Vivado Isolation Verifier

User Guide

UG1291 (v1.1) July 27, 2020

XILINX

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/27/2020	1.1	Added Table 1. Added a note in the Introduction section. Added three (3) notes in the Definitions section. Added links for XAPP1335 and XAPP1336 in the References section.
08/10/2018	1.0	Initial Xilinx release.

Table of Contents

Vivado Isolation Verifier

Introduction

Proof of correctness is required for sUG1291 (v1.1) July 27, 2020afety, security, and other high-reliability applications. The Xilinx® Isolation Design Flow (IDF) includes a set of design rule checks (DRCs) implemented by the Vivado® Isolation Verifier (VIV) that verifies the user and Vivado® software have fulfilled the requirements of IDF design methodology (See XAPP1335) [Ref 11].

The user needs to enable this feature explicitly (see Installation, page 16) since VIV is included with Vivado. When enabled, six additional DRCs appear under the Isolation category. The user invokes these DRCs, using the Vivado DRC interface just like any other built-in DRCs. Results are provided in tabular form in the GUI with hyperlinks to design elements related to potential isolation violations. The VIV DRCs also contribute to the text-based output of the Vivado DRC reporting system.

VIV aids in board development by checking that I/O pin assignments, I/O bank assignments, and floorplanning range constraints do not violate IDF rules. This is in addition to offering proof of isolation. The intent of these design constraint checks spares the designer from costly printed circuit board redesigns.

VIV is composed of six DRCs that perform checks related to a single aspect of isolation. The user can run all the IDF DRCs or any subset thereof.

- IDF_VIV2-1 provides provenance for VIV DRC results.
- IDF_VIV2-2, IDF_VIV2-3, and IDF_VIV2-4 checks the design constraints (pblocks, pads, pins, and banks).
- IDF_VIV2-5 and IDF_VIV2-6 checks the placement and routing of the implemented design.

Note: Although this document uses FPGA terminology, VIV applies equally to the Programmable Logic (PL) portion of Xilinx Zynq® devices, including the Zynq® UltraScale+™ MPSoC.

Supported Devices

Table 1: **Supported Devices**

Xilinx Device	VIV Support Status
Zynq® UltraScale+™ MPSoC devices	Supported
Zynq® UltraScale+™ RFSoCs	Not Supported
Kintex® UltraScale+™ FPGA devices	Supported
Virtex® UltraScale+™ FPGA devices	Supported
Virtex® UltraScale+™ HBM FPGAs	Not Supported
Virtex UltraScale+ 58G PAM4 FPGAs	Not Supported
UltraScale FPGA family	Not Supported
Zynq®-7000 SoC family	Supported
Spartan-7 FPGA	Only XC7S50 part is supported
Artix-7 FPGAs	All parts are supported except XC7A12T and XC7A25T
Kintex-7 FPGAs	Supported
Virtex-7 FPGAs	Supported

VIV History

Initially VIV was launched as a Tcl-based script (`viv.tcl`) and was not a part of the Vivado Design Suite which is distributed separately as an encrypted Tcl file. To use `viv.tcl` IDF DRCs, users need to source the `viv.tcl` script to load DRCs into Vivado. Refer to UG1290 [Ref 3] for complete details. VIV has been integrated with Vivado Design Suite starting with the 2018.2 release to improve its performance. This new version of VIV is also referred to as VIV2 in this document. To use VIV, users need to enable it by setting the *hd.enableIDFDRC* parameter to True.

Definitions

Many of the terms in this document are used in a specialized sense. The following glossary helps readers who are not well versed with Xilinx terminology related to FGPA architecture and configuration, or search algorithms.

area range – list of rectangular regions identifying a subset of the device resources in an FPGA. It is defined as a list of resource pairs in an XDC file. Each pair defines two opposite corners of an included rectangular region.

bitstream – contiguous sequence of bits that represents a stream of data.

FPGA bitstream – file that contains the programming information for an FPGA. A Xilinx FPGA device must be programmed using a specific bitstream in order for it to behave as an embedded hardware platform. This bitstream is typically provided by the hardware designer who creates the embedded platform.

Programming an FPGA is the process of loading a bitstream into the FPGA. During the development phase, the FPGA device is programmed using utilities such as Vivado® or using menu options in SDK. These tools transfer the bitstream to the FPGA on board. The bitstream is usually placed in non-volatile memory in the production hardware which is configured to program the FPGA when powered on.

device model – data and data structures that describe the potential programming of a specific model of an FPGA. The device model specifies the capacity of the device and all of the features that can be configured to realize an FPGA design. The device model is highly abstracted from the FPGA hardware schematics. Only aspects of the FPGA hardware that are programmable are represented. Although it is theoretically possible that a programmable feature of an FPGA might not be represented in the device model (for example, if testing shows the feature to be unreliable), in practice this is not applicable because it would make it impossible to perform the tests that would validate or invalidate the feature.

programmable unit (PU) – set of logical tiles such as CLEs, BRAMs, DSPs along with their shared interconnect tiles (one block RAM, five CLEs, and five interconnect tiles shared between block RAM and CLEs are one PU). When reserving resources during floorplanning it is strongly recommended to take the whole PU in the pblock. See (XAPP1335) [Ref 11] for more details on PU.

fence tile / PU– un-programmed tiles or PU, free of routing or logic, which is used to separate two or more tiles or PUs containing logic or routing from distinct isolated modules.

> ***Note:*** Refer to (XAPP1335) [Ref 11] for fencing rules with respect to UltraScale+ architecture, and (XAPP1222) [Ref 8] for fencing rules with respect to 7 series FPGAs.

function module– collection of logic that performs a specific operation, for example an encryption circuit.

interconnect tile – common hard IP block providing a programmable switching matrix connecting programmable logic elements of virtually all types to routing resources. The interconnect blocks are collectively referred to as the Global Switching Matrix in the end user documentation. Typically, individual interconnect blocks are not referred to in the documentation, but might occasionally be referred to as switch boxes.

inter-region signal – non-isolated net with one source and one load typically connects one isolated function/module to another, though sometimes connects one port of an isolated module to another port of the same isolated module or to top-level logic. An inter-region signal is not permitted to use routing resources containing programmable interconnect points (PIPs) in fence.

isolation – free from unintended influence. For the case of routing, the degree isolation is measured by the number of switch failures required to establish an unintended signal path between isolated circuits. For the case of floorplanning, isolation is determined by the presence of a "fence" of tiles/PUs free of isolated logic and routing between isolated portions of the design.

isolated function, isolated module – portion of the user design that is intended to be isolated.

isolated region, isolated – collection of tiles defined by area range constraints that can be used when implementing an isolated module.

I/O buffer, IOB – circuit in an FPGA that controls the behavior of the input/output pins on the FPGA package. An I/O buffer controls various communication-related settings, such as whether an associated pin is connected internally to an input circuit or an output circuit, or selects the voltage level expected by the pin.

I/O bank – collection of I/O buffers in an FPGA for settings and signals, common to the collection.

logic - circuits that implement a specific function; a flip-flop, look up table, and random access memory.

net – named collection of routing resources that creates signal paths among a collection of logic elements. A net may span levels in the design hierarchy.

node – indivisible unit of programmable routing. Note that a node may branch out to connect more than two points.

package pin – conductor on the outside of an FPGA package that powers or interfaces with the FPGA. It is shaped like a short wire protruding perpendicularly from the chip package, or shaped like a bump.

partition – collection of logic defined by the user that isolates one piece of hierarchy from another.

placement – assignment of a logical function to specific hardware resources.

derived range – derived range is the pblock boundary after considering all the tiles of the Programable Units (PU) in that pblock. When Snapping Mode property of a pblock is set to OFF, then both derived and XDC range compute the same pblock boundary. When Snapping Mode property of a pblock is set to FINE_GRAINED, the derived range might be different than the user-specified XDC range. If Snapping Mode is set to FINE_GRAINED, and if some of the tiles of a PU are left off in the XDC range, then all of the tiles in that PU are excluded from the pblock boundary in the derived range.

> ***Note:*** IDF requires FINE_GRAINED as the Snapping Mode property of pblocks. See (XAPP1335) [Ref 11] for more details.

Note: Programmable Unit and Derived Range concepts are applicable with respect to UltraScale+ architecture.

route – path that a signal follows within an FPGA as represented by a collection of nodes connected to one another by programmable interconnect points (PIPs).

site – physical location in the FPGA tile array that can be referenced in the floorplanning constraints, such as a SLICE, or RAMB16, etc.

switch matrix, global switch matrix, GSM – aggregate term for a programmable routing. The GSM is primarily composed of interconnect blocks.

trusted routing – connects isolated functions using routing resources with no programmable interconnection points within fence tiles. Trusted Routing is generated automatically without manual placement. Refer to (XAPP1335) [Ref 11] for more details.

Xilinx Design Constraints, XDC – SDC-based constraints in Tcl notation describing aspects of the design that includes floorplanning, pin assignments, electrical properties of I/O signals, and timing characteristics, but not the logic of the design.

wire – conductive path in a chip along which signals or power flow. A wire is the hardware that implements the node abstraction. The term wire also describes the software device model for a portion of a node that occupies exactly one tile.

FPGA Architecture

A field programmable gate array (FPGA) contains logic elements and routing. Both are controlled by configuration memory programmed by the user. Logic elements range in complexity from simple combinatorial logic functions up to complete embedded processors. Logic elements and routing are arrayed in a grid of tiles. The structure of an FPGA is extremely regular. Each tile contains one of a small variety of VLSI circuits dedicated to logic or routing.

Logic tiles include:

- Configurable logic blocks (CLBs) contains a small amount of programmable logic and memory.
- Input/output block circuitry (IOBs)
- Clock Management Tiles (CMTs)
- Other specialized circuitry that includes block RAMs, digital signal processors (DSPs), processors, etc.

Typically, Xilinx FPGA will have thousands of tiles, but only dozens of tile types. Common to all tiles is their association to a Global Switch Matrix (GSM). The GSM is composed of many interconnect tiles and interface tiles. Some logic tiles such as CLBs are associated with one interconnect tile, whereas other tiles such as block RAMs and DSPs are associated with multiple interconnect tiles. Interface tiles are used to adapt the various types of logic tiles to the common interconnect tile design.

In 7 series architectures, each user tile has a dedicated interconnect tile but UltraScale+™ architecture is different, as user tiles share interconnect tiles. In UltraScale+ devices two CLEs share one interconnect tile or one block RAM shares five interconnects with five CLEs, thus introducing the Programmable Unit (PU). The PU is a set of tiles that shares interconnect tiles. For example, two CLEs that share one interconnect tile is one PU. It is the same as one block RAM with five CLEs that share five interconnect tiles, and constitutes one PU. See (XAPP1335) [Ref 11] for more details on PU and UltraScale+ Architecture specific IDF concepts.

An FPGA is configured for a particular purpose by loading configuration memory with a particular bitstream. The bitstream specifies the exact function of each and every tile in the device. Whether used or not, logic tiles are configured to perform a specific function and the GSM is configured to provide the required routing between the logic tiles.

Note: Although the IDF focuses on FPGAs, the same methodology applies to the Programmable Logic (PL) portion of a Xilinx Zynq® UltraScale+ MPSoC.

The FPGA Development Flow with Isolation Analysis

To facilitate isolation analysis, the usual FPGA flow has a new set of constraints to control routing and additional floorplanning requirements. Firstly, the design is manually floorplanned. Secondly, constraints are applied to isolated regions of the floorplan to follow strict rules. Finally, VIV is used to demonstrate to ensure that the design is correctly implemented.

Figure 1 shows where isolation analysis fits in to the usual FPGA development flow. VIV is useful at two levels:

- VIV helps to avoid costly circuit board layout mistakes during floorplanning (shown on the right side blue boxes) and helps document the floorplan, a key part of the isolation approach of the design.
- VIV proves that the design is isolated per IDF rules when the design is complete (shown on the lower right green boxes).

The flowchart notation is as follows:

- Boxes represent processes
- Parallelograms represent data
- Trapezoids (quadrilaterals with one pair of parallel sides) represent manual input
- Shapes with curved bottoms represent output
- Arrows represent information flow
- Color is used specifically for grouping and emphasis

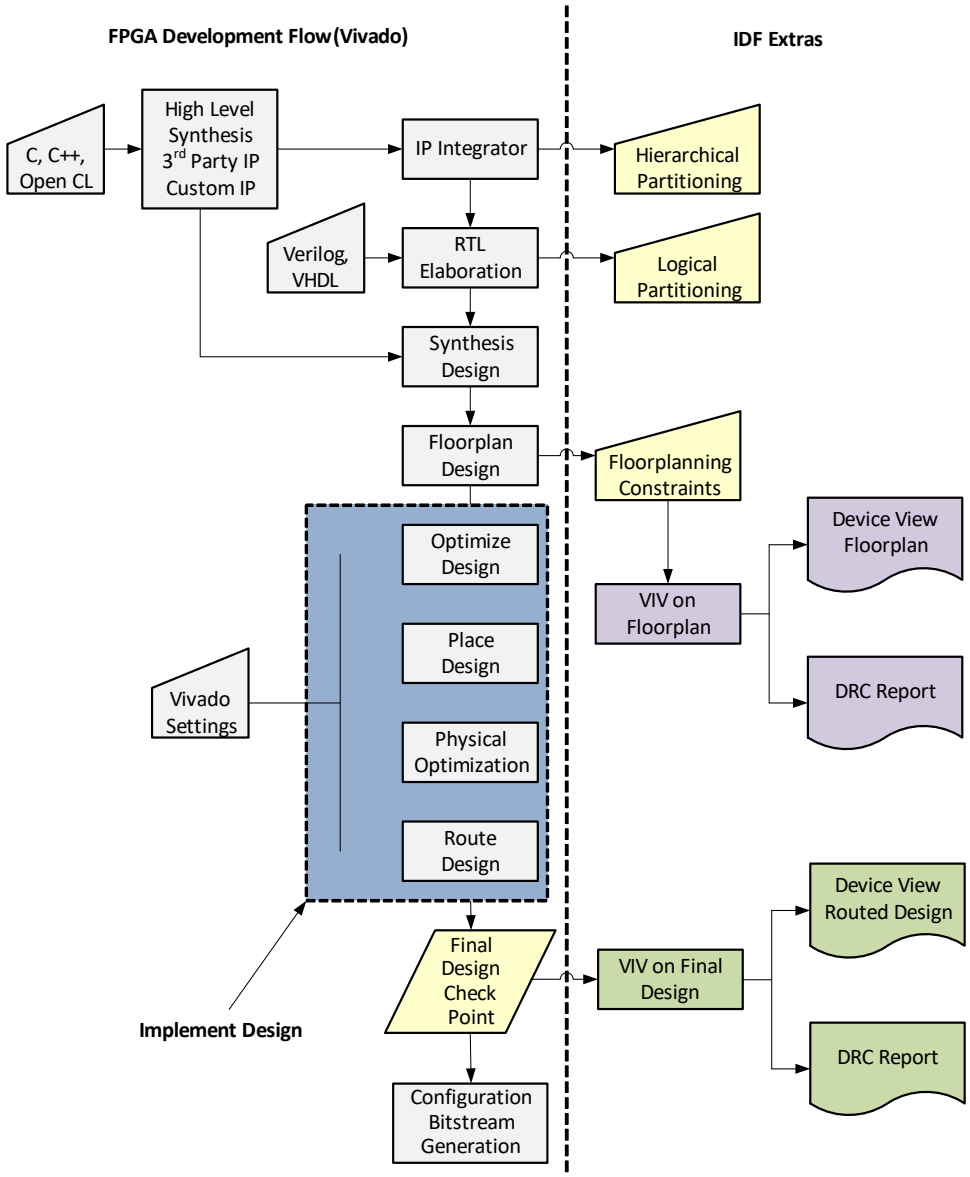

Figure 1: **Vivado Isolation Design Flow Relative to a Typical FPGA Design Flow**

IDF Design Rule Checks

VIV2 is available with Vivado framework in the form of built-in DRCs. This allows violations to be highlighted directly in the Vivado graphical user interface. The DRCs that apply to the pin assignments and floorplanning constraints are intended to aid board design. The DRCs that apply to the implemented placement and routing are intended to provide proof that isolation was achieved.

Constraint Checking (VIV – Constraints)

VIV checks the following on the pin constraints and floorplan:

- Pins from different isolation pblocks are not co-located in an I/O bank. This is checked in IDF_VIV2-2.

  Note: While VIV does report I/O bank sharing as a violation, this is a security precaution. - not mandated by analysis. The majority of applications allow for sharing of banks. Designers need to decide on a case-by-case basis if their design allows I/O bank sharing.
- Pins from different isolation pblocks are not physically adjacent on the package. Pins are considered adjacent if they share an edge or corner with no fence tile (unconstrained package pin) between them. This is checked in IDF_VIV2-3.
- The pblock constraints in the XDC file are defined such that a minimum of a one tile/PU wide fence exists between isolated regions. This is checked in IDF_VIV2-4.

  Note: For UltraScale+ architecture, fence is unprogrammed PUs and for 7 series FPGAs, fence is unprogrammed tile.

  Note: Refer to (XAPP1335) [Ref 11] for UltraScale+ architecture, and (XAPP1222) [Ref 8] for 7 series FPGA.

Because placement information is not used, VIV assumes 100% utilization of all constrained resources so that whatever resources are used in the implemented design, an isolation violation will not occur.

Final Isolation Verification (VIV – Implementation)

After the design is complete (placed and routed), VIV verifies that the required isolation is achieved in the design. VIV checks the placement and routing as follows:

- Tiles containing isolated logic must be separated by fence tiles. A fence tile cannot contain any logic (checked in IDF_VIV2-5) and cannot contain any routing (checked in IDF_VIV2-6) that could lead to an isolation violation with a single fault.
- Fence tiles cannot contain PIPs (checked in IDF_VIV2-6).

The six VIV DRCs are described in detail in the following sections.

IDF_VIV2-1 - Provenance

IDF_VIV2-1 is an advisory DRC documenting the circumstances of the run. It also validates that the design has at least two pblocks marked as isolated (using the `HD.ISOLATED` property). Nets driven by cells marked HD.ISOLATED_EXEMPT are exempt from inter-region isolation rules and are listed in the IDF_VIV2-1 output.

Here is an example of IDF_VIV2-1 output:

```
Vivado Isolation Verifier v2.0 (20180514)
Copyright (C) 2013-2018 Xilinx, Inc. All rights reserved.
Date(GMT): Tue May 15 12:50:38 2018
Top-level: design_1_wrapper
Isolated Partitions: pblock_uram_top_0 pblock_uram_top_1 pblock_uram_top_2
Part: xcvu13p-flga2577-1-i
Directory: c:/xilinx_design/implementation/idflab
User: <username>
Vivado Version: 2018.2
Platform: lnx64
Host: <hostname>

Top Level nets: CE_0, CE_1, CE_2, SCLR_0, SCLR_1, SCLR_2, clka_0, ena_0, clka_1,
clka_2, ena_1, ena_2, regcea_0, regcea_1, regcea_2 (the first 15 of 210 listed)

HD.ISOLATED_EXEMPT nets:
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK0,
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK1 and
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK2.

Inter-region nets:
design_1_i/keccak_0_ISO_Wrapper/buffer_data_reg[0]_0_ISOBUF_pblock_keccakCompare_0_
NewDrv,
M
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK0,
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK1,
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK2,
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_RESET0_N and
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_RESET1_N.
```

Note: For UltraScale+ architecture, IDF_VIV2-1 gives an error if there are any pblocks with Snapping Mode property not set to FINE_GRAINED.

IDF_VIV2-2 - I/O Bank Violation

IDF_VIV2-2 checks that each I/O bank is used by I/O pins of at most one isolation pblock. A violation of IDF_VIV2-2 is reported as:

```
Bank: <bank number> has pins from multiple isolated partitions: <partition name>
```

IDF_VIV2-3 - Package Pin Violation

IDF_VIV2-3 checks that no package pins from distinct isolation pblock are adjacent. A violation of IDF_VIV2-3 is reported as:

```
Package pin adjacency violation: site: <site name> pin: <pin name> vs site: <site name> pin:
<pin name>.
```

IDF_VIV2-4 - Floorplan Violation

Isolated pblocks must be separated by a valid fence. The definition of a valid fence is detailed in the Isolation Design Flow documentation. Users must follow the documented rules for the appropriate technology. Refer to the applicable reference documents identified in References, page 24. See (XAPP1335) [Ref 11] for additional details on fencing rules and see (XAPP1222) [Ref 8]) for 7 series fencing rules.

IDF_VIV2-4 checks that floorplan area ranges from distinct isolation pblocks have an appropriate gap between them. IDF_VIV2-4 checks Derived Range information, not the XDC Range, i.e. when the user gives in the XDC Range, Vivado computes internally the Derived Range by taking into account the Programmable Units and the Snapping Mode property of pblock.

IMPORTANT: *IDF requires that users create designs with Snapping mode FINE_GRAINED. Refer to* (XAPP1335) *[Ref 11] for detailed rules.*

Note: From 2019.2 onwards default value of SNAPPING_MODE for Isolated s is FINE_GRAINED.

Note: PU and Snapping Mode concept needs to be taken into account only for UltraScale+ architecture and not 7 Series.

The following example shows all of the constraints present in an XDC file that are needed to create a floorplan of an isolated function, into a specific region of the device (using pblocks).

```
create_pblock pblock_zup_ISO_Wrapper
add_cells_to_pblock [get_pblocks pblock_zup_ISO_Wrapper] [get_cells -quiet [list
design_1_i/zup_ISO_Wrapper]]
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {SLICE_X14Y150:SLICE_X15Y179
SLICE_X13Y125:SLICE_X13Y179 SLICE_X13Y35:SLICE_X13Y54 SLICE_X0Y0:SLICE_X12Y179}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{BUFCE_LEAF_X64Y8:BUFCE_LEAF_X87Y11 BUFCE_LEAF_X0Y0:BUFCE_LEAF_X63Y11}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{BUFCE_ROW_FSR_X12Y2:BUFCE_ROW_FSR_X16Y2 BUFCE_ROW_FSR_X0Y0:BUFCE_ROW_FSR_X11Y2}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{BUFGCE_HDIO_X0Y4:BUFGCE_HDIO_X1Y5}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {BUFG_PS_X0Y0:BUFG_PS_X0Y71}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {DSP48E2_X0Y0:DSP48E2_X0Y71}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{HARD_SYNC_X2Y4:HARD_SYNC_X3Y5 HARD_SYNC_X0Y0:HARD_SYNC_X1Y5}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{HDIOBDIFFINBUF_X0Y30:HDIOBDIFFINBUF_X0Y35}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{HDIOLOGIC_M_X0Y30:HDIOLOGIC_M_X0Y35}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{HDIOLOGIC_S_X0Y30:HDIOLOGIC_S_X0Y35}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add
{HDIO_BIAS_X0Y2:HDIO_BIAS_X0Y2}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {IOB_X0Y130:IOB_X0Y141}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {PS8_X0Y0:PS8_X0Y0}
```

```
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {RAMB18_X1Y50:RAMB18_X1Y71
RAMB18_X1Y14:RAMB18_X1Y21 RAMB18_X0Y0:RAMB18_X0Y71}
resize_pblock [get_pblocks pblock_zup_ISO_Wrapper] -add {RAMB36_X1Y25:RAMB36_X1Y35
RAMB36_X1Y7:RAMB36_X1Y10 RAMB36_X0Y0:RAMB36_X0Y35}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks pblock_zup_ISO_Wrapper]
set_property HD.ISOLATED true [get_cells */zup_ISO_Wrapper]
set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -filter {
PRIMITIVE_TYPE == CLOCK.BUFFER.BUFGCE }]
set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -filter
{PRIMITIVE_TYPE =~ CLOCK.BUFFER.BUFG_PS}]
set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -filter {
PRIMITIVE_TYPE == CLOCK.BUFFER.BUFGCE }]
```

A violation of IDF_VIV2-4 takes two forms:

```
Tile adjacency violation: pblock: <pblock name> tile: <tile name> vs pblock: <pblock
name> tile: <tile name>. Sites: <site name list>.
```

and

```
Tile occupancy violation: tile: <tile name> is in multiple isolated pblocks: <pblock
name list>. Sites: <site name list>.
```

IDF_VIV2-5 - Placement Violation

In contrast to IDF_VIV2-4 which checks XDC information, IDF_VIV2-5 checks placement of logic as actually implemented. Two simultaneous checks are performed. Firstly, a search for adjacent logic from distinct isolation modules is performed. In this context, logic is considered adjacent if the separation between the PUs/tiles containing logic of distinct isolated modules is not composed of a valid fence. Secondly, a check is performed to ensure top-level logic does not contain a potential path from one isolation group to another.

A violation of IDF_VIV2-5 takes two forms:

```
Tile adjacency violation: partition: <partition name> tile: <tile name> vs partition:
<partition name> tile: <tile name>. Sites: <site name list>.
```

and

```
Tile occupancy violation: tile: <tile name> is in multiple isolated partitions:
<partition name list>. Sites: <site name list>.
```

Note: Because IDF_VIV2-5 checks the implemented placement of logic, the results of running IDF_VIV2-5 are only useful if the design has been implemented. Prior to implementation there is nothing for IDF_VIV2-5 to check, and therefore no possibility a violation will be found.

IDF_VIV2-6 - Routing Violation

Isolated routing must be separated by an adequate fence and trusted routing must satisfy the following:

- Inter-region routes have loads in exactly one isolation group.

- No routing switches (PIPs) are used in the fence.
- Inter-region routes cannot share a tile unless source regions match and load regions match.
- An intra-region route cannot enter a fence tile or an isolated tile of another isolation group unless it is driven by a cell marked with the `HD.ISOLATED_EXEMPT` property.

Note: It might be useful to enable Routing Resources mode under **View > Routing Resources** in the menu or using the icon in the Device window.

Note: Because IDF_VIV2-6 checks the implemented routing, the results of running IDF_VIV2-6 are only useful if the design has been routed. Prior to implementation, there is nothing for IDF_VIV2-6 to check, and therefore no possibility a violation will be found.

Installation

Although VIV2.0 is being made available with Vivado since the 2018.2 Vivado release, it is disabled by default. Customers interested in IDF can enable it if they wish and thus VIV DRCs do not affect the normal development flow. Thus, the user must enable VIV explicitly by setting the `hd.enableIDFDRC` parameter to true by following any one of the following methods:

- Go to the Vivado Tcl Console, and enter the following command at the lower portion of the window (see Figure 2):

```
set_param hd.enableIDFDRC true
```

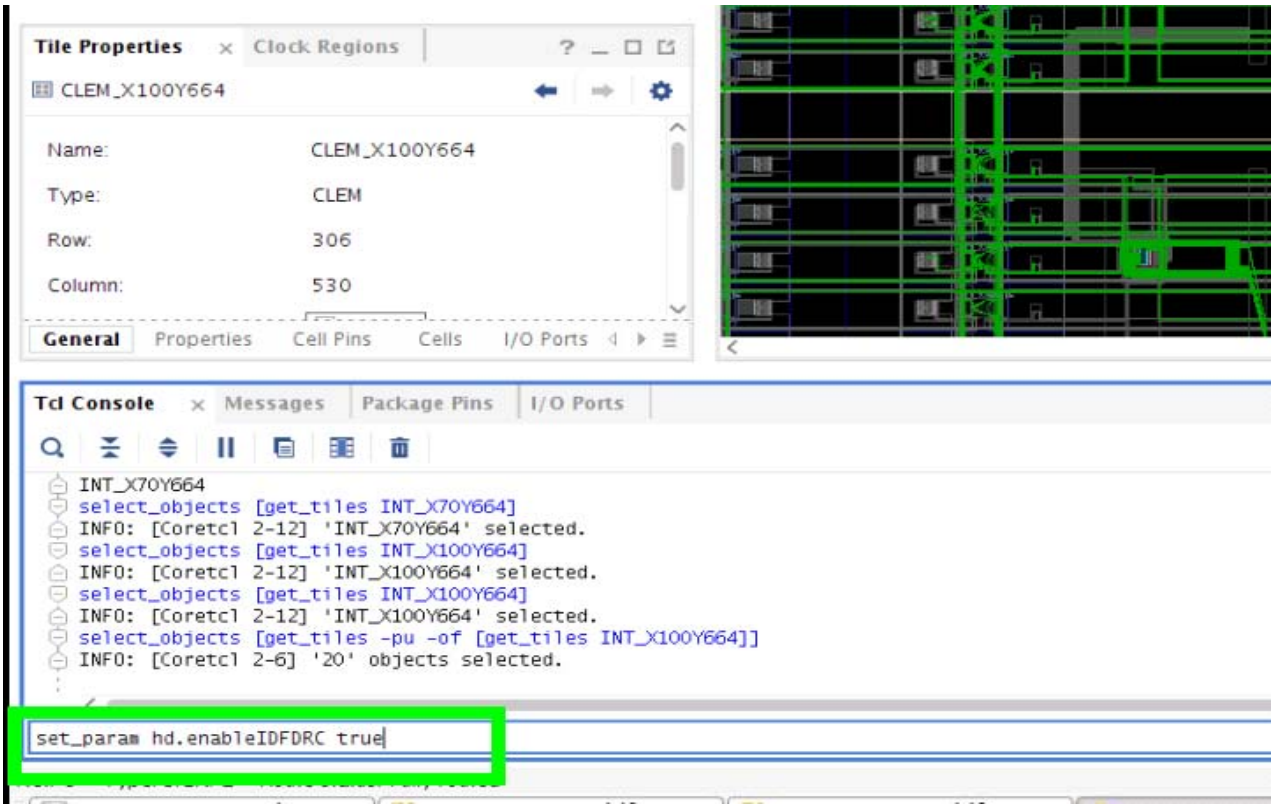

Figure 2: **Enable IDF DRCs by Setting the hd.enableIDFDRC Parameter**

The `hd.enableIDFDRC` parameter value will not be saved to project or DCP. Hence, user needs to set the parameter whenever they launch Vivado session to enable VIV DRCs, or alternately, add the *set param* command in the **`Vivado_init.tcl`** file to enable IDF DRCs permanently. This is helpful for customers always involved in IDF designs. The init file is under the **`~/.Xilinx/Vivado/2018.2`** directory. If you do not see the file under that directory, you need to create the file and add the following command to the init file:

```
set_param hd.enableIDFDRC true
```

Usage

When VIV is enabled by setting hd.enableIDFDRC to true, the Report DRC window will contain the six additional IDF DRCs under the category heading *Isolation* as shown in Figure 3. Note that the option under *Isolation* is not related to IDF and need not be checked.

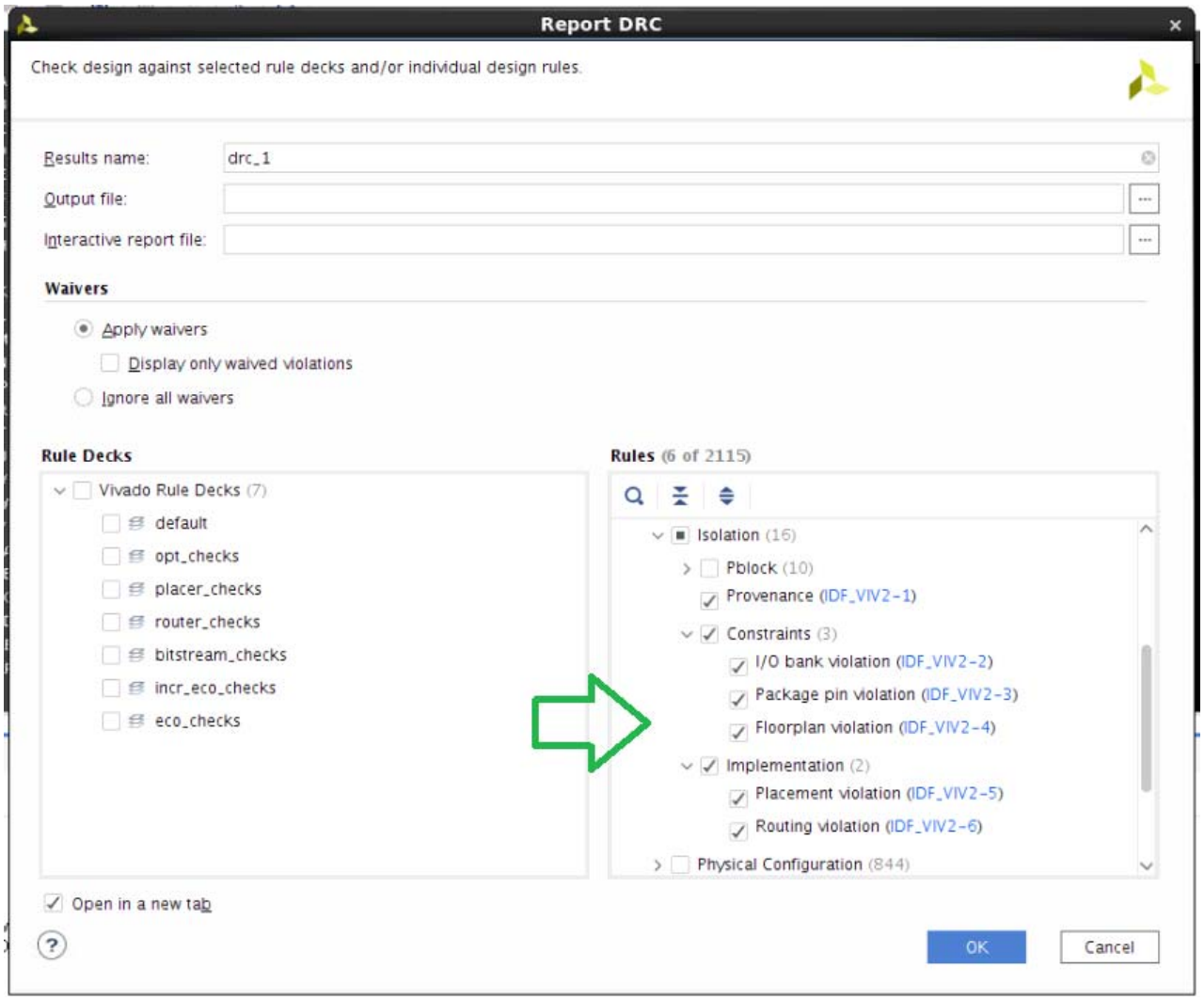

Figure 3: **IDF DRCs Selected in the Report DRC Dialog Box**

Invoking VIV DRCs

Users invoke these DRCs using the Vivado DRC interface just like built-in DRCs. Results are provided in tabular form in the GUI with hyperlinks to design elements related to potential isolation violations. The VIV DRCs also contribute to the text-based output of the Vivado DRC reporting system.

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** and **Package** windows.

Note: Specific DRCs can also be invoked using the following Tcl command:

```
report_drc -verbose -checks {IDF_VIV2-1 IDF_VIV2-2 IDF_VIV2-3
IDF_VIV2-4 IDF_VIV2-5 IDF_VIV2-6}
```

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.

- IDF_VIV2-1 is a provenance DRC and can be run at any time in the flow.
- IDF_VIV2-2, IDF_VIV2-3 & IDF_VIV2-4 can be run once synthesis has been performed.
- IDF_VIV2-5 and IDF_VIV2-6 examine the implementation, thus the Implementation step *must* be completed for IDF_VIV2-5 and IDF_VIV2-6 to have something to check.

An example DRC report in the Vivado GUI is shown in Figure 4.

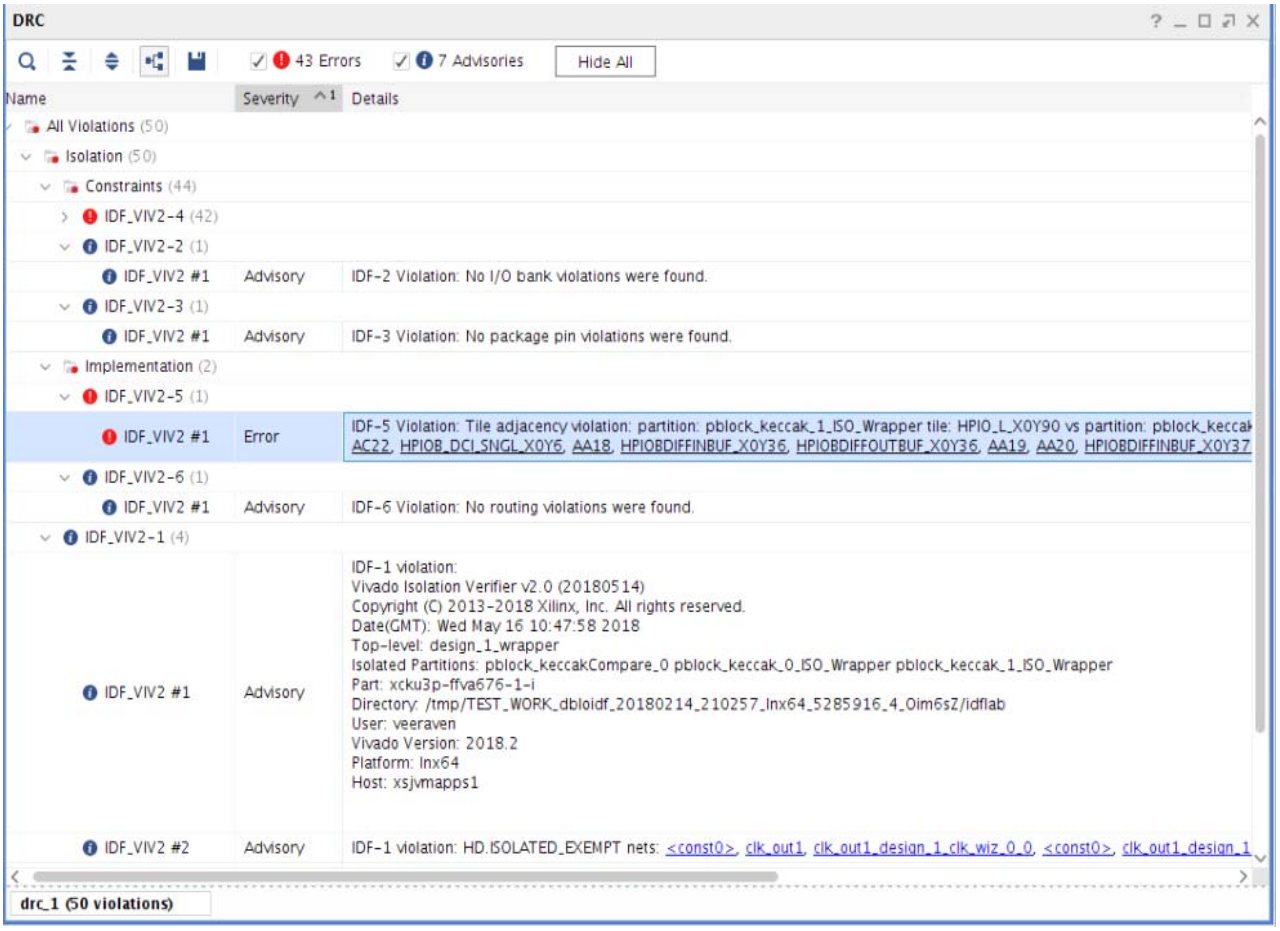

Figure 4: **IDF DRC Violations from Example Design**

Note that the highlighted DRC (IDF_VIV2 #1) in Figure 4 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 5.

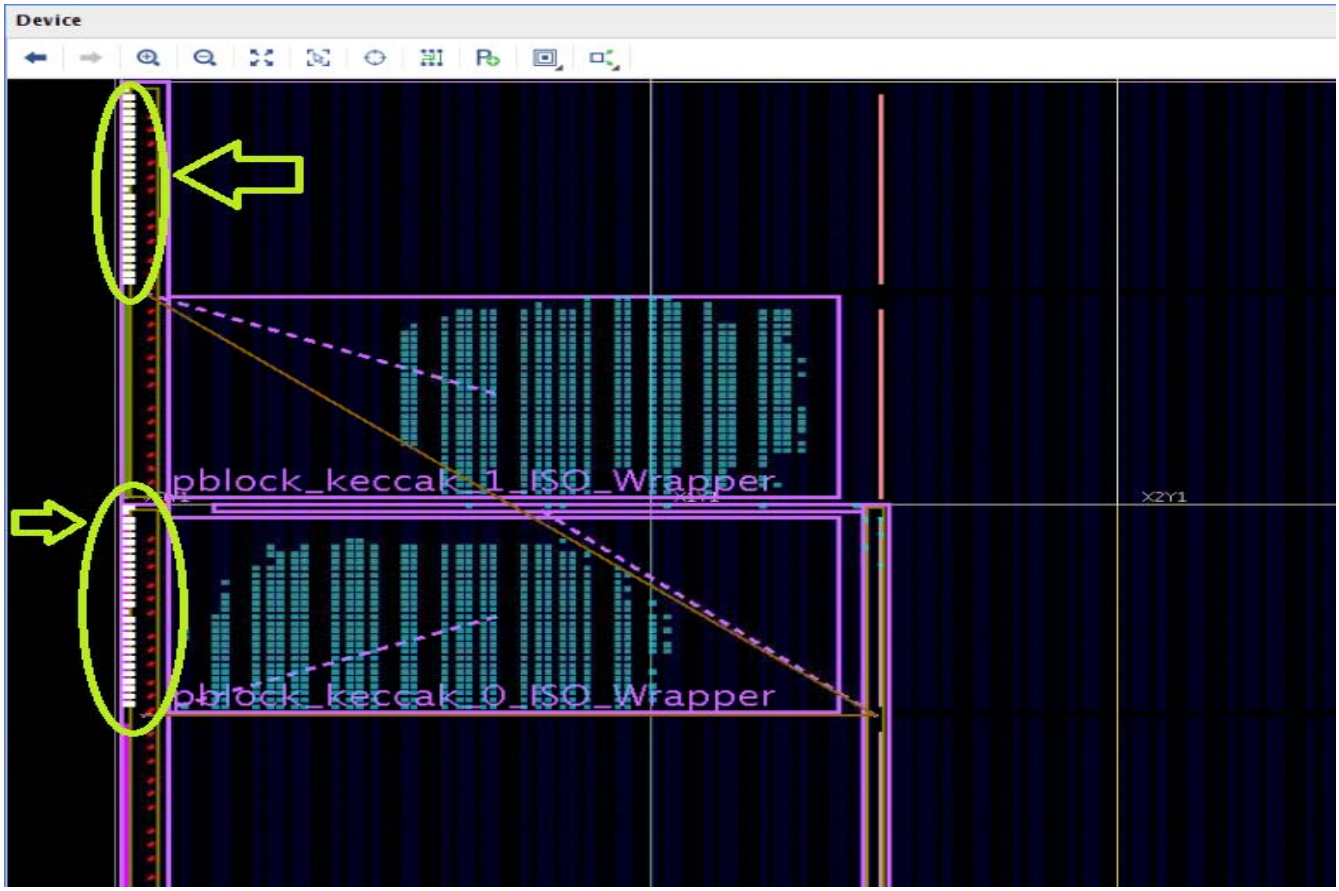

Figure 5: **Tile Adjacency Violations Highlighted in Device Window**

The text of the violation is displayed in two places in the GUI. The first line of the text is displayed in the DRC report table as shown in Figure 4. 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 6.

Figure 6: **Details Pane of the Violation Properties Window**

IMPORTANT: *In case of IDF violations with a large number of violating sites, by default all of those sites are not listed in the IDF violation text. The site list is truncated. You can enable VIV to list all the violating sites by setting drc.maxReportedNames to maximum value. Run the following command before running the VIV DRCs. This option is available in Vivado 2020.1 and later versions only:*
set_param drc.maxReportedNames 9999

IMPORTANT: *If DRC output is captured into a text file then the long violations will be displayed in multiple lines when above option is used. You can display the entire violation in a single line by setting maximum characters per line with the following parameter setting:*
set_param drc.maxReportedChars 9999

Application Notes

Table 2 lists the documentation available and upcoming through the Isolation Design Flow (IDF) website.

Table 2: **Isolation Design Flow Development Application Notes**

FPGA Family	Vivado Version	Application Note	Comments
7 series and Zynq-7000	Vivado 2015.2	XAPP1222	IDF Rules and Guidelines
7 series and Zynq-7000	Vivado 2015.2	XAPP1256	IDF Lab Tutorial App Note
UltraScale+	Vivado 2018.2	XAPP1335	IDF Rules and Guidelines for UltraScale+
UltraScale+	Vivado 2018.2	XAPP1336	IDF Lab Tutorial App Note for UltraScale+

Xilinx Resources

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

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

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

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

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

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

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

References

1. Isolation Design Flow website www.xilinx.com/idf
2. *Vivado Isolation Verifier User Guide (Tcl Based)* (UG1290)
3. *The Xilinx Isolation Design Flow for Fault-Tolerant Systems* (WP412)
4. *Vivado Design Suite Tcl Command Reference Guide* (UG835)

5. *Vivado Design Suite User Guide - Using Constraints* (UG903)
6. *Vivado Design Suite User Guide - Hierarchical Design* (UG905)
7. *Vivado Design Suite Tutorial - Hierarchical Design* (UG946)
8. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 SoCs (Vivado Tools)* (XAPP1222)
9. Zynq-7000 SoCs or 7 Series FPGAs Isolation Design Flow Lab (Vivado Design Suite) (XAPP1256)
10. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
11. *Isolation Design Flow for UltraScale+ Devices and the Zynq UltraScale+ MPSoC* (XAPP1335)
12. Isolation Design Example for the Zynq UltraScale+ MPSoC (XAPP1336)

Please Read: Important Legal Notices