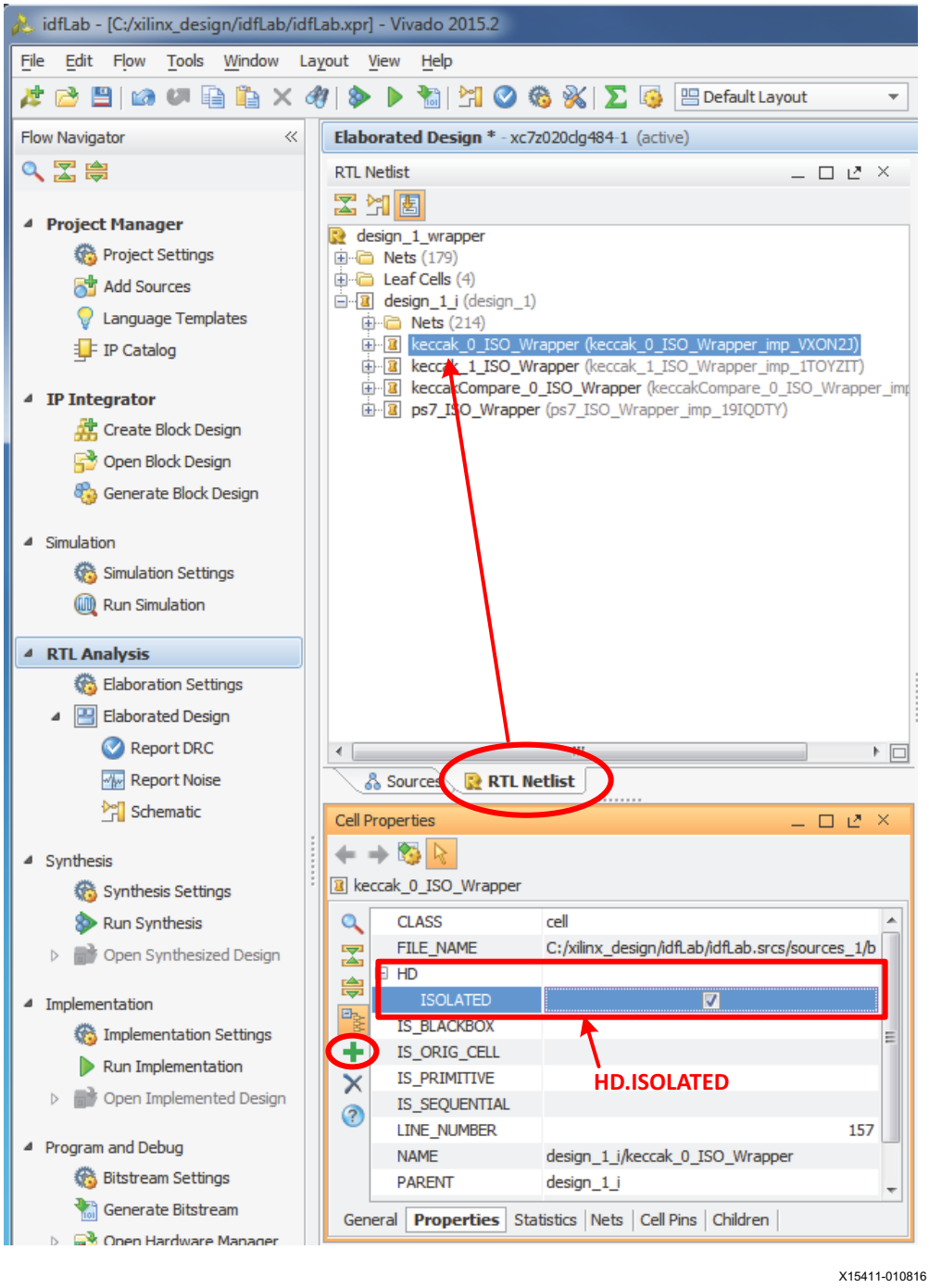


Figure 19: **RTL Schematic (expanded)**

3. Now you need to add some attributes to tell the tools which modules are going to be isolated using 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 **RTL Netlist** tab so that you can see each of the modules you created in IPI (Figure 20).
5. Select **keccak_0_ISO_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 green **+** symbol and add the attribute **HD.ISOLATED** from the **Add Properties** pop-up window.

d. Expand the newly added attribute and check the unchecked check box (Figure 20).



X15411-010816

Figure 20: **Setting HD.ISOLATED Attribute**

6. Repeat step 5 for **keccak_1_ISO_Wrapper**.
7. Repeat step 5 for **keccakCompare_0_ISO_Wrapper.**
8. Repeat step 5 for **ps7_ISO_Wrapper**.
9. Save the design and enter **Top** when requested, to enter the file name of the Xilinx design constraints (XDC) file.

10. Select **OK**.

Design Synthesis

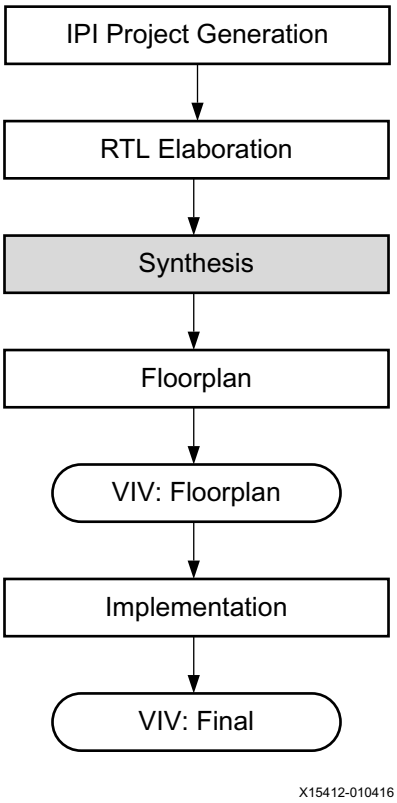


Figure 21: **Lab Flow Progression - Synthesis**

Synthesis Process

This section describes the synthesis process. In this lab, though time consuming, synthesis is not a significant part of the process. This is because all the IP was created earlier and you are just assembling the blocks. If done correctly, the synthesis schematic should look the same as the RTL schematic.

Launch Synthesis

1. Under the Synthesis menu on the left of the Vivado GUI, select **Run Synthesis**. If prompted, save the design.
2. When 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, do so by selecting **Yes**. This is good practice because memory might be limited.

Floorplanning the System

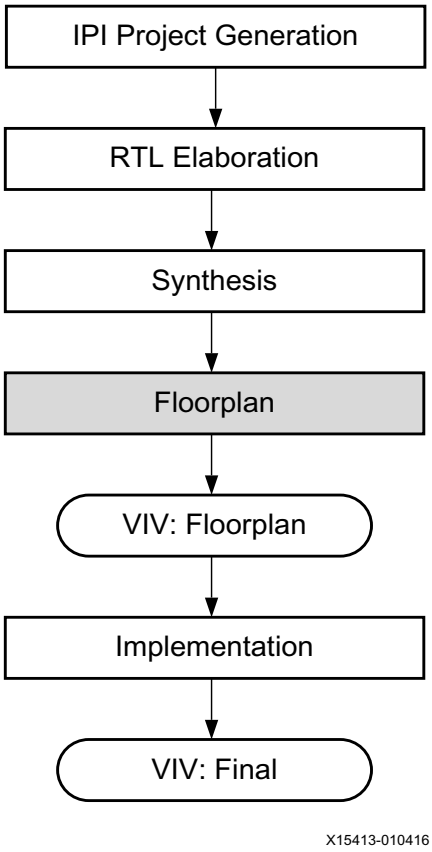


Figure 22: **Lab Flow Progression - System Floorplanning**

Floorplan Process

The floorplan of the reference design is shown in Figure 23. Where inter-module communication (via Trusted Routing) is necessary, regions must be coincident with each other with a fence tile between the two intended regions.

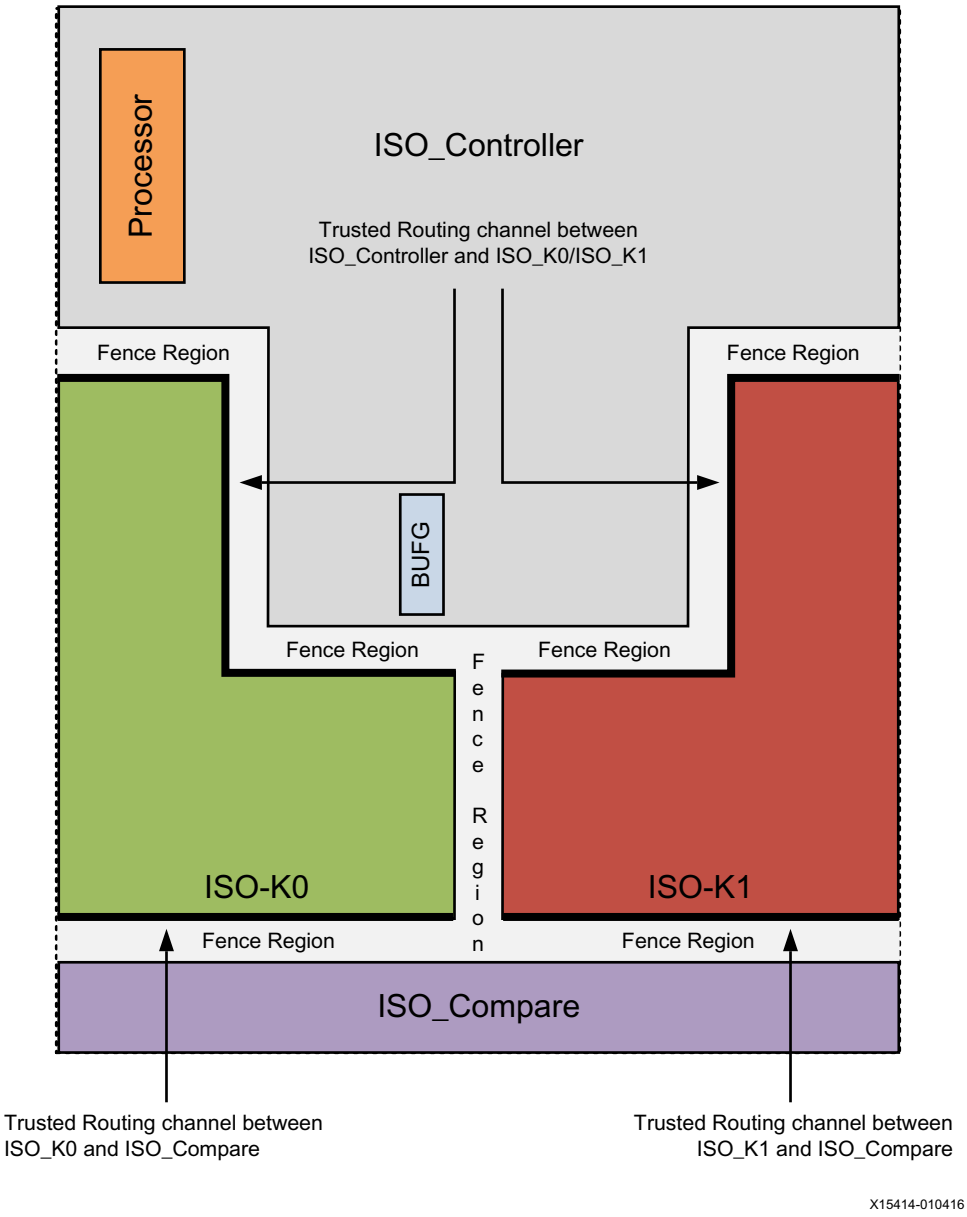


Figure 23: **Floorplan of the Reference Design Highlighting the Trusted Routing Channels**

Floorplanning a design is the most time consuming part of the Isolation Design Flow. The purpose of this lab is not to test your skills in creating pblocks and placing pins, but to familiarize you with the flow itself and how it integrates well in into IPI.

Scripts are provided that allow you to generate the floorplan in a few minutes rather than a few hours. That said, it is very important to understand the floorplanning rules and complexities that are associated with floorplanning any design (*Vivado Design Suite User Guide - Partial Reconfiguration* (UG909) [Ref 5] and *Vivado Design Suite User Guide - Hierarchical Design*

(UG905) [Ref 6]). More details on the IDF rules are in *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)* (XAPP1222) [Ref 3].

1. Select the **Tcl Console** tab at the bottom of the Vivado GUI.
2. Enter the following commands:
   a. **cd c:/xilinx_design** (if not already in this directory)

      Note: This assumes you extracted the lab design into `c:/xilinx_design`.

      Note the usage of the Linux / instead of the Windows \ when using the Vivado tool command language (Tcl). (See *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 7] for more information.).

   b. **source ./sources/xdc/pins.xdc**
   c. **source ./sources/xdc/k0.xdc**
   d. **source ./sources/xdc/k1.xdc**
   e. **source ./sources/xdc/compare.xdc**
   f. **source ./sources/xdc/controller.xdc**

      Note: There might be a warning for `No pblocks matched`, which is due to the first XDC command in the file querying if the pblock already exists. If so, delete it before creating a new pblock.

3. Save the design, selecting **OK** and **YES**. When complete, your floorplan looks like Figure 24.

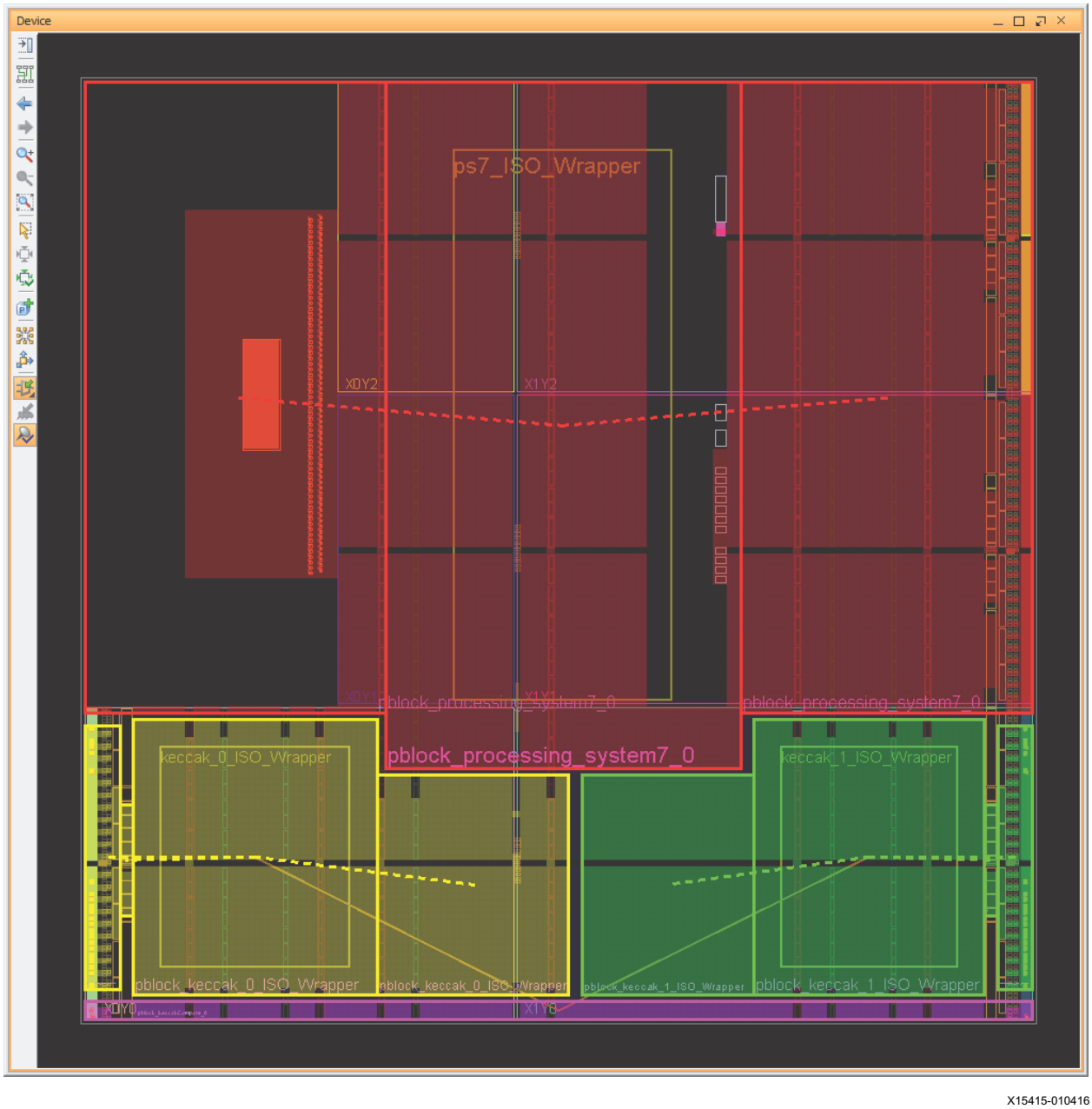


Figure 24: **Completed Lab Design Floorplan**

Due to a tool limitation, it is necessary to prohibit all sites not owned by a pblock, per *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)* (XAPP1222) [Ref 3]. Enter the following command:

source ./sources/xdc/topProhibits.xdc

4. Save the design by selecting **OK** and **YES**.

   Note: This limitation will be removed Vivado release 2016.1.

   An enhancement to IDF in Vivado is to allow global clocking components inside isolated modules. Global components cannot actually be isolated due to their global scope. Before IDF in Vivado, you had to modify their design and ensure such components were at the top level of their design. IDF in Vivado allows for attributes to be set to turn off isolation on them, allowing them to be nested but not isolated. This is particularly useful for clocks coming from the Zynq-7000 processing system (PS) region because they cannot be moved. In this design, all clocks and resets are generated by the PS region. To turn off isolation of the clocks, use the following command:

   ```
   set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -
   filter {PRIMITIVE_TYPE =~ CLK.gclk.*}]
   ```

   In this example the HD.ISOLATED_EXEMPT property is applied globally to any and all clock components in your design. If your design has some clock components that you desire to be isolated, the safer method is to exempt only the ones in the processor block. This is accomplished by adding an additional item to the `-filter` option. The more restrictive filter option now looks as follows:

   ```
   -filter {NAME =~ *ps7* && PRIMITIVE_TYPE =~ CLK.gclk.*}
   ```

5. The new Tcl command is shown below. Enter the following statement in the Tcl Console window:

   ```
   set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -
   filter {NAME =~ *ps7* && PRIMITIVE_TYPE =~ CLK.gclk.*}]
   ```

6. Save the design by selecting **OK** and **YES**.
7. Browse the floorplan around the fences, verifying the fence rules (no less than one user tile).

Running VIV against the Floorplan

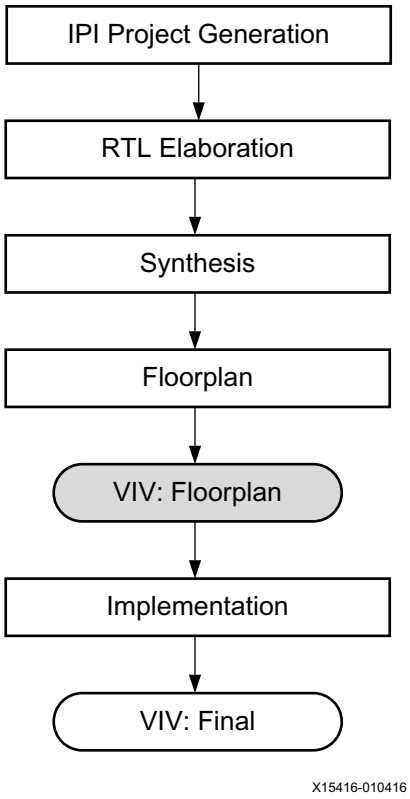


Figure 25: **Lab Flow Progression: VIV on Floorplan**

The Xilinx Vivado Isolation Verifier (VIV) software verifies that FPGA or SoC designs that have been partitioned into isolated modules meet the stringent standards for a fail-safe design. VIV is a Tcl script that runs in the Vivado tool framework in the form of DRCs. This allows for a strong GUI interface to the tool that was not available in the older ISE tools for IVT.

VIV is run on the floorplan to catch pin, I/O bank, and area group isolation faults early in the design, when changes are more easily integrated. The steps in this section guide you through the process. After implementation, the VIV is run against the routed design.

Constraint Checking (VIV - Constraints)

On the floorplan, VIV checks the following:

- Pins from different isolation groups are not physically adjacent, vertically or horizontally, at the die.
- Pins from different isolation groups are not physically adjacent at the package. Adjacency is defined in eight compass directions: north, south, east, west, northeast, southeast, northwest, and southwest.
- Pins from different isolation regions are not co-located in an I/O block (IOB) bank.

  Note: Though VIV does fault such conditions, only specific security-related applications require such bank isolation. The majority of applications allow for sharing of banks. Bank sharing is dependent on the specific application.
- The pblock constraints in the XDC file are defined so that a minimum of a one tile wide fence exists between isolated regions.

Verifying the Floorplan with VIV

Loading VIV into Vivado Framework

VIV is essentially a series of design rule checks coded in Tcl (see *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 7]. VIV is not part of the standard Vivado tools and must therefore be installed.

1. Type the following command in the Tcl Console (assuming the working directory is `c:/xilinx_design`):

   source ./sources/viv/viv.tcl

2. If not already open, open the synthesized run by selecting **Open Synthesized Design** under the Synthesis section on the left of the Vivado GUI.

   Note: If the option Open Synthesized Design is not available, it is either already open or there is no Synthesis run available (requiring synthesis to be re-run).

3. Under the Synthesized Design pull-down of the Synthesis section, select **Report DRC** (Figure 26).

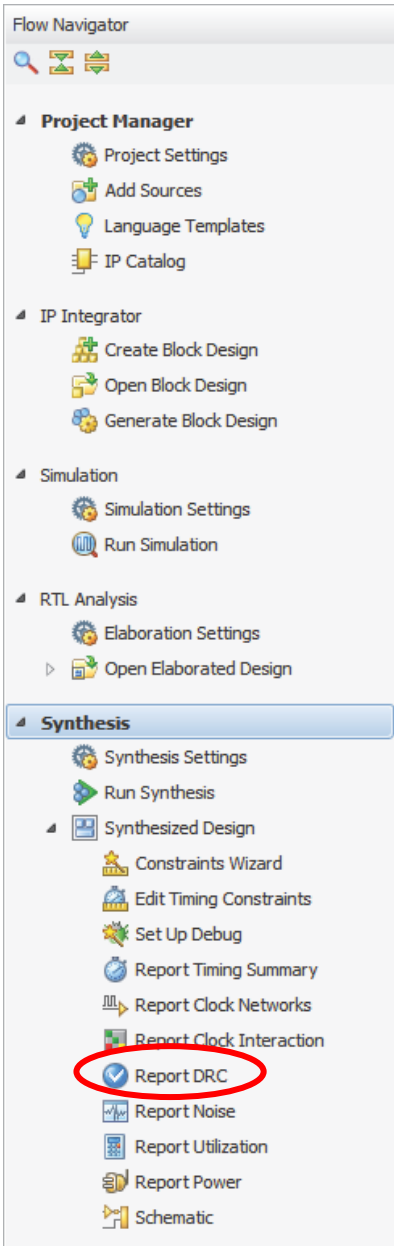


Figure 26: **Running DRCs after Synthesis**

4. Complete the following steps, as shown in Figure 27:
   a. Uncheck all DRC rules (deselect **All Rules**).
   b. Expand all of the **Isolation** rules.
   c. Select IDF rules **1–4** (IDF Floorplan rules).
   d. Change the **Results name** field to **viv_floorplan**.
   e. Change the **Output File** field to **c:/xilinx_design/viv_floorplan.txt**.
   f. Select **OK**.

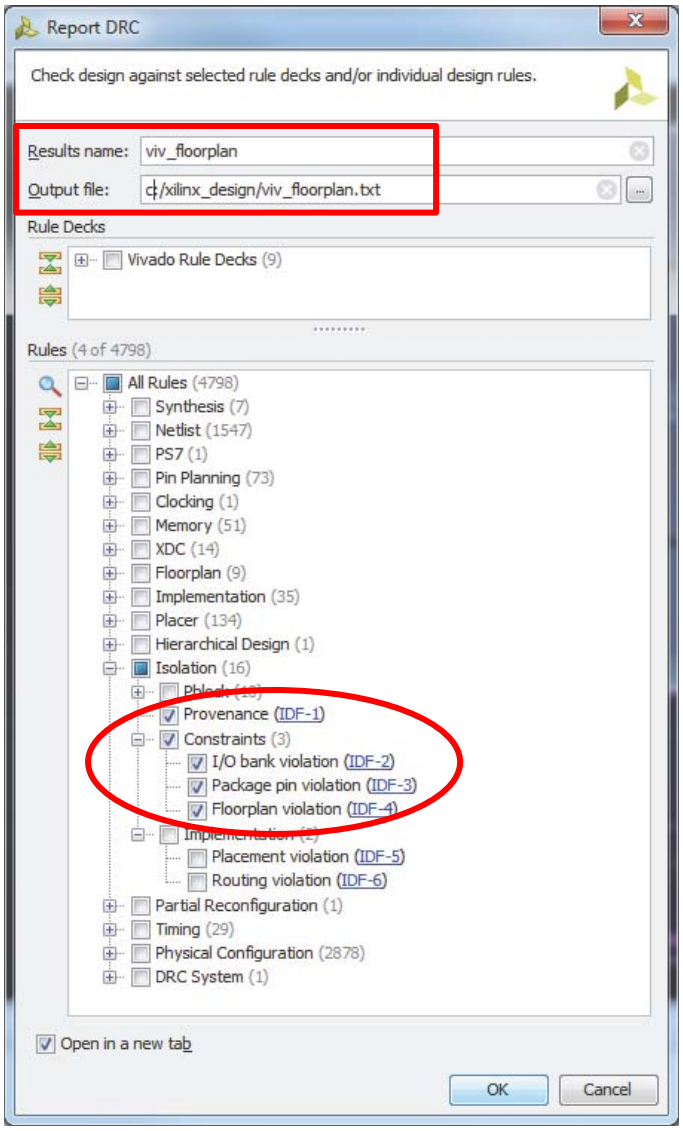


Figure 27: **VIV IDF Floorplan Rules (DRCs)**

5. All results are stored in the specified file (`c:/xilinx_design/viv_floorplan.txt`). For easier access, however, they are displayed in the DRC tab at the bottom of the Vivado GUI (Figure 28).

   Note: The DRC tab is only available after a DRC run has been executed.

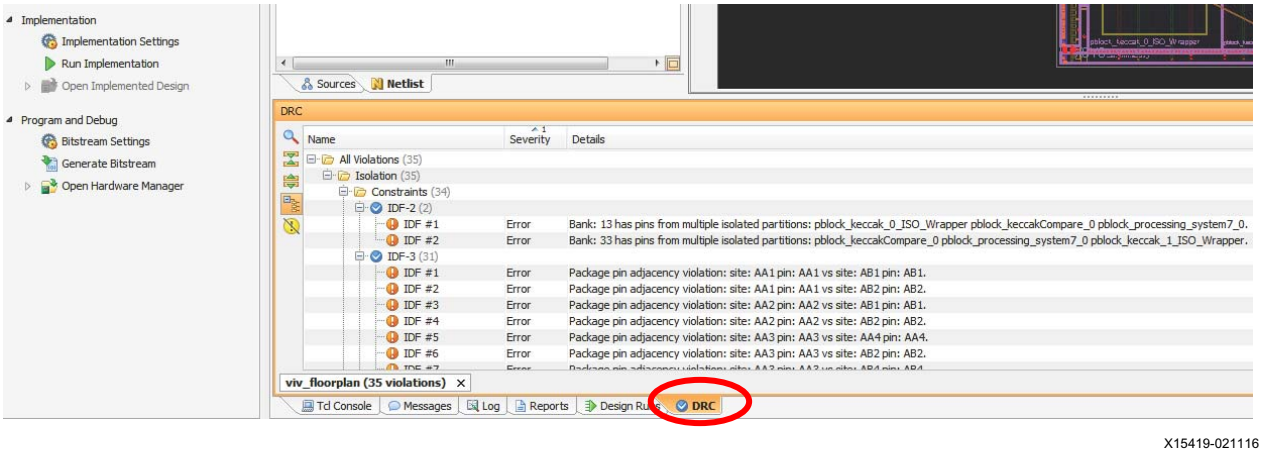


Figure 28: **DRC Tab after VIV Floorplan Run**

6. Inspect the results (35 violations are identified).

   a. IDF-1 (Provenance): IDF-1 documents the circumstances under which a DRC report was generated, including tool versions, date, design name, user, platform, and host. This DRC is informational. No errors are reported here.

   b. IDF-2 (I/O bank violation): IDF-2 reports all I/O banks that have IOBs from more than one isolated region. Two I/O banks are used in this lab design and both contain IOBs from two distinct isolated regions, hence the two IDF-2 errors shown in the DRC run. I/O bank sharing is not an actual error. It is informational. There are some cases where I/O banks cannot be shared and some where they can. It depends on the user application and if that application allows a common I/O power supply between isolated regions.

   c. IDF-3 (Package pin violation): IDF-3 reports errors where pins from different isolated regions are adjacent to each other at the package level. This lab has 31 pin adjacency errors. These are real errors and would be unacceptable for any design desiring physical isolation between such regions.

   d. IDF-4 (Floorplan violation): IDF-4 reports all locations where one (or more) isolated region is either adjacent or overlaps another isolated region. There are no such violations in this design.

      Note: The DRC engine reports 35 violations, but only 33 can be accounted for. This discrepancy is due to the way the DRC engine counts violations. Any report (such as saying `no violations found`) increments the DRC rule violation counter. As such, the first missing violation is from IDF-1 where VIV outputs the provenance of the design, while the second missing violation is from IDF-4 reporting that no violations were found.

7. Select each violation as desired, and notice that the violation is highlighted in the Vivado GUI.

   An example package pin adjacency violation (IDF-3), error IDF #1, is selected in the DRC tab. The corresponding device sites are highlighted in the GUI Device view, as shown in Figure 29.

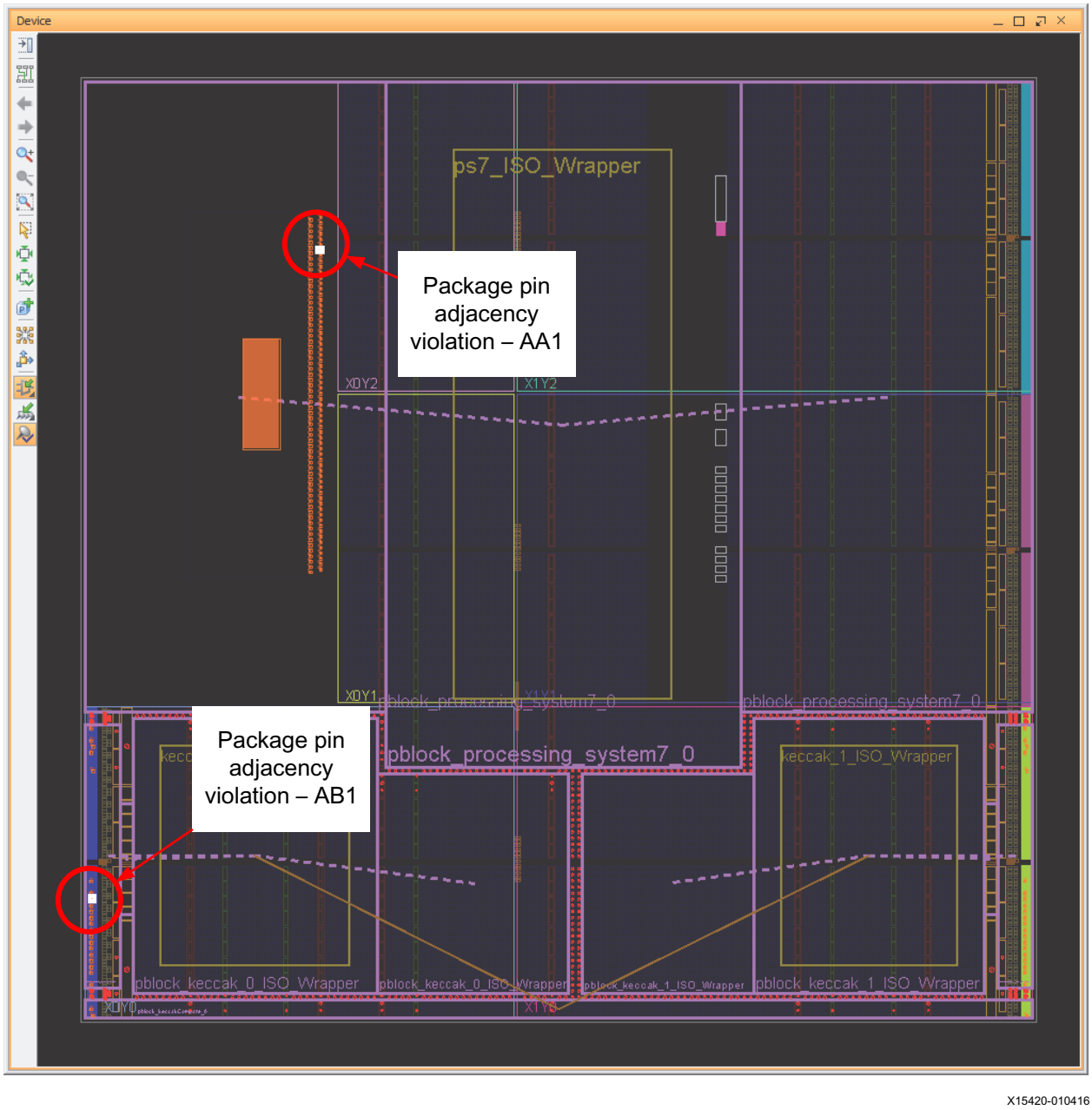


Figure 29: **Package Pin Adjacency Violation - Device View**

The corresponding pin adjacency violations for package pins AA1 and AB1 are highlighted in the GUI Package view as shown in Figure 30. The Package view shows that these pins are indeed physically adjacent in the package.

Figure 30: **Package Pin Adjacency Violation - Package View**

Implementing the Design

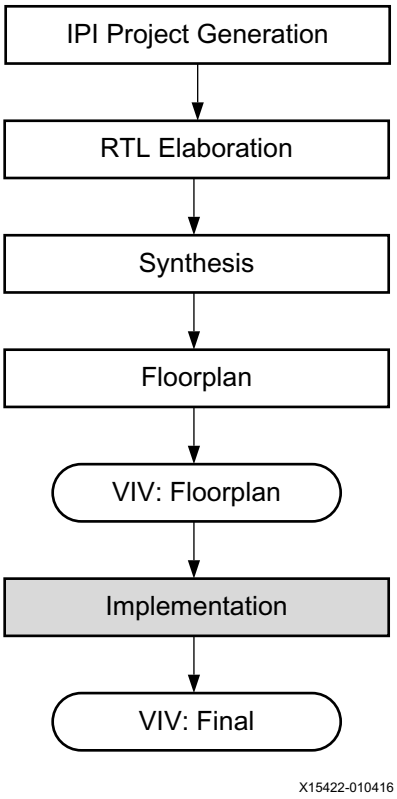


Figure 31: **Lab Flow Progression - Implementation**

Generating and Running an Implementation

Implementing the Design

These steps describe how to generate and run a design implementation.

1. Under the Design Runs tab at the bottom of the Vivado GUI, right-click **impl_1** and select **Change Run Settings**.
2. The lab design is densely populated for the K0 and K1 modules and the **Area Explore** run **Strategy** produces the best results.

a. Click the pull-down menu for **Strategy**, select **Area_Explore (Vivado Implementation 2015),** and click **OK** (as shown in Figure 32).

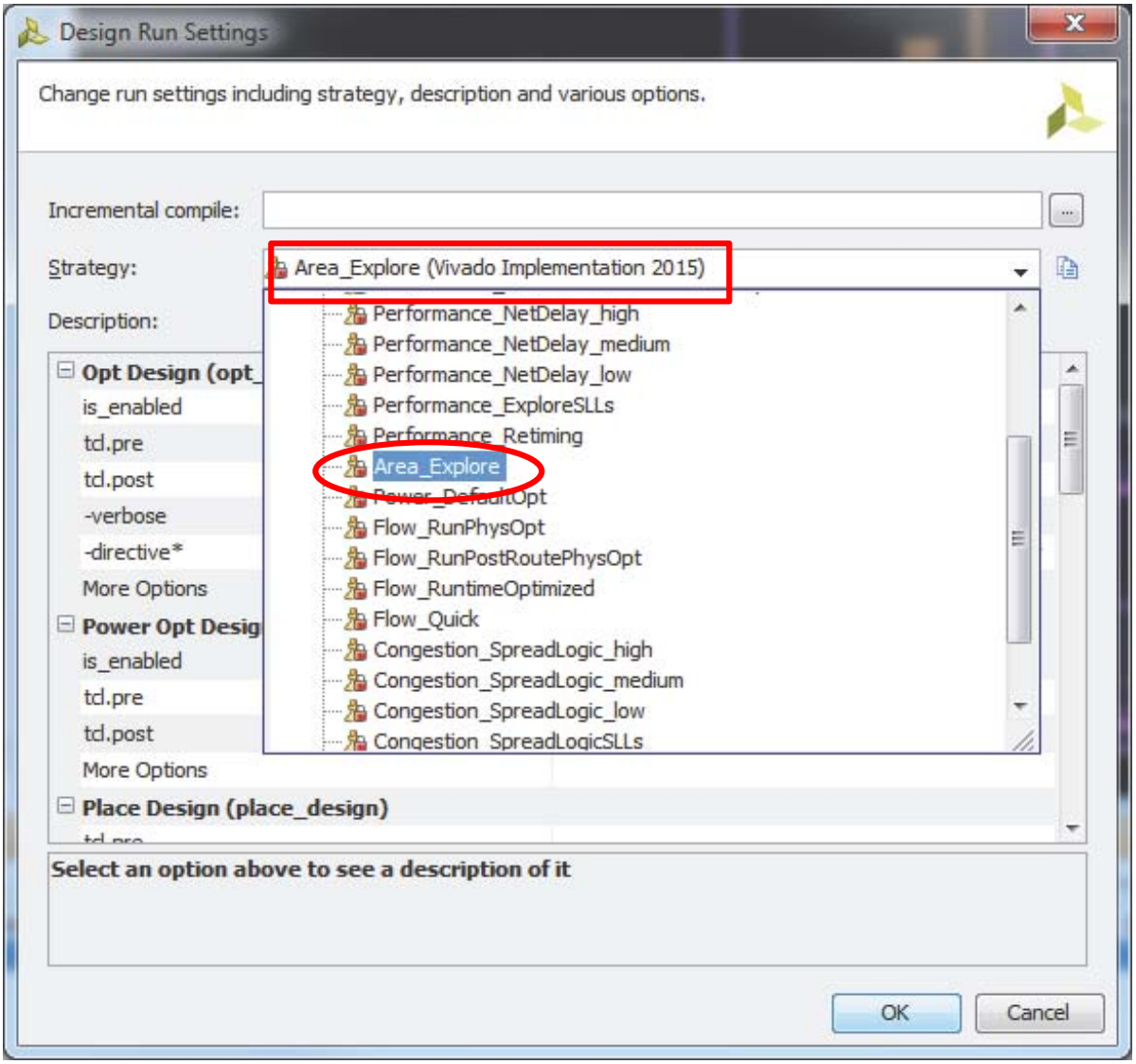


Figure 32: **Device View - Implemented Design**

3. Click **Run Implementation** under the Implementation menu on the left of the Vivado GUI.
4. Save the project or constraints if asked. If any constraints were modified, the Vivado tool requests starting from Synthesis. This is okay.

5. When implementation is complete, select the **Open Implemented Design** option and click **OK**. The device view pops up and looks like Figure 33 (which also shows the pblocks highlighted to clearly identify the isolated regions).
6. If asked to close the Synthesized Design before opening the Implemented Design, select **Yes**. This is good practice because memory might be limited.

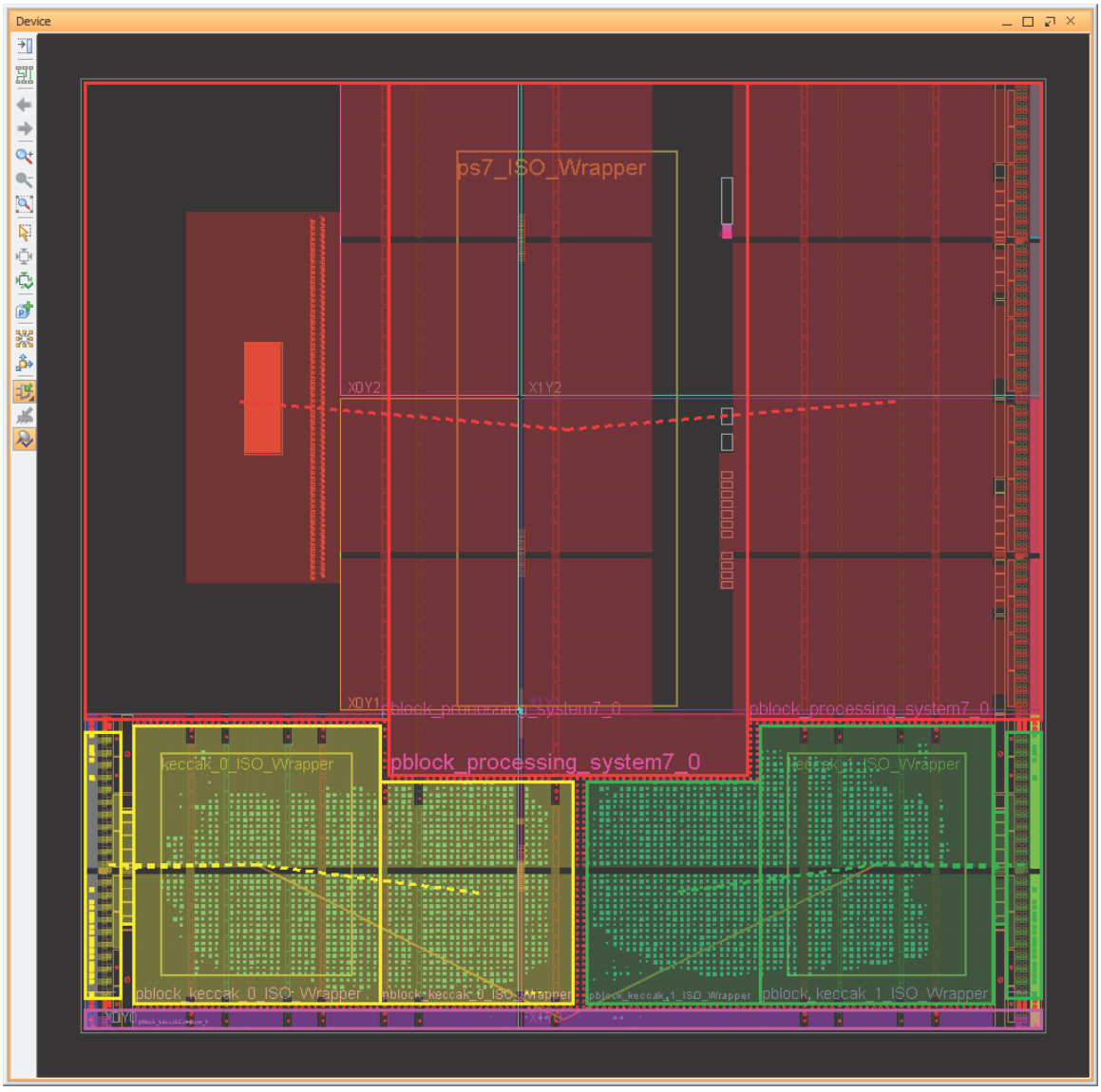


Figure 33: **Implemented Design - Device View**

Routing Resources View (Comments on the Mode)

Routing Resources view is the default when opening a routed design. However, it can be turned on and off by selecting the button in the top left of the Device View (Figure 34).

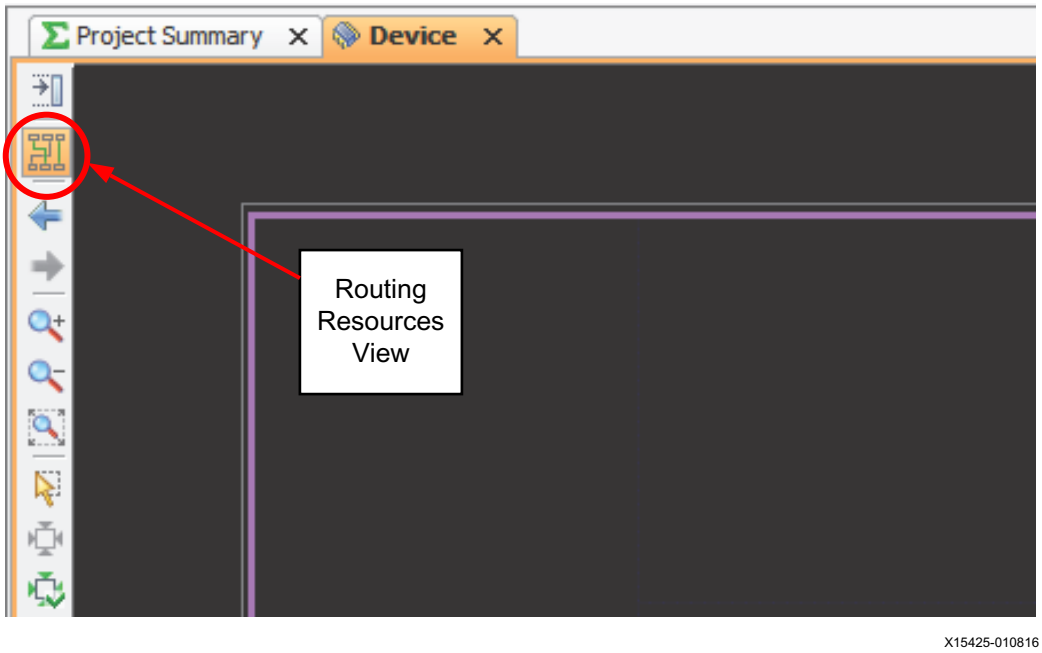


Figure 34: **Enabling/Disabling Routing Resources Mode – Device View**

Gaps in the floorplan appear in this view, because the tools do not consider the interconnect 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.

Verifying the Routed Design with VIV

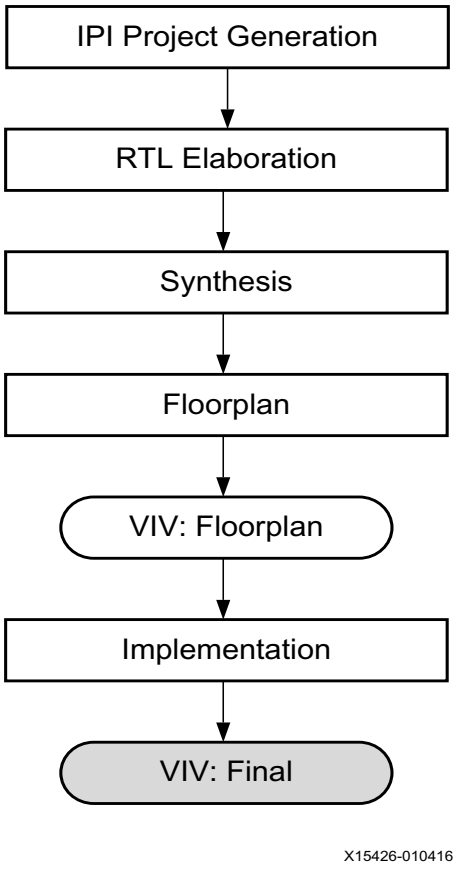


Figure 35: **Lab Flow Progression - VIV on Final Design**

VIV is run on the implemented design to catch isolation faults between isolated regions, as well as package pin and I/O bank violations (as when VIV was run on the floorplanned design). The steps in this section guide you through the process.

Final Isolation Verification (VIV - Implementation)

After the design is complete (placed and routed), VIV is used again on the implemented design to validate that the required isolation was built into the design.

At this step, VIV checks the following:

- VIV analyzes the complete placed and routed design.
- Tile-based isolation analysis looks for a barrier (fence) between isolated regions.
  - A valid user tile acts as a sufficient isolation barrier if:
    - It does not contain any isolated signals (from any isolated region).
    - It is configured in the default (unused) state.
- VIV does the same pin and I/O checking as in Floorplan mode.

Verifying the Implemented Design with VIV

Loading VIV into Vivado Framework

Note: VIV is essentially a series of design rule checks coded in Tcl (see *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 7]. VIV is not part of the standard Vivado tools and must therefore be installed.

1. If VIV is not already installed, type this command in the Tcl Console:

   source ./sources/viv/viv.tcl

2. If not already open, open the implementation run by selecting **Open Implemented Design** under the Implementation section in the Flow Navigator on the left side of the Vivado GUI.

IMPORTANT: *If the option Open Implemented Design is not available, it is either already open or there is no Implementation run available (requiring implementation to be re-run).*

3. Under the Implemented Design pull-down of the Implementation section, select **Report DRC** (Figure 36).

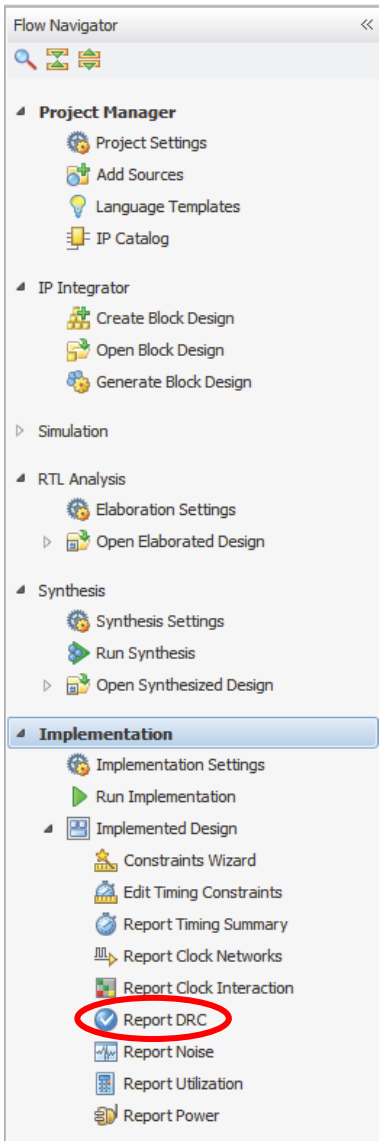


Figure 36: **Running DRCs after Implementation**

4. As shown in Figure 37, complete the following:
   a. Uncheck all DRC rules (deselect **All Rules**).
   b. Expand all of the Isolated rules.
   c. Select IDF rules **1–4** (IDF Floorplan rules).
   d. Select IDF rules **5–6** (IDF Implementation rules).
   e. Change the Results name field to **viv_implementation**.
   f. Change the Output File field to **c:/xilinx_design/viv_implementation.txt**.
   g. Select **OK**.

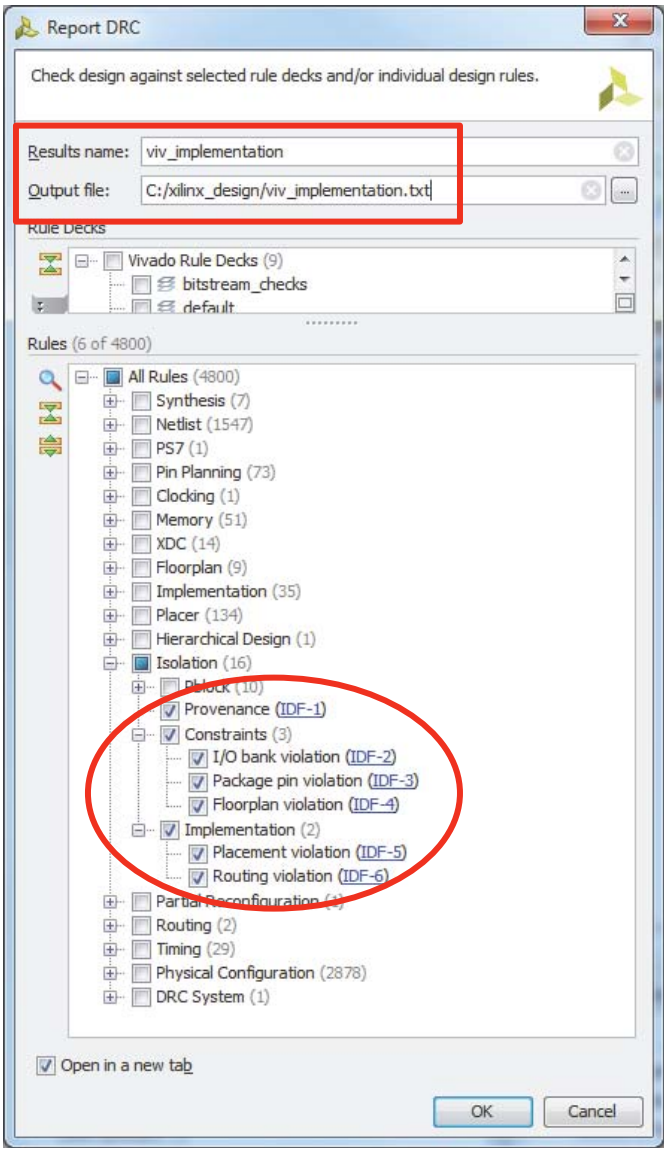
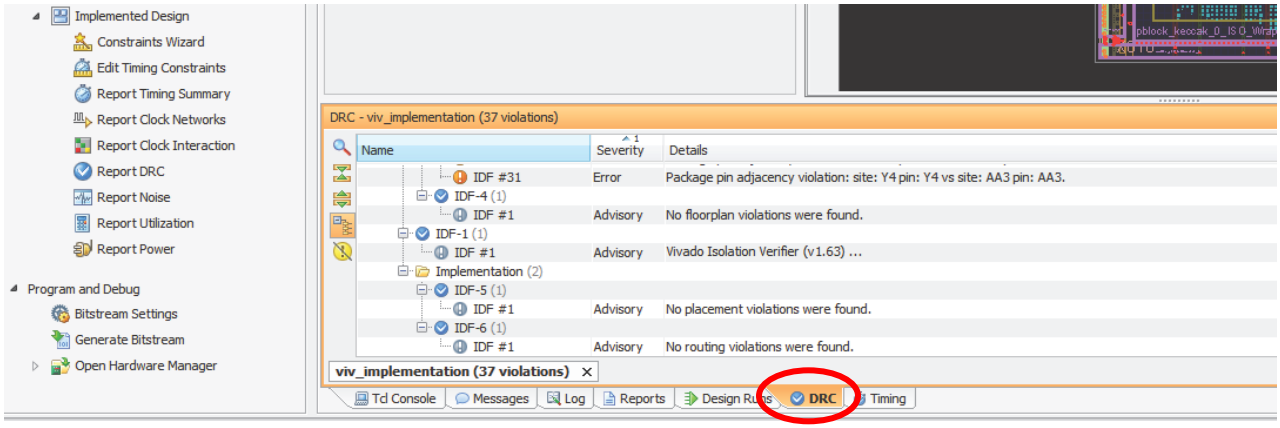


Figure 37: **VIV IDF Implementation Rules (DRCs)**

5. All results are stored in the specified file (`c:/xilinx_design/viv_implementation.txt`). For easy access, however, they are displayed in the DRC tab at the bottom of the Vivado GUI (Figure 38).

   Note: This tab is only available after a DRC run has been executed.



X15429-021116

Figure 38: **DRC Tab after VIV Implementation Run**

6. Inspect the results (37 violations are identified).

   a. IDF-1 (Provenance): IDF-1 documents the circumstances under which a DRC report was generated, including tool versions, date, design name, user, platform, and host. This DRC is informational. No errors are reported here.

   b. IDF-2 (I/O bank violation): IDF-2 reports all banks that have IOBs from more than one isolated region. Two IOB banks are used in this lab and both contain IOBs from two distinct isolated regions. Bank sharing is not an actual error. It is informational. There are some cases where banks cannot be shared and some where they can. It depends on the user application and if that application allows a common I/O power supply between isolated regions.

   c. IDF-3 (Package pin violation): IDF-3 reports errors where pins from different isolated regions are adjacent to each other at the package level. This lab has 31 pin adjacency errors. These are real errors and would be unacceptable for any design desiring physical isolation between such regions.

   d. IDF-4 (Floorplan violation): IDF-4 reports all location where one (or more) isolated region is either adjacent or overlaps another isolated region. There are no such violations in this design.

   e. IDF-5 (Placement violation): IDF-5 reports all placement violations. IDF-5 checks that no isolated logic or interconnect tile is adjacent to an isolated logic or interconnect tile of a different isolation group. There are no such violations in this design.

   f. IDF-6 (Routing violation): IDF-6 reports all routing violations and consists of three checks:

      - All inter-region nets must have loads in exactly one isolated region.

- No inter-region net can use nodes that have programmable interconnect points (PIPs) in the fence, except clock nets which can have unused PIPs in the fence.
- For any tile containing inter-region nets, all such nets must have a common source and load.

There are no such violations in this design.

Note: Isolation groups are defined by pblocks marked with the HD.ISOLATED property.

Note: The DRC engine reports 37 violations, but only 33 can be accounted for. This discrepancy is due to the way the DRC engine counts violations. Any report (such as saying `no violations found`) increments the DRC rule violation counter. As such, the first missing violation is from IDF-1 where VIV outputs the provenance of the design, while the second, third, and fourth missing violations are from IDF-4, IDF-5, and IDF-6 reporting that no violations were found.

7. Select each violation as desired. Notice that the violation is highlighted by the Vivado GUI.

Conclusion

This application note provides a step-by-step example of how to implement a complete Zynq-7000 IDF design. All of the necessary IDF steps are shown, highlighting the rules and guidelines detailed in *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)* (XAPP1222) [Ref 3]. This lab gives details on how functions are to be isolated, specific differences between a normal partition flow and a design using the IDF, information on IDF-specific HDL code mnemonics, and trusted routing rules. A designer wishing to create an IDF design should find all the necessary details using this application note and XAPP1222 [Ref 3].

Reference Design Files

You can download the Reference Design Files for this application note from the Xilinx website.

The reference design matrix in Table 1 indicates the tool flow and verification procedures used for the provided reference design.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target device	Zynq-7000 XC7Z020
Source code provided	Yes
Source code format	VHDL
Design uses code or IP from existing reference design, application note, third party, or Vivado software? If yes, list.	No

Table 1: **Reference Design Matrix *(Cont'd)***

Parameter	Description
Simulation	
Functional simulation performed?	Yes
Timing simulation performed?	Yes
Test bench used for functional and timing simulations?	Yes
Test bench format	VHDL
Simulator software and version used	Vivado Design Suite 2015.2
SPICE/IBIS simulations?	No
Implementation	
Implementation software tools and versions used	Vivado Design Suite 2015.2
Static timing analysis performed?	Yes
Hardware Verification	
Hardware verified?	No
Hardware platform used for verification	N/A

Table 2: **Device Utilization**

Device	Speed Grade	Package	Post-Synthesis Slices	Post-Implementation Slices
XC7Z020CLG484	-1	CLG484	9116	9028

References

The following sites are referenced in this document:

1. *7 Series Isolation Design Flow Lab Using ISE Design Suite 14.4* (XAPP1085)
2. Isolation Design Flow website
3. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)* (XAPP1222)
4. *Vivado Design Suite User Guide - Designing IP Subsystems Using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide - Partial Reconfiguration* (UG909)
6. *Vivado Design Suite User Guide - Hierarchical Design* (UG905)
7. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
8. Aerospace and Defense Security Monitor IP Core Product Marketing Brief

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/20/2016	1.0	Initial Xilinx release.
02/24/2016	1.1	Updated the title, Figure 28, and Figure 38.
03/21/2016	1.1.1	Updated the title to clarify support for 7 series FPGAs.

Disclaimer