Vivado Isolation Verifier (Tcl Based)

User Guide

UG1290 (v1.0) August 10, 2018

XILINX

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/10/2018	1.0	Initial Xilinx release.

Table of Contents

Vivado Isolation Verifier (Tcl Based)

Introduction

Proof of correctness is required for safety, 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 verify the user and Vivado® software have fulfilled the requirements insofar as the requirements can be automatically verified. VIV is a Tcl script (`viv.tcl`) that implements these DRCs. VIV is not included with Vivado. VIV is distributed separately as an encrypted Tcl file. Once loaded into Vivado, six additional DRCs appear under the Isolation category. The user invokes 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.

In addition to offering proof of isolation, 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. The intent of these design constraint checks is to spare the designer from costly printed circuit board redesigns.

VIV is developed separately from Vivado in an effort to achieve as much independence from Vivado development as practical. While it is true that VIV depends on Vivado for script execution, device models, and access to the design to be checked, VIV does not re-use Vivado code involved in logic placement or routing. VIV also parses XDC (Xilinx Design Constraint) information [Ref 4] [Ref 5] [Ref 6] instead of relying on the results of constraint interpretation available through the Vivado Tcl API.

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

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

Definitions

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

area range – a 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 – a bitstream is a contiguous sequence of bits, representing a stream of data.

FPGA bitstream – an FPGA bitstream is a file that contains the programming information for an FPGA. A Xilinx FPGA device must be programmed using a specific bitstream to enable 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. In production hardware, the bitstream is usually placed in non-volatile memory, and the hardware is configured to program the FPGA when powered on.

device model – the 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 show the feature to be unreliable), in practice this is not done because it would make it impossible to perform the tests that would validate or invalidate the feature.

fence tile – An un-programmed tile or site free of routing or logic used to separate two or more tiles containing logic or routing from distinct isolation groups.

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

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

inter-region signal – a non-isolated net with one source and one load typically used to connect one isolated function to another, though sometimes used to connect one port of an

isolated function to another port of the same isolated function or to top-level logic. An inter-region signal is not permitted to use routing resources containing programmable interconnect points (PIPs) in fence tiles.

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 free of isolated logic and routing between isolated portions of the design.

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

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

I/O buffer, IOB – a 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, etc.

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

logic - circuits used to implement a specific function, for example a flip-flop, look up table, random access memory.

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

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

package pin – a conductor on the outside of an FPGA package used for powering or interfacing with the FPGA, shaped like a short wire protruding perpendicularly from the chip package or shaped like a bump.

partition – a collection of logic defined by the user that can be used to isolate one piece of hierarchy from another.

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

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

site – a 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 – routing that connects isolated functions using routing resources with no programmable interconnection points within fence tiles. Trusted Routing is generated automatically without manual placement.

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

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

FPGA Architecture

A field programmable gate array (FPGA) is a chip that contains logic elements and routing both of which 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), each containing a small amount of programmable logic and memory
- Input/output block circuitry (IOBs)
- Clock Management Tiles (CMTs)
- Other specialized circuitry, including block RAMs, digital signal processors (DSPs), processors, etc.

A typical Xilinx FPGA might 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.

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.

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. First, the design must be manually floorplanned. Second, constraints must be applied to isolated regions of the floorplan to constraint routing to follow strict rules. Finally, VIV must be used to demonstrate 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 points:

- During floorplanning (right side blue boxes), VIV helps to avoid costly circuit board layout mistakes and helps document the floorplan, a key part of the isolation approach of the design.
- When the design is complete (lower right green boxes), VIV is used to help prove that the design is isolated per IDF rules.

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, and
- Color is used only for grouping and emphasis

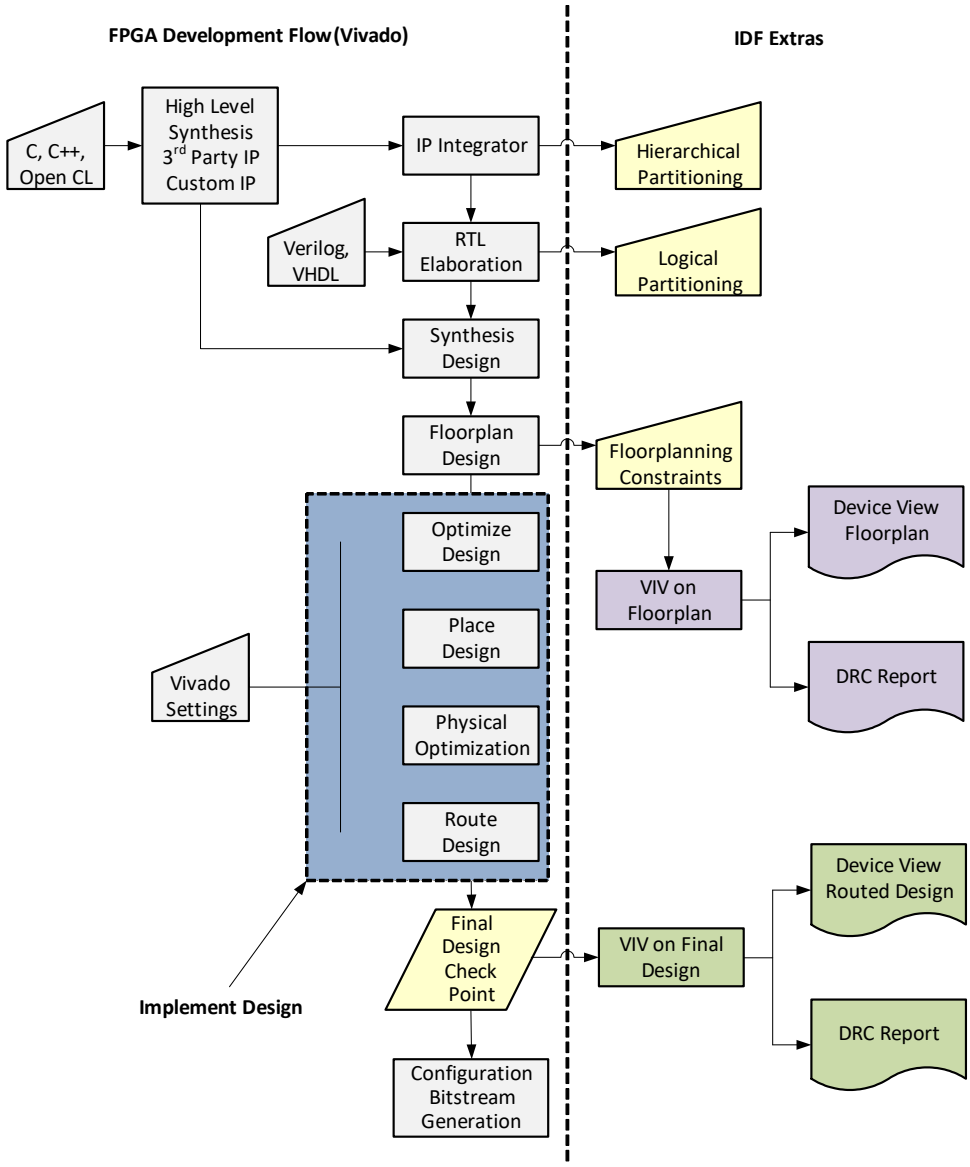

Figure 1: **Vivado Isolation Design Flow Relative to a Typical FPGA Design Flow**

IDF Design Rule Checks

VIV is a Tcl script that runs in the Vivado framework in the form of DRCs. This allows VIV to take advantage of the Vivado IDE for the display of violations in the context of the design as it is normally seen. The VIV DRCs are described in detail in the following sections.

IDF-1 - Provenance

IDF-1 is an advisory DRC documenting the circumstances of the run. It also validates that the design has at least two partitions 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-1 output.

Here is an example of IDF-1 output:

```
Vivado Isolation Verifier (v1.72)
Copyright (C) 2013-2016 Xilinx, Inc. All rights reserved.
Date: Mon May 16 20:30:32 GMT 2016
Project: project_xapp1256_routed
Design: checkpoint_xapp1256_routed
Top-level: design_1_wrapper
Isolated Partitions: pblock_keccakCompare_0 pblock_keccak_0_ISO_Wrapper
pblock_keccak_1_ISO_Wrapper pblock_processing_system7_0
Part: xc7z020clg484-1
Directory: c:/xilinx_design/implementation/idflab
User: username
Vivado Version: 2016.3.0
Platform: win64
Host: hostname

Global clock 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,
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK2,
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK_unbuffered[0],
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK_unbuffered[1] and
design_1_i/ps7_ISO_Wrapper/processing_system7_0/inst/FCLK_CLK_unbuffered[2].

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,
⋮
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.
```

IDF-2 - I/O Bank Violation

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

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

IDF-3 - Package Pin Violation

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

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

IDF-4 - Floorplan Violation

Isolated regions must be separated by an adequate fence. The definition of an adequate fence is detailed in the Isolation Design Flow documentation [Ref 1] [Ref 7]. IDF-4 checks that floorplan area ranges from distinct isolation groups have an appropriate gap between them. IDF-4 checks XDC information, not the actual implementation.

The following example shows all of the constraints, resident 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_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 {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}
```

IDF-5 - Placement Violation

In contrast to IDF-4 which checks XDC information, IDF-5 checks placement of logic as actually implemented. Two checks are performed. First, a search for adjacent logic from distinct isolation groups is performed. In this context logic is considered adjacent if the separation between the logic tiles is not composed of a valid set of fence tiles. Second, a check is performed that top-level logic does not contain a potential path from one isolation group to another.

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

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

Installation

VIV is not supplied with Vivado. VIV is installed by loading `viv.tcl`. This can be accomplished in one of several ways.

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

  `source viv.tcl`

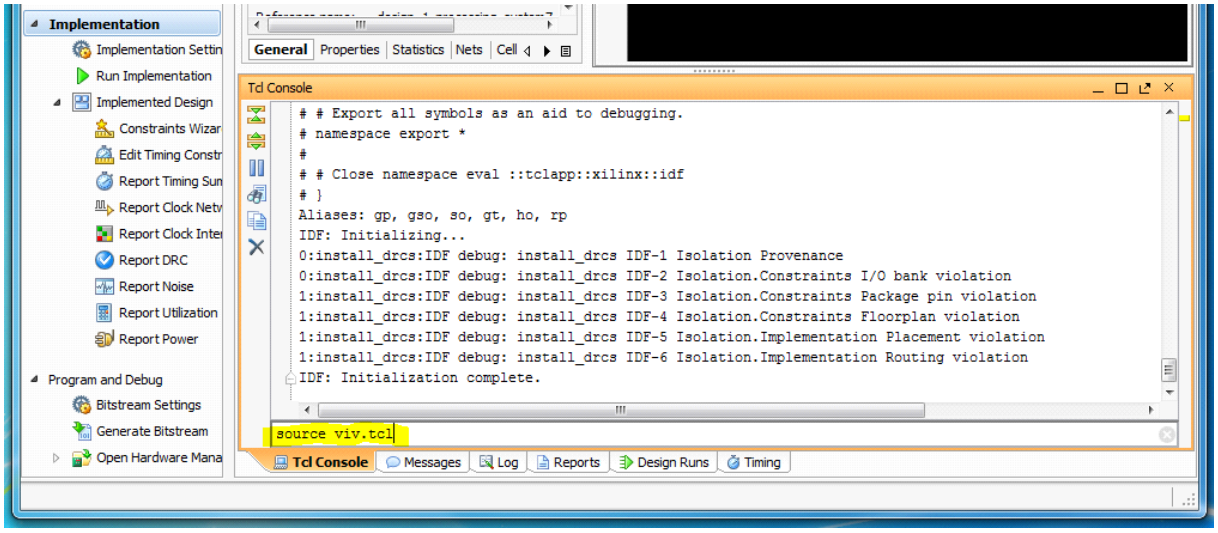

Figure 2: **IDF DRCs Selected in the Report DRC Dialog Box**

- Go to menu item **Tools > Run Tcl Script** from the Vivado GUI and navigate to `viv.tcl`.
- Install `viv.tcl` as a startup script as described in the Vivado documentation [Ref 3].
- Include `viv.tcl` on the Vivado command line [Ref 3].

Usage

After `viv.tcl` is loaded, the Report DRC window will contain the six additional IDF DRCs under the category heading *Isolation* as shown in Figure 3. Note that the *Pblock* option under *Isolation* is not related to IDF and need not be checked.

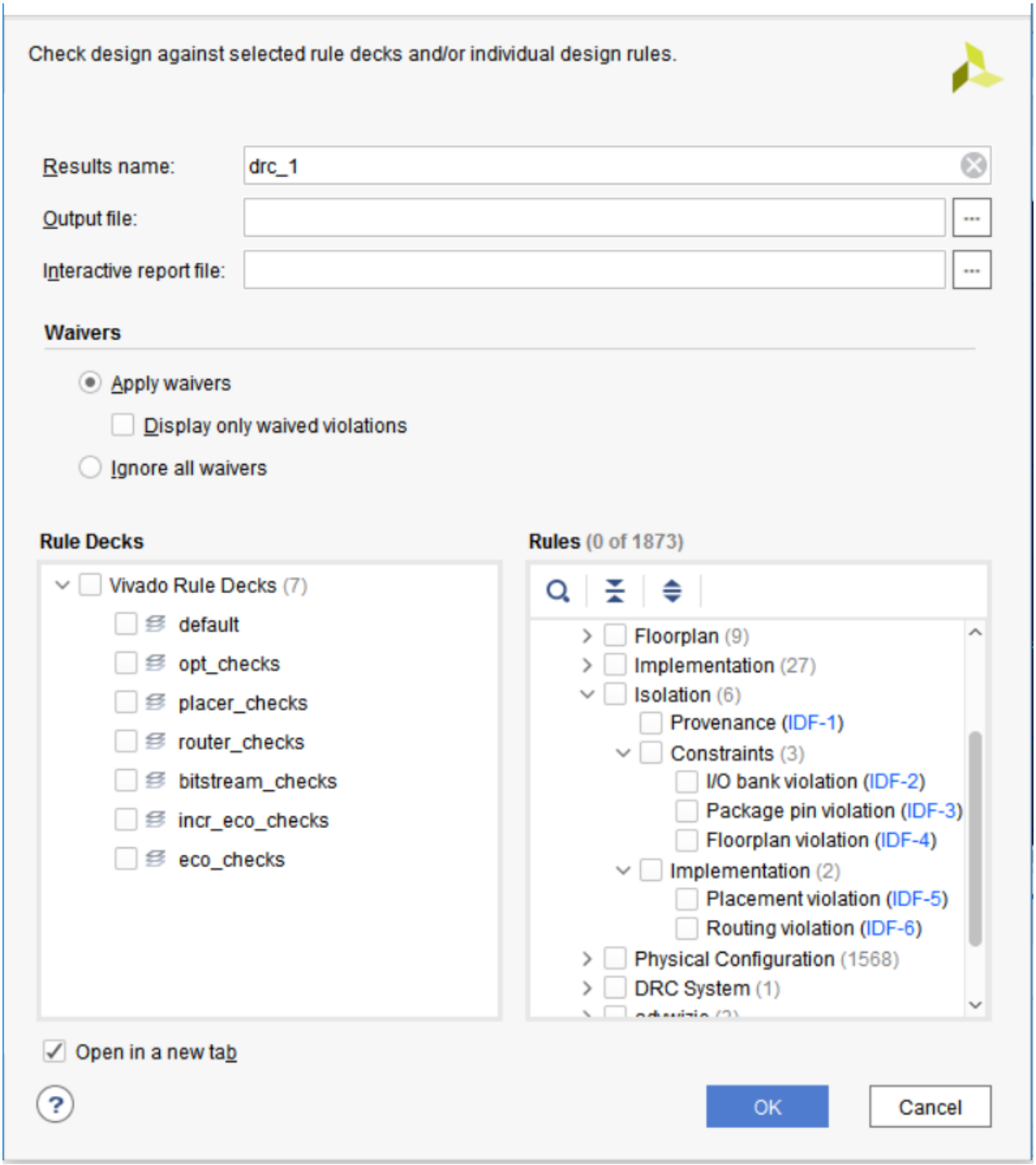

Figure 3: **IDF DRCs Selected in the Report DRC Dialog Box**

Invoking VIV DRCs

The user invokes 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** window.s

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-2 and IDF-3 can be run after synthesis has been performed.
- IDF-5 and IDF-6 examine the implementation, thus the Implementation step *must* be completed for IDF-5 and IDF-6 to have something to check.

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

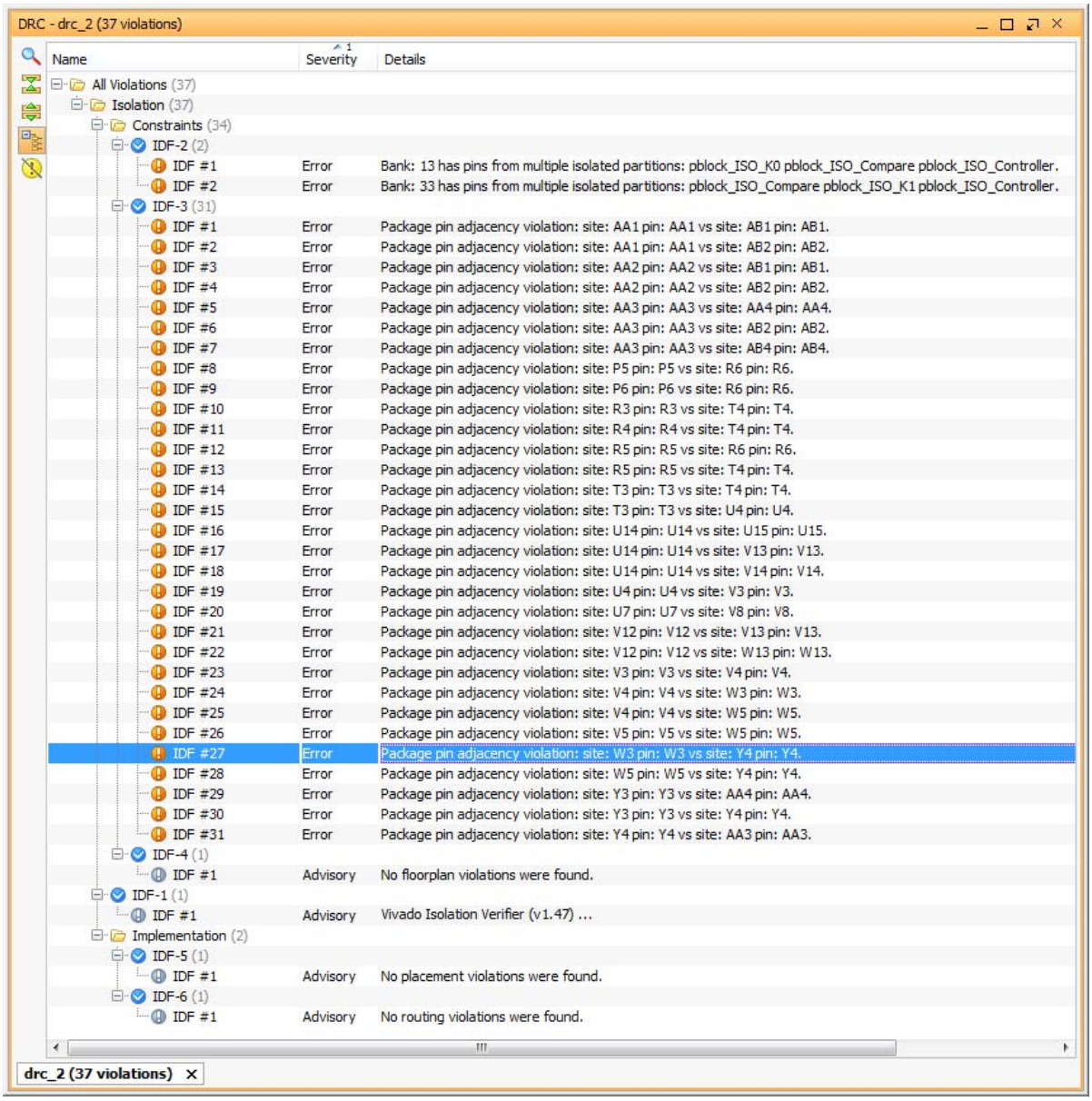

Figure 4: **IDF DRC Violations from Example Design**

Note that the highlighted DRC (IDF #27) 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 and the corresponding package pins are selected in the **Package** window as shown in Figure 6.

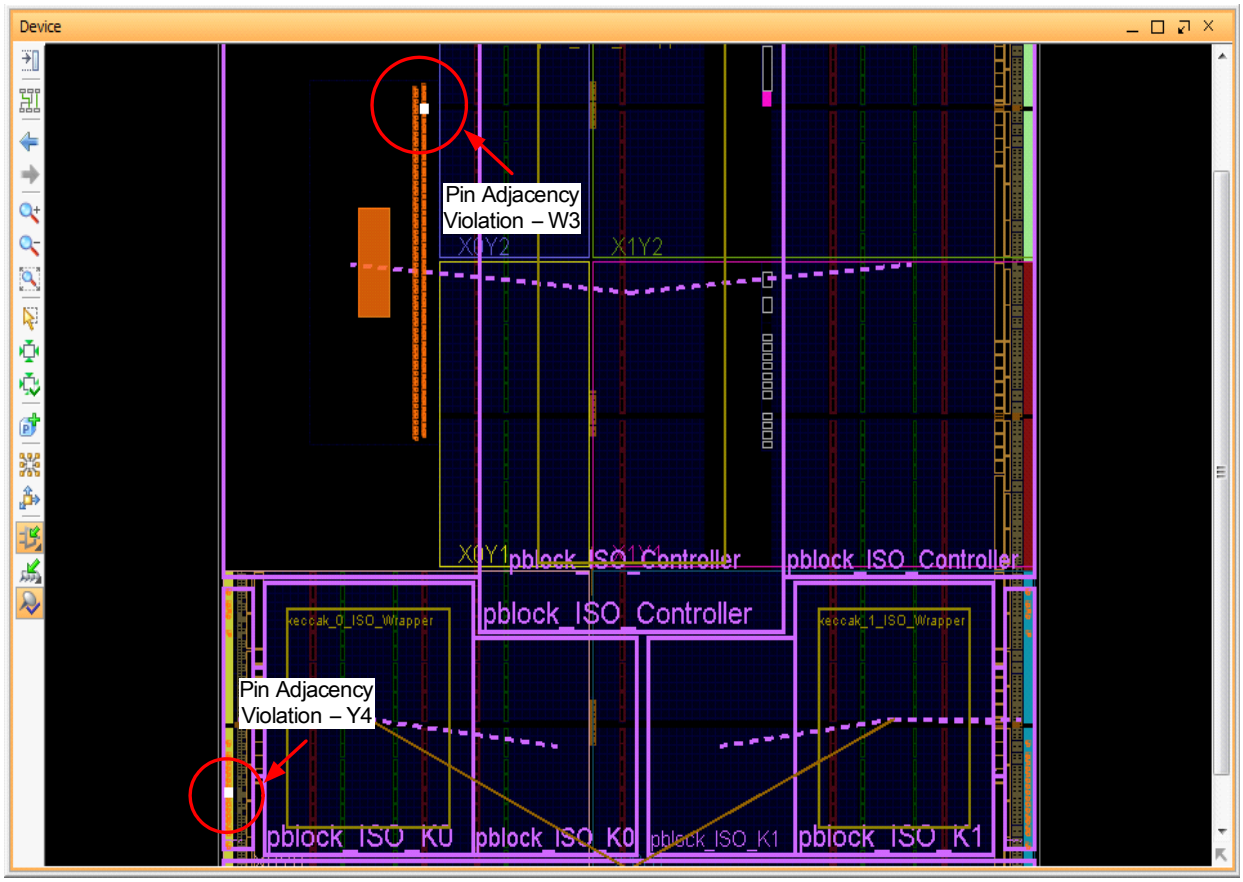

Figure 5: **Pin Adjacency Violations with I/O Buffers Highlighted in Device Window**

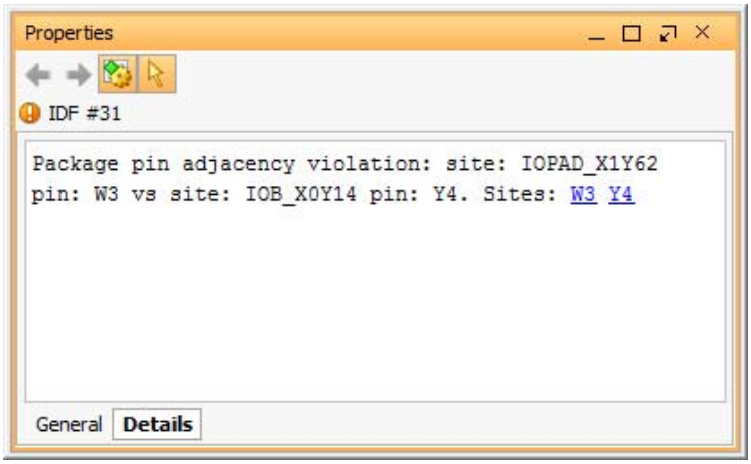

Figure 6: **Pin Adjacency Violations Highlighted in Package Window**

VIV Design Stages

The Vivado Isolation Verifier (VIV) verifies that an FPGA design partitioned into isolated regions meets applicable safety or security requirements. 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 framework in the form of 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 (IOBs) from different isolation groups are not physically adjacent on the die.
- Pins from different isolation groups are not physically adjacent on the package. Pins are considered adjacent if they share an edge or corner with no fence tile in between.
- Pins from different isolation regions are not co-located in an I/O bank.

  Note: Although 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.
- The pblock constraints in the XDC file are defined such that a minimum of a one tile-wide fence exists between isolated regions.

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 is used to verify that the required isolation was achieved in the design. VIV checks the placement and routing as follows:

- Isolated logic tiles must be separated by fence tiles. A fence tile cannot contain any logic and cannot contain any routing that could lead to an isolation violation with a single fault.
- Routing does not depend on programmable connections in fence tiles.

Application Notes

Table 1 lists the documentation available at the Isolation Design Flow (IDF) website [Ref 1].

Table 1: **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

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 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. Isolation Design Flow website www.xilinx.com/idf
2. *The Isolation Design Flow for Fault-Tolerant Systems* (WP412)
3. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
4. *Vivado Design Suite User Guide - Using Constraints* (UG903)
5. *Vivado Design Suite User Guide - Hierarchical Design* (UG905)
6. *Vivado Design Suite Tutorial - Hierarchical Design* (UG946)
7. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)* (XAPP1222)
8. *Zynq-7000 AP SoC Isolation Design Flow Lab (Vivado Design Suite 2015.2)* (XAPP1256)
9. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)

Please Read: Important Legal Notices