Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 SoCs (Vivado Tools)

Author: Satya Pitaka

Summary

This application note describes how to implement security- or safety-critical designs using the Xilinx® Isolation Design Flow (IDF) with the Xilinx Vivado® Design Suite. Design applications include information assurance (single chip cryptography), avionics, automotive, and industrial. This document explains how to:

- Implement isolated functions in a single Xilinx 7 series FPGA or Zynq®-7000 SoCs [1] in commercial, defense, industrial, and automotive grades using IDF.

  ◦ For example, implementation might include red/black logic, redundant Type-I encryption modules, or logic processing multiple levels of security. Or for safety applications, implementation might include 1oo2, 1oo2D, and 2oo3 modules.

- Verify the isolation using the Xilinx Vivado Isolation Verifier (VIV).

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 high-assurance applications. If you wish to add additional security to your design, the Security Monitor IP, developed by Xilinx, can be purchased. If you embed this IP, modifications to the steps in this document must be made as described in *Integration and Verification of Security Monitor 3.0 for 7 Series FPGAs and Zynq-7000 All Programmable SoC* (XAPP796). Refer to the *Aerospace and Defense Security Monitor IP Core Product Marketing Brief* [Ref 1] or contact your local Xilinx representative for more information. If the target application requires mask control, a defense-grade (XQ) device might be needed.

This application note is similar to the application note *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (ISE Tools)* (XAPP1086) [Ref 2] with the primary difference being this document is specific to using the Xilinx Vivado Design Suite, whereas XAPP1086 is specific to using the Xilinx ISE® Design Suite for developing IDF designs for the 7 series FPGA devices and Zynq-7000 SoC devices. The rules for IDF defined in this application note do not differ from those defined in XAPP1086, but the methodology for implementation using Vivado tools does.

---

1. The FPGAs and SoC are called FPGA/SoC in the rest of the document.

EΣ XILINX®

Note: Not all 7 series FPGA devices are supported by IDF. For the supported devices list, see Table 1.

Table 1: **IDF Supported Devices**

Xilinx Device	IDF Support Status
Zynq®-7000 SoC family	Supported
Spartan®-7 FPGA	Only XC7S50 device is supported
Artix®-7 FPGAs	All devices are supported except XC7A12T and XC7A25T
Kintex®-7 FPGAs	Supported
Virtex®-7 FPGAs	Supported

This application note is accessible from the Xilinx Isolation Design Flow website [Ref 3]. 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.

Introduction

The flexibility of programmable logic affords security- and safety-critical industries many advantages. However, before IDF, in applications such as information assurance, government contractors and agencies could not realize the full capability of programmable logic due to isolation, reliability, and security concerns, and were therefore forced to use multichip solutions. To address these concerns, the isolation design flow (IDF) was developed to allow independent functions to operate on a single chip. Examples of such single chip applications include, but are not limited to, redundant Type-I cryptographic modules or resident safety- and non safety-critical functions. The successful completion of the Xilinx Isolation Design Flow has allowed Xilinx to provide new technology for the information assurance (IA) industry as well as provide safety-critical functions in avionics, automotive, and industrial applications.

Isolation Design Flow

Developing a safe and secure single chip solution containing multiple isolated functions in a single FPGA is made possible through Xilinx isolation technology. Special attributes such as HD.ISOLATED and the features it enables are necessary to provide controls to achieve the isolation needed to meet certifying agency requirements. To better understand the details of the IDF, the designer should have a solid understanding of the hierarchical design flow. Many of the terms and processes in the partition flow are utilized in the IDF. Areas that are different supersede the partition design flow and are identified in this application note.

Common Terminology

Throughout this document the terms *ownership*, *function*, *logic*, *region*, and *fence* are used extensively. These terms are defined as follows:

Ownership (physical/logical)—The concept of physical versus logical ownership is an important concept to understand when using the IDF. This concept is covered in detail in the section Trusted Routing Design Guidelines.

Function—A collection of logic that performs a specific operation (example: an AES encryptor).

Logic—Circuits used to implement a specific function (examples are: flip-flop, look up table, RAM, to name a few).

Isolated Region/Pblock—A physical area to implement logic.

Fence—A set of unused tiles in which no routing or logic is present.

Trusted Routing—Trusted routing is automatically enabled after the HD.ISOLATED attribute is set to "true" on at least one isolated module. These routes are a subset of existing routing resources that meet the following restrictions:

- No entry or exit point in the fence between isolated regions

- One source and one destination region

- Its entirety stays contained in the source/destination regions

- It does not come within one fence tile from another unintended isolation region

These rules act as a filter to all available routes in a given design. An example of routes that will be filtered is shown in Figure 1. Example routes excluded for programmable interconnect points (PIPs) outside the intended isolation regions or proximity to unintended isolation regions are shown.

Note: Routes depicted are for demonstration purposes only.

Send Feedback
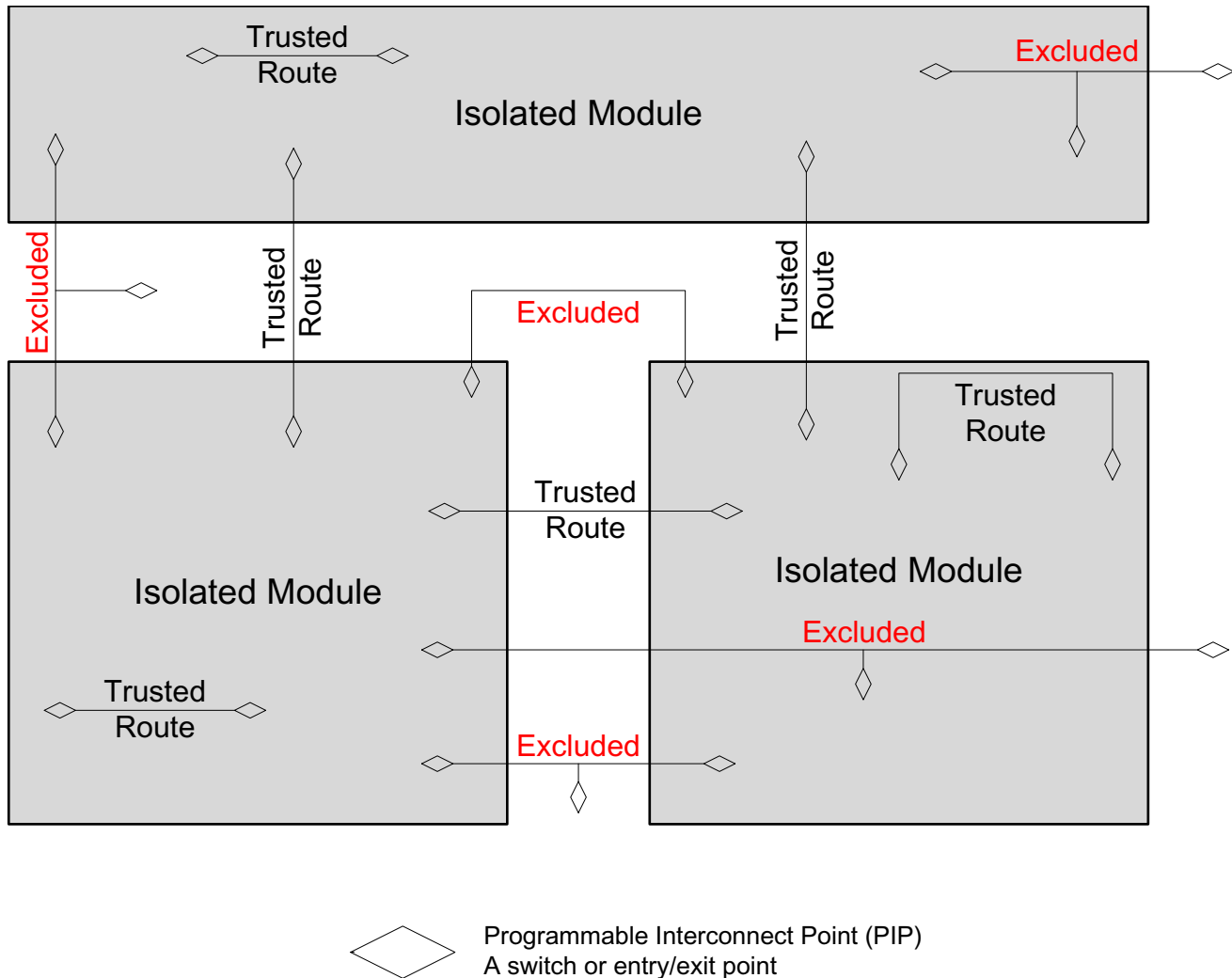
Figure 1: **Available Routes after Applying Trusted Routing Rules**

Rules

A secure or safety-critical solution can be achieved while using FPGA design techniques and coding styles with only moderate modifications to the development flow. IDF development requires the designer to consider floorplanning much earlier in the design process to ensure that proper isolation is achieved in logic, routing, and I/O buffers (IOBs). In addition to early floorplanning, the development flow is based on hierarchy, that is, each function you wish to isolate must be at its own level of hierarchy. Although this flow requires additional steps, the hierarchical approach has certain advantages.

There are a few unique design details that must be adhered to in order to achieve an FPGA-based IDF solution. Carefully consider all aspects of the design details explained in subsequent sections of this application note. These considerations include:

• Each function to be isolated must be in its own level of hierarchy.

Send Feedback

- A fence must be used to separate isolated functions within a single chip.

- IOBs must be instantiated inside isolated modules for proper isolation of the IOB. This can be achieved by manual user instantiation or automatically by the tools.

   Note: Automatic logical inferencing by the tools is unique to the Vivado Design Suite.

- On-chip communication between isolated functions is achieved through the use of trusted routing (Tools automatically choose trusted routes along coincident physical borders).

Top Level Logic

Isolated designs must take care to keep the amount of top level logic to a minimum. In a typical IDF design, the only logic at the top should be Clock logic. Any component that is not part of an isolated module in the design hierarchy is optimized to the top level. Because isolation is defined by the HD.ISOLATED attribute being set on a hierarchical module, all top logic is, by default, NOT isolated. This has the following implications:

- There are no placement constraints on top level logic other than it will not be placed in the fence.

   ◦ Top level logic can be placed in any isolated region.

- There are no routing restrictions on top level logic other than it will not violate the fence with used PIPs.

   ◦ Top level routes can route to, from, and through any isolated region.

While IDF states top level logic should be very minimal (clocks), there are cases where it cannot be avoided. To prevent top level logic from being placed in the fence, it is necessary to add prohibits to all sites not ranged by a Pblock. A *prohibit* is a directive to the placer that disallows (prohibits) any placement in that site. This should be done after initial floorplanning and IOB pin placement using the following Tcl commands.

TIP: Copying and pasting from PDF files does not always work. Retyping is advised.

```
#First, select all sites:

    set_property prohibit 1 [get_sites]

#Now, subtract the ones that have been ranged by a pblock

    set_property prohibit 0 [get_sites -of_objects [get_pblocks]]

#Now, subtract out the BUFH and BUFG tiles (IDF rules state not to add BUFH and BUFG tiles
to pblock).

    set_property prohibit 0 [get_sites -filter {SITE_TYPE =~ "*BUFH*"
    || SITE_TYPE =~ "*BUFG*"}]
```

For cases where you need some top level sites not ranged in a Pblock, those sites can have their prohibits cleared as per the above syntax.

Send Feedback

Note: This limitation is fixed in the Vivado 2016.1 release. For versions 2016.1 and beyond, Vivado automatically prohibits logic from being placed in the fence.

Reference Design

For clarity, an example Single Chip Cryptographic (SCC) design is used throughout this application note to describe the design details and tool flow. Figure 2 shows the floorplan for the lab design as implemented in an XC7Z020CLG484-1 device. It consists of four isolated regions. Additionally, this design has been implemented with Xilinx Vivado Design Suite 2015.1, verified by the VIV, and provided to the designer as a reference. For more details, see Reference Design Files.
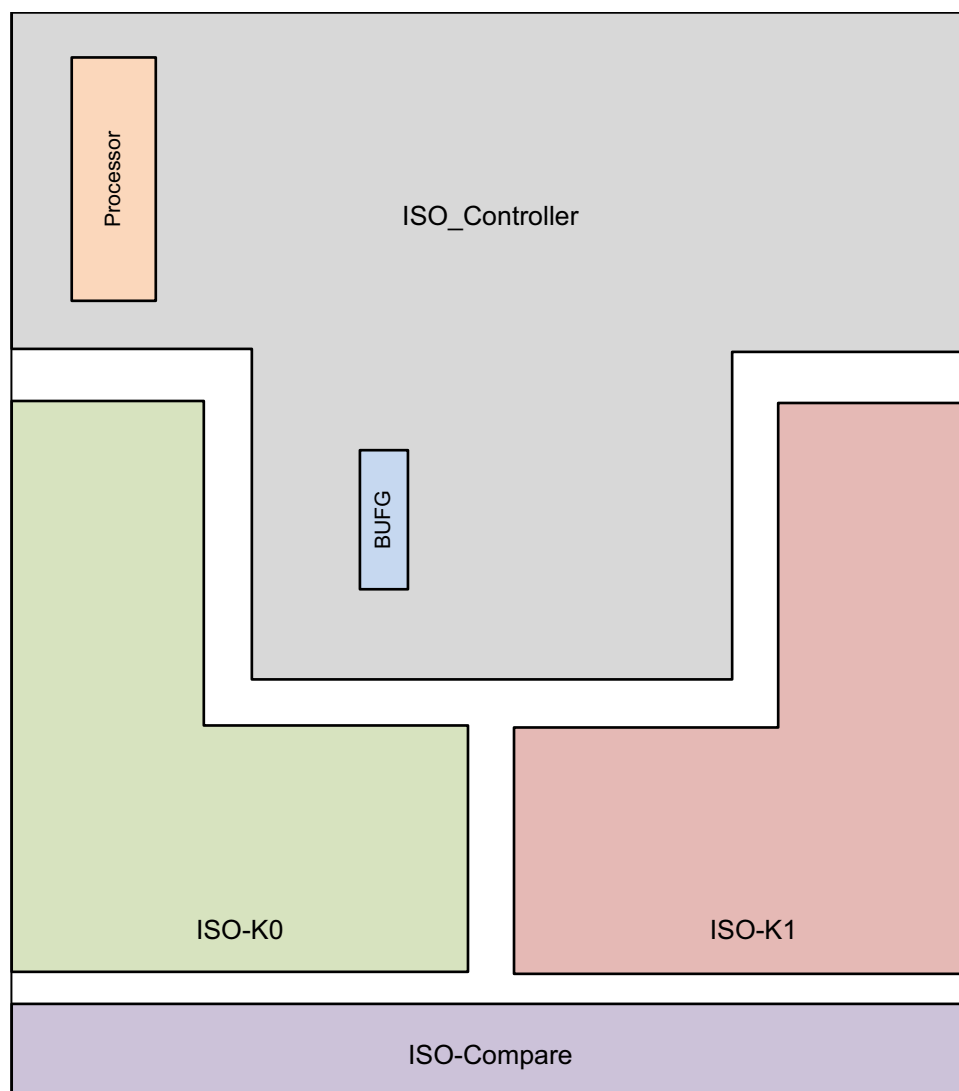
X17814-090616

Figure 2: **SCC Solution using IDF - Implementing Two Redundant Keccak Modules and Compare Logic**

Send Feedback

The example design consists of two redundant Keccak cryptographic hash modules (ISO_K0 and ISO_K1), whose outputs are sent to a comparator (ISO_Compare) block, and a processor control (ISO_Controller) module used to supply clocks and resets. The redundant Keccak hash modules, compare function, and processor module are all isolated within a single Zynq-7000 SoC device as shown in the Vivado tool routing resources view in Figure 3.
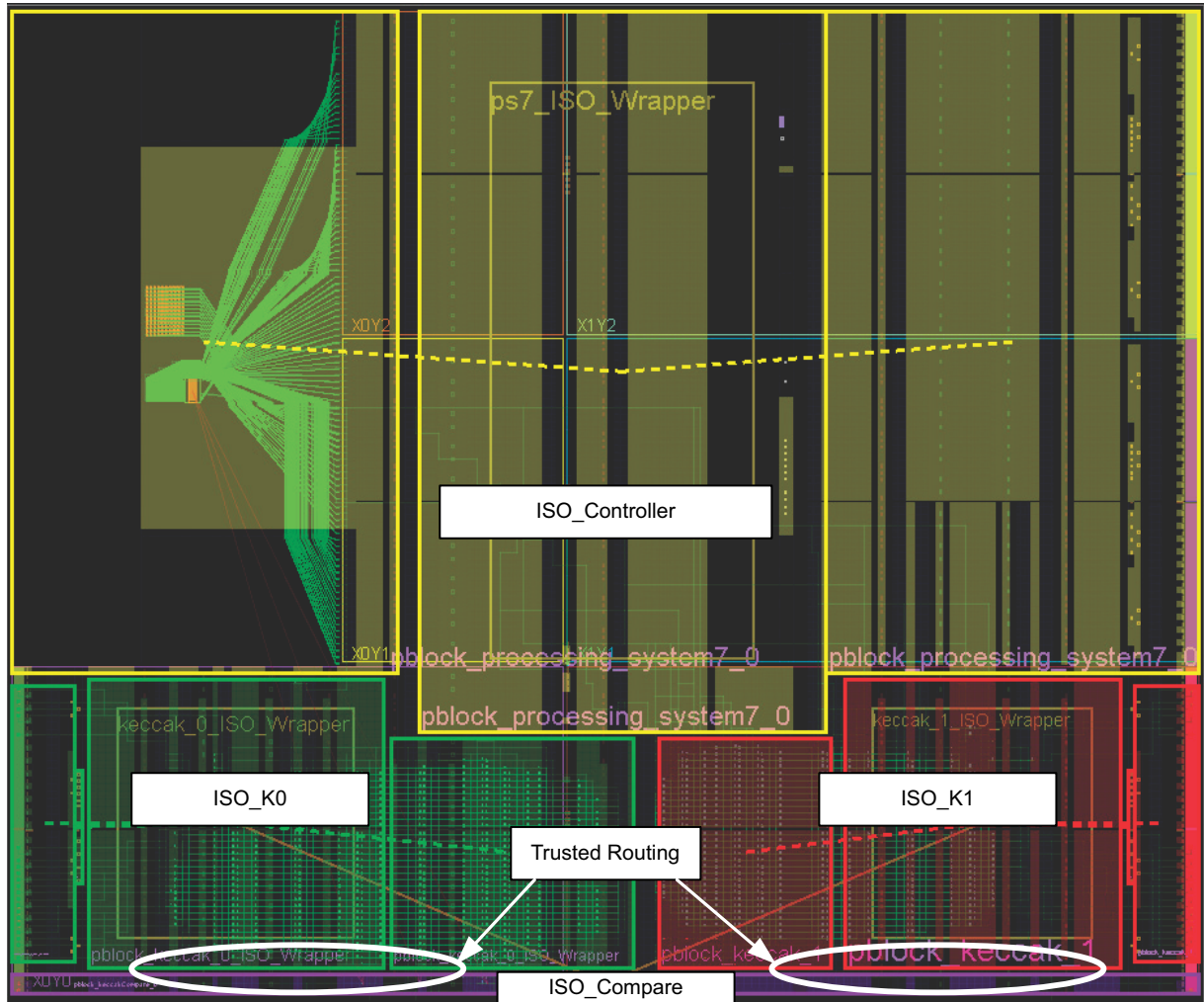
X17815-090616

Figure 3:   **Implemented SCC Solution from Figure 2**

To illustrate the IDF and its capabilities, this design implements isolated, redundant Keccak hash modules with a compare block. While this architecture is chosen somewhat arbitrarily and is not meant to represent an actual user design, it does provide a valuable example in explaining SCC design details.

The Vivado tool, version 2015.1 or later, allows you to import RTL source code directly into the project so the project can be done using just the Vivado tool. Figure 4 is a hierarchical diagram of the various VHDL sub-blocks used in the implementation of this design.
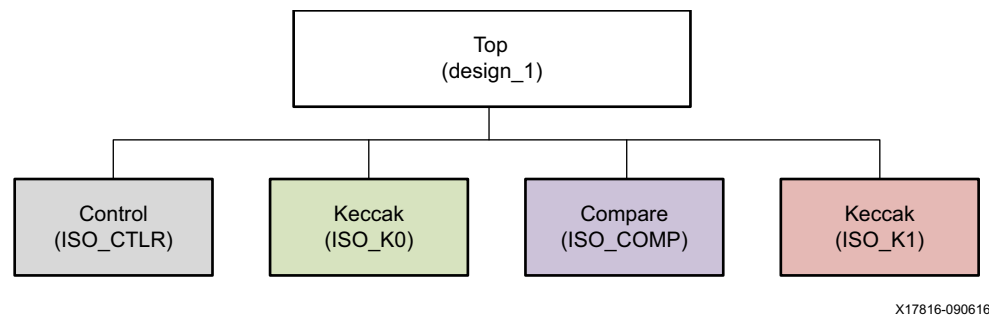

X17816-090616

Figure 4: **Hierarchy for the SCC Implementation of the Reference Design**

Additional lower level hierarchy is allowed for each function. There is no theoretical limit on the number of functions allowed. For example, if the Security Monitor is to be integrated into this design, it is a fifth function with its own hierarchy under Top.

Isolation Properties

The Vivado tool uses two specific properties to create an isolated design: HD.ISOLATED and HD.ISOLATED_EXEMPT.

In ISE, Isolated="true" was set in the `xpartition.pxml` file to enable isolation of a specified function. The Vivado tool, however, enables isolation of a specific function by the application of the HD.ISOLATED property on the function. This property can be set either in the XDC file or the Vivado Integrated Design Environment (IDE).

The format is:

```
set_property HD.ISOLATED true [get_cells
<function_hierarchical_path>/<function_instance_name>]
```

or

```
set_property HD.ISOLATED 1 [get_cells
<function_hierarchical_path>/<function_instance_name>]
```

An example of the command is:

```
set_property HD.ISOLATED 1 [get_cells */keccak_0_ISO_Wrapper]
```

By default, when HD.ISOLATED is enabled on a function, all components and routing that belong to that function are isolated. This means that routing is contained within that function, unless it is communicating with another isolated function. This also means that all components of that function are placed in the corresponding isolated region. With this, global logic instantiated within an isolated function could not be routed globally. Vivado IDF allows you to instantiate global logic at any level. To support this and enable this routing of the global logic, the property HD.ISOLATED_EXEMPT can be set on the global instance to override the default

isolation behavior. By setting this property on the global logic instance, Vivado routes it as global logic instead of isolated logic.

The format is:

```
set_property HD.ISOLATED_EXEMPT true [get_cells
<global_logic_hierarchical_instance_path>/<global_logic_instance_na
me>]
```

or

```
set_property HD.ISOLATED_EXEMPT 1 [get_cells
<global_logic_hierarchical_instance_path>/<global_logic_instance_na
me>]
```

An example of the command is:

```
set_property HD.ISOLATED_EXEMPT 1 [get_cells
*/ps7_ISO_Wrapper/bug_inst_1]
```

See the Guidelines for Controlling Global Clocking Logic for more information regarding the HD.ISOLATED_EXEMPT property, including an example of setting HD.ISOLATED_EXEMPT on all global logic in the design.

Isolation Modules

Each function requiring isolation must be in its own hierarchical block. The reason for this is that each function has an isolation attribute set and that attribute applies to that function and its entire logical hierarchy. It is not possible to have nested isolated functions.

IDF has many additional constraint rules as opposed to traditional Partial Reconfiguration (PR) [Ref 4] or Hierarchical Design [Ref 5] flows. It is necessary to define one key attribute for each function to be isolated to invoke these rules.

The HD. ISOLATED attribute needs to be set for each function for the backend implementation tools to use the isolated rules. This allow for floorplanning of an isolated design when using the IDF, and also protects redundant functions from undesired optimization. If the HD.ISOLATED attribute is not set, you can only floorplan a limited subset of the FPGA components— specifically configurable logic block (CLB) slices, block RAM (BRAM), and DSP sites. Because IDF requires the addition of all user tiles, this is a clear indicator that the HD.ISOLATION has not been set.

Send Feedback

Isolation Fence

Xilinx FPGAs and SoCs are modular tile-based devices. No matter which device is chosen, the FPGA/SoC consists of the same basic building blocks tiled over and over again. The basic building blocks of the 7 series family are composed of CLBs (type LM and type LL), block RAM, and DSP etc. Each tile is constructed of some hardened logic and one or more Global Routing Matrix (GRM) blocks. Analysis of the FPGA is done at the tile level and this is how isolation fences are determined (what tiles are valid and how many). A high-level view of this construction is shown in Figure 5.
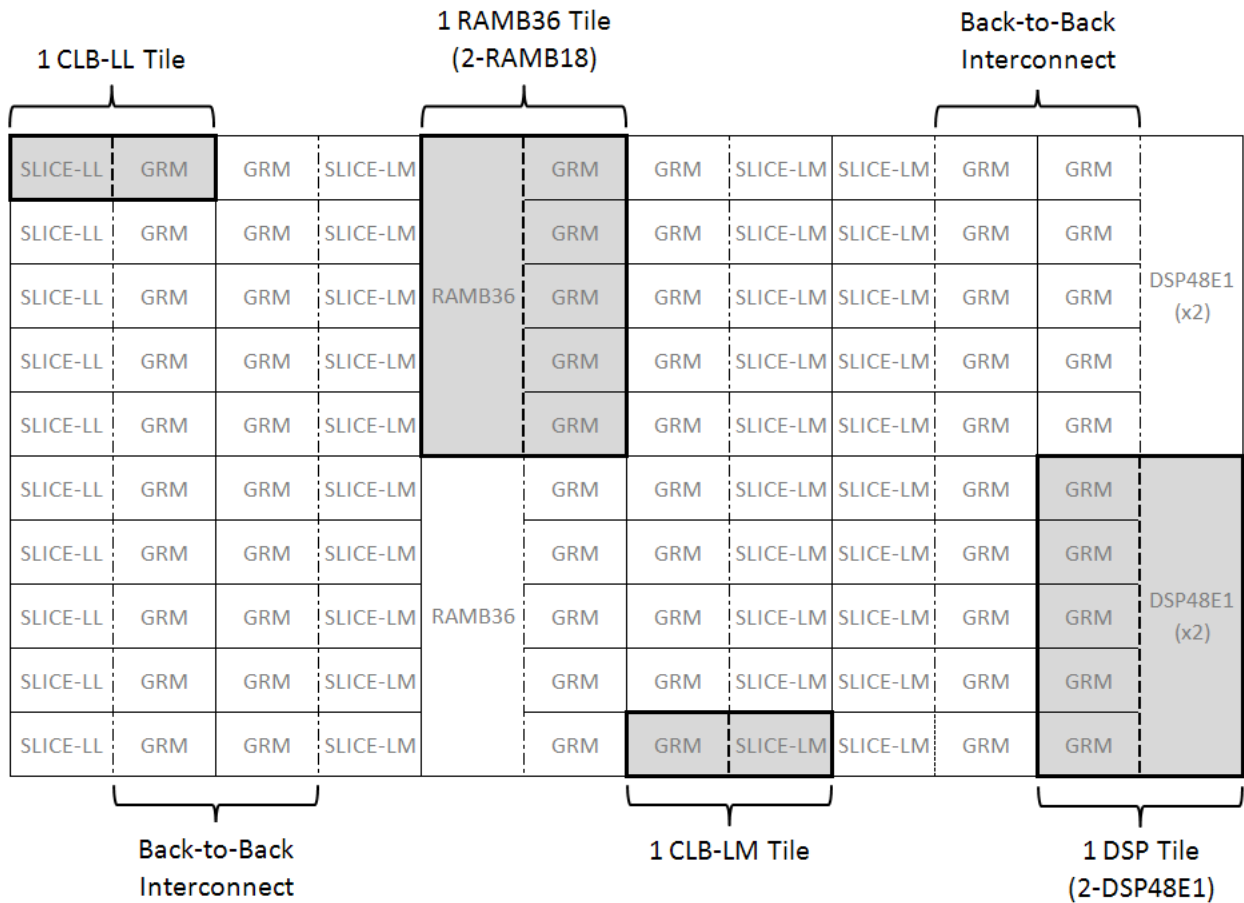
Figure 5:   **Representation of the Tile-Based 7 Series Architecture**

To achieve isolation within a single device, the concept of a fence is introduced. The fence is just a set of unused tiles, as described, in which no logic is present. The results of the isolation analysis performed by Xilinx shows that one, or sometimes two, such tiles placed between isolated regions guarantees that no single point of failure exists that would compromise the isolation between the two regions. An example of fence placement between two isolated modules is shown in Figure 6.
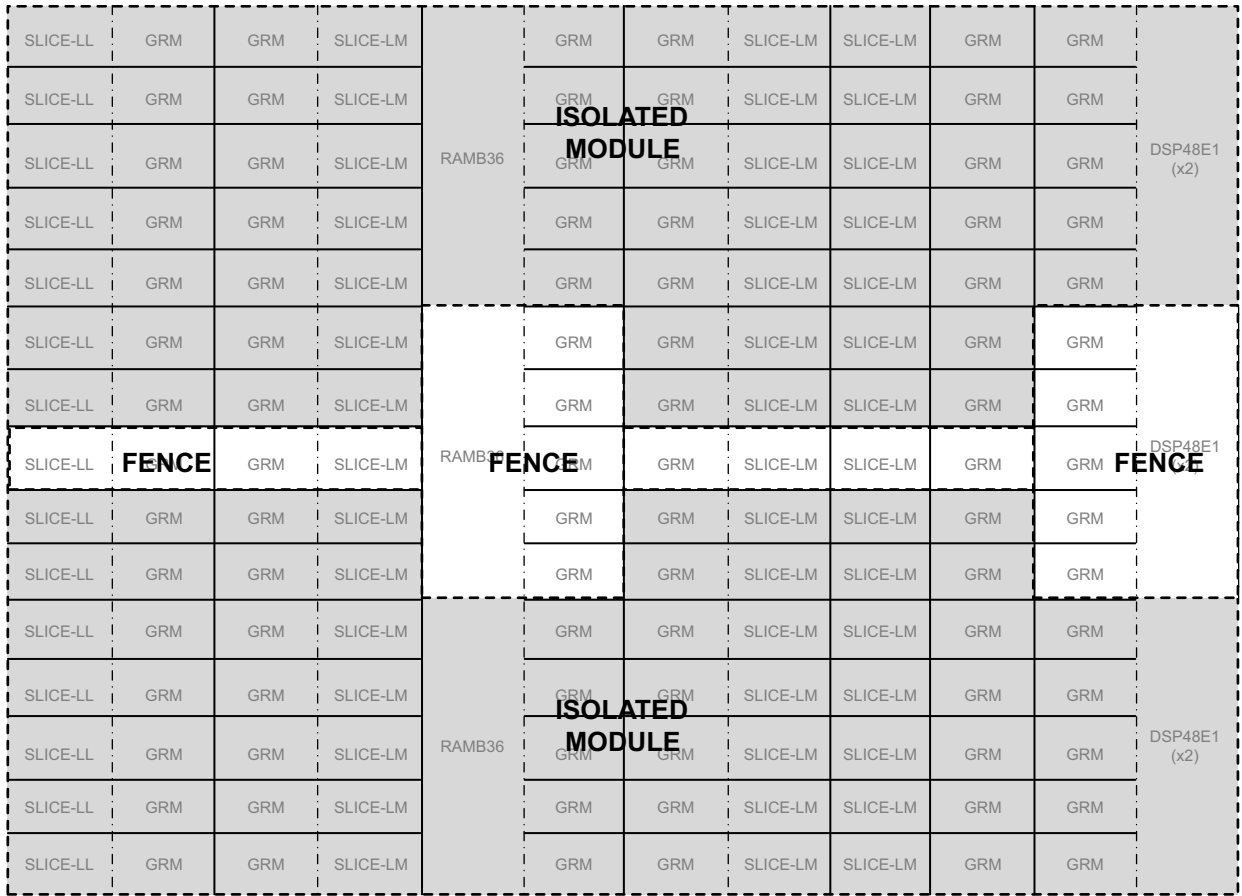
X17818-090616

Figure 6: **Isolated Functions Separated by a Fence of Unused Logic Tiles**

The fence location is not directly specified by the designer; rather it is created indirectly by applying the appropriate physical constraints to each user-defined region to be isolated. In the reference design, the functions that must be isolated are the two redundant Keccak hash modules, the Keccak comparator block, and the PS7_Controller block.

The minimum fence width is determined by a detailed schematic analysis. This analysis provides for a minimum of two faults (though in most cases many more) before the separation between two isolated functions is violated. While it is not mandatory to keep the size of the fence to this minimum (one tile), you can incur a stiff penalty for using larger fences. Because IDF rules prevent routing touchdowns in the fence, a fence size of one (minimum allowed) prohibits the

Send Feedback

use of all routes (to cross the fence) that span only a single tile (singles). Using larger fences have the consequences listed in Table 2.

*Table 2:*   **Consequences of Fence Sizes**

Horizontal Fence Size	Consequence
1 (minimum required)	All single (1) routes are removed from available routing resources necessary to cross the fence.
2-3	All single (1) and double (2) routes are removed from available routing resources necessary to cross the fence.
4-5	All single (1), double (2), and quad (4) routes are removed from available routing resources necessary to cross the fence.
6 or more	A fence this wide cannot be crossed with trusted routes. (Isolation requirements prevent it.)
Vertical Fence Size	**Consequence**
1 (minimum required)	All single (1) routes are removed from available routing resources necessary to cross the fence.
2-3	All single (1) and double (2) routes are removed from available routing resources necessary to cross the fence.
4-5	All single (1), double (2), and quad (4) routes are removed from available routing resources necessary to cross the fence.
6-8	All single (1), double (2), quad (4), and hex (6) routes are removed from available routing resources necessary to cross the fence.
9 or more	A fence this wide cannot be crossed with trusted routes. (Isolation requirements prevent it.)

The Trusted Routing rules act as a filter to the router. Any route that violates the rules of Isolation get removed from the list of valid resources to the router. This is what creates a *Trusted Route*. If the fence is too wide, all routes are removed and the design will not be able to route (if communication across that boundary is intended). It is recommended to keep the minimum size fence width allowed to maximize the routing resources available between two isolated functions.

Floorplanning the Reference Design

The floorplan of the reference design is shown in Figure 7. Where inter-module communication (Trusted Routing) is necessary, regions must be coincident with each other with a fence tile between the two intended regions. See Communicating Between Isolated Functions for more details.
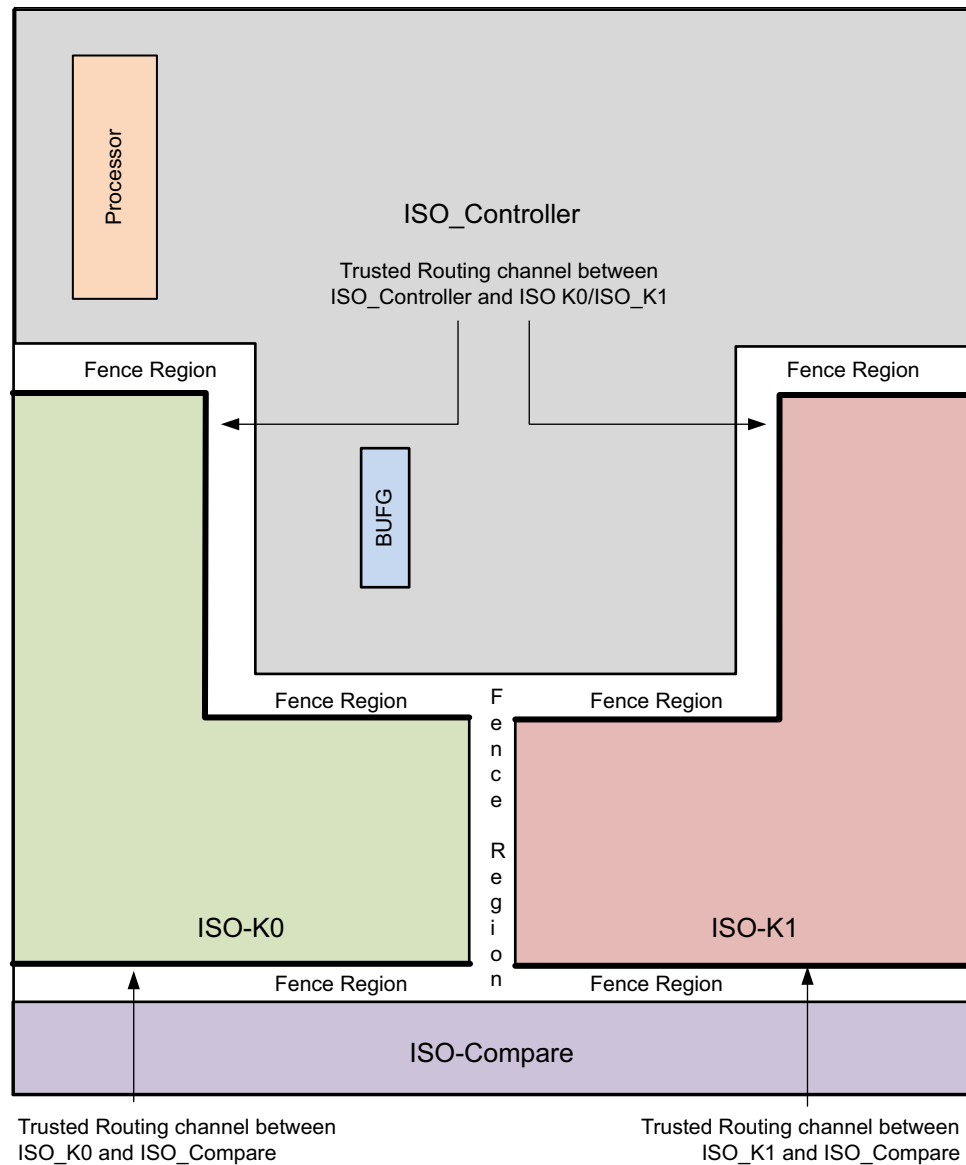
X17819-090616

Figure 7: **Floorplan of the Reference Design Highlighting the Trusted Routing Channels**

To create an isolated region, the logic needs to be constrained to a specific region of the FPGA and Tcl commands are used in the Xilinx Design Constraints (XDC) file to accomplish this. The following example uses one of the Keccak modules to show how the Tcl commands are used to create an isolated region.

First, the Pblock is created via the `create_pblock` command, to allow adding logic instances:

```
create_pblock pblock_ISO_K0
```

Next, the logic elements are added to the Pblock using the `add_cells_to_pblock` command:

```
add_cells_to_pblock [get_pblocks pblock_ISO_K0] [get_cells -quiet
[list */keccak_0_ISO_Wrapper]]
```

Next, isolation regions must be assigned to a specific range of logic in the FPGA. Isolation regions can be defined in terms of SLICEs, RAMB18s, RAMB36s, IOBs, ILOGICs, OLOGICs, IDELAYs, ODELAYs, PLLs, MMCME2s, IN_FIFOs, OUT_FIFOs, BUFGCTRLs, GTXs, DSP48s, and so on. Adding components (specified range of sites) to an isolated region (Pblock) is done by using the Tcl command `resize_pblock` as follows:

```
resize_pblock [get_pblocks pblock_ISO_K0] -add
{SLICE_X32Y4:SLICE_X55Y38 SLICE_X0Y4:SLICE_X31Y47}
```

The generic format for the Tcl command for adding the ranges (`resize_pblock`) is:

```
-add {<comp name>_XaYb:<comp name>_XcYd}
```

Where <comp name> = name of desired component, for example: SLICE, DSP48, RAMB36, and so on.

A full listing of this syntax can be found in *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6].

The component name can be identified by pointing at it in the Device view in the Vivado tool and reading it off the bottom right side of the screen.

Coordinates can also be identified from the same location that the component name was identified.

a, b, c, d = Coordinates of the starting component and the ending components

Although the reference design does not use DSPs or block RAMs, they have been added to the isolated region so that the routing resources contained by these blocks can be used. As a general rule, all available resources (except BUFGs and BUFHs) should be assigned to the isolation region unless there is specific need to exclude that resource. This is the default selection when generating the XDC file using the Vivado tool. An exception to this guideline is the clock management tile (CMT) components (MMCME2_ADV, PLLE2_ADV, PHASER_IN_PHY, PHASER_OUT_PHY, PHY_CONTROL, IN_FIFO, and OUT_FIFO) should only be added to an isolation region if the specific site is used in the design.

Send Feedback

Communicating Between Isolated Functions

When communication between isolated functions is required, there are two possible solutions:

- Trusted routing (preferred method). For details on the user rules required when using Trusted routing, refer to Trusted Routing Rules.

- Signals can be taken off-chip from one isolated region, routed through the PCB, and then brought back on-chip in a separate isolated region. While possible, this method is not preferred due to both complexity and practicality.

Off-Chip Communication - Input/Output Buffer Control

If an isolated function has inputs or outputs that must come from or go off-chip, these signals must have their IOBs inferred or instantiated inside an isolated partition. While this is different from standard FPGA design practice, it is required in order to have control over the routing of the signals from the IOB to the function. If the IOB is not part of the isolated logic, there is no control of how the signal is routed from the IOB to the logic.

It is a synthesis default that all IOBs be inserted towards the end of the process and at the top level of the hierarchy. This contradicts the requirements of the Isolation Design Flow. Before Vivado Synthesis, this required you to instruct the synthesis engine to keep from inserting these buffers and manually instantiating each IOB at the proper level of logical hierarchy. This is a non-trivial task and a burden on the developer.

Previous to Vivado Synthesis, this required the developer to instruct the synthesis engine to not insert these buffers. The developer would then have to manually instantiate each IOB at the proper level of hierarchy.

A more detailed description of Vivado tools insertion of IOBs is described in Hints and Guidelines under Automatic Movement of IOB into Isolated Hierarchy, page 58.

Note: If you do not want to use this feature, follow the older methodology of disabling synthesis IOB insertion and manually instantiating the IOBs.

Mapping the Logical View to the Physical View

One of the most difficult concepts in IDF is the relationship between the logical view and the physical view of a user design. *Logical* refers to the actual Hardware Description Language (HDL) design. *Physical* refers to where that design resides in an FPGA/SoC. The mapping from the logical view to the physical view happens when a Pblock is created and a logical instance is assigned to it. This binding of a logical component, or portion of hierarchy, to a physical location in the device is the foundation of the Isolation Design Flow. This is achieved with the following commands:

```
create_pblock pblock_ISO_K0

add_cells_to_pblock [get_pblocks pblock_ISO_K0] [get_cells -quiet
[list */keccak_0_ISO_Wrapper]]
```

Send Feedback

After the Pblock has been created and logic has been assigned to it, the Pblock must be defined (floorplanned) to add the necessary resources as defined in the Floorplanning the Reference Design. Note that any component or routing resource that is not ranged by the Pblock definition cannot be used by the logic assigned to that Pblock. Routing resources associated with ranged components are added automatically though not specifically defined by the Pblock definition.

Constraint Summary

Some initial architecting and floorplanning of the FPGA/SoC, coupled with a small list of constraints, is all that is required to achieve isolation of specific functions within a single FPGA or SoC. It is important to note that any logic not isolated is, by definition, unconstrained logic. As such it can place or route in any isolated region. Due to this, it is highly recommended that only global logic remain unconstrained. With a few exceptions, this is mandatory. The following example shows the constraints generated by the tools when an isolated function is floorplanned into a specific region of the device (using Pblocks).

Putting together all of the constraints, isolation of the Keccak(0) function is achieved by the following statements (commands):

```
create_pblock pblock_ISO_K0
add_cells_to_pblock [get_pblocks pblock_ISO_K0] [get_cells -quiet [list
*/keccak_0_ISO_Wrapper]]
resize_pblock [get_pblocks pblock_ISO_K0] -add {SLICE_X32Y4:SLICE_X55Y38
SLICE_X0Y4:SLICE_X31Y47}
resize_pblock [get_pblocks pblock_ISO_K0] -add {BUFHCE_X0Y0:BUFHCE_X1Y11}
resize_pblock [get_pblocks pblock_ISO_K0] -add {BUFIO_X0Y0:BUFIO_X0Y3}
resize_pblock [get_pblocks pblock_ISO_K0] -add {BUFMRCE_X0Y0:BUFMRCE_X0Y1}
resize_pblock [get_pblocks pblock_ISO_K0] -add {BUFR_X0Y0:BUFR_X0Y3}
resize_pblock [get_pblocks pblock_ISO_K0] -add {DSP48_X2Y2:DSP48_X2Y13
DSP48_X0Y2:DSP48_X1Y17}
resize_pblock [get_pblocks pblock_ISO_K0] -add {IDELAY_X0Y5:IDELAY_X0Y46}
resize_pblock [get_pblocks pblock_ISO_K0] -add {IDELAYCTRL_X0Y0:IDELAYCTRL_X0Y0}
resize_pblock [get_pblocks pblock_ISO_K0] -add {ILOGIC_X0Y5:ILOGIC_X0Y46}
resize_pblock [get_pblocks pblock_ISO_K0] -add {IOB_X0Y5:IOB_X0Y46}
resize_pblock [get_pblocks pblock_ISO_K0] -add {OLOGIC_X0Y5:OLOGIC_X0Y46}
resize_pblock [get_pblocks pblock_ISO_K0] -add {PMV_X0Y1:PMV_X0Y1}
resize_pblock [get_pblocks pblock_ISO_K0] -add {PMVBRAM_X0Y0:PMVBRAM_X3Y0}
resize_pblock [get_pblocks pblock_ISO_K0] -add {PMVIOB_X0Y0:PMVIOB_X0Y0}
resize_pblock [get_pblocks pblock_ISO_K0] -add {RAMB18_X2Y2:RAMB18_X3Y13
RAMB18_X0Y2:RAMB18_X1Y17}
resize_pblock [get_pblocks pblock_ISO_K0] -add {RAMB36_X2Y1:RAMB36_X3Y6
RAMB36_X0Y1:RAMB36_X1Y8}
```

Vivado Isolation Verifier

The Vivado Isolation Verifier (VIV) verifies that an FPGA design partitioned into isolated regions meets stringent standards for fail-safe design. VIV is a collection of design rule checks (DRCs) intended to aid FPGA developers in producing and documenting fault-tolerant FPGA applications developed with the Xilinx Isolation Design Flow (IDF).

VIV is a Tcl script that runs in the Vivado tool framework in the form of DRCs. This allows for a strong Vivado IDE interface to the tool that was not previously available in the older ISE Isolation Verification Tool (IVT).

Constraint Checking (VIV - Constraints)

VIV, on the floorplan, 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 IOB bank.

    Note: While 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 such that a minimum of a one tile-wide fence exists between isolated regions.

Final Isolation Verification (VIV - Implementation)

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

At this step, VIV checks the following:

- Analyzes the complete placed and routed design.

- Tile-based isolation analysis is looking for a barrier (fence) between isolated regions.

    ◦ A user tile acts as a sufficient isolation barrier:

        - If it does not contain any isolated signals (from any isolated region)

        - If it Is configured in the default (unused) state

- VIV does the same pin and I/O checking as in Floorplan mode.

Vivado Isolation Verifier Tcl Script

VIV is a Tcl script that runs in the Vivado tool framework in the form of DRCs. This allows for a strong Vivado IDE interface to the tool that was not previously available in older ISE tools for IVT.

The VIV Tcl script is comprised of six DRCs named IDF-1 through IDF-6. They are invoked like any other DRCs in the Vivado tool:

- From the Vivado IDE under **Tools > Report > Report DRC**

- By clicking the blue checkmark icon in the task bar

- From the Tcl console or from a Tcl script, using `report_drc` and related commands

The IDF DRCs are described as follows:

IDF-1 Provenance—IDF-1 documents the circumstances under which a DRC report was generated including tool versions, date, design name, directory, user, platform, and host.

IDF-2 I/O Bank violations—IDF-2 reports all IOBs in banks that have IOBs from multiple isolation groups.

IDF-3 Package pin violation—IDF-3 reports all package pin adjacency violations. IDF-3 checks that no two adjacent package pins are from distinct isolation groups.

IDF-4 Floorplan violation—IDF-4 reports all floorplan violations. IDF-4 checks that no isolated region is adjacent to or overlaps another isolated region.

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.

IDF-6 Routing violation—IDF-6 reports all routing violations and consists of three checks:

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

2. No inter-region net can use nodes that have PIPs in the fence, except clock nets which can have unused PIPs in the fence.

3. For any tile containing inter-region nets, all such nets must have a common source and load.

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

To ensure VIV runs on the final design and not some cached version of it, it is strongly recommended to run VIV on an exported Design Checkpoint. This can be done by using the **File > Write Checkpoint** action and saving the checkpoint to your desired location. This checkpoint can be opened in a new Vivado Integrated Design Environment (IDE) window by the **File > Open Checkpoint** operation.

VIV is not supplied with Vivado tools. It must be loaded using the **Tools > Run Tcl Script** from the Vivado IDE or using the `source` command in the Vivado Tcl console or in a Tcl script

Send Feedback

(source `viv.tcl`). After `viv.tcl` is loaded, the Report DRC window should be opened using the **Tools > Report > Report DRC** option. This window now contains the IDF DRCs under the heading **Isolation** as shown in Figure 8. Note that the Pblock option under **Isolation** is not related to IDF and should be unchecked.
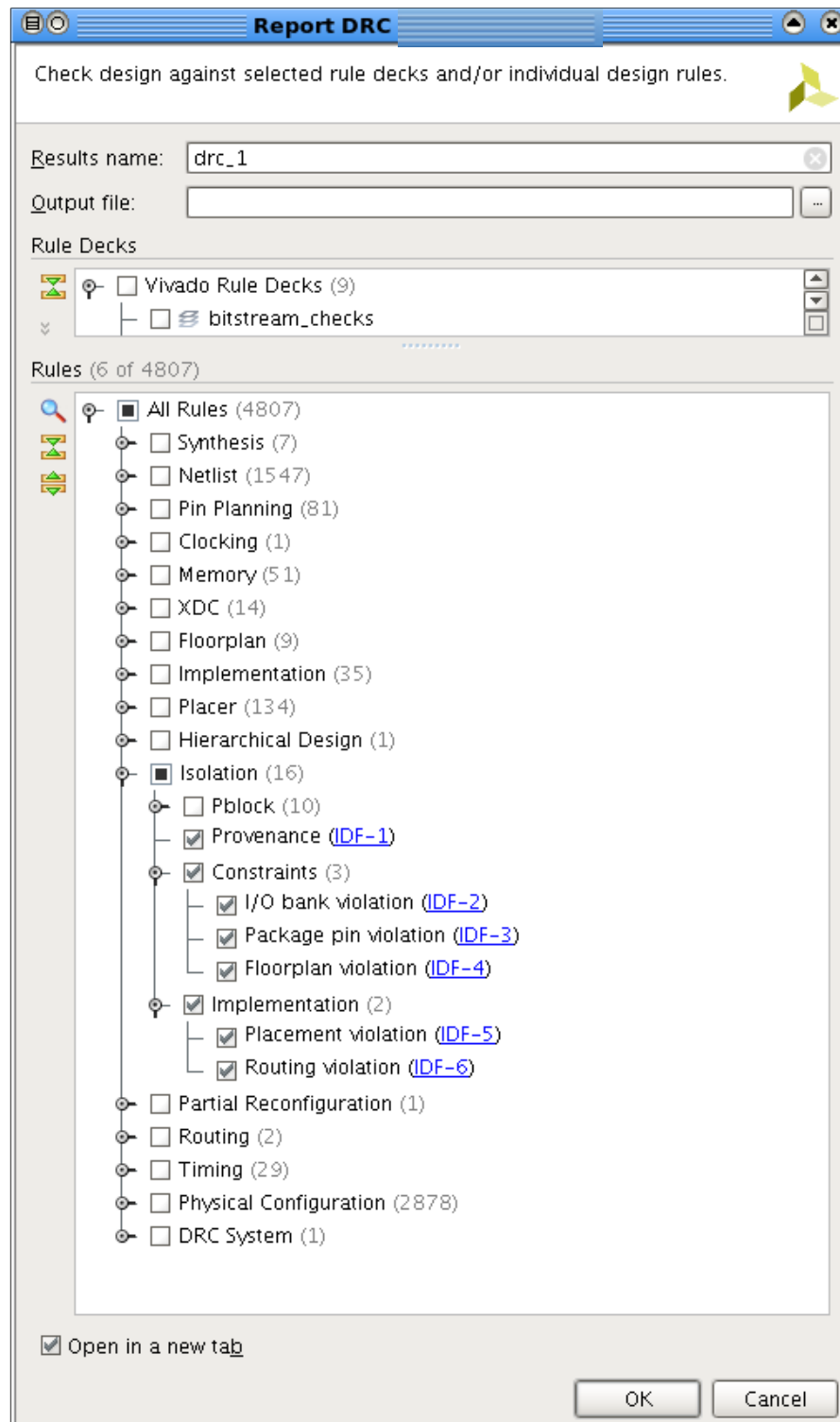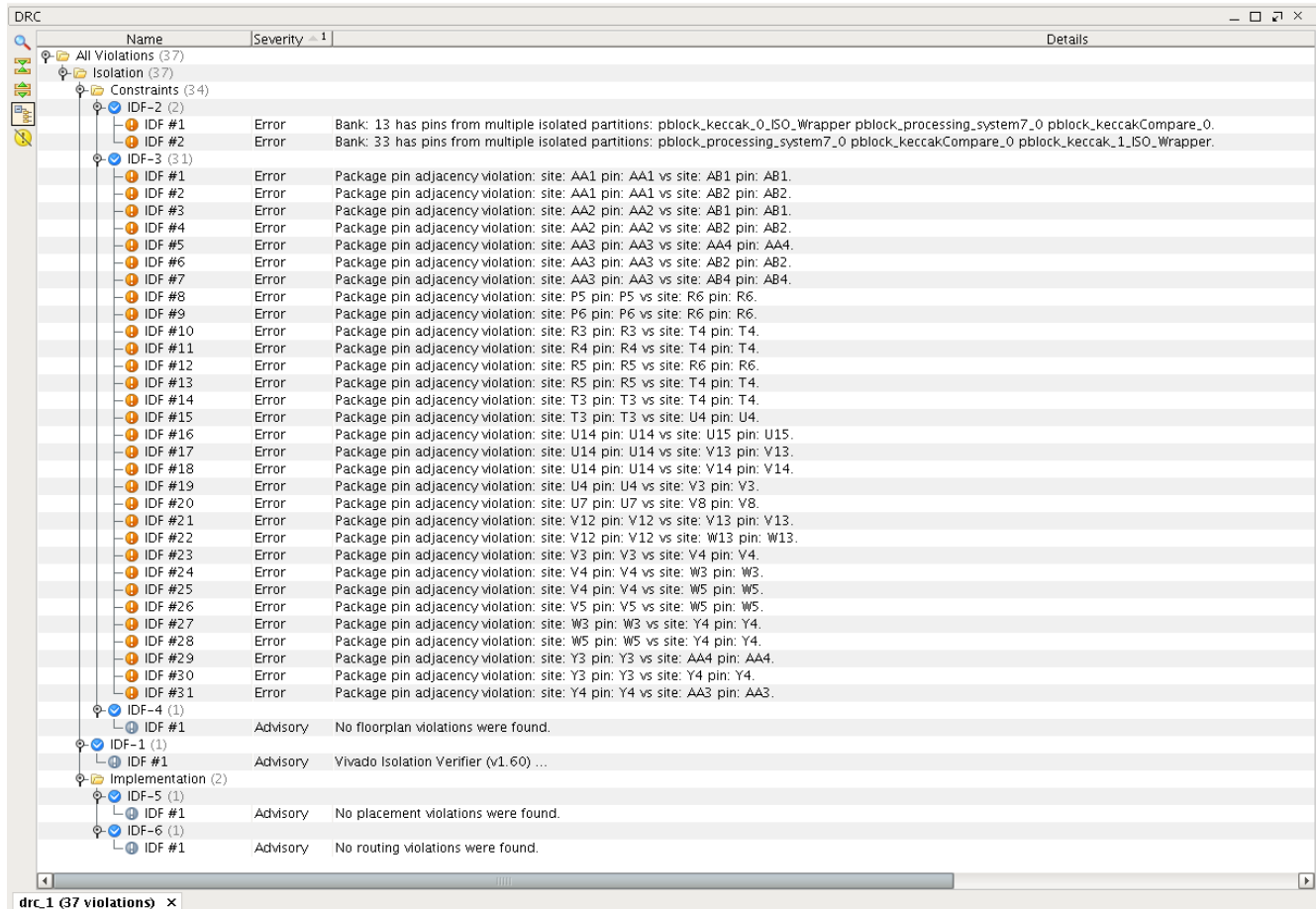
X17820-090616

Figure 8: **IDF DRCs Selected in the Report DRC Dialog Box**

Send Feedback

When run under the Vivado IDE, the IDF DRCs show up in tabular form in the DRC pane just like other Vivado tool DRCs. Each line can be selected for additional detail in the Design Rule Properties window. Design objects associated with the DRC are automatically highlighted in the Device window and Package window. Additionally, some DRCs provide blue underlined links to individual design elements.

DRCs can be invoked at several stages of the flow. Some IDF DRCs can be run on the design constraints. IDF DRCs for the implemented design can be run after the design is implemented.

An example of a DRC report in the Vivado IDE is shown in Figure 9.

X17821-090616

Figure 9:   **IDF DRC Violations from Example Design**

Send Feedback

The highlighted DRC (IDF #31) in Figure 9 refers to a package pin adjacency violation. When this line is highlighted, the corresponding I/O buffers are selected in the Device window as shown in Figure 10 and the corresponding package pins are selected in the Package window as shown in Figure 11.
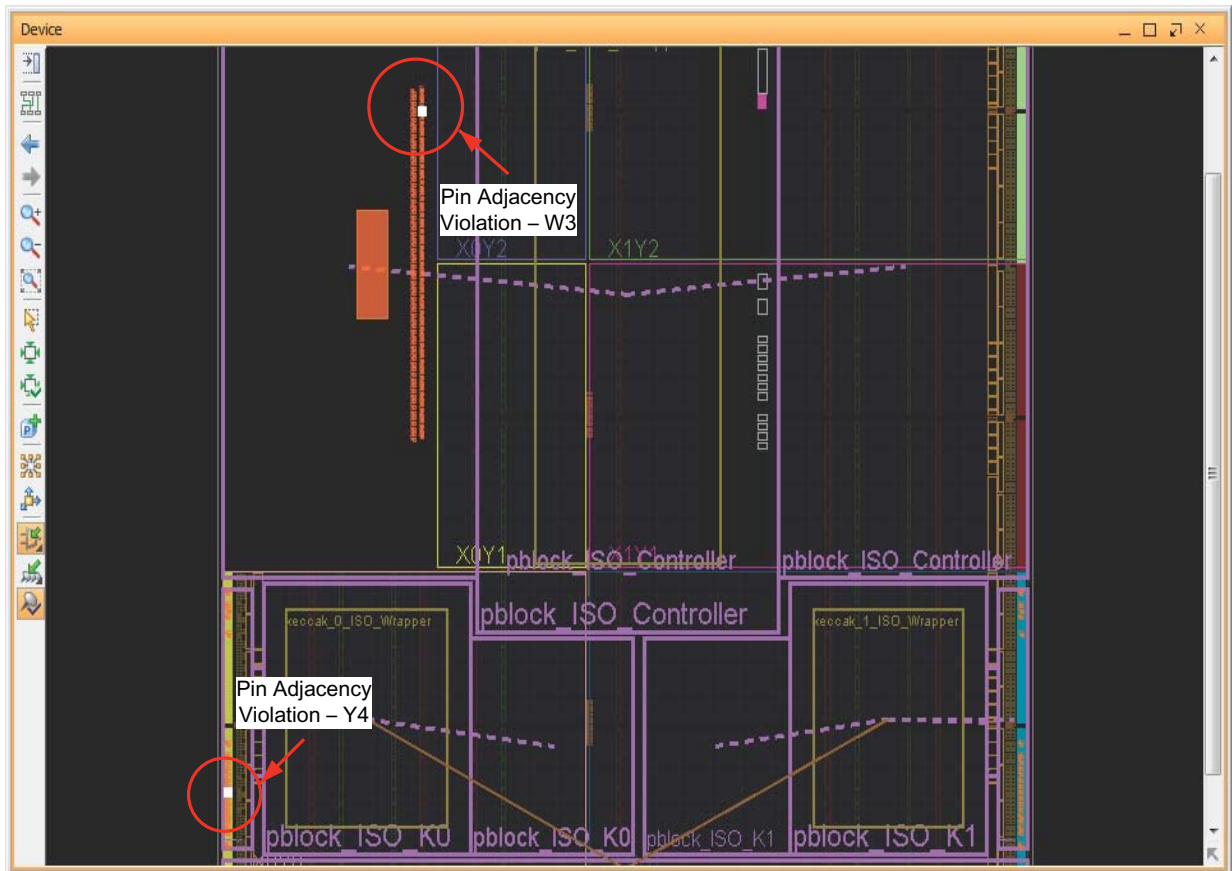
X17822-090616

Figure 10: **Pin Adjacency Violations with I/O Buffers Highlighted in Device Window**

X17823-090616

Figure 11: **Pin Adjacency Violations Highlighted in Package Window**

The text of the violation is displayed in two places in the Vivado IDE. The first line of the text is displayed in the DRC report table as shown in Figure 9. The complete text, along with links to the sites associated with the violation, are displayed in the Details pane of the Violation Properties window as shown in Figure 12.


X17824-090616

Figure 12: **Details Pane of the Violation Properties Window**

Send Feedback

Design Guidance

Rules for the Isolation Design Flow

The Isolation Design Flow (IDF) software methodology allows for logical and physical separation of hierarchical designs. There are many applications requiring the IDF, such as government-grade cryptographic systems, safety-critical applications, and high-availability systems. The guidance in this section represents one particular application using the IDF, in this case, Single Chip Cryptography (SCC). The IDF is not that different from a traditional design flow. Figure 13 shows a flow diagram of both flows side by side for the Vivado tools.

www.xilinx.com

Send Feedback

EXILINX®

FPGA Development Flow (Vivado)   **IDF Extras**
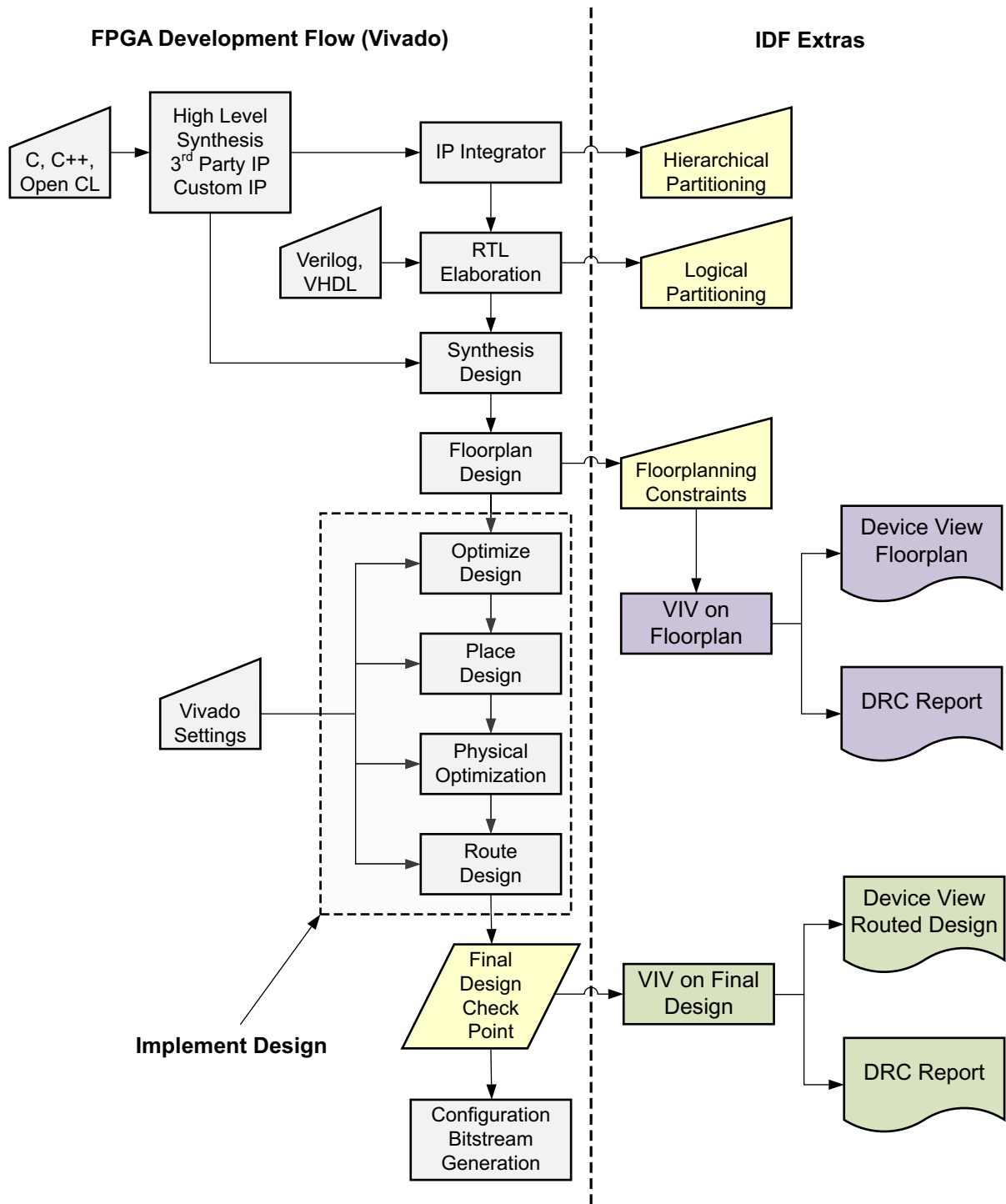
X17825-090616

Figure 13:   **Vivado Isolation Design Flow Relative to a Typical FPGA Design Flow**

Send Feedback

www.xilinx.com

Trusted Routing Design Guidelines

Trusted routing is automatic, but there are a few rules and concepts to be aware of. The following sections describe the concept of ownership and the IDF rules to use to ensure proper communication between isolated regions.

Concepts - Ownership

Logical ownership relates to HDL partition (left side of Figure 14):

- Each HDL file logically owns the logic instantiated within it.

- Only global logic (BUFG, MMCM, PLL, and so on) is allowed at the top logical level of design.

- Only global logic can be at the top, but it can also be instantiated in an isolated hierarchy as long as the HD.Isolated_Exempt property is used.

Physical ownership relates to Isolated Regions/Pblocks (right side of Figure 14):

- No user logic component can be used if it is not physically owned by an isolated region.

  ◦ Physical ownership is defined by addition of the available components to the Pblock/Isolated Region definition in the XDC file.

  ◦ For example, the resize_pblock constraint allows a range of slices placed in the ISO_Controller Pblock to be used by the developer:

    - `resize_pblock [get_pblocks pblock_ISO_Controller] -add {SLICE_X0Y49:SLICE_X113Y149 SLICE_X32Y40:SLICE_X81Y48}`

- Global logic, while not typically owned by any isolated logic, must be associated with some (any) isolated region, with the exception of BUFGs and BUFHs. If the global logic is part of an isolated function and not a BUFG or BUFH, it must be associated with the isolated region of that isolated function.

Send Feedback

www.xilinx.com

Local Ownership
(User HDL)

Physical Ownership
(User Floorplan)
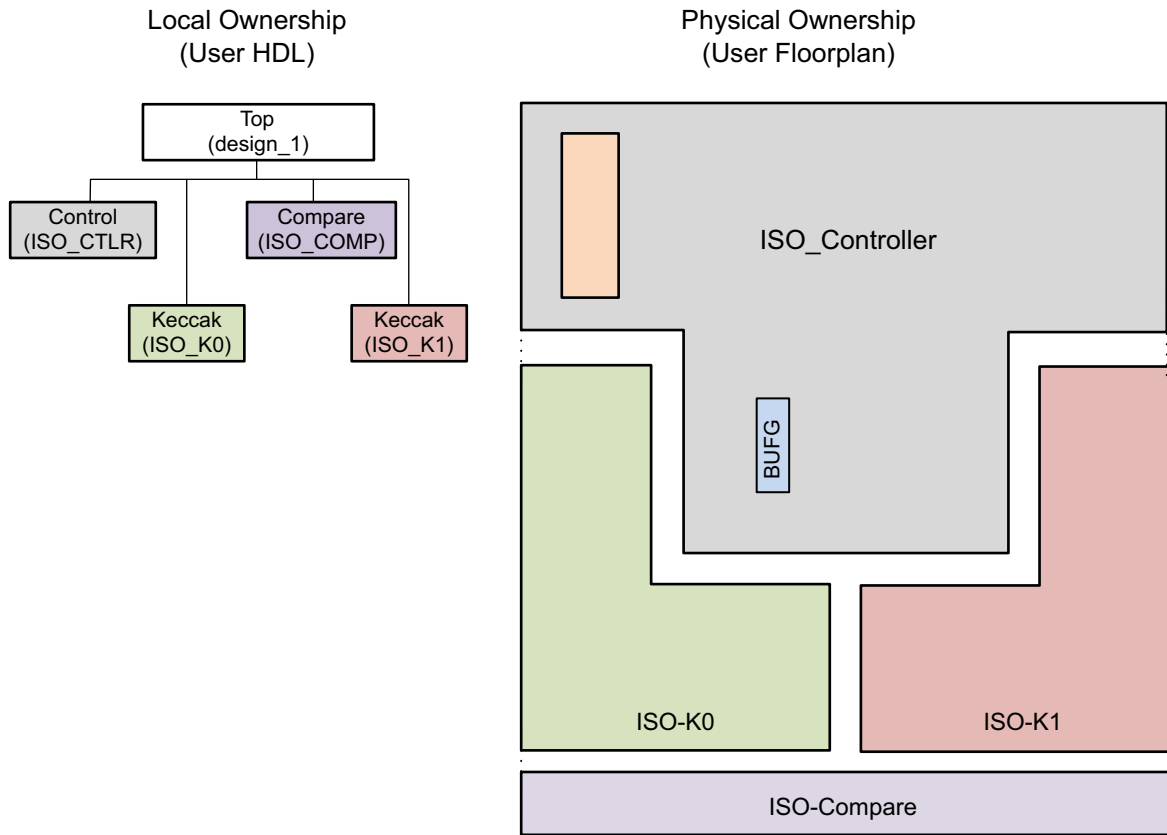
X17826-090616

Figure 14: **Side by Side View of Ownership Concepts**

See Guidelines for Controlling Global Clocking Logic for how to prevent Vivado tools from attempting to isolate the global logic.

Trusted Routing Rules

Trusted routing is automatic when `HD.ISOLATED true` is set in the design. The design tools recognize the communication between isolated regions and use the trusted routing resources. However, some rules must be adhered to if safe communication between isolated functions is to be guaranteed.

Note: The Vivado tools now take into account and adjust for each of the following rules; the ISE tools do not. It is recommended as a good design practice to understand these IDF rules and take the precautions suggested to avoid violating them, even though Vivado tools take the necessary steps to ensure they are followed. These changes show in the netlist after synthesis.

If the developer wishes, they can manually modify the HDL to meet the IDF rules and manually split the nets and add LUTs as described below. Additionally, you can disable this automatic modification by setting the parameter HD.ISOLATED_DISABLE_NETSPLIT to 1 using the following syntax:

```
set_parameter HD.ISOLATED_DISABLE_NETSPLIT 1
```

Rule 1: Feed-through signals are not allowed without buffering of some kind (LUT, FF for example):

- If a signal is directly connected to both an input port and an output port, it must be buffered.

- Direct instantiation of a buffer (LUT1, for example) is recommended. This isolates the wire segments in each of the isolated regions with the LUT buffer, preventing a common (shorted) net throughout both regions.

Problem: The statement `reset_out <= reset` violates IDF rules by creating a short between the isolated regions for this example (Figure 15).
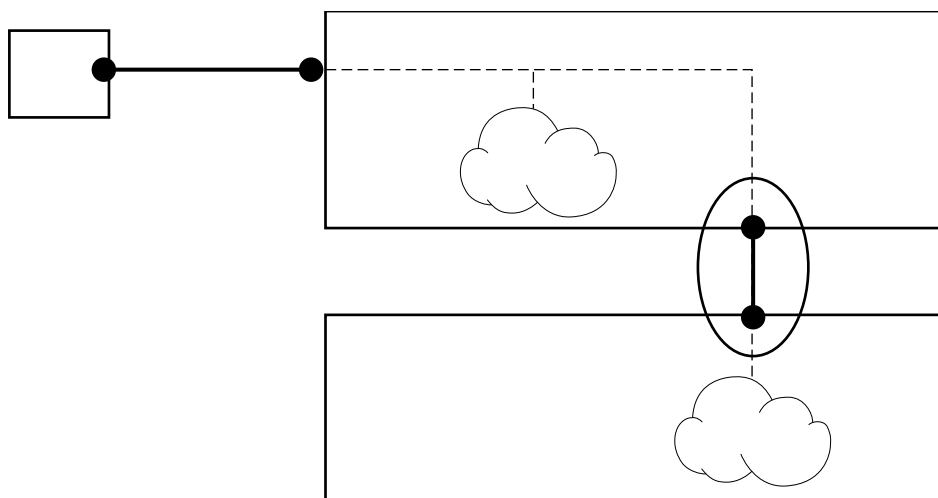

X17827-090616

Figure 15: **Short Created by Feed-Trough Signal**

Send Feedback

Solution: Feed-through signals need to be buffered. This can be achieved either through HDL coding to ensure that there is some unique driver on the output port, by direct instantiation of a LUT or flip-flop buffer (Figure 16), or by letting the Vivado tools address the issue using separate wire segments. The problematic code can be fixed as follows:

```
lut_reset_out: LUT1

GENERIC MAP (INIT => X"2")

PORT MAP (I0 => reset, O => reset_out);
```
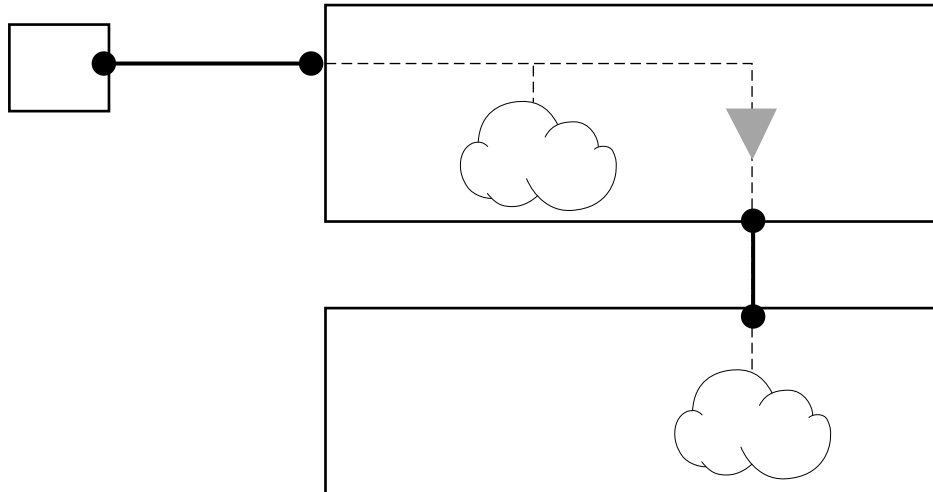

X17828-090616

Figure 16: **Elimination of a Short by Instantiation of a LUT Buffer**

Rule 2: A function output port cannot connect to more than one function input port. Stated differently, port-to-port connections must be singular:

- You must create two different ports for such a connection.
- Each port must not violate **Rule 3**.

Send Feedback

Problem: The segment of VHDL code that creates a common connection between unintended isolated functions is shown in Figure 17, for this example.
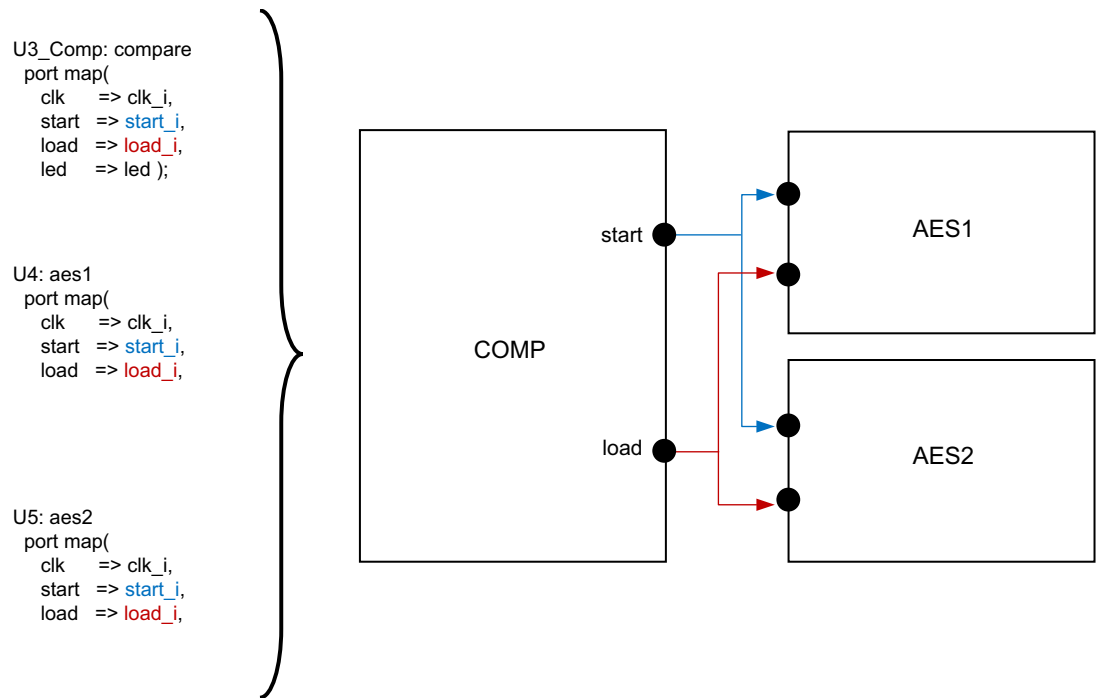

Figure 17: **Multi-Port Connection Causing Connectivity between AES1 and AES2**

Send Feedback

Solution: You must create multiple output ports if required to drive multiple input ports of other isolated functions (Figure 18), or let the Vivado tools split the offending ports. If you choose to let Vivado do this for you, the HDL does not need to be modified to achieve this. The tools split the ports at the netlist level.
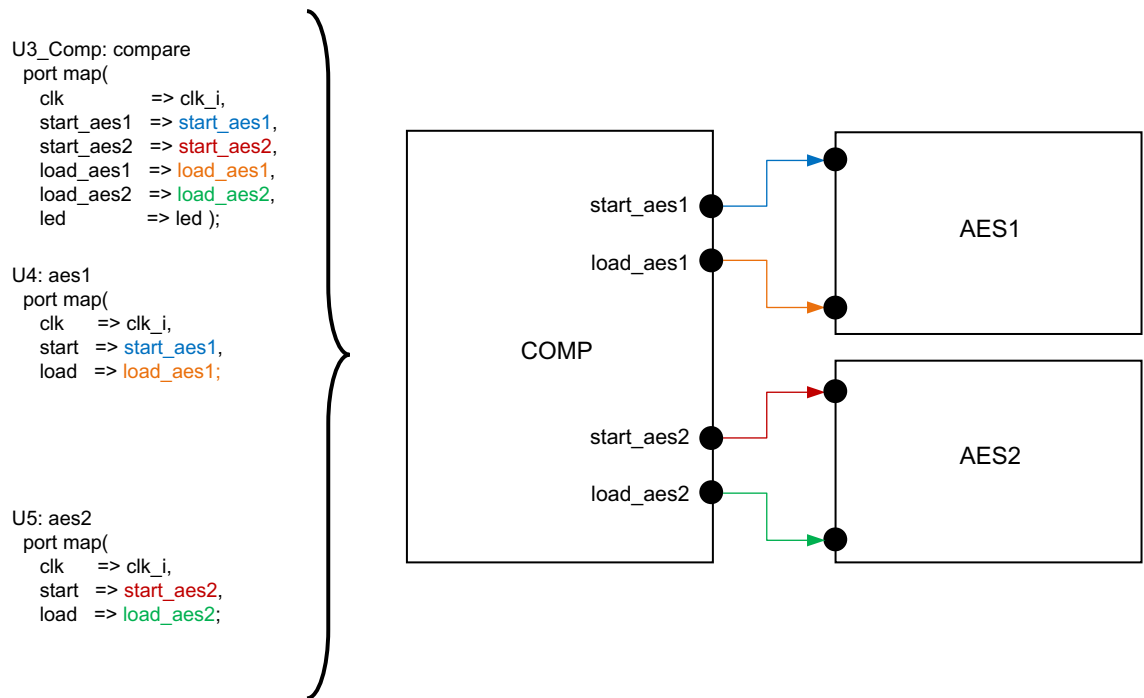

Figure 18: **Elimination of Unintended Connection Using Multiple Output Ports**

Rule 3: One signal cannot drive two different output ports of the same function:

- Each port must have a unique driver.

- Direct instantiation of a buffer (LUT1, for example) is necessary. This isolates regions, with the LUT preventing a shorted net.

Problem: These example statements create unintended connectivity between two isolated functions (Figure 19):

```
start_aes1 <= start_i;

start_aes2 <= start_i;
```
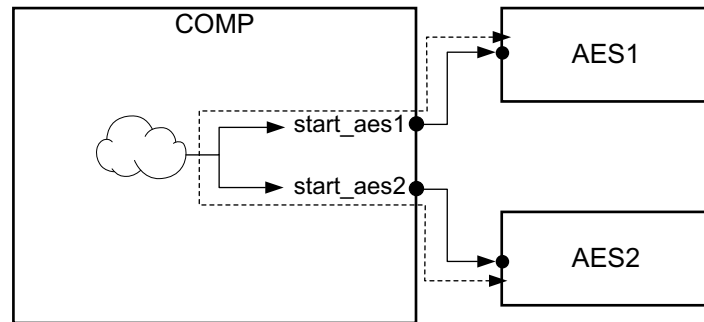

X17831-090616

Figure 19: **Unintended Connectivity between AES1 and AES2 inside COMP**

Solution: Each port driver needs to be buffered. This can be achieved through HDL coding to ensure that there is some unique driver for each output port, by direct instantiation of a LUT buffer or flip-flip (Figure 20), or by letting the Vivado tools address the issue using separate wire segments. The problematic code can be fixed as follows:

```
lut_start_out: LUT1

GENERIC MAP (INIT => X"2")

PORT MAP (I0 => start_i, O => start_aes1);


lut_start_out: LUT1

GENERIC MAP (INIT => X"2")

PORT MAP (I0 => start_i, O => start_aes2);
```
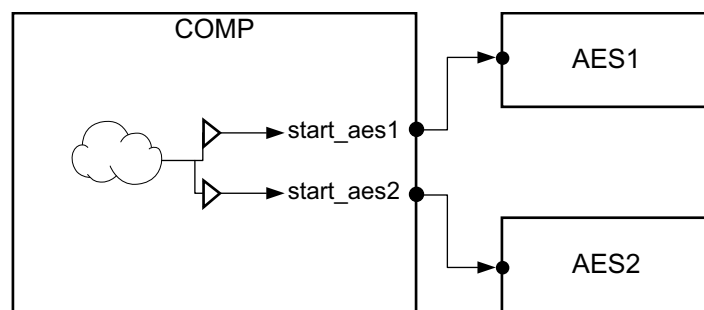

X17832-090616

Figure 20: **Elimination of Unintended Connectivity Using LUT Buffers**

7 Series FPGA Fencing Rules

To achieve isolation between floorplanned regions, a fence (isolated region) is necessary to separate them. This fence is not specifically drawn so much as it is what is left over between drawn regions. This empty space must meet certain criteria to be considered a fence:

• The fence tile must be empty. While the tools enforce this, manual edits can be made to violate this rule. Such edits cause a reduction in fault immunity and are identified by IVT as a failure.

• The fence tile must be a valid one as listed in the table of user tiles with the appropriate number of tiles (Table 3). Not all fence tiles are available in all FPGA and SoC devices.

Table 3: **Xilinx 7 Series FPGA and Zynq-7000 SoC Fence Tile Requirements**

User Tile	User Tile Description	Fence Tile?
CLB (LL and LM)	**Configurable Logic Block**. The key logic unit of an FPGA.	Yes Vertical: 1 Horizontal: 1
BRAM	**Block RAM**. User-accessible high speed RAM (BRAM tile is RAMB36 which is two RAMB18 blocks; Vivado tools only allow selection of a BRAM tile).	Yes Vertical: 1 Horizontal: 1
DSP	**Digital Signal Processor**. A programmable math function (DSP tile is two DSP48E1 slices; analysis was done on DSP48E1 slice; Vivado tools only allow selection of a DSP tile).	Yes Vertical: 1 Horizontal: 1
PCIe® Gen2	PCI Express Gen2 Endpoint block.	Yes Vertical: 1 Horizontal: 1
PCIe® Gen3	PCI Express Gen3 Endpoint block.	Yes Vertical: 1 Horizontal: 1
GTX transceiver	**High speed transceiver** for Kintex-7 FPGAs (up to 12.5 Gb/s). The GTX Quad tile is made up of four GTXE2_CHANNELs and one GTXE2_COMMON block.	Yes Vertical: 1 GTX **Quad** Horizontal: 1 GTX **Channel**
GTH transceiver	**High speed transceiver** for Virtex-7 FPGAs (up to 13.1 Gb/s). The GTH Quad tile is made up of four GTHE2_CHANNELs and one GTHE2_COMMON block.	Yes Vertical: 1 GTH **Quad** Horizontal: 1 GTH **Channel**
GTP transceiver	**High speed transceiver** for Artix-7 FPGAs (up to 6.6 Gb/s). The GTP Quad tile is made up of four GTPE2_CHANNELs and one GTPE2_COMMON block.	Yes Vertical: 1 GTP **Quad** Horizontal: 1 GTP **Channel**
IOB/IOI (Master/Slave pair)	**IOB** = Input receivers with output drivers (in master/slave pairs). **IOI** = All other IOB logic such as capture and send flip-flops (I/O tile pair: all I/O except at top and bottom of each I/O bank).	Yes Vertical: 1 Horizontal: 1

Send Feedback

*Table 3:* **Xilinx 7 Series FPGA and Zynq-7000 SoC Fence Tile Requirements** *(Cont'd)*

User Tile	User Tile Description	Fence Tile?
IOB/IOI (Single)	**IOB** = Input receivers with output drivers. **IOI** = All other IOB logic such as capture and send flip-flops (I/O tile single at top and bottom of each I/O bank).	Yes Vertical: 1 Horizontal: 1
IO_FIFO (CMT)	**I/O FIFO.** Fence tiles for the CMT clocking components. The CMT column is adjacent to the IOI/IOB column, so a vertical fence does not make sense for the 7 series CMT.	Partial Vertical: Not allowed Horizontal: 1
PS (Zynq-7000 device only)	**Processor System (PS).** The PS block is located in the upper left corner of the Zynq-7000 SoC, so being part of a vertical or horizontal fence does not make sense.	Vertical: Not allowed Horizontal: Not allowed

Fence Tile Usage Overview

All of the user tiles specified in Table 3 are shown in the following figures for fence tile usage. Figure 21, Figure 22, and Figure 23 show high-level Vivado tool views for all of the user tiles in an example Kintex-7 K325T or Zynq 7Z020 device, and Figure 24 through Figure 36 show valid fence tile usage for each user tile type in both horizontal and vertical fences.

The CMT user tile is the most complicated tile and set of components in the 7 series FPGA and Zynq-7000 SoC architectures to use as a user tile and a fence tile. This is in part due to the complexity of efficiently supplying high-speed clocking from the phase-locked loops (PLLs) and mixed-mode clock managers (MMCMs) to the whole FPGA and in part for the new integrated I/O FIFOs in the 7 series FPGA to aid in creating high-speed memory interfaces. This tile is covered in the last fence tile section, CMT User Tile Fence Usage.

Send Feedback

EX XILINX®

Figure 21 shows the Vivado tool high-level view for the following user tiles: CLB, BRAM, DSP, I/O pair, I/O Single, and IO_FIFO (for the CMT components). As stated earlier, the CMT and clocking components are discussed in a later section. One CMT spans a complete I/O bank in a 7 series FPGA and in a Zynq-7000 SoC.
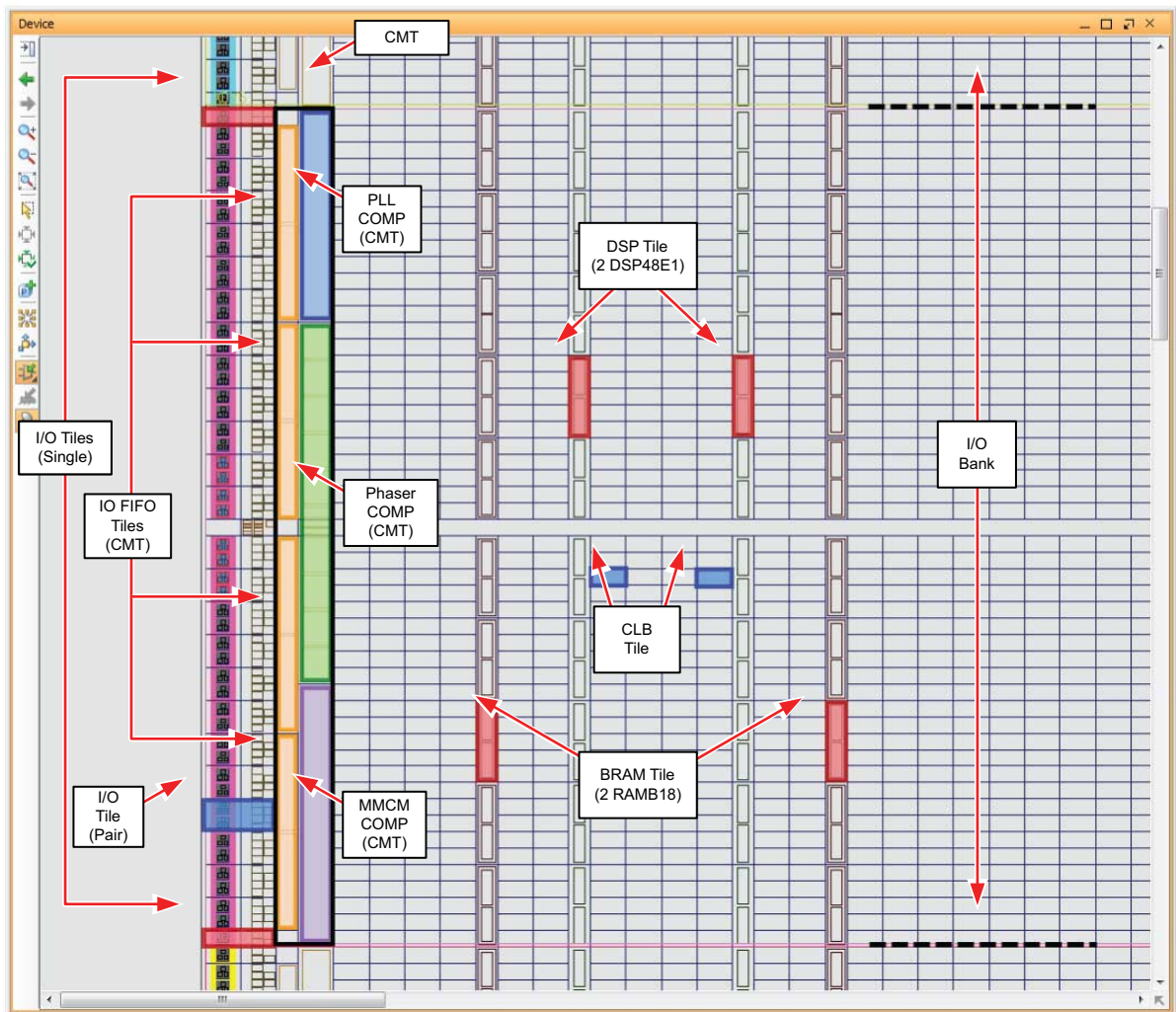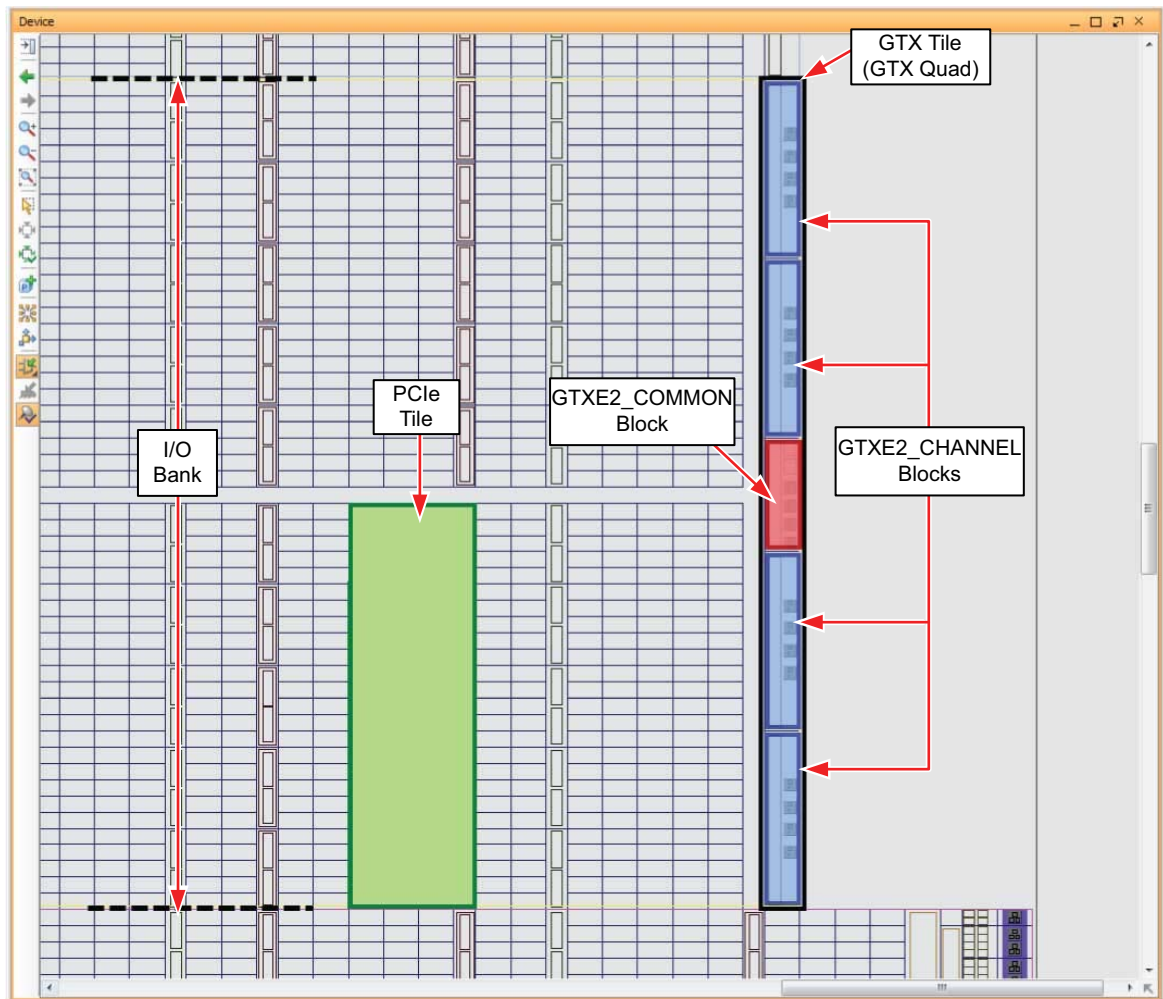

Figure 21: **High-Level Vivado Tool View for CLB, BRAM, DSP, I/O, and CMT User Tiles**

Figure 22 shows the Vivado tool view for the following user tiles in Kintex-7: PCIe and GTX Quad.

The GTX Quad tile is composed of four GTXE2_CHANNEL blocks and one GTXE2_COMMON block. The GTX Quad tile is the equivalent of fifty CLB tiles high or one complete I/O bank. The GTH and GTP Quad tiles are similar.

In some Xilinx 7 series and Zynq-7000 SoC devices, the GT tiles are on the extreme edge of the device. In some devices the GT tiles are in the middle of the device, so it might not always make sense to be part of a vertical fence.

X17834-090616

Figure 22:   **High-Level Vivado Tool View for PCIe Tile and GTX Quad Tile**

Send Feedback

Figure 23 shows the Vivado tool high-level view for the PS block. One PS block spans two complete I/O banks in a Zynq-7000 SoC device.
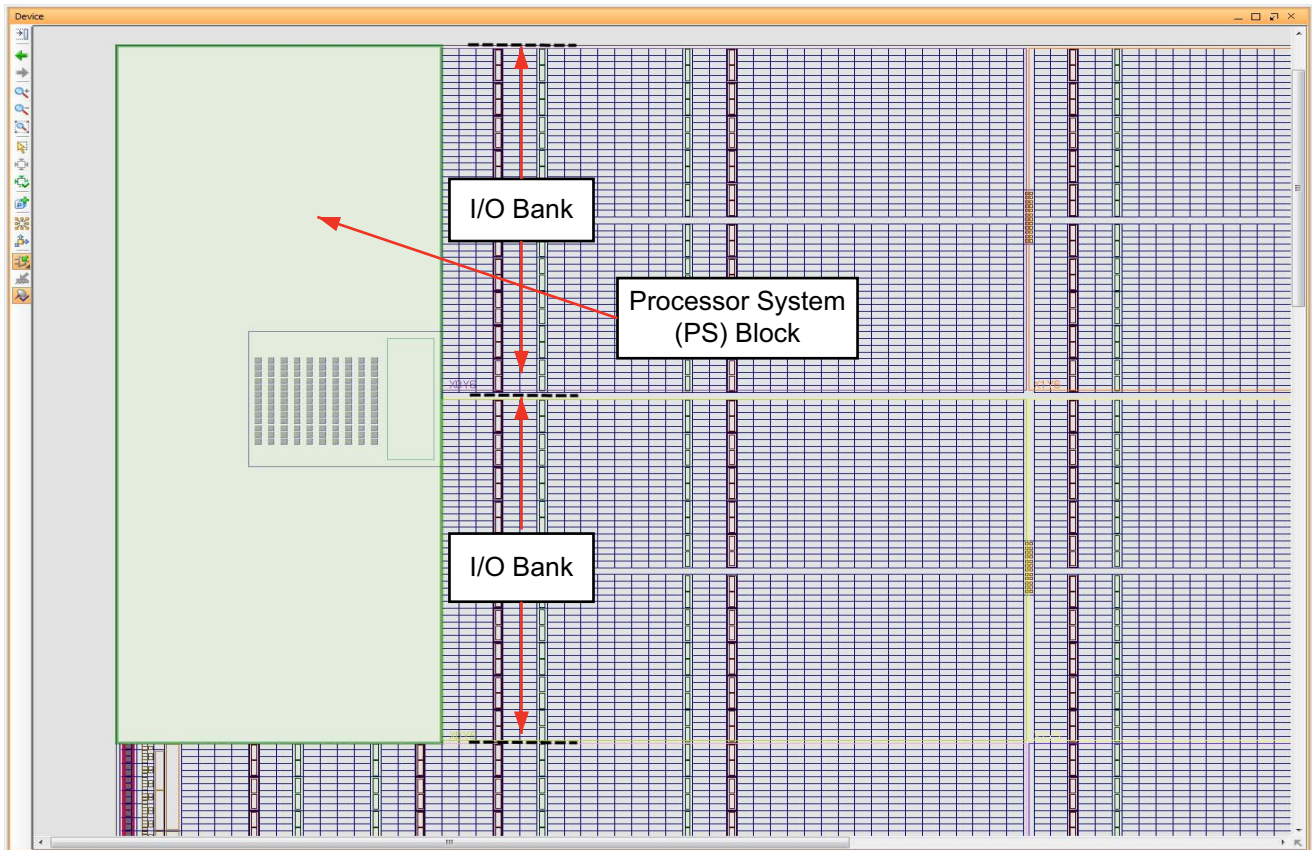
X17835-090616

Figure 23:   **High-Level Vivado Tool View for Zynq-7000 Processor System (PS) Block**
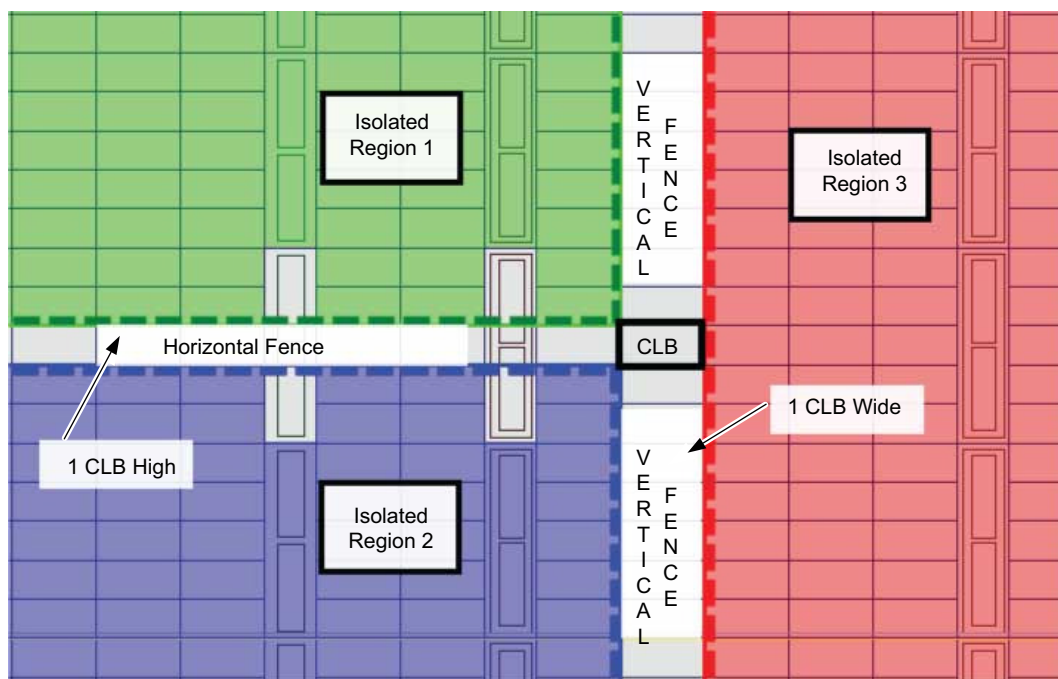
CLB Fence Tile Usage

The CLB (LM and LL) tile is one of the most basic and straightforward tiles to use as a fence tile. It can be used as a minimum fence of one CLB tile in both a horizontal and vertical fence. Figure 24 shows the CLB being used as a fence tile in a vertical and a horizontal fence.

Figure 24 shows the horizontal fence (white area) isolates Isolated Region 1 from Isolated Region 2. The vertical fence (white area) isolates Isolated Region 1 from Isolated Region 3 and Isolated Region 2 from Isolated Region 3. The horizontal fence is one CLB high and the vertical fence is one CLB wide.

Note: The CLB tile encompasses two slices, SLICE_L and SLICE_M for a CLB_LM, and two SLICE_Ls for a CLB_LL. For purposes of a fence tile, the CLB cannot be split at a slice boundary, but instead the complete CLB must be selected, as shown in Figure 24.
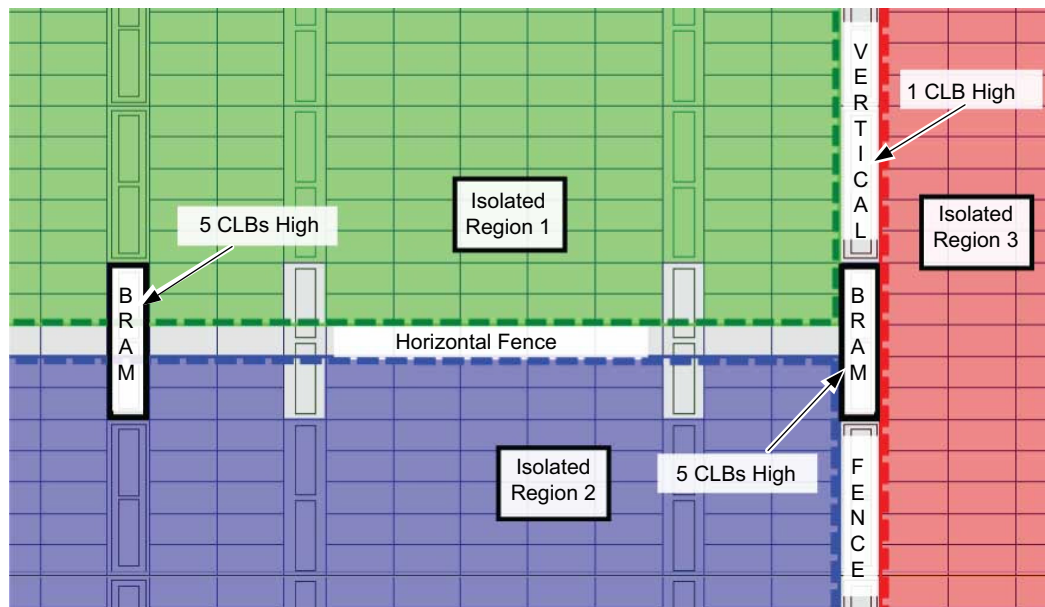

X17836-090616

Figure 24: **Vivado Tool View for Horizontal and Vertical Isolation Fences Using the CLB Tile**

Send Feedback

BRAM Fence Tile Usage

The BRAM tile (RAMB36) is the equivalent of five CLB tiles high and one CLB tile wide. It can be used as a minimum fence of one BRAM tile in both a horizontal and vertical fence. Figure 25 shows the BRAM on the right being used as a fence tile in a vertical and a horizontal fence.

Figure 25 shows the horizontal fence (white area) isolates Isolated Region 1 from Isolated Region 2. The vertical fence (white area) isolates Isolated Region 1 from Isolated Region 3 and Isolated Region 2 from Isolated Region 3. The horizontal fence is one CLB tile high and the vertical fence is one CLB tile wide, except at the BRAM tile. The BRAM on the far left in Figure 25 is part of the horizontal fence. Even though the height of the fence (distance between isolated regions) is one equivalent CLB tile, the whole BRAM tile is part of the fence and not used. If the whole BRAM tile cannot be used in an isolated region, none of it is used and it is part of the horizontal fence.

Note: The BRAM tile encompasses two RAMB18 blocks, but for purposes of a fence tile, the BRAM tile (RAMB36) cannot be split at the RAMB18 boundary. Instead the whole BRAM tile must be selected, as shown in Figure 25.

X17837-090616

Figure 25:   **Vivado Tool View for Horizontal and Vertical Isolation Fences Using the BRAM Tile**

Send Feedback

DSP Fence Tile Usage

The DSP tile (2 DSP48E1 slices) is the equivalent of five CLB tiles high and one CLB tile wide. It can be used as a minimum fence of one DSP tile in both a horizontal and vertical fence. Figure 26 shows the DSP on the right being used as a fence tile in a vertical and a horizontal fence.

Figure 26 shows the horizontal fence (white area) isolates Isolated Region 1 from Isolated Region 2. The vertical fence (white area) isolates Isolated Region 1 from Isolated Region 3 and Isolated Region 2 from Isolated Region 3. The horizontal fence is one CLB tile high and the vertical fence is one CLB tile wide, except at the DSP tile. The DSP on the left in Figure 26 is part of the horizontal fence. Even though the height of the fence (distance between isolated regions) is one equivalent CLB tile, the whole DSP tile is part of the fence and not used. If the whole DSP tile cannot be used in an isolated region, none of it is used and it is part of the horizontal fence.

Note: The DSP tile encompasses two DSP48E1 blocks, but for purposes of a fence tile, the DSP tile cannot be split at the DSP48E1 boundary. Instead the whole DSP tile must be selected, as shown in Figure 26.
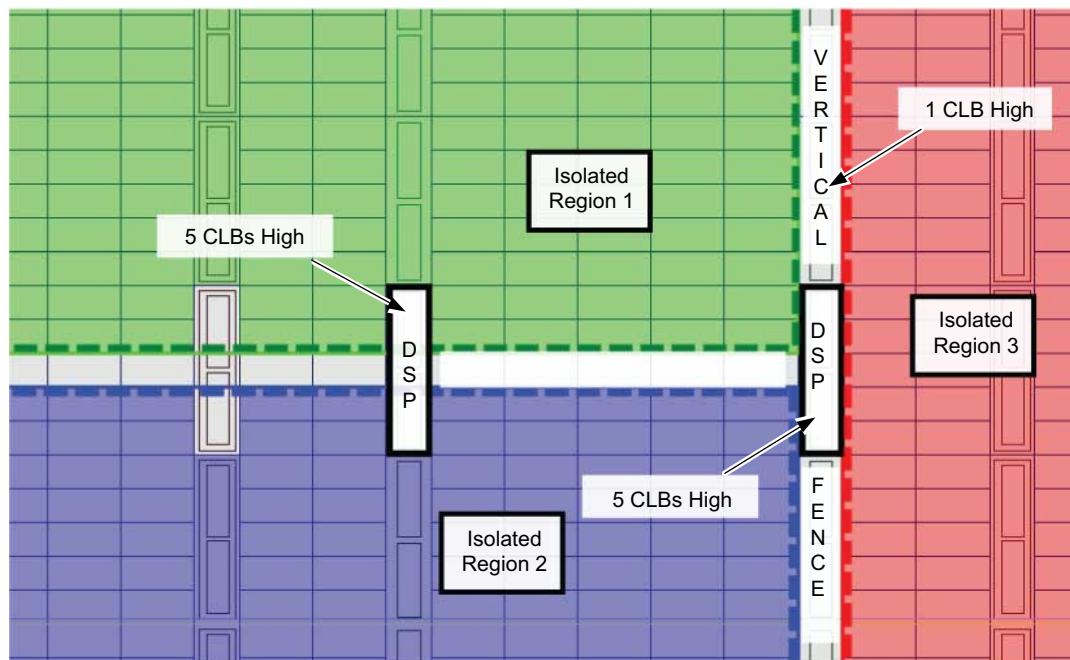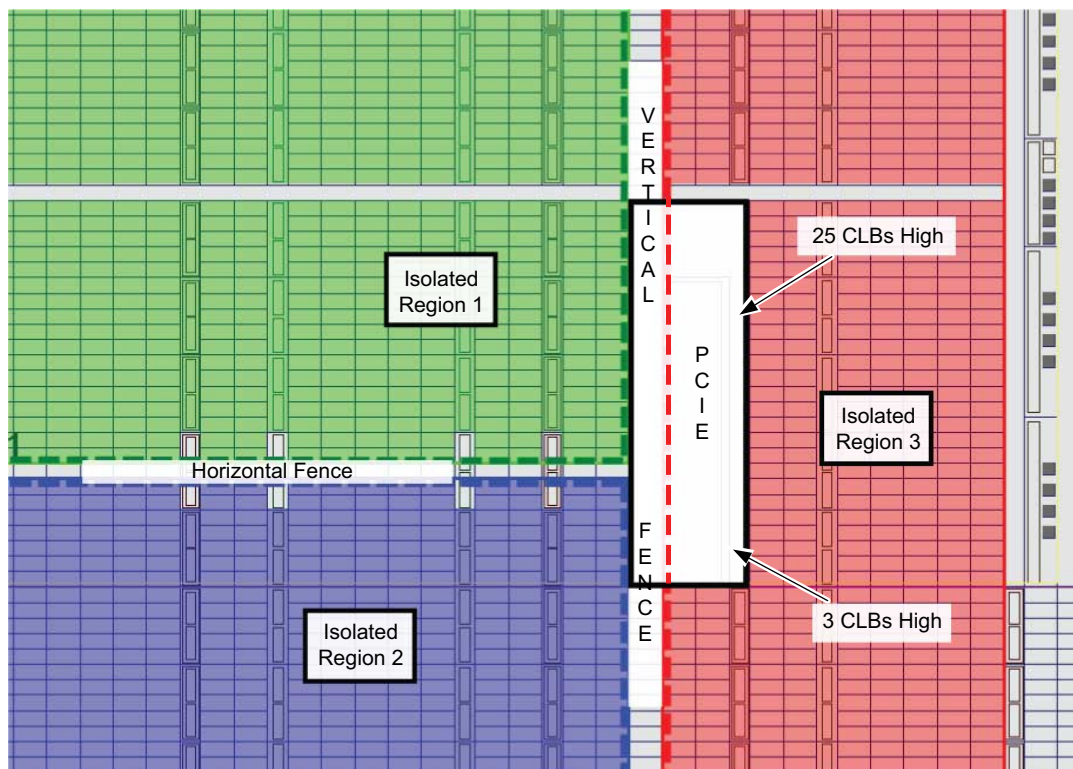

X17838-090616

Figure 26: **Vivado Tool View for Horizontal and Vertical Isolation Fences Using the DSP Tile**

Send Feedback

PCIe Gen2 and Gen3 Fence Tile Usage

The PCIe Gen2 and Gen3 tile is the equivalent of 25 CLB tiles high and the equivalent of four CLB tiles wide (three CLBs and one BRAM tile). It can be used as a minimum fence of one PCIe Gen2 or Gen3 tile in both a horizontal and vertical fence. Figure 27 shows the PCIe Gen2 being used as a fence tile in a vertical and a horizontal fence.

Figure 27 shows the horizontal fence (white area) isolates Isolated Region 1 from Isolated Region 2. The vertical fence (white area) isolates Isolated Region 1 from Isolated Region 3 and Isolated Region 2 from Isolated Region 3. The horizontal fence is one CLB tile high and the vertical fence is one CLB tile wide, except at the PCIe Gen2 tile.


X17839-090616

Figure 27: **Vivado Tool View for Horizontal and Vertical Isolation Fences Using the PCIe Gen2 Tile**

GT Fence Tile Usage

The following example from Kintex-7 shows the GTX tile, but the GTH and GTP tiles are similar. There is not sufficient isolation at the GTXE2_COMMON block level so it cannot be used as a horizontal fence. A minimum of one GTXE2_CHANNEL block can be a valid horizontal fence tile on its own, as shown in Figure 28. Additionally, only one isolation region can occupy a GTX Quad tile.

In some Xilinx 7 series and Zynq-7000 SoC devices, the GT tiles are on the extreme edge of the device and it does not make sense to be used as part of a vertical fence. However, in some devices where the GT tile is in the middle of the device, the GT Quad can be used as a vertical fence.
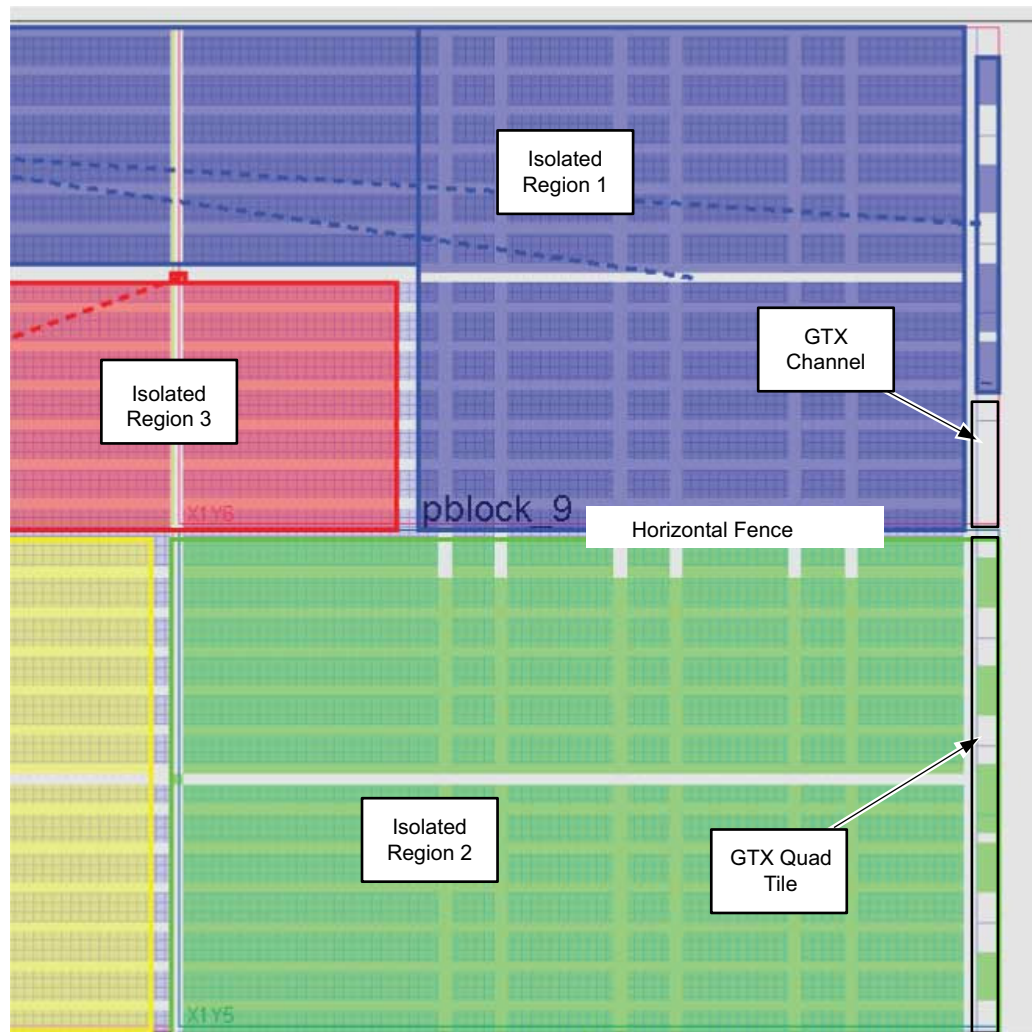
X17840-090616

Figure 28: **Vivado Tool View for Horizontal Fencing Using the GTX Tile**

Send Feedback

Where a GTX Quad abuts logic tiles (for example, BRAM, CLB, and I/O), the fence could either be a GTX channel block or the logic tiles, per the appropriate logic tile fence rules. Figure 29 shows an example of using the GTX channel block as a fence on the left side of the figure and using the logic tiles as a fence on the right side of the figure.
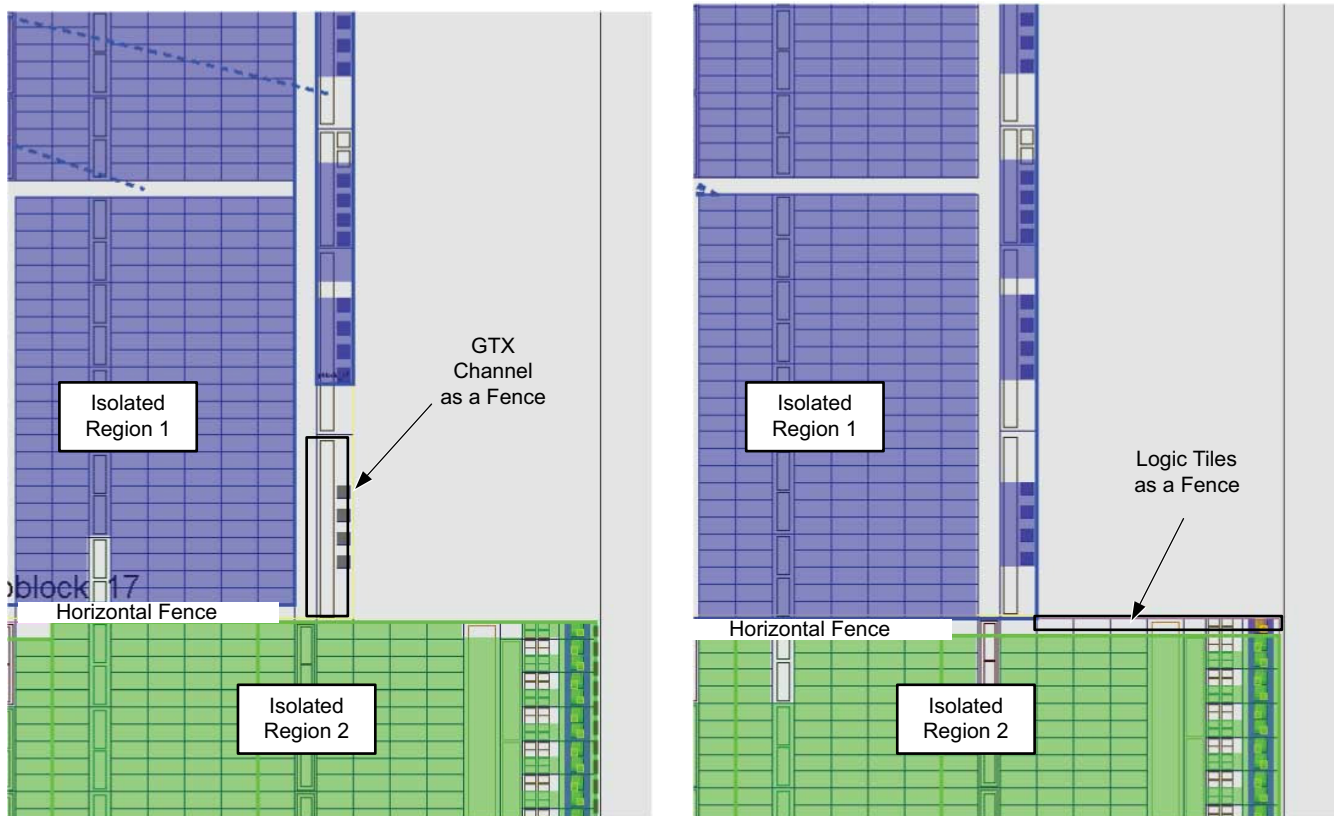

Figure 29: **Example of GTX Channel Block and Logic Tiles as Fences**

Processor System Block Fence Tile Usage

The following basic processor system (PS) block fence tile usage example is for when the PS block is in its own isolated region, with no other logic included from the programmable logic (PL). The PS block does not provide the required isolation between itself and a different isolation region directly to the right of the PS block. Therefore, a column of valid vertical fence tiles is required, as shown in Figure 30.

The PS block analysis also concludes that although the interface between the bottom of the PS block and the PL does have adequate isolation for most of the user tiles, the CLB_ll tile below the far right corner of the PS block does not provide the required isolation on its own. Therefore, the CLB_ll tile is required as a fence tile between the PL isolated region and the PS block (see Figure 30). Also, a minimum of one valid user fence tile is needed as part of a horizontal fence on both sides of the CLB_ll tile in order to provide the required diagonal isolation between isolation regions (PS block and the user tiles in the PL isolated region below). See Figure 30, which shows separate isolation regions with the PS block (green region), another isolation region in the PL (blue region), and the three valid fence tiles (white tiles) required for proper horizontal fence isolation.
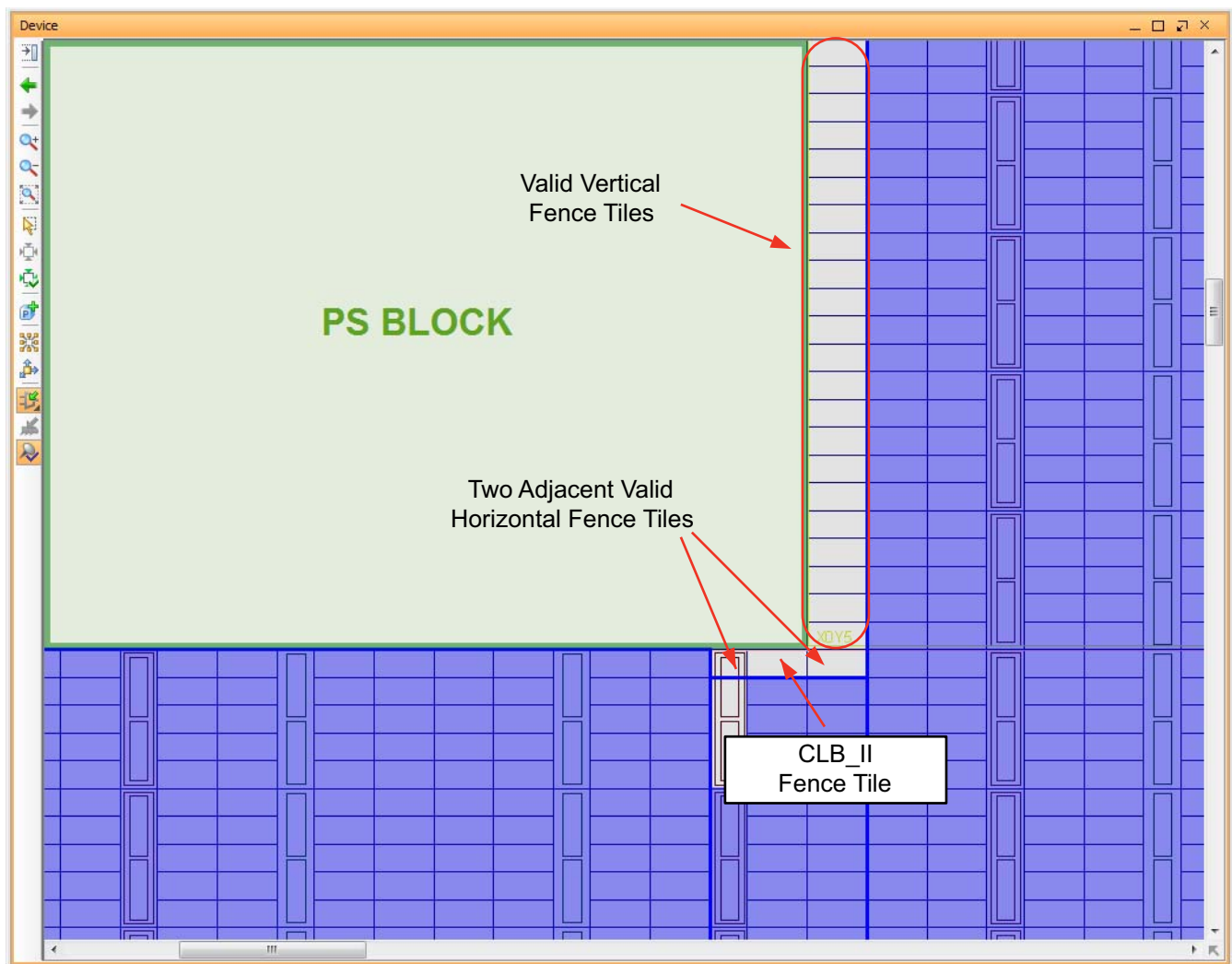
X17842-090616

Figure 30: **Vivado Tool View for Basic Fence Tile Usage Around the PS Block**

Send Feedback

Figure 31 shows a more realistic fence tile usage example where the PS block is in an isolated region with other logic from the PL (pblock_9). In this example, the same minimum three valid horizontal fence tiles are required for proper isolation of the PS block, near the lower right side of the PS block, as in the previous example. Figure 31 also shows the standard required vertical fence between the two isolated regions.
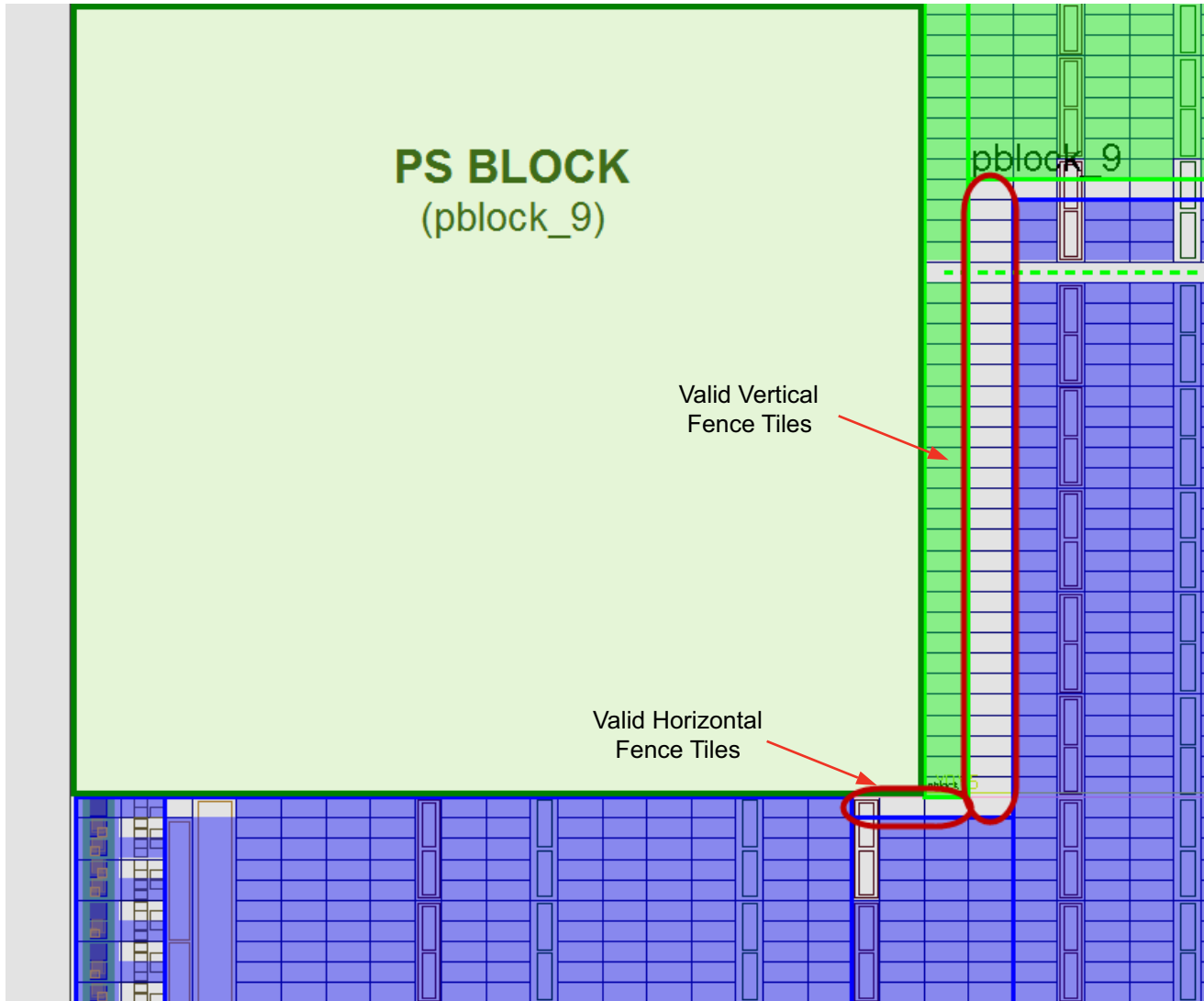

Figure 31: **Vivado Tool View for More Realistic Fence Tile Usage Around the PS Block**

Send Feedback

I/O Fence Tile Usage

With some exceptions, each I/O bank in 7 series FPGAs and Zynq-7000 SoCs contains 50 SelectIO™ interface pins. The two pins at the very ends of each I/O bank can only be used with single-ended I/O standards. The remaining 48 pins can be used with either single-ended or differential standards using two SelectIO interface pins grouped together as positive/negative (P/N) pairs or Master/Slave pairs. Figure 32, is a high-level block diagram showing the I/O single (single-ended only) versus I/O pair (differential) IOB/IOI structures for one 7 series I/O bank. The I/O tile is composed of the IOB block and one IOI block.
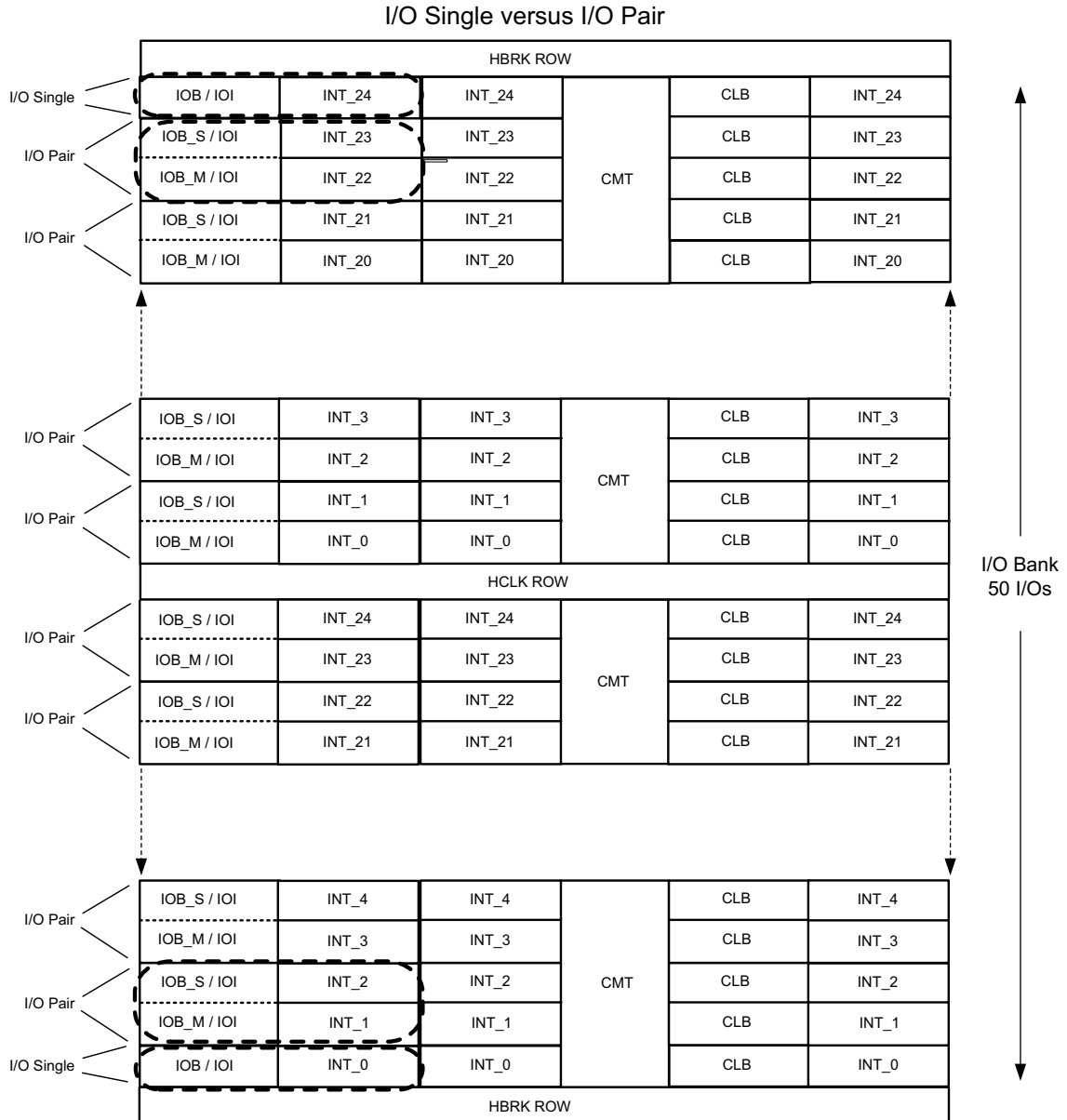
Figure 32: **7 Series and Zynq-7000 Devices—I/O Single versus I/O Pair Tiles**

An I/O single tile is the equivalent of one CLB tile high. It can be used as minimum fence of one I/O single tile in a horizontal fence. In most Xilinx 7 series and Zynq-7000 SoC devices, the I/O single tile is on the extreme edge of the device and it does not make sense for it to be used as part of a vertical fence. However, in some devices where the I/O single tile is in the middle of the device, the I/O single tile can be used as part of a vertical fence. Figure 33 shows the I/O Single tile being used as a fence tile in a horizontal fence.
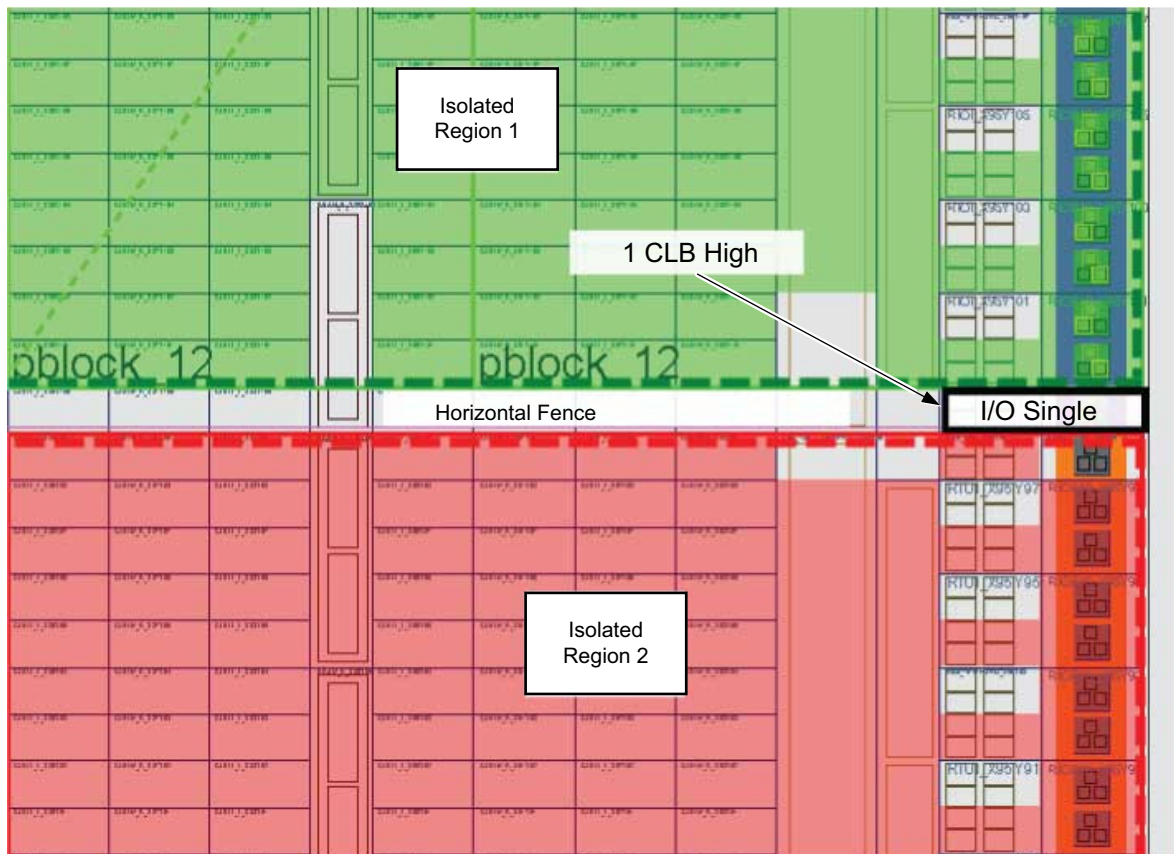

X17845-090616

Figure 33:   **Vivado Tool View for Horizontal Isolation Fence Using the I/O Single Tile**

An I/O pair tile has to be used as a horizontal fence tile as a pair of I/Os because the I/O pair has Master/Slave logic that is common to both I/Os in the pair. An I/O pair tile is the equivalent of two CLB tiles high. It can be used as minimum fence of one I/O pair tile in a horizontal fence. An I/O pair tile cannot be used as a vertical fence because it is on the edge of the device. Figure 34 shows the I/O pair tile being used as a fence tile in a horizontal fence.

CMT User Tile Fence Usage
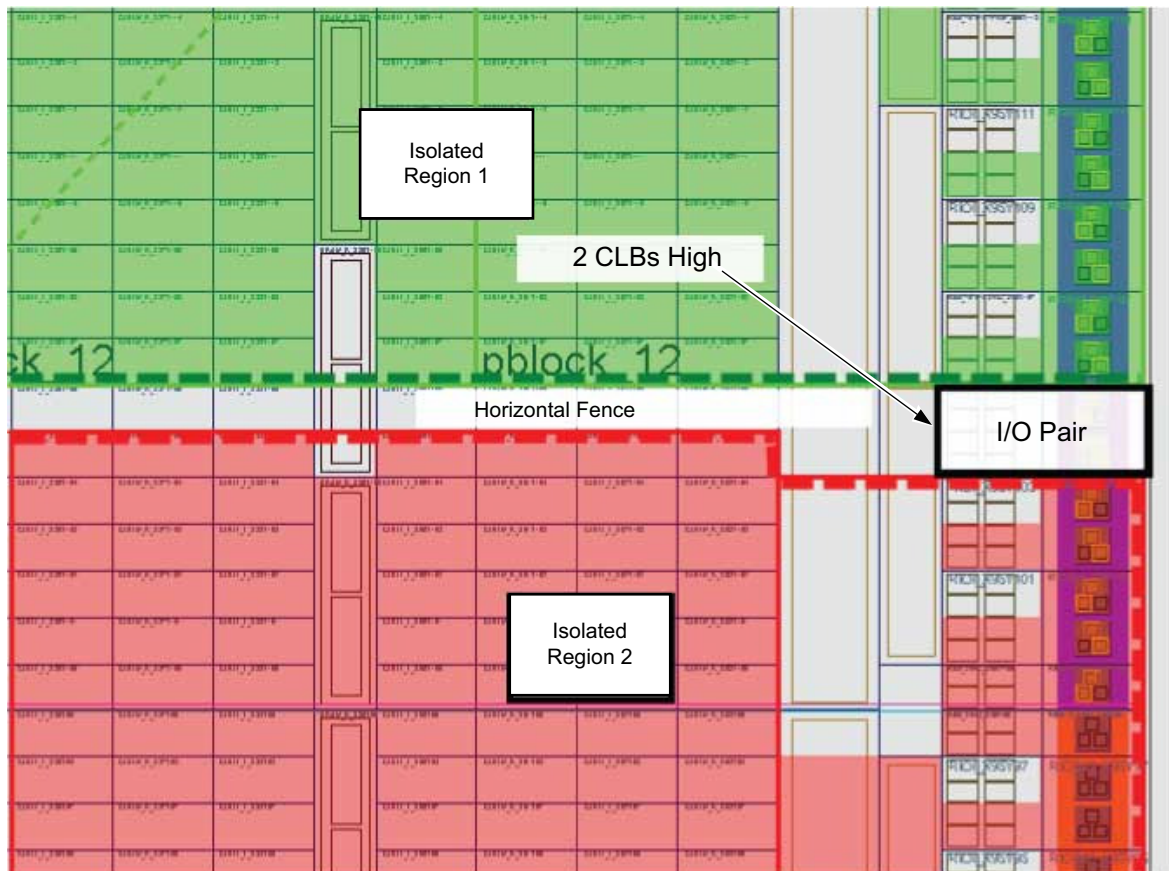
X17846-090616

Figure 34: **Vivado Tool View for Horizontal Isolation Fence Using the I/O Pair Tile**

Each 7 series FPGA has up to 24 clock management tiles (CMTs), each consisting of one MMCM and one PLL. The 7 series CMT also includes a Phaser component. The Phaser component is designed to be used with the PLL component in high-speed memory interfaces. The PLL provides the clocking references to the Phaser component which provides the high-speed interface clocking and control signals, so the Phaser component cannot be used without the adjacent PLL component.

The I/O-specific FIFOs (IO_FIFOs) work in conjunction with the Phaser and PLL components for use in high-speed interfaces. Each IO_FIFO tile has an IN_FIFO and an OUT_FIFO. The Xilinx Vivado tool shows an independent IN_FIFO and OUT_FIFO as separate sites within the CMT_FIFO_R/L tile. However, they cannot be separated into different isolated regions or be split up as part of a fence.

To understand the fencing rules for the CMT clocking components, the concept of adjacent IO_FIFOs must first be understood, because the fencing rules are based on the IO_FIFOs. The concept of an adjacent IO_FIFO is an IO_FIFO that shares part or all of a boundary directly with a CMT clocking component. The coincident border between the IO_FIFO/s and the three different CMT clocking components is shown in the three drawings in Figure 35.

Send Feedback

The phaser and the MMCM components share a common IO_FIFO. Adjacent IO_FIFOs for each of the CMT clocking components are:

* PLL component: One adjacent IO_FIFO

* Phaser component: Two adjacent IO_FIFOs

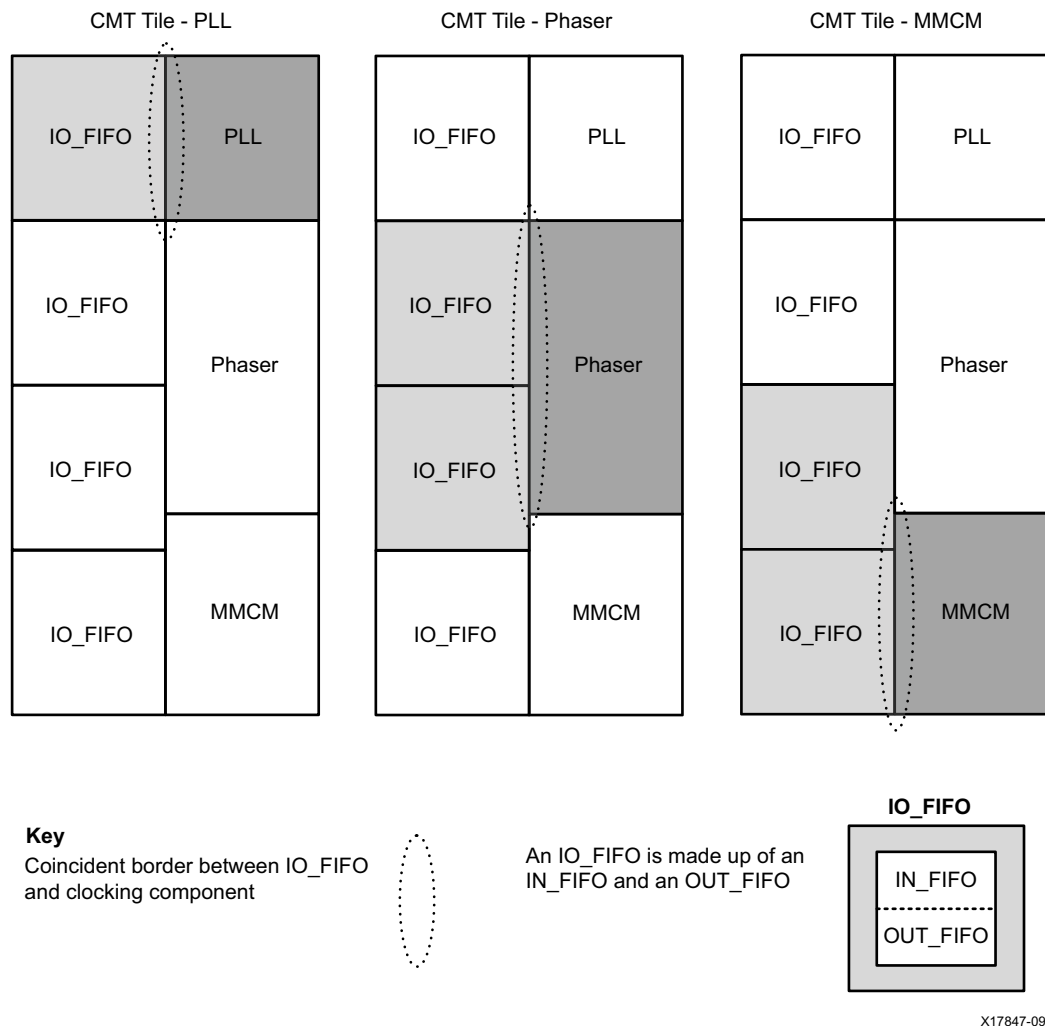* MMCM component: Two adjacent IO_FIFOs

Adjacent IO_FIFOs in the CMT Tiles

Figure 35: **CMT Clocking Components and Adjacent IO_FIFOs**

Notes relevant to Figure 35:

Each drawing shows the adjacent IO_FIFOs for each CMT clocking component type. The phaser and the MMCM share an IO_FIFO. Adjacent IO_FIFO for each of the clocking components are:

◦ PLL has one adjacent IO_FIFO

◦ Phaser has two adjacent IO_FIFOs

Send Feedback

◦ MMCM has two adjacent IO_FIFOs

Based on the adjacent IO_FIFO concept, the basic rule for an IO_FIFO fence is that the minimum number of IO_FIFOs used as an isolation fence is one. However, for a used or instantiated CMT clocking component (MMCM, PLL, Phaser), the IO_FIFOs included must be all of the adjacent IO_FIFOs for that CMT clocking component. In other words, none of the adjacent IO_FIFOs for a used clocking component can be part of the fence.

A high-level block diagram showing the coincident IO_FIFOs and borders for each clocking component, as well as the minimum IO_FIFO fence tile needed for the required isolation in each case, is shown in Figure 36.

7 Series Minimum IO_FIFO Fence Tiles for CMT
Left Side - PlanAhead Tool View

Note: Phaser and PLL are used in this CMT. MMCM is not used.

Note: PLL and MMCM are used in this CMT (in different isolated regions). Phaser is not used.

Note: MMCM is used in this CMT. PLL (and associated phaser) is not used.

NOTE: Minimum fence is one IO_FIFO but phaser cannot be used without the PLL, so unused IO_FIFO is in fence also.

| IOB / IOI Tiles | IO_FIFO | PLL | CLB Tiles | CLB Tiles | I/O Bank |

Figure 36: **Minimum IO_FIFO Fence Tiles for the 7 Series CMT Clocking Components**

X17848-090616

Legend

Fence Tile

Isolated Region #1 Used Tile

Isolated Region #2 Used Tile

Isolated Region #3 Used Tile

Isolated Region #4 Used Tile

Unused CMT Tile

CMT

Hints and Guidelines

General Hints and Guidelines

I/O Pair Tile versus I/O Single Tile

The I/O pair *cannot* be cut in half to create two I/O singles to use as a fence tile. The I/O pair tile has common Master/Slave logic, which does not give the required isolation, so the fence tile must be the complete I/O pair tile.

The I/O single tile does have the required isolation for a fence tile.

Resources to Add in an IDF Isolated Region

It is generally advised to add all available resources within an isolated region, even if the logic is not used in the design, because excluding them also excludes using their respective routing resources. Failure to do so, while not an error, might produce designs that are unnecessarily difficult to route. This includes the regional and I/O clock buffers (BUFR, BUFMRCE, and BUFIO).

However, in the Vivado tool, BUFGs and BUFHs should not be included in the isolated region unless you want the global clock isolated. All other components should be included.

CMT Resource Exception Guideline

If not adding the entirety of a CMT site to a specific isolation region, it is recommended that the entirety of the CMT be left out of that region. A frequent cause of the `overlapping pblock` error is due to the large scope of various sites in the CMT. Care must be taken when adding only a portion of the CMT to an isolated region. The rules for breaking up CMT for fencing reasons is described in the IO_FIFO Fencing Guideline section.

Note: The CMT column is directly adjacent to the I/O column and the routing resources to the I/O are not dense. There are enough routing resources to jump over an unused CMT column to gain access to the I/O routing without causing any routing issues.

CMT User Tile IO_FIFO Fencing

IO_FIFO Fencing Guideline

The Phaser and the MMCM share a common IO_FIFO, as shown in Figure 37 with the dashed line around the common IO_FIFO. A fence tile in the CMT must include at least one IO_FIFO, therefore, the MMCM and the Phaser cannot both be used in different isolated regions Pblocks). However, the MMCM and the Phaser could be used in the same isolated region.

Note: Figure 37 is only highlighting the common IO_FIFO tile between the MMCM and the Phaser. The Phaser always needs to be used with the PLL, which is not shown here.

Common IO_FIFO
MMCM and Phaser
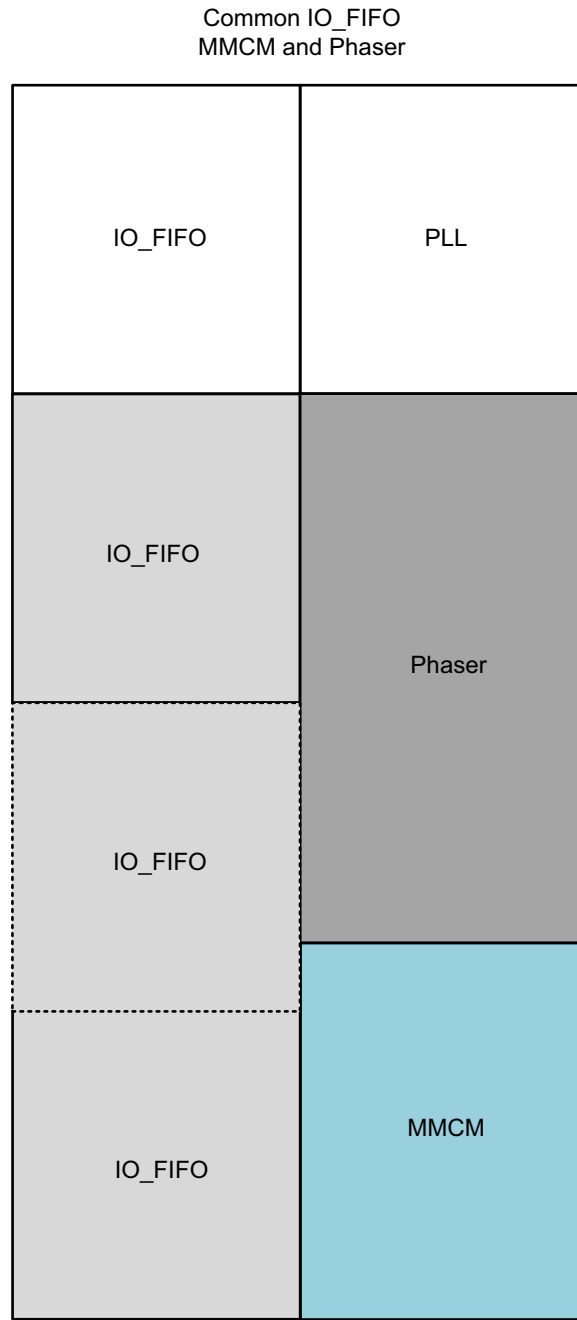
X17849-090616

Figure 37: **Common IO_FIFO Between MMCM and Phaser Components**

www.xilinx.com

Send Feedback

Guidelines for Controlling Global Clocking Logic

Before Vivado tools, you had to ensure that all global logic was at the top level of the logical hierarchy. More specifically, you had to ensure no global logic was logically contained in modules that were to be isolated. This is due to the 7 series architecture that prohibits the isolation of such components due to their large scope. While this architecture limitation has not changed, how the tools deal with this architecture has. Vivado tools allow you to keep your global logic as is (without HDL modification) by giving you the option to target said logic with an attribute that exempts it from isolation. This is a key feature in Vivado tools, because there are many IPs that have global logic embedded in them (BUFGs and MMCMs for example). In such cases, you identify them with a search and add the HD.ISOLATED_EXEMPT attribute to them.

Examples

Global Clocks:

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

Regional Clocks:

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

Note: The filtering structure (that is, `CLK.rclk.*`) shown in the example is correct for Vivado Design Suite 2015.2, however this could change in future versions. Please verify the correct structure for the version of Vivado used for your design.

These filter statements are very generic (wild cards). More specific operations can be performed if there are some clocks that need exemption and some that do not.

Note: Though you cannot realistically isolate a global clock like a BUFG due to its global scope, it is possible to isolate regional clocks like BUFRs. In such a case, ensure the entire region being driven by the regional clock is in the isolated region (that is, floorplan on clock boundaries).

Vivado IP Integrator Differences

Vivado IP integrator is fully compatible with the Isolation Design Flow. In some cases it even makes it easier, for example, with the ability to add hierarchy with a single click/command.

However, there is one feature to be mindful of when creating isolated designs in Vivado IP integrator. To preserve the IP being instantiated, IP integrator automatically adds the DONT_TOUCH property to all instances of IP. This is a good thing, because it adds the IP vendors' IP without modification. However, this impacts the feature of IDF where isolated nets can drive multiple isolated regions. In this case, the Vivado tool attempts to split that net to follow IDF rules but is prevented from doing so due to the DONT_TOUCH attribute.

If an IP is to be marked as an isolated function, that IP must be wrapped with a level of hierarchy that does not have the DONT_TOUCH attribute. In such cases, follow the *Vivado Design Suite*

Send Feedback

User Guide - Designing IP Subsystems Using IP Integrator (UG994) [Ref 7] to add a level of hierarchy to a module (a simple right-click operation).

Gaps in the Floorplan

There are apparent gaps in the floorplan as seen by the Vivado tool Routing Resources view (Figure 38). These gaps are an artifact of how the software model is depicted in the Vivado IDE. Such gaps do not represent any form of isolation and cannot be used as a fence tile because they are not, in fact, a user tile at all.
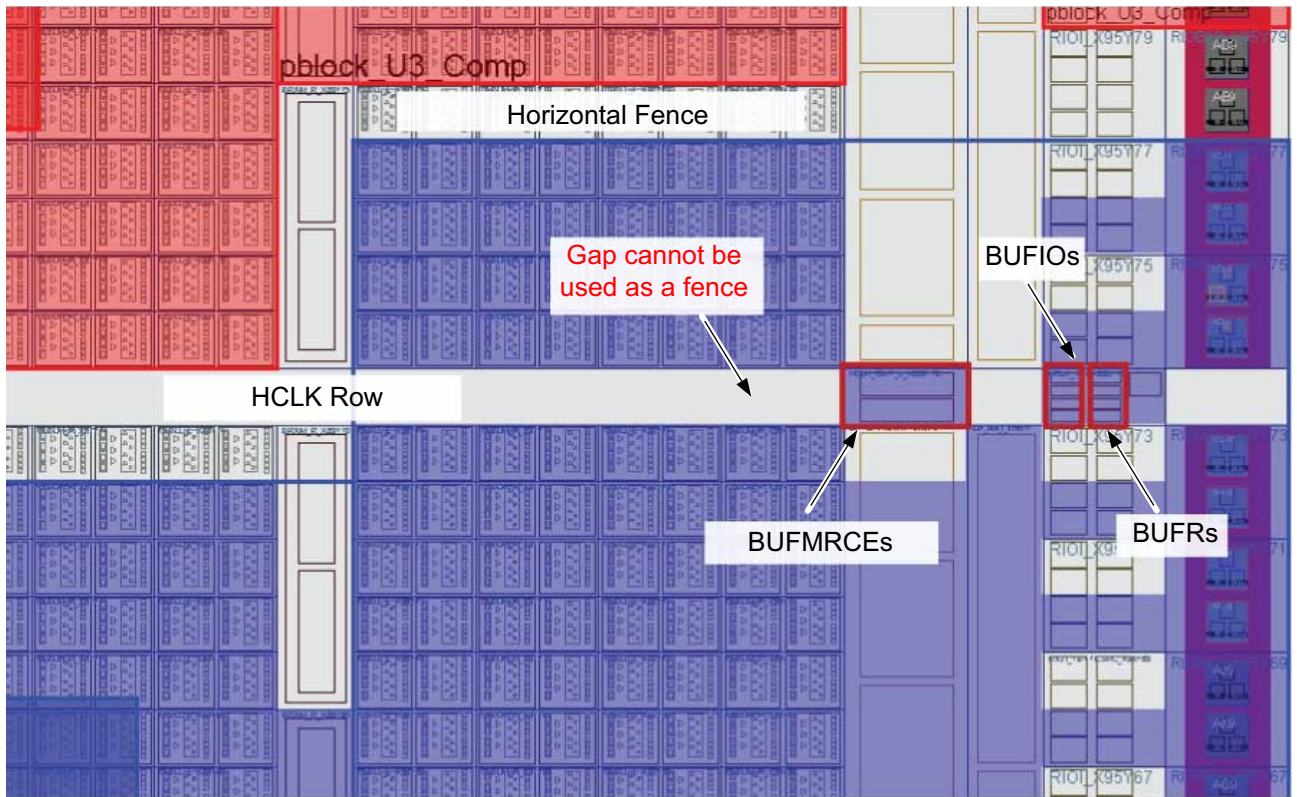
The following are the most common gaps seen in the floorplan:

- HCLK row (horizontal clock row)
- Vertical clock center column
- Configuration blocks near the center of the device floorplan

HCLK Row Gaps

The HCLK row, in the center of each I/O bank, appears as a gap in every user tile type it crosses, as shown in Figure 38. It contains clock routing and the following clock and I/O buffers near the I/Os:

- BUFIO (I/O buffers)
- BUFR (regional clock buffers)

Send Feedback

Figure 38: **Gaps in the Floorplan—HCLK Row**

www.xilinx.com

Send Feedback

Vertical Clock Center Column Gap

The vertical clock column, in the center of the device, appears as a gap between CLB tiles, as shown in Figure 39. It contains clock routing and the following clock buffers:

- BUFHCE: Horizontal clock drivers for each HCLK row

- BUFGCTRL: Global clock buffers located in the center, spanning the whole device
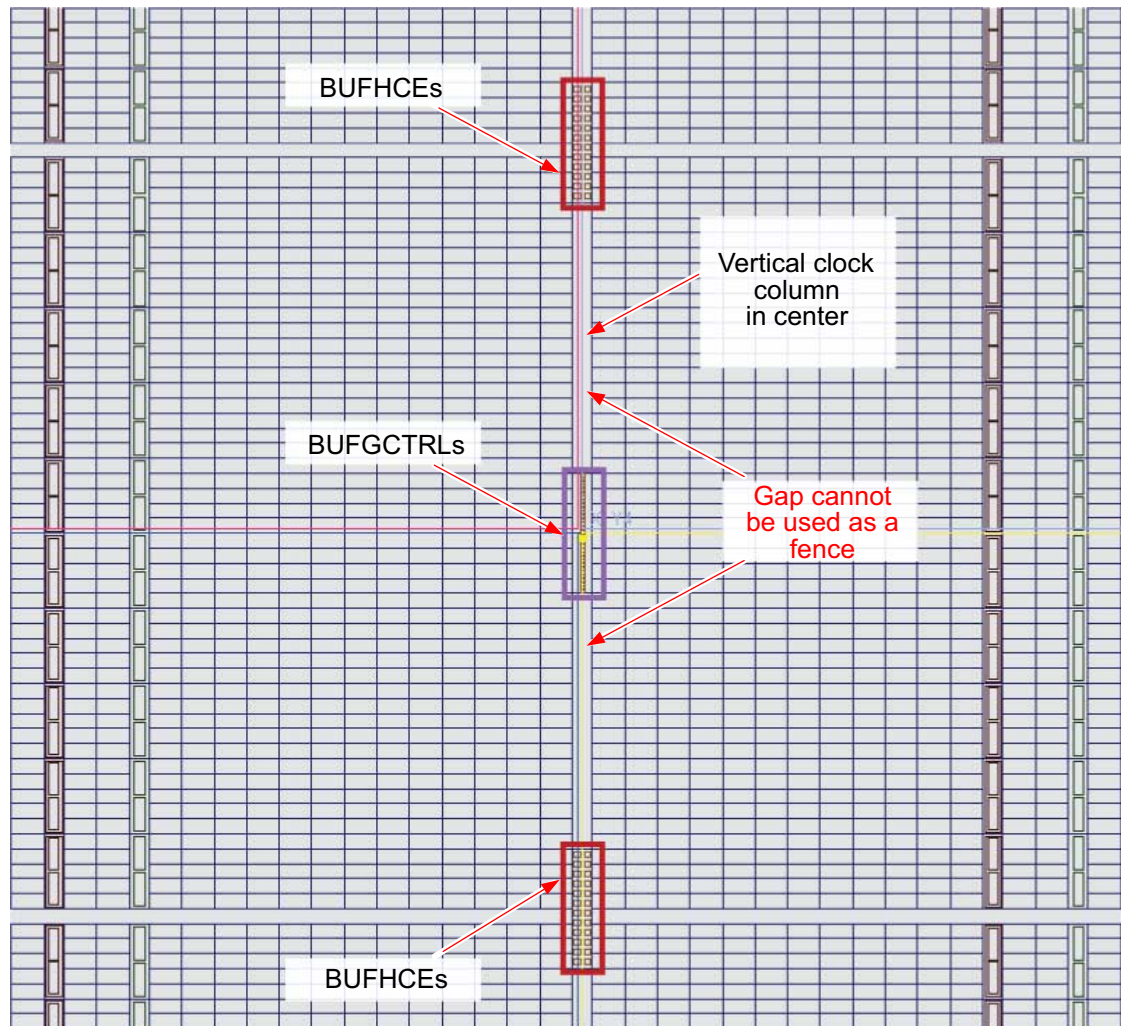
X17851-090616

Figure 39: **Gaps in the Floorplan—Vertical Clock Center Column**

Configuration Blocks Gap

The configuration block, near the center of the device, appears as a gap between CLB tiles vertically and between CLB and BRAM tiles horizontally, as shown in Figure 40. This block cannot be used as isolation. It contains configuration and analog blocks such as:

• XADC

• ICAP

• BSCAN

• EFUSE

• FRAME_ECC
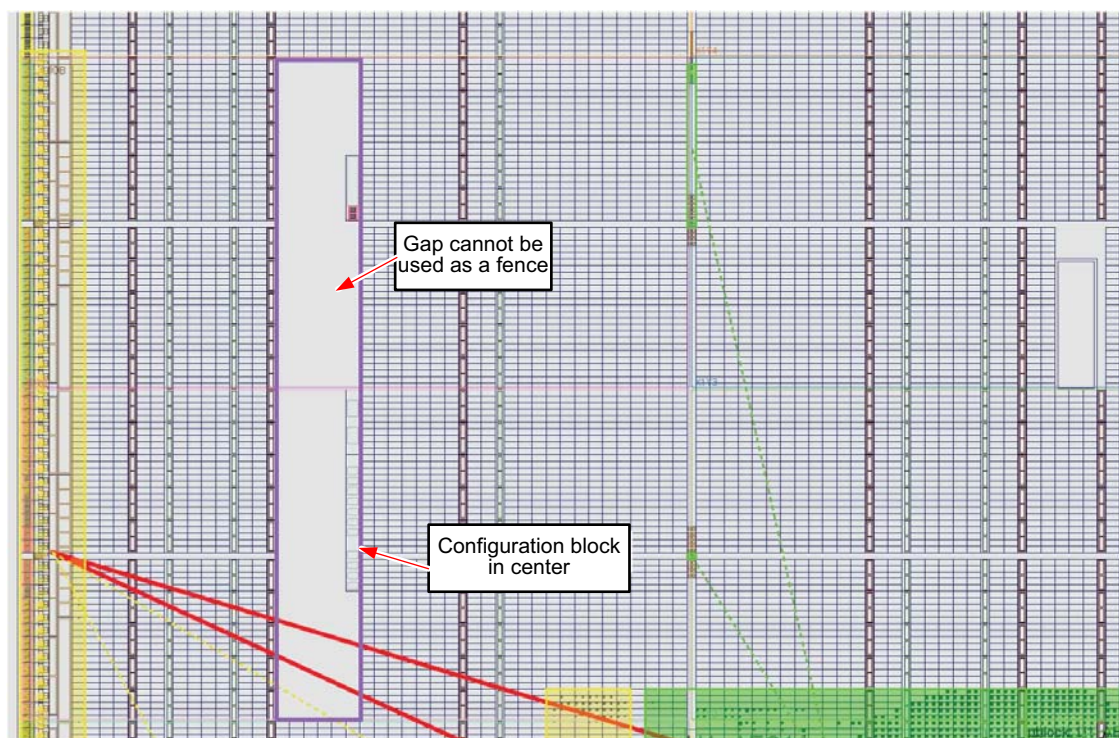
Figure 40: **Gaps in the Floorplan—Configuration Blocks**

As stated, the configuration block cannot be used as an isolation fence tile. Because it contains configuration and analog components that might be used in the design, the IDF rules require that if any of the logic components are used in an isolated region, all of the logic components need to be included in that isolated region. In other words, the configuration logic components cannot be split between different isolated regions or be part of the isolation fence. Xilinx recommends that whether or not you make use of configuration block components, the components are included in an isolated region.

Because the configuration block cannot be used as an isolation fence tile, ensure there is an isolation fence, using valid fence tiles, completely surrounding the isolated region.

www.xilinx.com

Send Feedback

Automatic Movement of IOB into Isolated Hierarchy

The automatic insertion of IOBs into the intended isolated hierarchy is new to IDF with the introduction of Vivado Design Suite. Before Vivado tools, you had to manually instantiate IOB inside of the HDL for each isolated module. Vivado tools, however, do this for you provided you have not already instantiated the IOB themselves. At initial inference, all IOB are at top level. Where Vivado IDF can see a clear usage for the IOB, it automatically moves this IOB from the top level into the desired isolated region. This movement happens in the opt_design step after synthesis but before placement.

However, there are some limitations to this feature because not all possible cases allow for such movement. These cases are as follows:

- **Case 1:** Input with multiple destinations (including top level). See Figure 41.

    In this case, a "reset" input to the top level MMCM also connects to at least one isolated region.
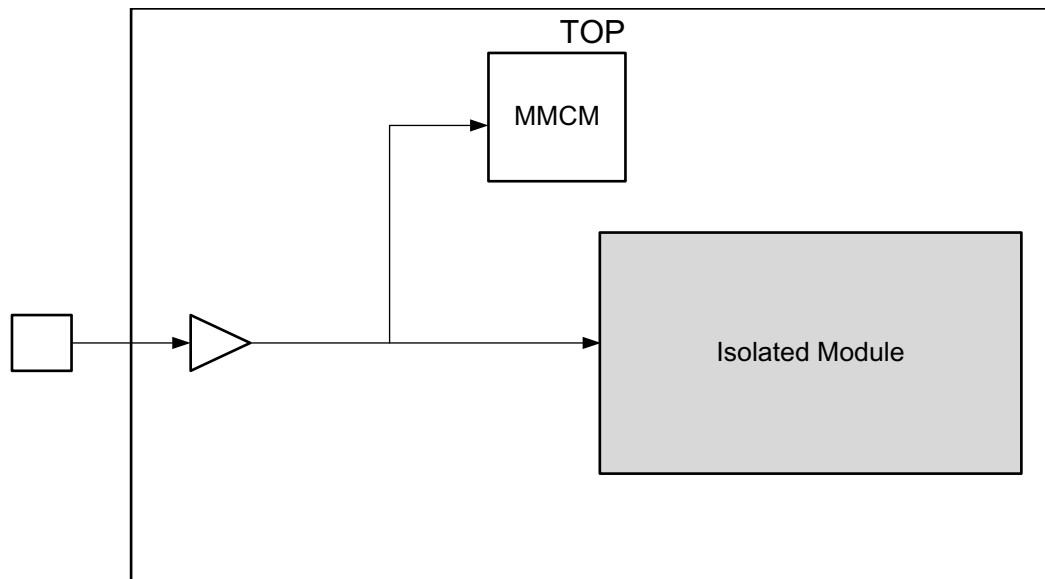
X17853-090616

Figure 41:   **Case 1: Input with Multiple Destinations (including top level)**

Send Feedback

- **Case 2:** Output whose input drives the IOB in question and at least one other region (including top level). See Figure 42.
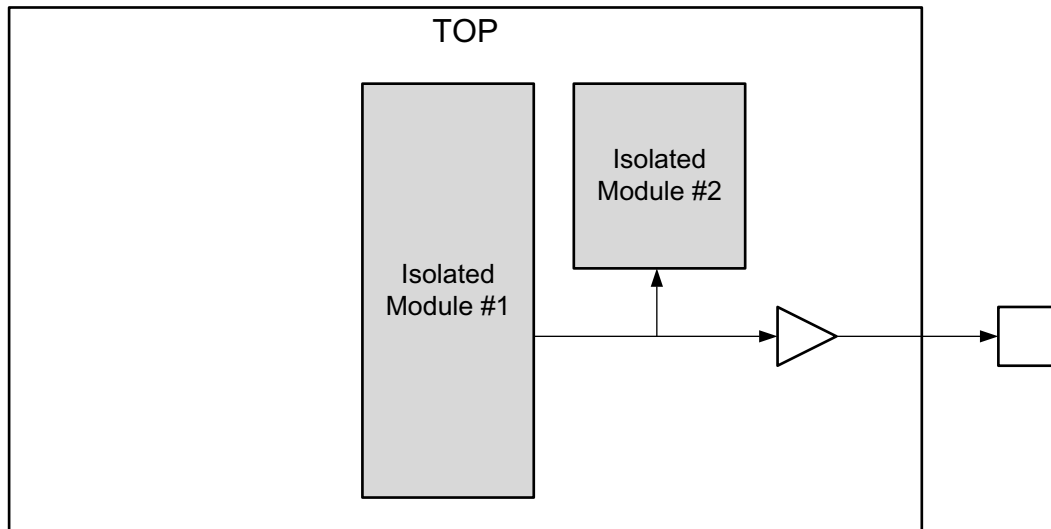
X17854-090616

Figure 42: **Case 2: Output Whose Input Drives the IOB in Question and at Least One Other Region (including top level)**

- **Case 3:** Output whose input comes directly from an input port of the isolated module in question. See Figure 43.

  It does not matter if that signal is used in the isolated module in question or not.
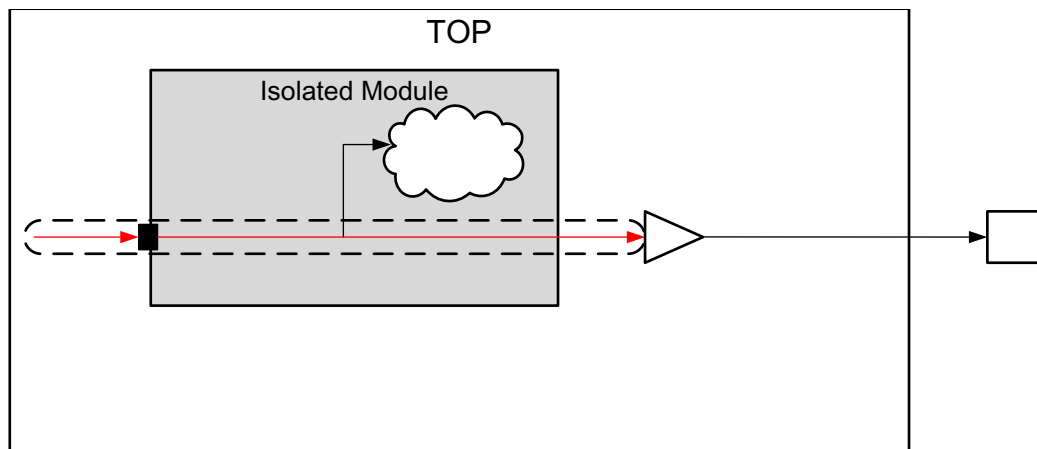
X17855-090616

Figure 43: **Case 3: Output Whose Input Comes Directly from an Input Port of the Isolated Module in Question**

Send Feedback

![XILINX®]

• **Case 4:** Bidirectional IOB where input and output do not go to the same isolated region. See Figure 44.
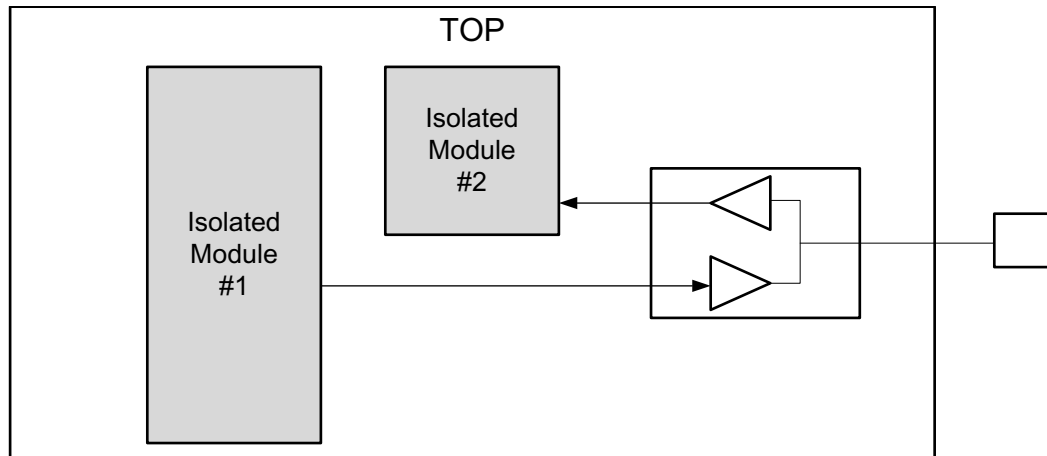
X17856-090616

Figure 44: **Case 4: Bidirectional IOB Where Input and Output Do Not Go to the Same Isolated Region**

• **Case 5:** IOB with connections only to top level logic. See Figure 45.

Top level IOB driving only top level logic will not be moved.
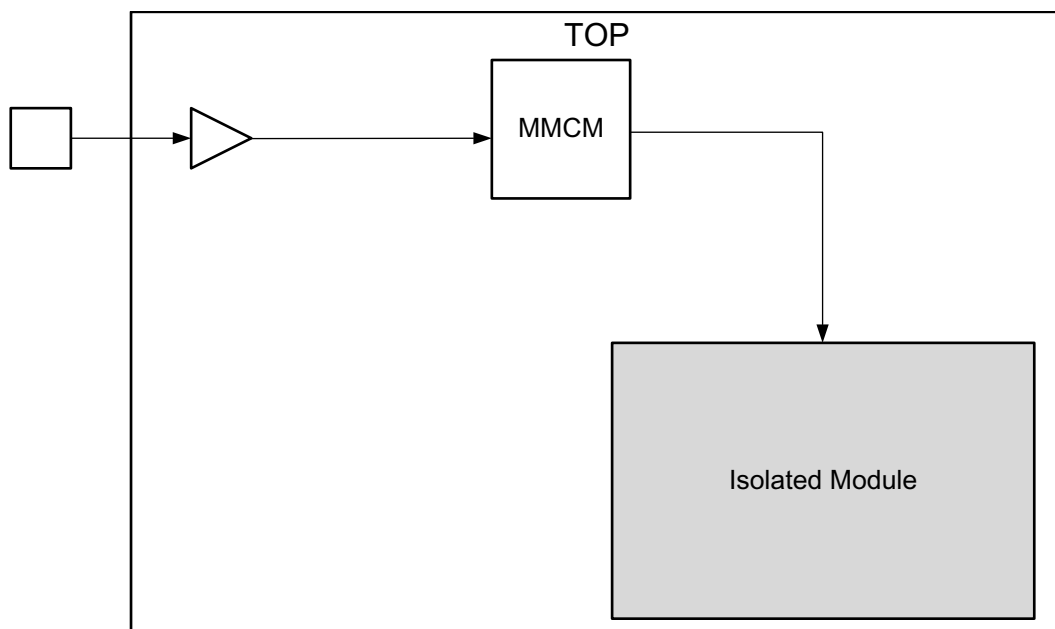
X17857-090616

Figure 45: **Case 5: IOB with Connections Only to Top Level Logic**

• **Case 6:** IOB was manually instantiated by the user or their IP.

Directly instantiated IOB with the DONT_TOUCH property set on the buffer will not be moved. The DONT_TOUCH property can be inherited by the IP that instantiates the IOB or you can add it directly.

Send Feedback

IDF Rules Checklist

The following checklist is a helpful set of items you can use to verify that all the basic rules and guidelines for an IDF design are followed. The checklist can be used before, during, or after project design to help ensure the rules have been followed and to determine if overall good IDF design practices have been considered. A good IDF lab tutorial example to reference would be *Zynq-7000 AP SoCs or 7 Series FPGAs Isolation Design Flow Lab (Vivado Design Suite)* (XAPP1256) [Ref 8].

IDF Checklist

- The HD.ISOLATED property is applied to all isolated modules (except global clocking modules).

- All isolated modules have corresponding Pblocks associated with them.

- Pblocks must contain the required resource types in sufficient quantities to implement the intended function or functions.

- Each isolated module must be in its own level of hierarchy.

  ◦ Isolated modules cannot be nested.

- As a general rule, to increase routing resources, all available resources (except BUFGs and BUFHs) should be assigned to the isolation region unless there is a specific need to exclude that resource.

  ◦ An exception to this is that the Clock Management Tile (CMT) components should only be added to an isolation region if the specific site is used in the design.

- A valid isolation fence (unused tiles) must exist between Pblocks.

  ◦ The fence is constructed indirectly by the space between isolated regions.

  ◦ Refer to Table 3 Xilinx 7 Series FPGA and Zynq-7000 SoC Fence Tile Requirements to create a valid fence.

    - This is determined through schematic analysis and stated in rules and guidelines throughout this document.

- Isolated modules that communicate with each other must share a coincident Pblock edge separated by a valid fence.

- Pblocks must enclose all available tiles in the device (only the fence remains).

  ◦ Do not leave large gaps because the gaps do not provide an advantage and can even result in routing difficulty.

  ◦ Making the fence larger than necessary can make it difficult or impossible for inter-region signals (trusted routes) to cross the fence.

    - For example, IDF rules prevent routing touchdowns in the fence. A fence size of one (the minimum fence size allowed) prohibits the use of all routes (to cross the fence)

Send Feedback

> that span only a single tile (singles). Larger fences further limit the routing resources available.

- Only global clocking logic is allowed to not be isolated. All other logic must be associated with an isolated module and placed in the associated isolation region.

- For global logic instantiated within an isolated function, the HD.ISOLATED_EXEMPT property must be set (TRUE) for the tools to route to or from that global logic.

- Feed-through signals are not allowed without buffering of some kind (LUT, FF, etc.).

  ◦ Vivado does this automatically unless you choose to disable this feature.

- Ports communicating between two isolated modules can only have one source and one destination.

  ◦ Vivado does this automatically unless you disable this feature

- IOBs must be instantiated or inferred inside isolated modules for proper isolation of the IOB.

  ◦ Vivado does this automatically unless you choose to disable this feature.

- These are the limitations on Vivado automatically inserting IOBs:

  ◦ Input with multiple destinations, including top-level (i.e., reset to MMCM and isolated module)

  ◦ Output whose input drives the IOB in question and at least one other region, including top-level

  ◦ Output whose input comes directly from an input port of the isolated module in question

  ◦ Bidirectional IOB where input and output do not go to the same isolated region

  ◦ IOB with connections only to top level. Top level IOB driving only top-level logic will not be moved.

  ◦ IOB was manually instantiated by you or the IP.

    - Directly instantiated IOB with the DONT_TOUCH property set on the buffer will not be moved. The DONT_TOUCH property can be inherited by the IP that instantiates the IOB or you can add it directly.

- Run Xilinx Vivado Isolation Verifier (VIV) on the elaborated design (constraint checking) to check for DRC violations for device and package pins (pin adjacency) and for IOB bank violations (if required).

- Run Xilinx VIV on the placed and routed (implemented) design to check for DRC violations for Isolation violations.

- Run Xilinx VIV on an unmodified design checkpoint (to ensure VIV is not run on a cached version of the design).

- The VIV Tcl script must be loaded into the Vivado tool.

  ◦ The script is not included in the base delivery but is available at no charge from the Isolation Design Flow site.

Device Ordering Instructions

Contact your local Xilinx representative for more information.

Reference Design Files

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

The trusted routing design files (including the instantiation templates), files, and simulation models are available on the Isolation Design Flow page [Ref 3]. The design matrix in Table 4 includes simulation, implementation, and hardware details for the reference design.

Table 4: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Satya Pitaka
Target device	Zynq-7000 XC7Z020 SoCs
Source code provided	Yes
Source code format	VHDL
Design uses code or IP from existing Xilinx application note reference designs or third party	No
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 2015.1
SPICE/IBIS simulations?	No
Implementation	
Synthesis software tools and version used	Vivado Synthesis
Implementation software tools and versions used	Vivado Design Suite 2015.1
Static timing analysis performed?	Yes
Hardware Verification	
Hardware verified?	No
Hardware platform used for verification	N/A

Table 5: **Device Utilization**

Device	Speed Grade	Package	Post-Synthesis Slices	Post-Implementation Slices
XC7Z020	-1	CLG484	9116	9028

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (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 the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

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

References

1. Aerospace and Defense Security Monitor IP Core Prouduct Marketing Brief
2. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (ISE Tools)* (XAPP1086)
3. Isolation Design Flow website
4. *Vivado Design Suite User Guide - Partial Reconfiguration* (UG909)
5. *Vivado Design Suite User Guide - Hierarchical Reference* (UG905)
6. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
7. *Vivado Design Suite User Guide - Designing IP Subsystems Using IP Integrator* (UG994)
8. *Zynq-7000 AP SoCs or 7 Series FPGAs Isolation Design Flow Lab (Vivado Design Suite)* (XAPP1256)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/17/2020	1.4	Added Table 1 to show IDF supported devices.
09/23/2016	1.3	Updated the note in Top Level Logic. Added the IDF Rules Checklist. Updated the device in Table 4 and Table 5 (XC7Z020).
06/21/2016	1.2.1	Corrected a typo in the target device number in Table 4. Added [Ref 8].
10/05/2015	1.2	Updated Top Level Logic and added information about the consequences of fence size to Isolation Fence. Updated Final Isolation Verification (VIV - Implementation) and Figure 9. Updated Case 6 in Automatic Movement of IOB into Isolated Hierarchy.
09/01/2015	1.1	Added Trusted Routing description to Common Terminology, Added Figure 1, *Available Routes after Applying Trusted Routing Rules*. Added sections Top Level Logic and Automatic Movement of IOB into Isolated Hierarchy.
06/09/2015	1.0.1	Revised reference design links.
05/26/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices