



XAPP1086 (v1.3.1) February 5, 2015

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

Author: Ed Hallett

## Summary

This application note is written for FPGA designers wishing to implement security or safety critical designs, that is, information assurance (single chip cryptography), avionics, automotive, and industrial applications, using the Xilinx Isolation Design Flow (IDF). This document explains how to:

- Implement isolated functions in a single Xilinx 7 series FPGA or Zynq®-7000 All Programmable SoC (AP SoC)<sup>(1)</sup> in commercial-grade or defense-grade using IDF.
  - For example, implementation might include red/black logic, redundant Type-I encryption modules, or logic processing multiple levels of security.
- Verify the isolation using the Xilinx Isolation Verification Tool (IVT).

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-grade, high-assurance applications. If the user wishes to add additional security to their design, the Security Monitor IP, developed by Xilinx, can be purchased. Should the user decide to 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.

The Kintex®-7 FPGA and Zynq-7000 AP SoC families are currently supported for the IDF. For more information about the support time line for the rest of the Xilinx 7 series FPGA/SoC families, contact your Xilinx representative for currently supported 7 series devices. This application note is accessible from the Isolation Design Flow page on Xilinx.com [Ref 2].

You can download the [Reference Design Files](#) for this application note from the Xilinx website. For detailed information about the design files, see [Reference Design Files](#).

---

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

## Introduction

The flexibility of programmable logic affords the security and safety critical industries many advantages. However, prior to this work, 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. To address these concerns, the Isolation Design Flow 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, resident red and black data, and functionality operating on multiple independent levels of security. The successful completion of this work 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 partition technology. Special attributes such as SCC\_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 standard partition 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 partition are used extensively.

They are defined thus:

- 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 (that is, an AES encryptor)
- Logic
  - Circuits used to implement a specific function (that is, flip-flop, look up table, RAM, and so on)
- Region/Area Group/Pblock
  - A physical area to implement logic

- Partition
  - A user-defined logical construct that can be used to isolate one piece of hierarchy from another

## Isolation Design Flow

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 partition based (that is, each function a user desires to isolate must be at its own level of hierarchy). From here the designer can take one of two approaches. If the designer wishes to ensure unwanted optimization of redundancy does not occur, each isolated function must be synthesized and implemented independently of the other partitions. After each partition is implemented the design is merged into a flattened FPGA design for device configuration. If the designer wishes to use other techniques to prevent such optimization, they can synthesize the full design while being careful to maintain at least one level of hierarchy such that IDF constraints can be applied to each partition that requires isolation. While this flow requires the FPGA designer to break away from traditional FPGA development flows, the partition approach does have certain advantages. If an isolated partition requires a change late in the design cycle, only that specific function is modified while the remaining partitions remain unchanged.

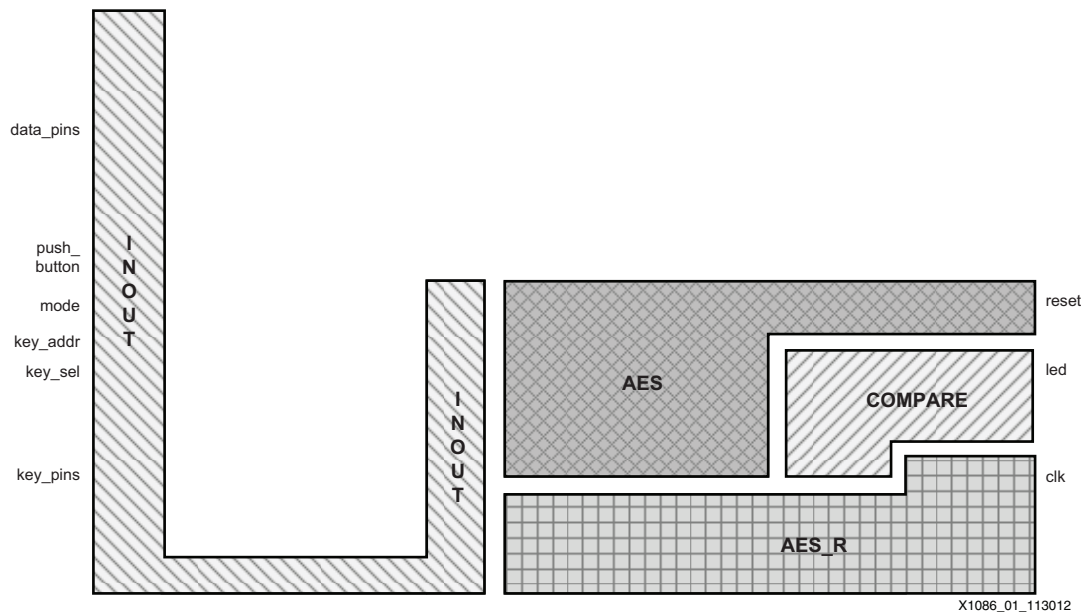
**Note:** All logic should belong to an isolated partition except for global clocks, resets, and IOBs.

There are a few unique design details that must be adhered to in order to achieve an FPGA based IDF solution. The designer should carefully consider all aspects of these design details, which are explained in detail in subsequent sections of this application note. These considerations include:

- Each isolated function must be in its own partition
- Each partition must consist of a single module instantiation
- A fence must be used to separate isolated partitions within a single chip
- IOBs must be inferred or instantiated inside isolated partitions for proper isolation of the IOB
- On-chip communication between isolated functions is achieved through the use of trusted routing

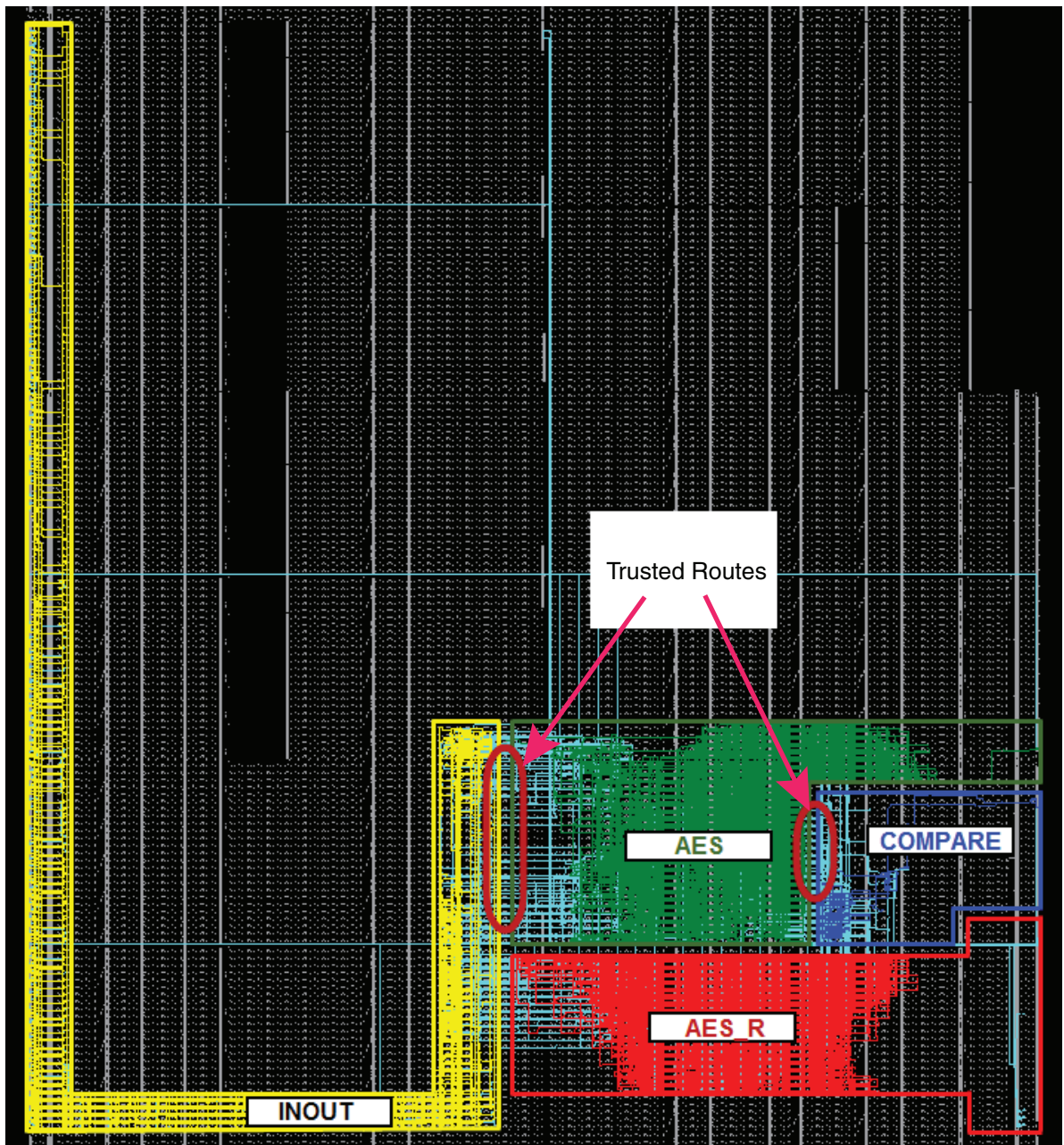
## Reference Design

For clarity, an example Single Chip Cryptographic (SCC) design (Figure 1) is used throughout this application note to describe the design details and tool flow. In addition, this design has been implemented with the Xilinx ISE® tools, verified by the Isolation Verification Tool (IVT), and provided to the designer as a reference. For more details, see [Reference Design Files](#), page 55.



*Figure 1: SCC Solution Using IDF Implementing Two Redundant AES Modules and Compare Logic*

The example design consists of two redundant advanced-encryption-standard (AES) encryption modules, whose outputs are sent to a comparator (COMPARE) block, and an I/O (INOUT) module used to buffer and isolate the data and key inputs. The redundant AES encryption modules, compare functions, and I/O (INOUT) module are all isolated within a single FPGA as shown in the FPGA Editor view in [Figure 2](#).



X1086\_02\_092513

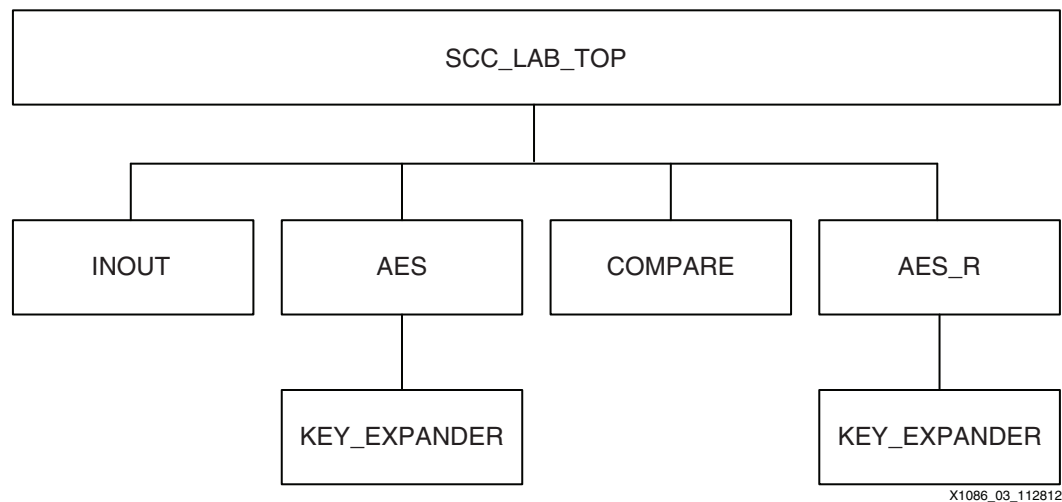
Figure 2: Implemented SCC Solution from Figure 1

For simplicity, the AES modules are 128-bit engines running in codebook mode with a static plain-text word. The design can inject errors through a pushbutton located on one of the AES engines. An LED, driven by the comparison block, indicates when the outputs of the AES modules do not match. 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. Table 1 lists the ports and IOBs of each module.

**Table 1: Reference Design IOB for Each Isolated Module Including the Top Level**

Module	Module Inputs	Module Outputs
TOP	clk_p/clk_n, reset, push_button, mode, key_pins, data_pins, key_sel, key_addr	led
AES	clk, reset, push_button, start, mode, load, key, data_in	data_out, done, reset_out, reset_iob
AES_R	clk, reset, push_button, start, mode, load, key, data_in	data_out, done
COMPARE	clk, reset, done1, done2, data_in1, data_in2	reset_out, start_aes1, start_aes2, load_aes1, load_aes2, led
INOUT	clk, reset, push_button, mode, key_sel, key_addr, data_pins, key_pins	push_button_aes, push_button_aes_r, key_out1, key_out2, data_out1, data_out2, mode_aes, mode_aes_r

Per isolation design flow details, the reference design implementation is performed using a traditional (in-context) synthesis flow, using the PlanAhead™ tool, which is the flow used for IDF designs to maintain isolation. The PlanAhead tool, version 14.4 or later, allows the user to import RTL source code directly into the project so the project can be done using just the PlanAhead tool. [Figure 3](#) shows the hierarchy of the reference design.



**Figure 3: Hierarchy for the SCC Implementation of the Reference Design**

Additional lower level hierarchy is allowed for each function. In addition, 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. As a companion to this reference design, *7 Series Isolation Design Flow Lab Using ISE Design Suite 14.4* (XAPP1085) [\[Ref 3\]](#) details step by step instructions on generating a similar reference design.

## Isolation Partitions

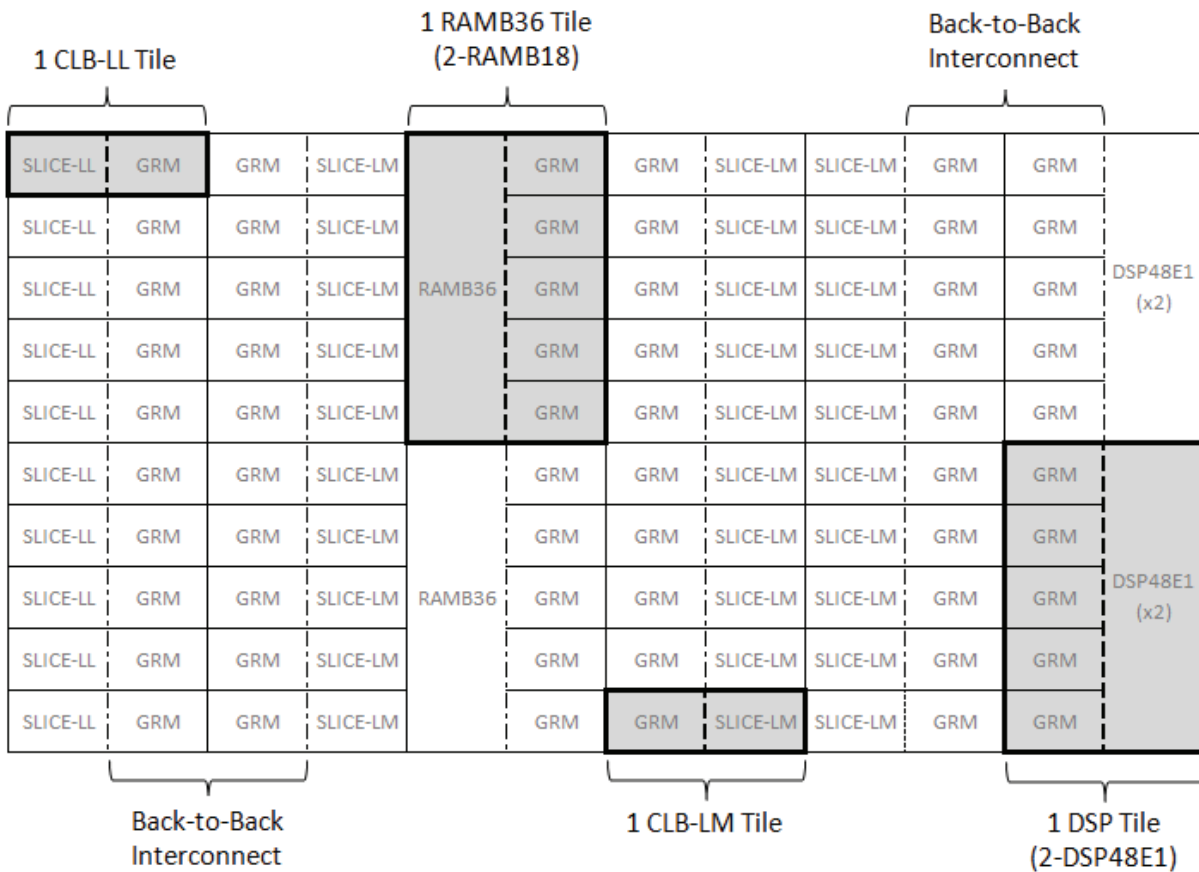
Each of the four separate functions in the hierarchy, below the top-level, is put into isolated partitions. IDF rules have many additional constraint rules as opposed to a traditional Partial

Reconfiguration (PR) or Hierarchical Design flow. It is necessary to define one key attribute for each partition to invoke these rules.

The **SCC\_ISOLATED** attribute needs to set for each partition in order for the backend implementation tools to use the special Isolated Partition rules, which allow for floorplanning of an isolated design when using the IDF. If the **SCC\_ISOLATED** attribute is not set, a user can only floorplan a limited subset of the FPGA components (CLB slices, RAMB, and DSP48E).

## Isolation Fence

The complexity and density of the 7 series FPGAs might seem to preclude any efficient isolation methodology. In addition, the difficulty of performing an isolation analysis is assumed to be magnified considering there are many device types in each of the Xilinx 7 series FPGA/SoC families (Kintex-7, Virtex®-7, and Artix®-7 FPGAs, and the Zynq-7000 AP SoC). However, Xilinx FPGAs are modular tile based devices. No matter which device is chosen, the FPGA 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 (LM and LL), block RAM, DSP, and so on. 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 4](#).



X1086\_04\_112812

Figure 4: 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 routing or 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 5](#).

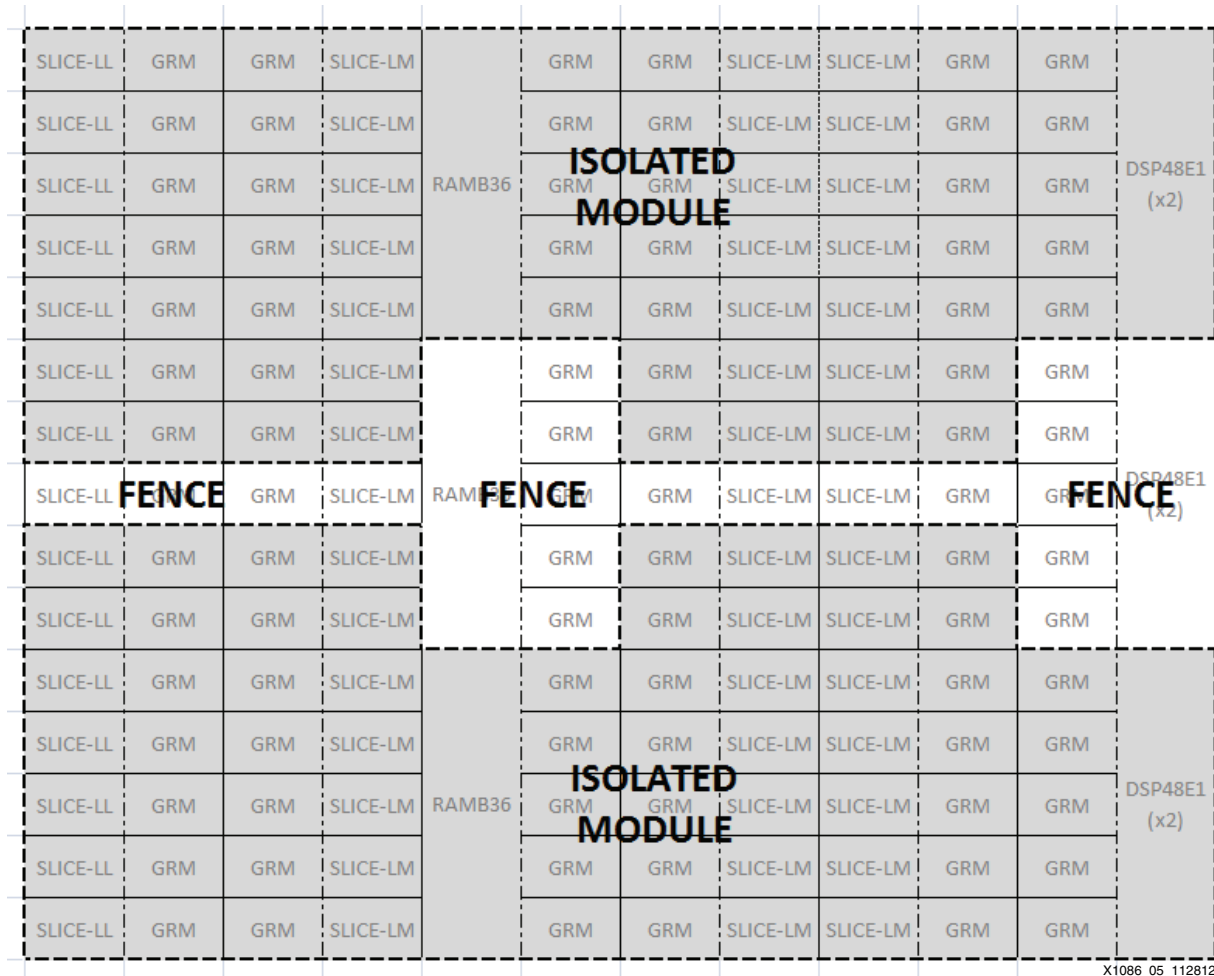


Figure 5: Isolated Functions Separated by a Fence of Unused Logic Tiles

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

## Floorplanning the Reference Design

The floorplan of the reference design is shown in [Figure 6](#). Where inter-module communication is necessary, regions must be coincident with each other with a fence tile between the two intended regions. See [On-Chip Communication, page 12](#) for more details.



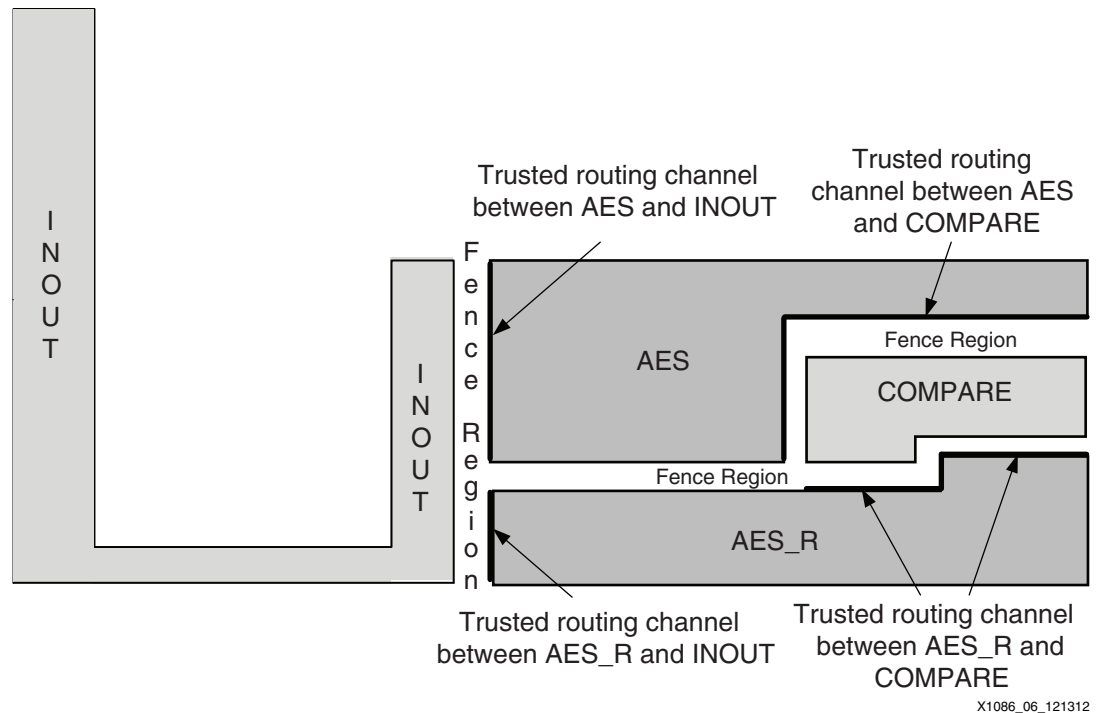


Figure 6: Floorplan of the Reference Design Highlighting the Trusted Routing Channels

To constrain the logic to a specific region of the FPGA, the AREA\_GROUP constraint is utilized. This constraint is applied to an instance in the HDL representing the function to be isolated. In this example, each of the AES encryption modules and comparator functions has an AREA\_GROUP constraint applied to it. For more information on the AREA\_GROUP constraint and the FPGA coordinate system, see the *Constraints Guide* (UG625) [Ref 4]. The first step in creating the AREA\_GROUP constraint is to analyze the region the function is to be mapped into to ensure that enough dedicated resources exist. The dedicated resources can be IOB, FFs, LUTs, block RAM, DSP48E slices, and so on. The PlanAhead tool, which relates AREA\_GROUPS with physical blocks (PBLOCKS), can be used to assist the designer in this analysis.

The next step is to assign the HDL instance name of the module to a user-specified AREA\_GROUP name. In the SCC reference design, the HDL module instance names are U1\_AES1, U2\_AES2, U3\_Comp, and U4\_IOB. The assignment of the instance name to the AREA\_GROUP for the reference design is:

```
"INST "U1_AES1" AREA_GROUP = "pblock_U1_AES1";
"INST "U2_AES2" AREA_GROUP = "pblock_U2_AES2";
"INST "U3_Comp" AREA_GROUP = "pblock_U3_Comp";
```

Next, AREA\_GROUPS must be assigned to a specific range of logic in the FPGA. AREA\_GROUPS 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. The generic syntax for adding components to an area group is:

```
AREA_GROUP "<AG Name>" RANGE=<comp name>_XaYb:<comp name>_XcYd;
```

Where:

- <AG Name> = name of area group
- <comp name> = name of desired component, for example:
  - SLICE
  - RAMB18
  - DSP48
  - IOB
  - ILOGIC
  - OLOGIC
  - PLL
  - MMCM

**Note:** This is not a complete set of components.

The component name can be identified by pointing at it in the PlanAhead tool and reading it off the bottom right hand of the screen.

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

Coordinates can also be identified from the same location in the bottom right-hand corner of the PlanAhead tool that the component name was identified. Some examples of this syntax are as follows:

- AREA\_GROUP "pblock\_U1\_AES1" RANGE=SLICE\_X122Y97:SLICE\_X153Y104;
- AREA\_GROUP "pblock\_U1\_AES1" RANGE=DSP48\_X5Y40:DSP48\_X5Y41;
- AREA\_GROUP "pblock\_U1\_AES1" RANGE=RAMB18\_X5Y40:RAMB18\_X6Y41;
- AREA\_GROUP "pblock\_U1\_AES1" RANGE=ILOGIC\_X1Y97:ILOGIC\_X1Y104;
- AREA\_GROUP "pblock\_U1\_AES1" RANGE=OLOGIC\_X1Y97:OLOGIC\_X1Y104;

Although the reference design does not use DSPs or block RAMs, they have been added to the area group so that the routing resources contained by these blocks can be used. As a general rule, all available resources should be assigned to the area group unless there is specific need to exclude that resource. This is the default selection when generating the constraint file using the PlanAhead tool. An exception to this rule are the CMT components (MMCME2\_ADV, PLLE2\_ADV, PHASER\_IN\_PHY, PHASER\_OUT\_PHY, PHY\_CONTROL, IN\_FIFO, and OUT\_FIFO), which should only be added to an area group if the specific site is used in the design.

## Communicating with Isolated Functions

Communication between isolated functions in the FPGA and other components within a system occurs through user IOBs. However, to maintain isolation from logic to the IOBs, and ultimately the PCB, the IOBs must be considered part of the isolation function and contained in the isolated region associated with that function.

Communication between two isolated functions within the FPGA typically occurs internally using trusted routing. A less preferable alternative is to use IOBs to go off-chip and then back on again into a different region. Each of these alternatives has specific constraints that must be considered before choosing how to communicate with isolated functions.

## Off-Chip Communication

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 partition, then there is no control of how the signal is routed from the IOB to the logic. Standard design practice is to define all of the FPGA IOBs in the top-level port map (for VHDL users). During synthesis, all of the IOBs are inferred at the highest level of the target hierarchy. However, in the Isolation Design Flow, IOB insertion is specifically controlled and is typically in the lower level partitions. For the Isolation Design Flow, the only signals that should have IOB buffers inferred at the top level are global clocks, resets, and other control signals that do not require isolation. All other signals that go off-chip should have their IOBs inferred or directly instantiated at the lower level modules.

In the flow used in previous SCC/IDF application notes, where each of the isolated functions were synthesized independently (*out-of-context*), the I/O buffers were inferred by default for each of the I/O pins in that module unless specifically inhibited, through the **buffer\_type is none** attribute.

The PlanAhead tool, version 14.4 or later, gives the option to use an RTL Source Import Flow (hierarchical or in-context). This flow is used in SCC reference design so the I/O Buffers are specifically instantiated in each isolated function at the HDL level, as needed. The global clock input buffers are instantiated in the top level block (`scc_lab_top.vhdl`).

The *Add I/O Buffers (-iobuf)* synthesis option enables or disables I/O buffer insertion. The option is left at the default, which is yes, so I/O buffers are automatically inserted during synthesis, as long as they are not already instantiated in the isolated function. XST does not place two I/O buffers on an I/O pin.

The following is an example snippet of VHDL code, for the IOB instantiation, for an input buffer and an output buffer.

```
-- Instantiate Input Buffer for PUSH_BUTTON
  PushButton_IBUF : IBUF
  port map(
    O => push_button_inbuf,
    I => push_button
  );
-- Instantiate Output Buffer for LED
  Led_OBUF : OBUF
  port map(
    O => led,
    I => led_outbuf
  );
```

## On-Chip Communication

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, page 26](#).
- 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.

## Constraining Placement

Placement constraint is now achieved by the addition of components to the area group. Examples of this syntax with respect to the AES1 isolated function are listed below. A full listing of this syntax can be found in *Constraints Guide* (UG625) [\[Ref 4\]](#).

```
AREA_GROUP "pblock_U1_AES1" RANGE=SLICE_X122Y97:SLICE_X153Y104;
AREA_GROUP "pblock_U1_AES1" RANGE=DSP48_X5Y40:DSP48_X5Y41;
AREA_GROUP "pblock_U1_AES1" RANGE=RAMB18_X5Y40:RAMB18_X6Y41;
```

Components and resources not in an area group cannot be used. Because this includes top-level clock components such as MMCMs, an additional constraint is necessary. The constraint PRIVATE = NONE is a directive to the top-level function that allows top-level routes to use components owned by each area group. In the included reference design, the PRIVATE constraint is set to NONE for each isolated function.

**Note:** NONE is the default setting for the PRIVATE constraint. Even if not specifically set, this is the presumed value.

Because PRIVATE = NONE is the default setting, it does not need to be specifically set and it does not show up in the user constraints file (UCF). If the constraint were to be set, it would have the following example syntax in the UCF:

```
AREA_GROUP "pblock_U1_AES1" PRIVATE = NONE;
```

The INCLUSIVE attribute needs to be set to ALL for each partition or Physical block (Pblock) to allow the backend implementation tools to include all the resources within the Area Range. The syntax for this constraint in the UCF is:

```
AREA_GROUP "pblock_U1_AES1" INCLUSIVE=ALL;
```

## Constraining Routing

Routing control is no longer controlled using constraints. Rather, routing is controlled using physical ownership. No resource can be controlled if it is not physically owned (included in an area group statement). Global logic (MMCM, PLL, BUFG, and so on) can use any resource as long as it is physically owned. Local logic (logic and routing of an isolated function) can only use resources physically owned by its own isolated region.

## Constraint Summary

Some initial architecting and floorplanning of the FPGA, coupled with a small list of constraints, are all that is required to achieve isolation of specific functions within a single 7 series FPGA or SoC. The definition of whether a function is reconfigurable or simply isolated is no longer contained in the user constraint file (UCF). It is determined in the PlanAhead tool by setting the module as a partition and then attaching the SCC\_ISOLATED attribute to it.

All constraints applied to the compare function are shown here:

```
INST "U3_Comp" AREA_GROUP = "pblock_U3_Comp";
AREA_GROUP "pblock_U3_Comp" RANGE=SLICE_X146Y80:SLICE_X153Y95;
AREA_GROUP "pblock_U3_Comp" RANGE=DSP48_X5Y30:DSP48_X5Y37;
AREA_GROUP "pblock_U3_Comp" RANGE=IDELAY_X1Y81:IDELAY_X1Y94;
AREA_GROUP "pblock_U3_Comp" RANGE=ILOGIC_X1Y81:ILOGIC_X1Y94;
AREA_GROUP "pblock_U3_Comp" RANGE=IOB_X1Y81:IOB_X1Y94;
AREA_GROUP "pblock_U3_Comp" RANGE=ODELAY_X1Y81:ODELAY_X1Y94;
AREA_GROUP "pblock_U3_Comp" RANGE=OLOGIC_X1Y81:OLOGIC_X1Y94;
AREA_GROUP "pblock_U3_Comp" RANGE=RAMB18_X6Y32:RAMB18_X6Y37;
AREA_GROUP "pblock_U3_Comp" RANGE=RAMB36_X6Y16:RAMB36_X6Y18;
AREA_GROUP "pblock_U3_Comp" INCLUSIVE=ALL;
```

**Note:** Even though the constraint PRIVATE = NONE is set for this isolated partition, it does not show up in the UCF. The attribute INCLUSIVE is set to ALL and it does show up in the UCF.

---

## Isolation Verification Tool

The Isolation Verification Tool (IVT) verifies that an FPGA design partitioned into isolated regions meets stringent standards for fail-safe design. IVT is used at two stages in the FPGA design process. Early in the design flow, IVT, in UCF mode, is used to perform a series of design rule checks on floorplans and pin assignments. The goal of the UCF checking is to identify potential isolation problems before commitment to board layout.

After the design is complete (placed and routed), IVT, in NCD (Native Circuit Description) mode, is used again to validate that the required isolation is built into the design. IVT is a command line tool, running under Windows® 2000, Windows XP, Windows 7, and Linux operating systems. The directory containing the Xilinx executables and dynamic link libraries must be listed in the PATH environment variable for IVT to locate them.

### Constraint Checking (IVT - UCF Mode)

IVT, in UCF mode, 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 groups are not co-located in an IOB bank.

- While IVT 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 AREA\_RANGE constraints in the UCF are defined such that a minimum of a one tile-wide fence exists between isolated regions.

Because the UCF file might not specify the target device or package, they must be specified through the command line. The association between IVT isolation groups and user-defined area groups is also specified on the command line. The ability for the user to associate an IVT isolation group to the user-defined area groups allows for more than one area group to belong to a single IVT isolation group. An example of IVT execution is:

```
ivt -f -verbose parameter_file_name.ivt
```

Available command-line arguments for IVT operation on the UCF are shown in [Table 2](#).

**Table 2: IVT Command-Line Arguments for UCF Mode**

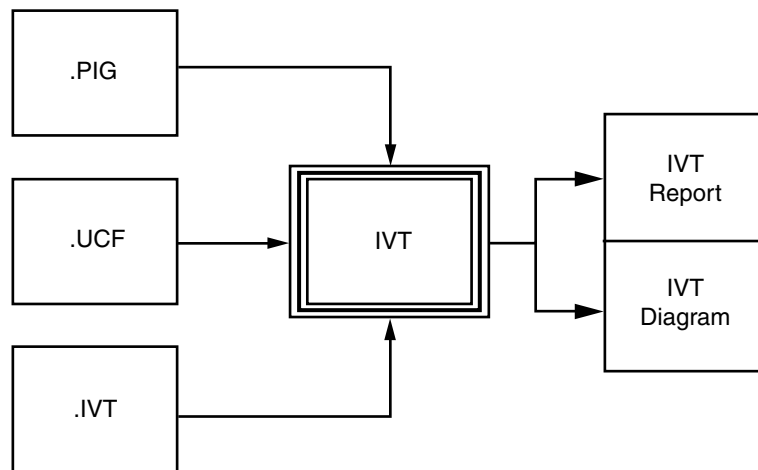
IVT Command Line Argument	Description
-device <i>device_name</i>	Specifies the device.
-package <i>package_name</i>	Specifies the package.
-group <i>isolation_group</i> <i>area_group</i>	Associates the area group name to an arbitrary isolation group name. At least two distinct isolation groups are required.
[-pig <i>pin_isolation_groups</i> [.pig]]	If omitted, no pin-related analysis is performed.
[-output <i>output</i> [.rpt]]	Name of the output file.
[-verbose]	Writes out all IVT reporting information.
[-f <i>argument_file</i> ]	Command-line arguments are stored in an external file.
<i>constraint_file</i> [.ucf]	Constraint file for the design (UCF).

**Notes:**

1. Optional arguments are in brackets "[ ]". Italics denote variables.

### IVT UCF Input Files

IVT, in UCF mode, requires two input files but accepts three, with the third one being a parameter file to reduce the length of the command line. From these inputs, IVT outputs two unique reports. The text report is the official output for use in an evaluation while the second, a graphical output, is useful for quick location of any isolation faults. [Figure 7](#) represents a block diagram of the inputs and outputs of IVT while in UCF mode.



X1086\_07\_121212

Figure 7: IVT Inputs and Outputs in UCF Mode

### User Constraint File (.UCF)

The UCF is the floorplan of the user design.

### ITV Parameter File (.IVT)

The command-line arguments for the IVT are somewhat long, especially for checking UCFs. Therefore, it is recommended that a file containing the arguments be created and supplied to IVT rather than typing them directly at the command line. Arguments can be spread across multiple lines, with blank lines and lines beginning with # ignored (comments). The UCF mode parameter file for the reference design is shown here.

```
-device xc7k325t -package ffg900

# Groups  Isolation Group  Area Group
# -----
-group    AES             pblock_U1_AES1
-group    AES_r           pblock_U2_AES2
-group    COMPARE         pblock_U3_Comp
-group    INOUT           pblock_U4_IOB

# Pin Isolation Groups
-pig scc_lab_top.pig

# User Constraint File
..\planahead\FloorPlan_SCC\scc_lab_top.ucf

# Output file
-output scc_lab_top_ucf.rpt
```

### Pin Isolation Group File (.PIG)

Designs containing red and black logic and routing can require isolation at the device inputs and outputs. In this scenario, a pin isolation group (PIG) file is required. The PIG file specifies pins belonging to each isolation group, allowing IVT to perform design rule checks. The PIG file format is simple and allows an FPGA designer to cut and paste the I/O location constraints

directly from the UCF into the PIG text file file extension (.pig). An isolation group is defined with this syntax:

```
ISOLATION_GROUP group_name BEGIN
.
. #Cut and paste IOB location
. #constaints from UCF here
.
END ISOLATION_GROUP
```

**Note:** Only the net names are significant. All other syntax within the IOB location constraints is ignored.

Here is an example PIG file from the SCC reference design. Only some of the INOUT Isolation Group NETs are shown. This example PIG file is for illustrative and format purposes only. If creating a new PIG file, the user should copy each NET statement directly from the UCF.

```
# Place all Global (top level) signals here (each commented out)
# NET "clk_p"          LOC = AD12;
# NET "clk_n"          LOC = AD11;

ISOLATION_GROUP AES BEGIN
NET "reset"           LOC = AK4;
END ISOLATION_GROUP

ISOLATION_GROUP AES_r BEGIN
# There are no pins in AES_r
END ISOLATION_GROUP

ISOLATION_GROUP COMPARE BEGIN
NET "led"             LOC = AB9;
END ISOLATION_GROUP

ISOLATION_GROUP INOUT BEGIN
NET "push_button"    LOC = P27;
NET "mode"           LOC = Y25;
NET "key_sel"        LOC = Y28;
NET "key_addr[2]"    LOC = Y29;

...

NET "data_pins[56]"  LOC = R23;
NET "data_pins[58]" LOC = T20;
END ISOLATION_GROUP
```

**Note:** The NET syntax is a copy/paste from the actual UCF. The clock net is commented out because it is a top-level net that is not associated with any specific isolated region. The other IOB declarations are embedded in their appropriate ISOLATION\_GROUP in the UCF.

## Final Isolation Verification (IVT - NCD Mode)

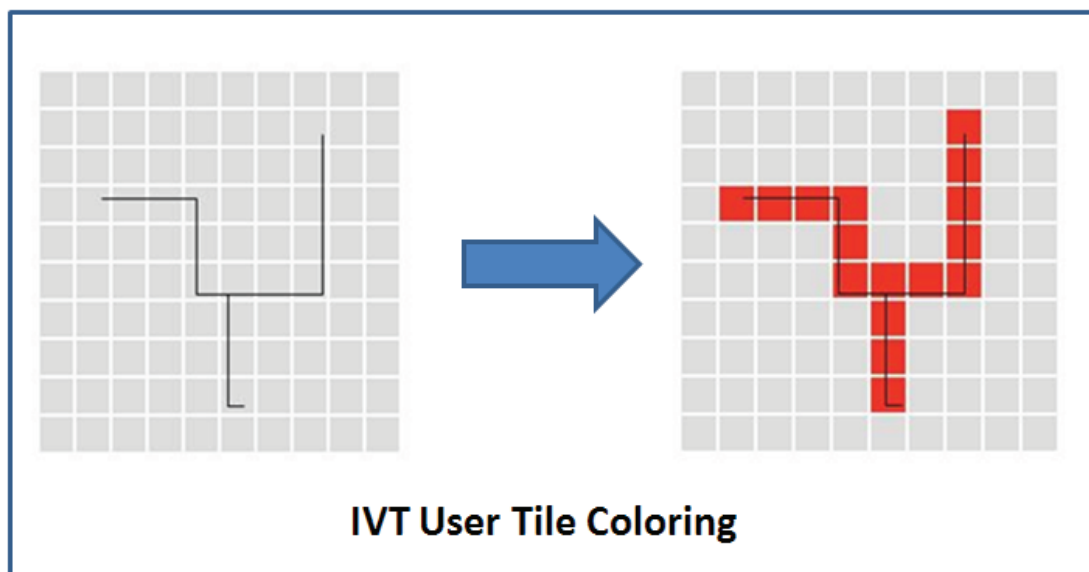
IVT, in NCD mode, checks the following:

- Analyzes the complete Placed and Routed design.
- Tile-based isolation analysis is looking for a barrier (Fence) between Isolated Regions.



- If analysis shows that a user tile:
  - Does not contain any isolated signals (from any isolated region),
  - And, the user tile is configured in the default (unused) state,
- Then, it acts as a sufficient isolation barrier.
- Also, it does the same Pin and I/O checking as in UCF mode.

IVT, in NCD mode, uses tile-based isolation analysis. IVT iterates through all the networks in the design coloring each user tile (Figure 8) according to the isolation region of the networks that pass through it. For most tiles, exactly one isolation region is represented. Trusted Bus routes, clocks, and other global signals produces multi-colored tiles, which needs to be evaluated to assure they were intended.



X1086\_08\_112812

Figure 8: IVT Tile-Based Analysis

To run IVT (NCD mode) with the parameter file, use the *-f* option:

```
ivt -f parameter_file_name.ivt
```

Available command-line arguments for IVT operation on the user design file (NCD) are shown in Table 3.

Table 3: IVT Command-Line Arguments for NCD Mode

IVT Command Line Argument	Description
<i>-group isolation_group user_instance_name</i>	Associates an arbitrary group name to an isolated instance in the user routed design. At least two distinct isolation groups are required.
<i>-out output[.rpt]</i>	Name of the output file.
<i>-f parameter_file</i>	Command line arguments stored in an external file.

Table 3: IVT Command-Line Arguments for NCD Mode (Cont'd)

IVT Command Line Argument	Description
<i>user_design</i> [.ncd]	Fully routed user design file (NCD).

**Notes:**

- Optional arguments are in brackets "[ ]". Italics denote variables.

**IVT NCD Input Files**

IVT, in NCD mode, requires one input file but accepts two, with the second one being a parameter file to reduce the length of the command line. From these inputs, IVT outputs two unique reports. The text report is the official output for use in an evaluation while the second, a graphical output, is useful for quick location of any isolation faults. Figure 9 represents a block diagram of the inputs and outputs of IVT while in NCD mode.

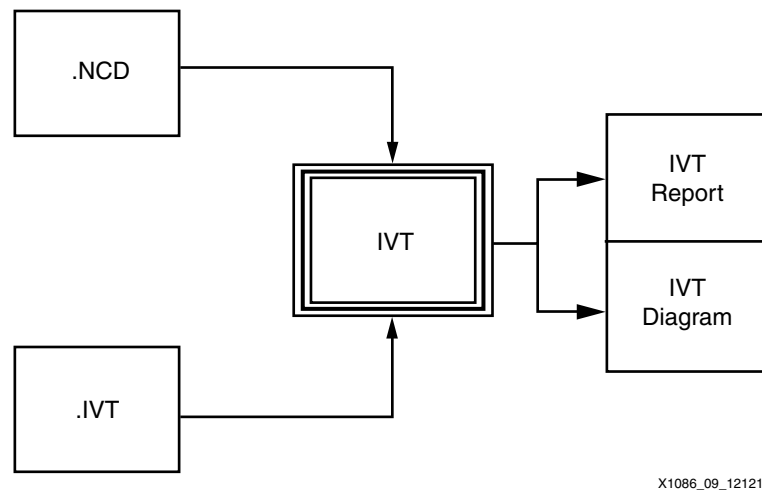


Figure 9: IVT Inputs and Outputs in NCD Mode

**User Netlist file (.NCD)**

The user design netlist, or NCD file, contains the complete placement and routing information, including the names of blocks, pins, and nets. There is no need to specify the device, package, or mapping from the I/O pins to area groups to IVT, because it is contained in the NCD file.

**IVT Parameter file (.IVT)**

The command-line arguments for IVT are somewhat long, especially for checking UCFs. Therefore, it is recommended that a file containing the arguments be created and supplied to IVT rather than typing them directly at the command line. Arguments can be spread across multiple lines, with blank lines and lines beginning with # ignored (comments). The NCD mode parameter file for the reference design is shown here. # comment the next line for reduced detail in the report file.

```

-verbose

# Groups      Isolation Group      Instance Name in Final NCD
#-----
-group      AES                    U1_AES1
-group      AES_r                  U2_AES2
-group      COMPARE                U3_Comp
-group      INOUT                  U4_IOB

# Combined design
scc_lab_top_routed.ncd

# Output Report File
-output scc_lab_top_ncd.rpt

```

## IVT Output

IVT has two outputs: The first, a text file, is a detailed report that is to be used in any official evaluation. Faults and other items of concern are described in this report. The second, a graphical file, is useful as a high-level overview of the user design and is helpful in quick identification of isolation faults. The content of these reports is dependent on the mode in which IVT was called (UCF or NCD).

### *UCF Mode*

IVT, in UCF mode, verifies the user floorplan. Specifically, IVT ensures that user area groups have the required fence width between them and that the IOBs are properly isolated at both the die and package level.

### Text Report

A typical UCF mode report contains the following sections:

- **Provenance:** Includes the date, IVT version, ISE tool version used for the run, ISE tool version against which IVT was compiled, the location of the ISE tool installation, the command line, the current working directory, the output report location, and the part/package.
- **Area range constraints:** For each isolation group, lists the corresponding area groups and associated site ranges used to define the floorplan of the design.
- **Package pins, I/O buffers, and I/O banks (fault):** The pin assignments of the design are presented with coordinates, I/O banks, isolation groups, and the net names.
- **I/O buffer isolation violations (fault):** Lists pairs of I/O buffers from distinct isolation groups that are adjacent on the die.
- **Package pin isolation violations (fault):** Lists pairs of pins from distinct isolation groups that are members of a single I/O bank.
- **I/O bank violations (fault):** Lists examples of pins from distinct isolation groups that are adjacent on the package.
- **Isolation verification summary:** Lists the total number of constraints violated, reports completion, and reports the elapsed time to perform the analysis and generate the report.

See the IVT user guide provided with the reference design for more detailed descriptions and examples.

### Graphical Report

For ease of use, IVT outputs a Silicon Vector Graphics (SVG) file. This generic output allows for quickly identifying where faults are. While the text report is all-encompassing, the SVG file gives a high-level view of any faults and is useful in debugging the user floorplan. [Figure 10](#) shows a typical example of the SVG output as run on the Kintex-7 reference design.

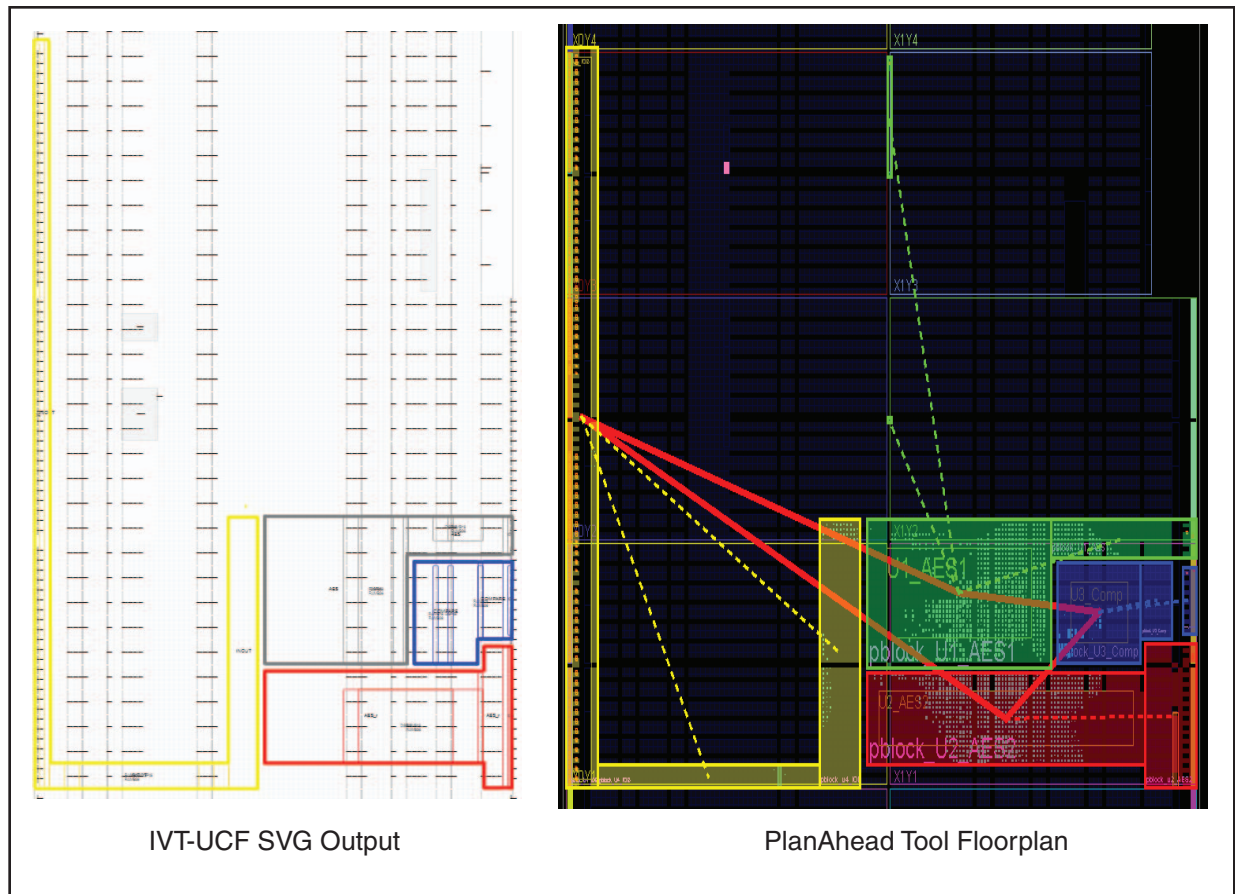


Figure 10: SVG UCF Output Run (Left) Next to the Actual PlanAhead Tool Floorplan (Right)

Figure 11 shows a typical example of the SVG output as run on a Zynq-7000 SoC design.

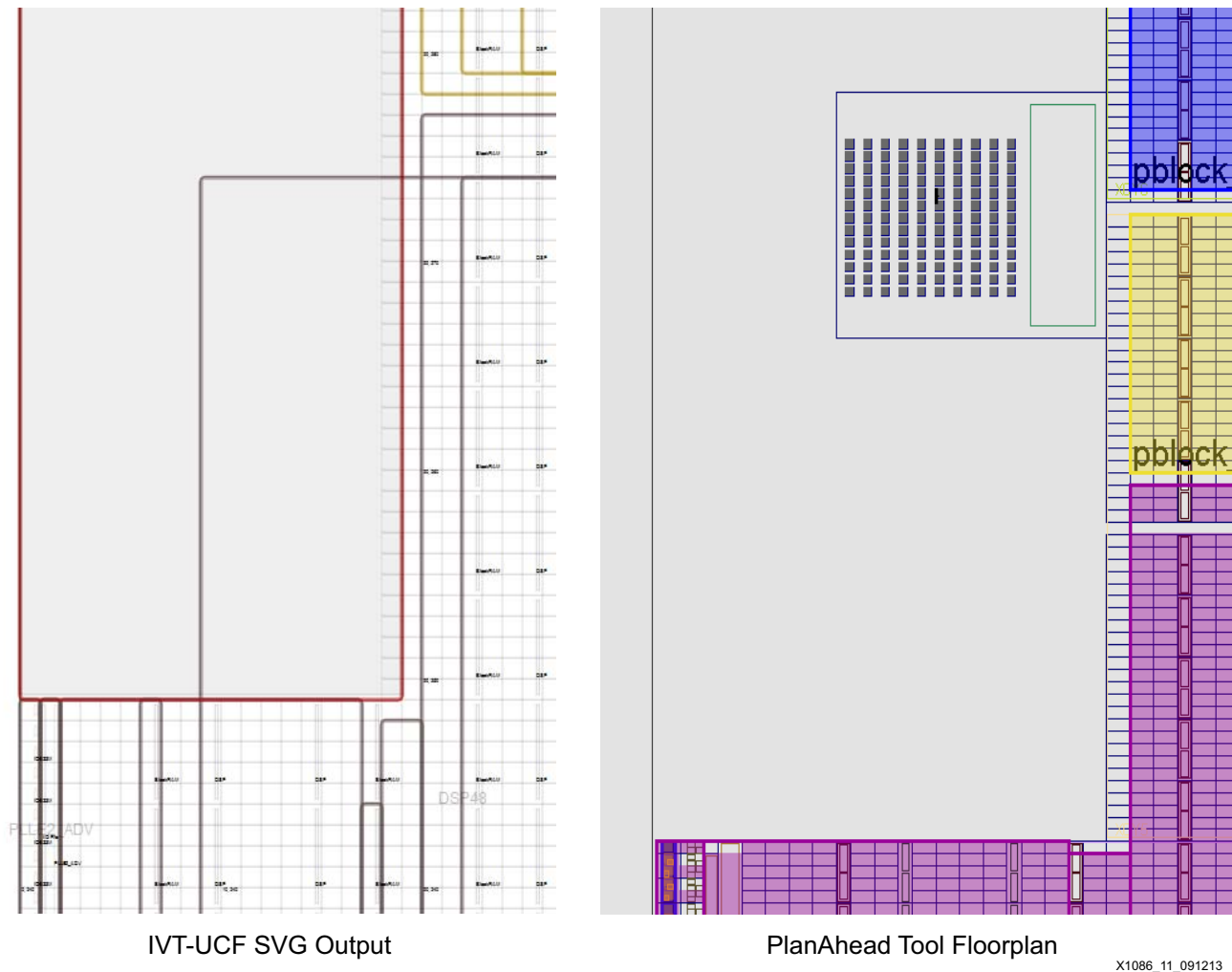


Figure 11: Zynq-7000 SVG UCF Output Run (Left) Next to the Actual PlanAhead Tool Floorplan (Right)

## NCD Mode

### Text Report

A typical NCD mode report contains these sections:

- **Provenance:** Includes the date, IVT version, ISE tool version used for the run, ISE tool version against which IVT was compiled, location of the ISE tool installation, command line, current working directory, output report location, part/package, and the name of the combined design.
- **Isolated Modules (Informational):** Lists the isolation groups and associated design blocks or partial NCD files.
- **Uncategorized User Global Nets (Informational):** Lists nets (signals) in the design that are above the isolated modules in the design hierarchy and might connect isolated functions. All such nets must be examined for their impact on the data separation and

independence requirements of the system. Ideally, the only nets listed in this section would be nets specifically intended to connect isolated functions. In practice, some global clock signals appear here as well. In the report, uncategorized global nets are said to be “found in multiple isolation groups.” This is an artifact of the original implementation of IVT in which multiple partial NCD files were used to specify isolation groups. Global resources are duplicated in all the partial NCD files.

- **Categorized Nets (Informational):** Has many possible subsections corresponding to various categories of nets that for one reason or another are benign with respect to isolation. Examples include constants, global clocks, trusted bus macros, and clocking inserted automatically to mitigate negative-bias temperature instabilities.
- **Trusted Bus Macros (Informational):** Lists all the instances of Trusted Bus Macros in the design and the nets connected to them.
- **Area Range Constraints (Informational):** For each isolation group, lists the corresponding area groups and associated site ranges used to define the floorplan of the design.
- **Tiles with Net Content Violations (Fault):** Lists the contents of all tiles that contain isolated logic or routing from more than one isolation group.
- **Tiles in the Fence Containing Nets (Fault):** Lists the contents of tiles that are outside all the isolated area ranges. This list is advisory. It is permissible for inter-region signals and clocking to exist in the fence. Akin to the list of Uncategorized Global Nets, nets in tiles that are outside of all isolation groups must be vetted for their impact on data separation and independence requirements. This section provides low-level details of the specific nodes and wire segments used to implement routing in the fence.
- **Tiles in the Fence Containing Programming (Fault):** Lists tiles in the fence that are associated with components and therefore are not entirely unused. All such tiles must be examined for their impact on the data separation and independence requirements of the system.
- **Tiles in the Fence Containing Used PIPs (Fault):** Lists tiles containing Programmable Interconnect Points (PIPs); nodes that have the potential to connect to other nodes. For example, suppose a certain type of node spans five horizontal tiles called A, B, C, D, and E. Further suppose that this node has PIPs in tiles A, C, and E. It is allowed to use this node to connect two isolated regions provided the only tiles in the fence are B or D
- **Tiles with Net Adjacency Violations (Fault):** Lists all pairs of tiles that are adjacent and contain isolated logic or routing resources from distinct isolation groups, in other words, pairs of tiles that should be separated by a fence tile.
- **Package Pins, I/O Buffers, and I/O banks (Fault):** The pin assignments of the design are presented with coordinates, I/O banks, isolation groups, and the net names.
- **I/O Buffer Isolation Violations (Fault):** Lists pairs of I/O buffers from distinct isolation groups that are adjacent on the die.
- **Package Pin Isolation Violations (Fault):** Lists pairs of pins from distinct isolation groups that are adjacent on the package.

- **I/O Bank Violations (Fault):** Lists examples of pins from distinct isolation groups that are members of a single I/O bank.
- **Isolation Verification Summary:** Lists the total number of constraints violated by category, reports completion, and reports the elapsed time to perform the analysis and generate the report. For each section of the report containing violations, there is a line in the summary with a tally of the violations in that section.

A more detailed description and examples can be found in the IVT user guide.

### Graphical Report

For ease of use, IVT outputs an SVG (Silicon Vector Graphics) file. This generic output allows for quickly identifying where faults are located. While the text report is all encompassing, the SVG file gives a high-level view of any faults and is useful in debugging the user floorplan. [Figure 12](#) shows a typical example of the NCD SVG output as run on the SCC reference design, alongside of the FPGA Editor view of the same design.

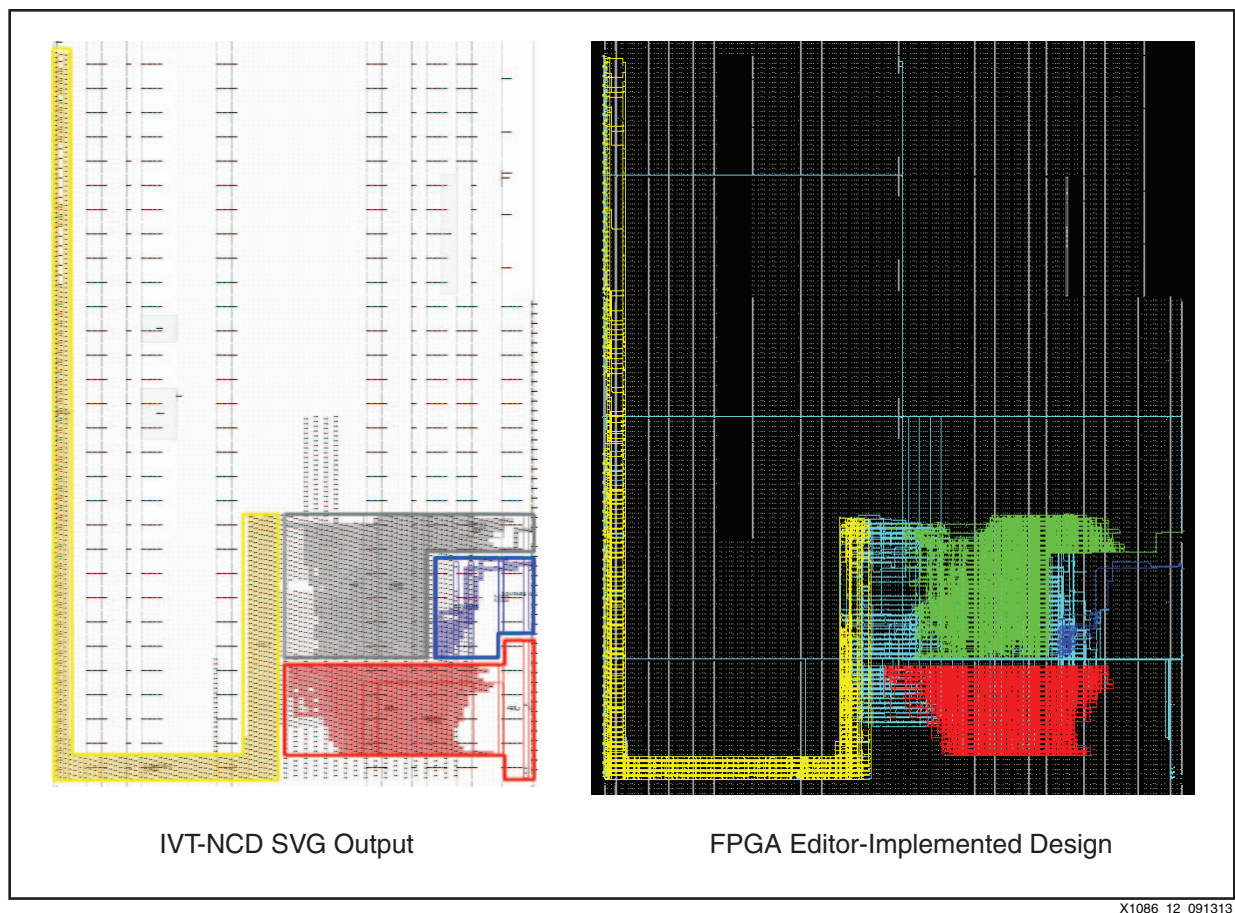


Figure 12: IVT NCD SVG Run (Left) Next to the FPGA Editor View (Right)

# Design Guidance

## Rules for the Isolation Design Flow

The Isolation Design Flow (IDF) is a software methodology that 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 following guidance represents one particular application using the IDF, in this case, Single Chip Cryptography (SCC). The IDF is not that dissimilar from a traditional design flow. [Figure 13](#) shows a flow diagram of both flows side by side.

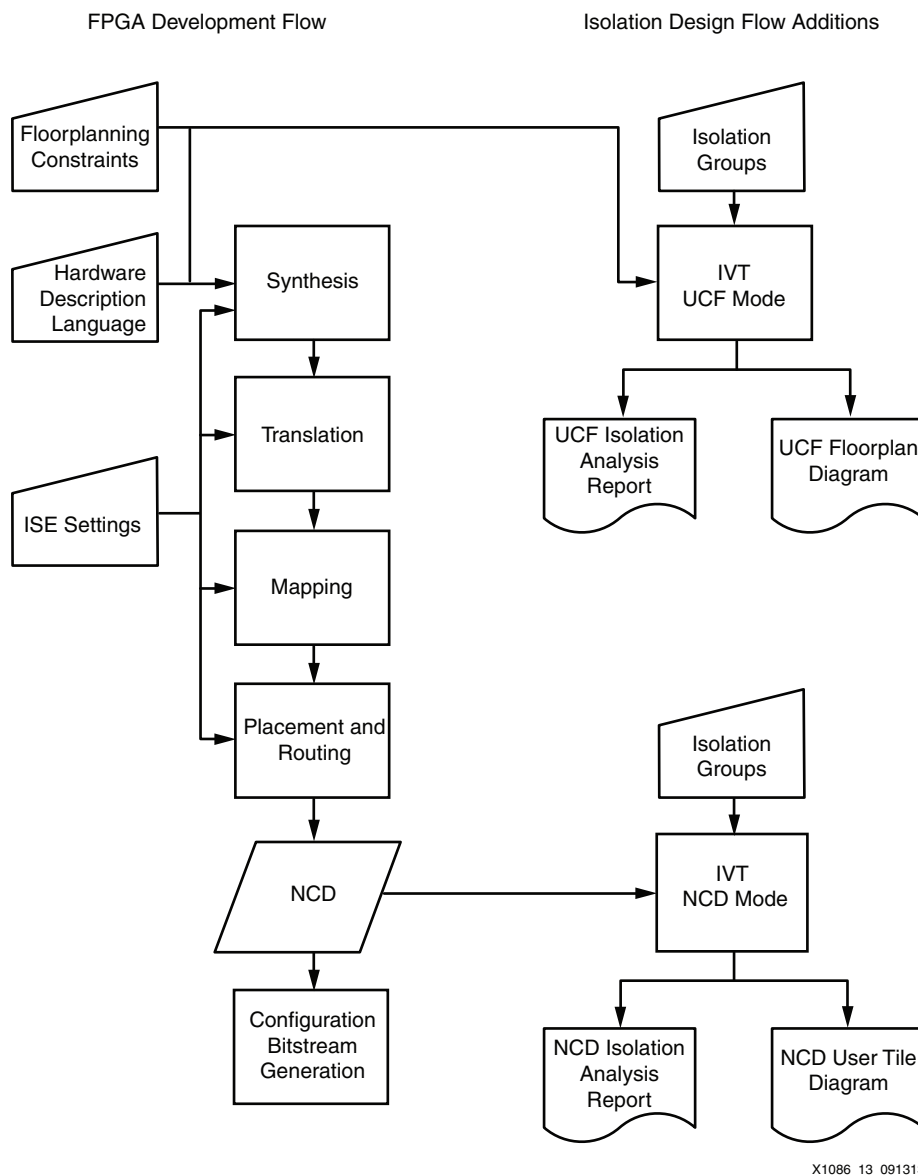


Figure 13: The Isolation Design Flow Relative to a Typical FPGA Design Flow



## Partition Overview

All the partition definitions and controls are done in the `xpartition.pxml` file. This file can be created by hand or automatically with the PlanAhead tool GUI, and is described in the Design Preservation Flows chapter in *Hierarchical Design Methodology Guide* (UG748) [Ref 5]. Creating the `xpartition.pxml` file automatically is the easier method. The file can be created automatically when launching from the PlanAhead tool. The ISE tools, versions 14.4 and later, are required.

To implement the `xpartition.pxml` file automatically using the PlanAhead tool, the new file is copied by the included implementation scripts of the reference design up to the top level of the project for future command-line use. The commands in the `xpartition.pxml` file are case sensitive.

## Trusted Routing Design Guidelines

While trusted routing is automatic, there are a few rules and concepts the user must be aware of. The following sections touch on the concept of ownership and then detail the IDF rules that the user must follow to ensure proper communication between isolated regions.

### Concepts - Ownership

- Logical ownership: Relates to HDL partition (right side of [Figure 14](#)):
  - Each HDL file logically owns the logic instantiated within it.
  - Only global logic (BUFG, MMCM, PLL, and so on) allowed at the top logical level of design and the global logic cannot logically be placed in a partition.
  - Global logic must be logically owned by the top level of the design.
- Physical ownership: Relates to area groups/Pblocks (left side of [Figure 14](#)):
  - No user logic component can be used if it is not physically owned by an isolated region (area group).
    - Physical ownership is defined by addition of the available components to the AREA\_GROUP definition in the UCF.
  - No global logic component can be used if it is not physically *owned* by some Isolation Region. For example, the BUFG constraint allows a range of BUFGs placed in the U1\_AES1 area group to be used by the designer:

```
AREA_GROUP "pblock_U1_AES1" RANGE=BUFGCTRL_X0Y0:BUFGCTRL_X0Y15;
```

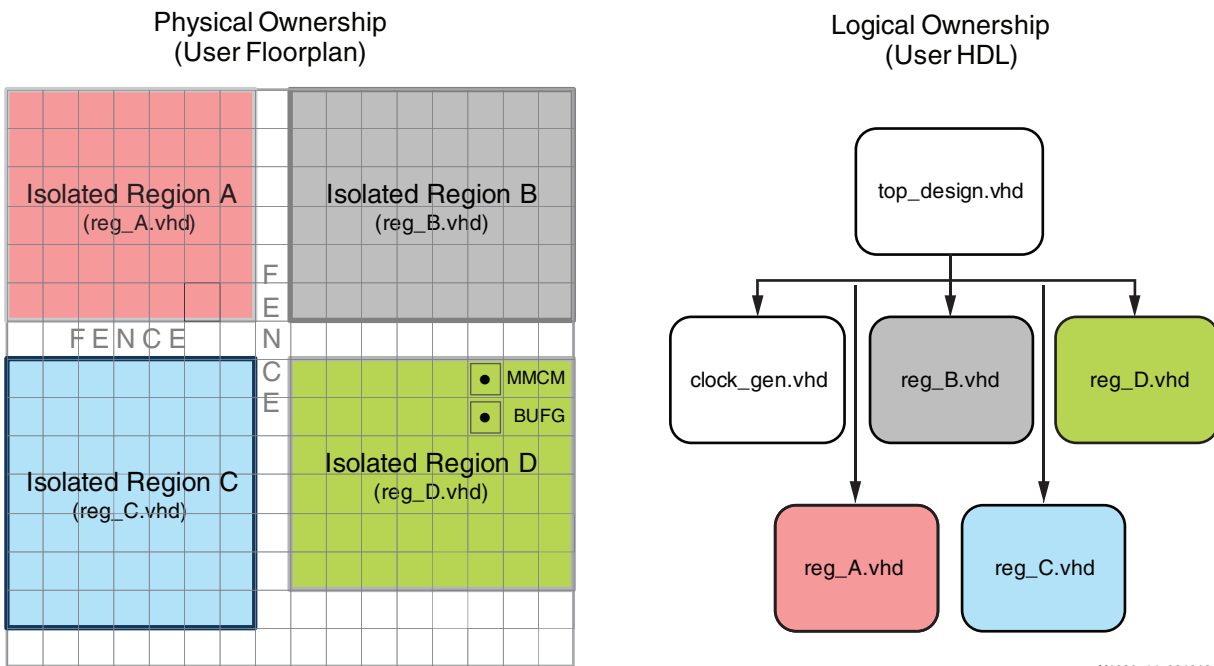


Figure 14: Side by Side View of Ownership Concepts

**Note:** The MMCM and BUFG instances in Figure 14 are logically owned by `clock_gen.vhd`, because that is the level of HDL that instantiated the primitives, and physically owned by Isolated Region D because their physical sites are encompassed by the area group `Isolated_Region_D`.

### Trusted Routing Rules

Trusted routing is automatic. 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:

1. Feed-through signals are not allowed without buffering of some kind (LUT, FF, and so on):
  - a. If a signal is directly connected to both an input port and an output port, it must be buffered.
  - b. Direct instantiation of a buffer (LUT1, for example) is necessary. This isolates area groups, with the LUT preventing a shorted net.

**Problem:** The statement `reset_out <= reset` violates IDF rules by creating a short between isolated area groups (Figure 15).

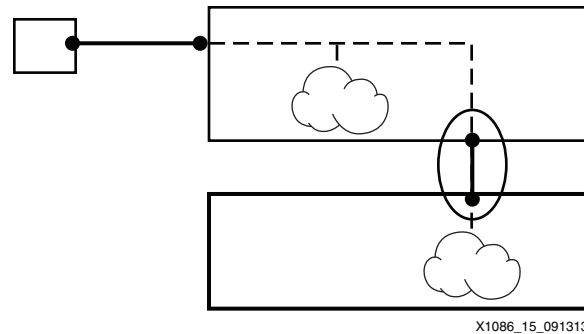


Figure 15: Short Created by Feed-Trough Signal

**Solution:** Feed-through signals need to be LUT buffered. This can be achieved either through HDL coding to ensure that there is some unique driver on the output port or by direct instantiation of a LUT buffer (Figure 16). The problematic code above can be fixed as follows:

```
lut_reset_out: LUT1
GENERIC MAP (INIT => X"2")
PORT MAP (I0 => reset, O => reset_out);
```

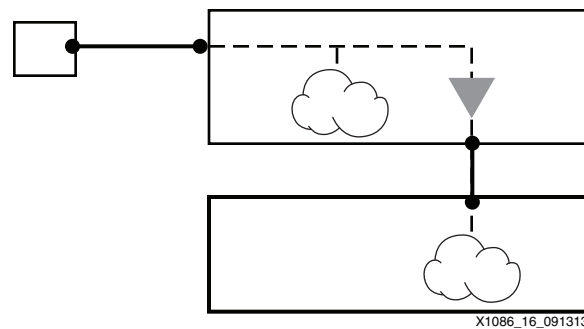
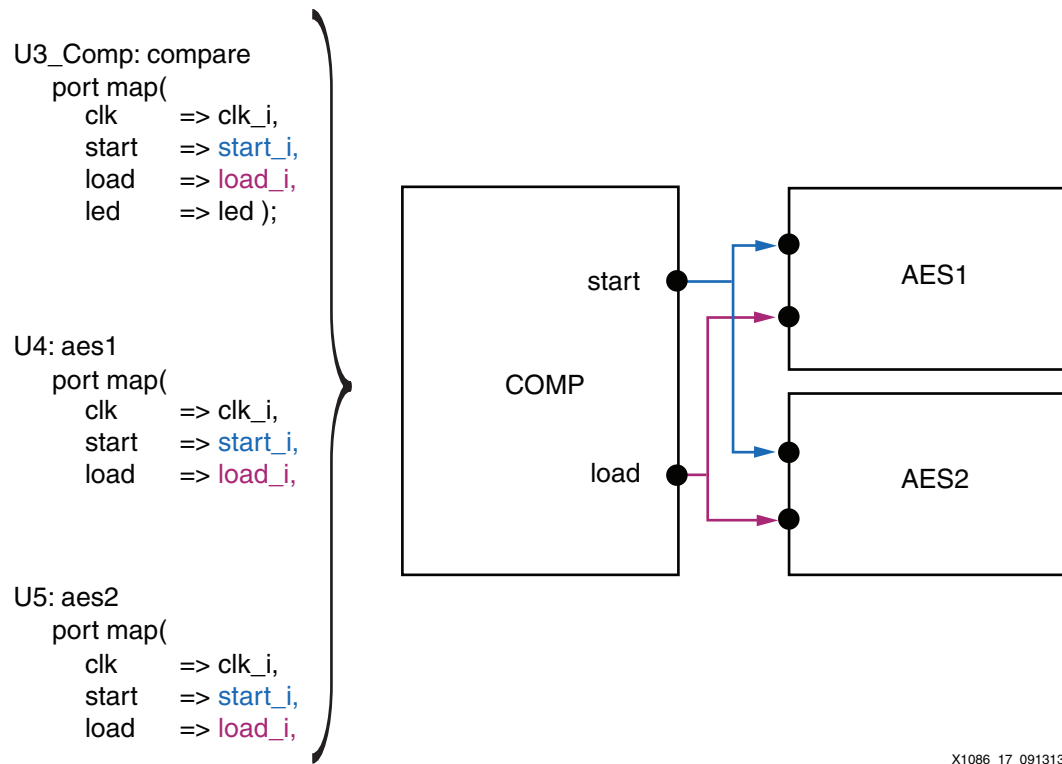


Figure 16: Elimination of a Short by Instantiation of a LUT Buffer

2. A function output port cannot connect to more than one function input port. Stated differently, port-to-port connections must be singular:
  - The user must create two different ports for such a connection.
  - Each port must not violate Rule 3 below.

**Problem:** The segment of VHDL code that creates a common connection between unintended isolated functions is shown in [Figure 17](#).



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

**Solution:** The user must create multiple output ports if required to drive multiple input ports of other isolated functions (Figure 18).

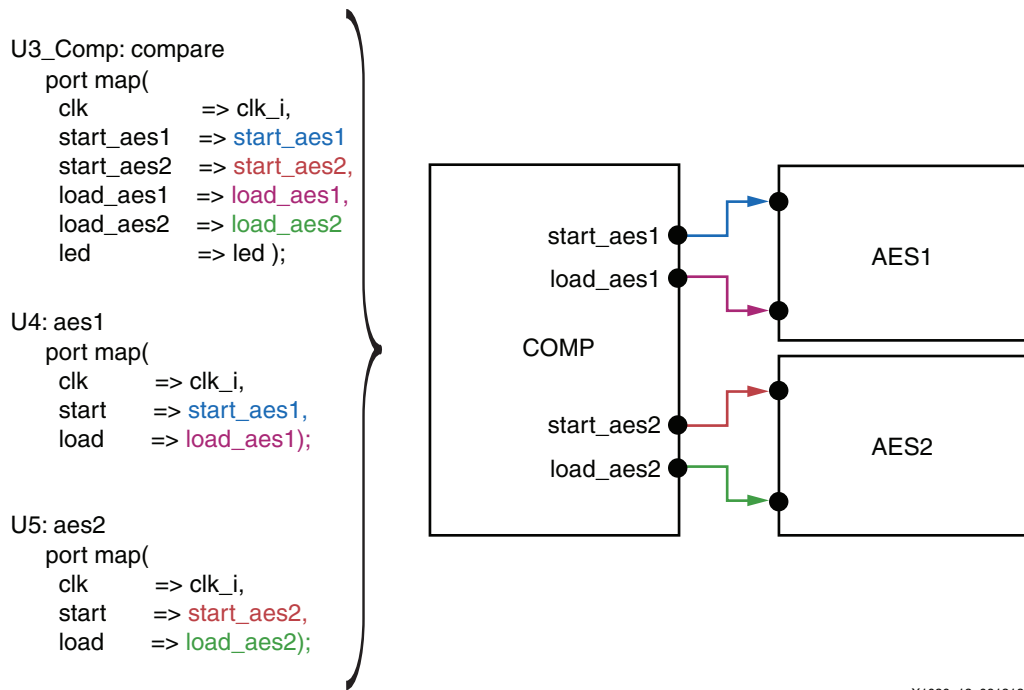


Figure 18: Elimination of Unintended Connection Using Multiple Output Ports

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 area groups, with the LUT preventing a shorted net.

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

```

start_aes1 <= start_i;
start_aes2 <= start_i;
    
```

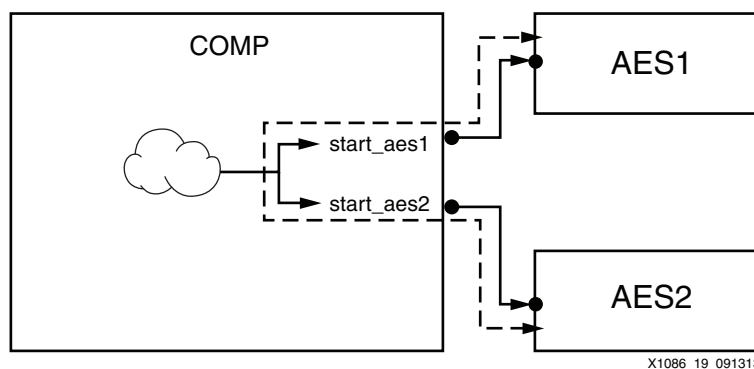


Figure 19: Unintended Connectivity between AES1 and AES2 inside COMP

**Solution:** Each port driver needs to be LUT buffered. This can be achieved either through HDL coding to ensure that there is some unique driver for each output port, or by direct instantiation of a LUT buffer (Figure 20). The problematic code above 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);
```

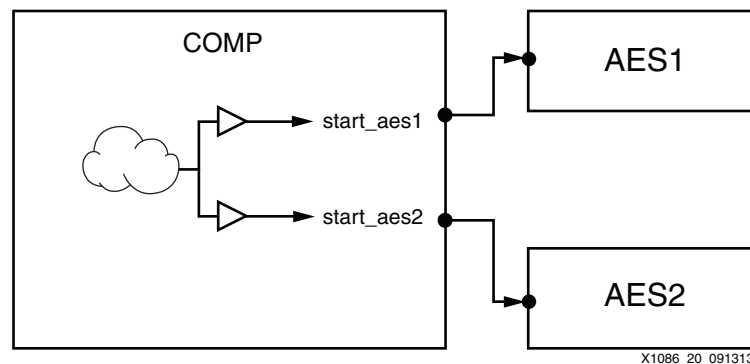


Figure 20: Elimination of Unintended Connectivity Using LUT Buffers

## 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 4).

Table 4: Xilinx 7 Series FPGA and Zynq-7000 AP SoC Fence Tile Requirements

User Tile	User Tile Description	Fence Tile?
CLB (LL and LM)	<b>Configurable Logic Block.</b> The key logic unit of an FPGA.	Yes Vertical: 1 Horizontal: 1
BRAM	<b>Block RAM.</b> User accessible high speed RAM (BRAM tile is RAMB36 which is two RAMB18 blocks; PlanAhead tools only allow selection of a BRAM tile).	Yes Vertical: 1 Horizontal: 1
DSP	<b>Digital Signal Processor.</b> A programmable math function (DSP tile is two DSP48E1 slices; analysis was done on DSP48E1 slice; PlanAhead tools only allow selection of a DSP tile).	Yes Vertical: 1 Horizontal: 1
PCIe®	PCI Express endpoint block	Yes Vertical: 1 Horizontal: 1
GTX transceiver	<b>High speed transceiver for</b> Kintex-7 (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 <b>Quad</b> Horizontal: 1 GTX <b>Channel</b>
IOB/IOI (Master/Slave pair)	<b>IOB</b> = Input receivers with output drivers (in master/slave pairs). <b>IOI</b> = All other IOB logic such as capture and send FFs (I/O tile pair: all I/O except at top and bottom of each I/O bank)	Yes Vertical: 1 Horizontal: 1
IOB/IOI (Single)	<b>IOB</b> = Input receivers with output drivers <b>IOI</b> = All other IOB logic such as capture and send FFs (I/O tile Single: at top and bottom of each I/O bank).	Yes Vertical: 1 Horizontal: 1
IO_FIFO (CMT tile)	<b>I/O FIFO</b> (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 Only)	<b>Processor System (PS)</b> The PS block is located in the upper left corner of the Zynq-7000 device 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 4 are shown in the following figures for fence tile usage. Figure 21, Figure 22, and Figure 23 show high-level PlanAhead tool views for all of the user tiles 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 architecture 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 PLLs and 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.

Figure 21 shows the PlanAhead 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 tile and clocking components is discussed in a later section. One CMT tile spans a complete I/O bank in a 7 series FPGA.

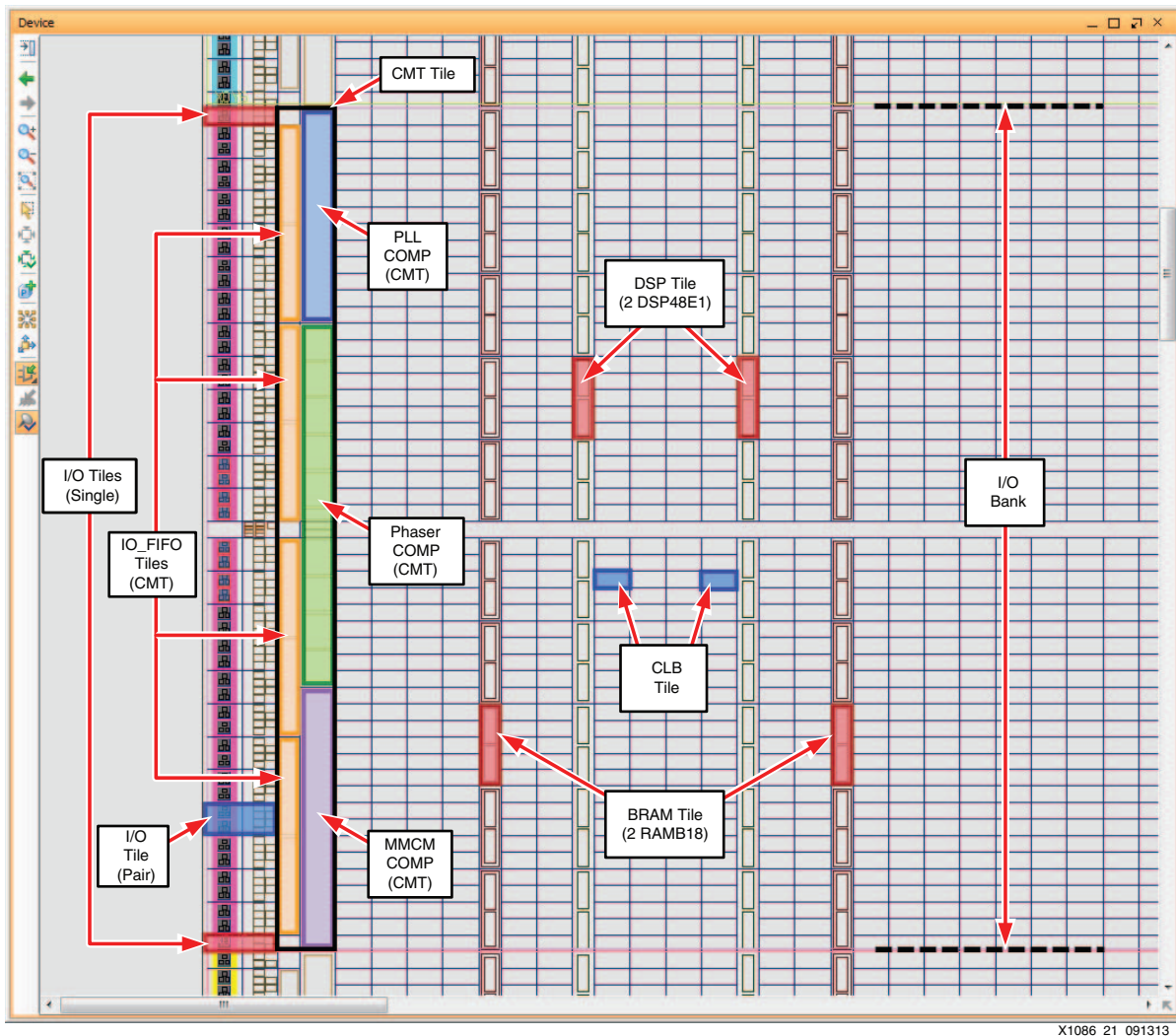


Figure 21: High-Level PlanAhead Tool View for CLB, BRAM, DSP, I/O, and CMT Tiles



Figure 22 shows the PlanAhead tool view for the following User Tiles: 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.

In most Xilinx 7 series devices, the GTX tile is on the extreme edge of the device. In some devices the GTX tile is in the middle of the device, so it might not always make sense to be part of a vertical fence.

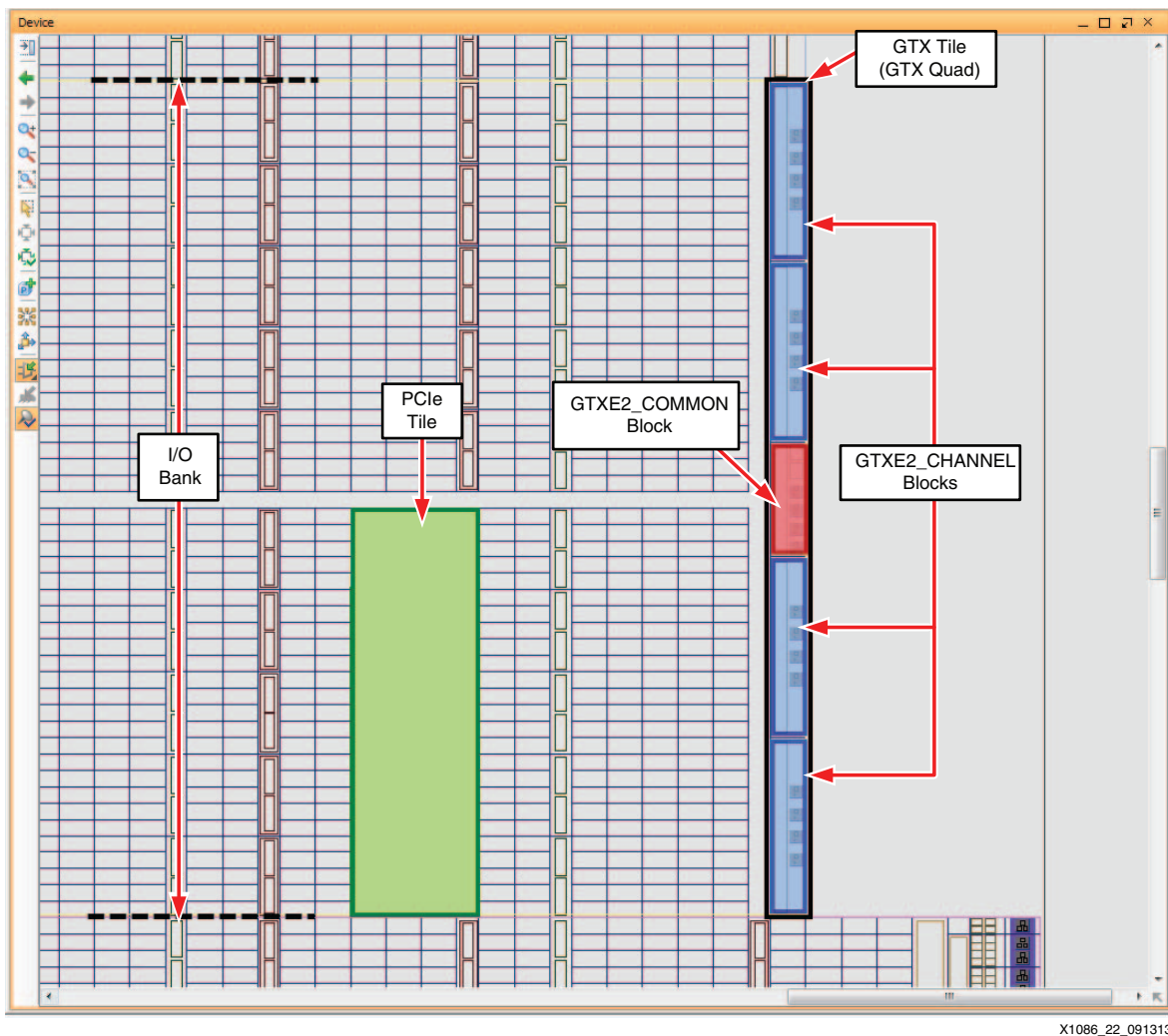


Figure 22: High-Level PlanAhead Tool View for PCIe Tile and GTX Quad Tile

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

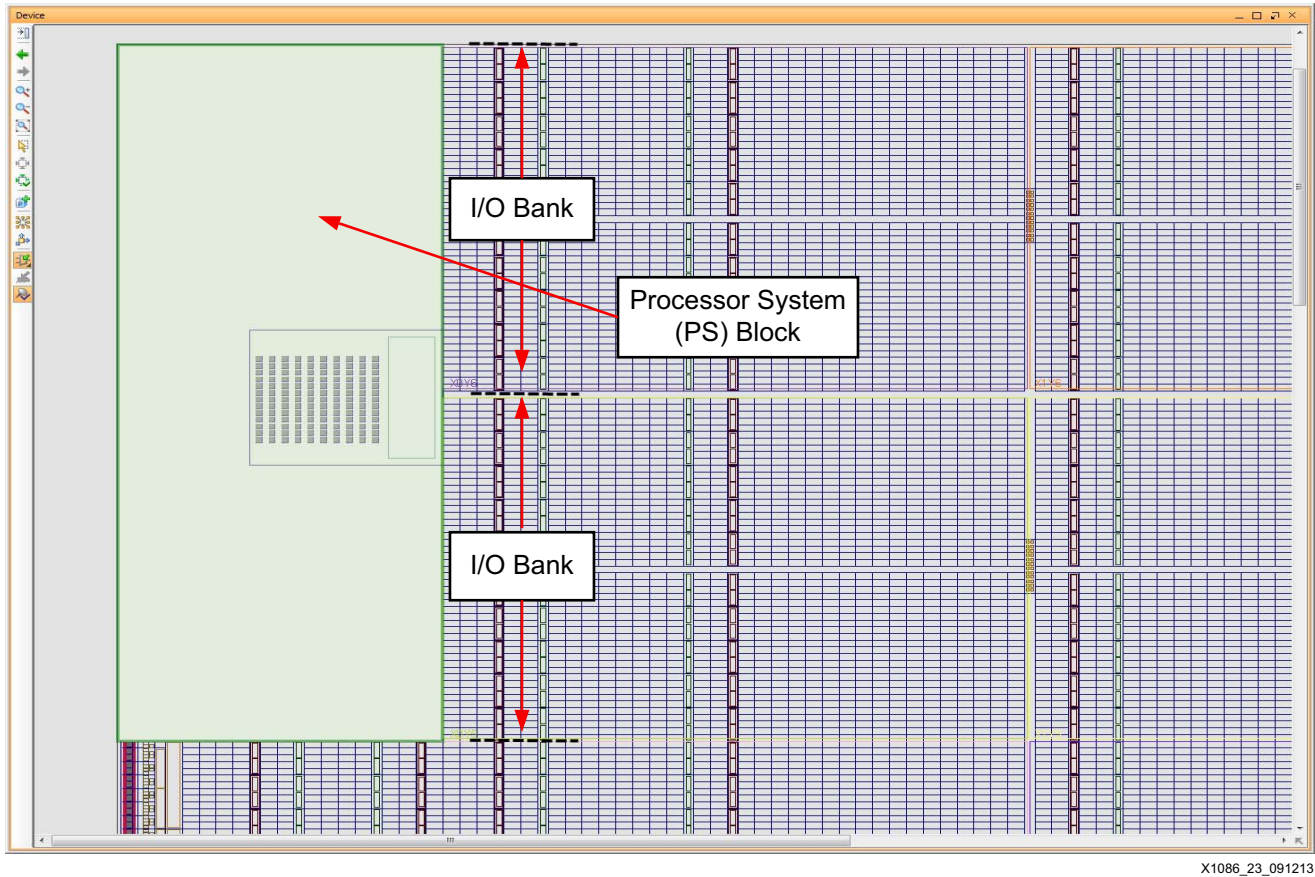


Figure 23: High-Level PlanAhead Tool View for Zynq-7000 Processor System (PS) Block

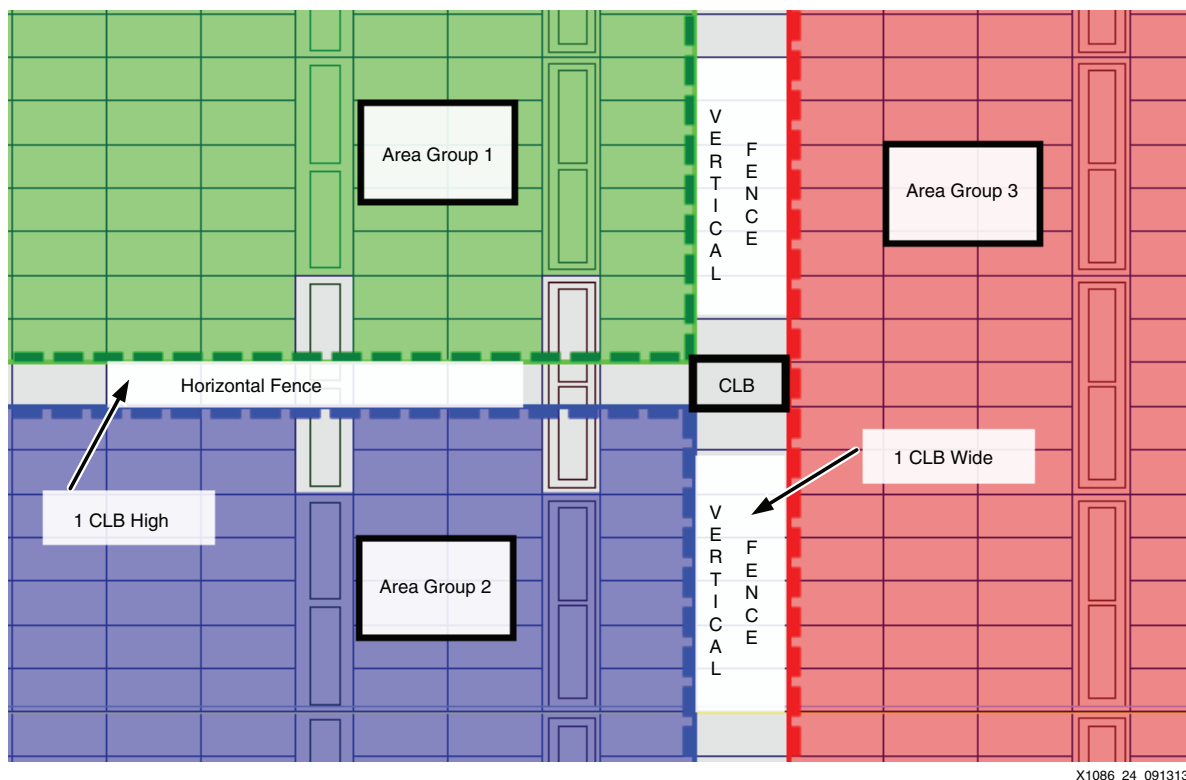
## CLB Tile Fence Tile Usage

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.

As can be seen in the Figure 24, the horizontal fence (white area) isolates Area Group 1 from Area Group 2. The vertical fence (white area) isolates Area Group 1 from Area Group 3 and Area Group 2 from Area Group 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.



X1086\_24\_091313

Figure 24: PlanAhead Tool View for Horizontal and Vertical Isolation Fences using the CLB Tile

## BRAM Tile 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.

As can be seen in [Figure 25](#), the horizontal fence (white area) isolates Area Group 1 from Area Group 2. The vertical fence (white area) isolates Area Group 1 from Area Group 3 and Area Group 2 from Area Group 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 area group than 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, but instead the whole BRAM tile must be selected, as shown in [Figure 25](#).

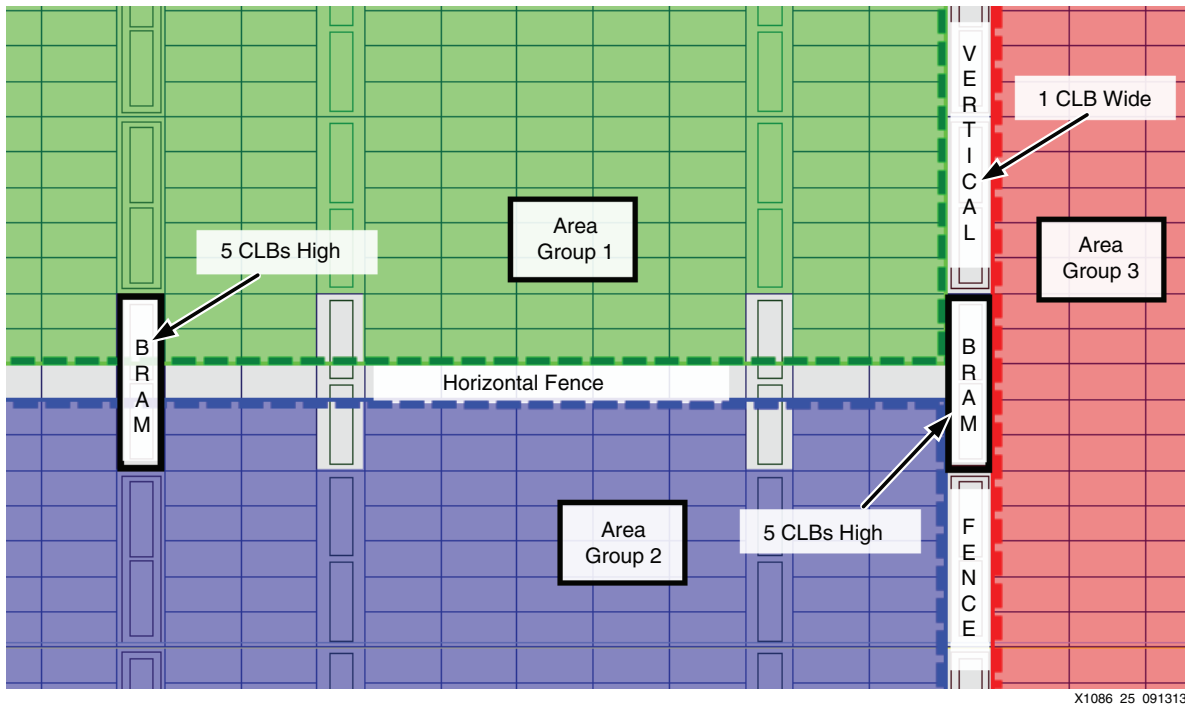


Figure 25: PlanAhead Tool View for Horizontal and Vertical Isolation Fences using the BRAM Tile

## DSP Tile Fence Tile Usage

The DSP tile (2 DSP48E1s) 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.

As can be seen in the [Figure 26](#), the horizontal fence (white area) isolates Area Group 1 from Area Group 2. The vertical fence (white area) isolates Area Group 1 from Area Group 3 and Area Group 2 from Area Group 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 area group than 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, but instead the whole DSP tile must be selected, as shown in [Figure 26](#).

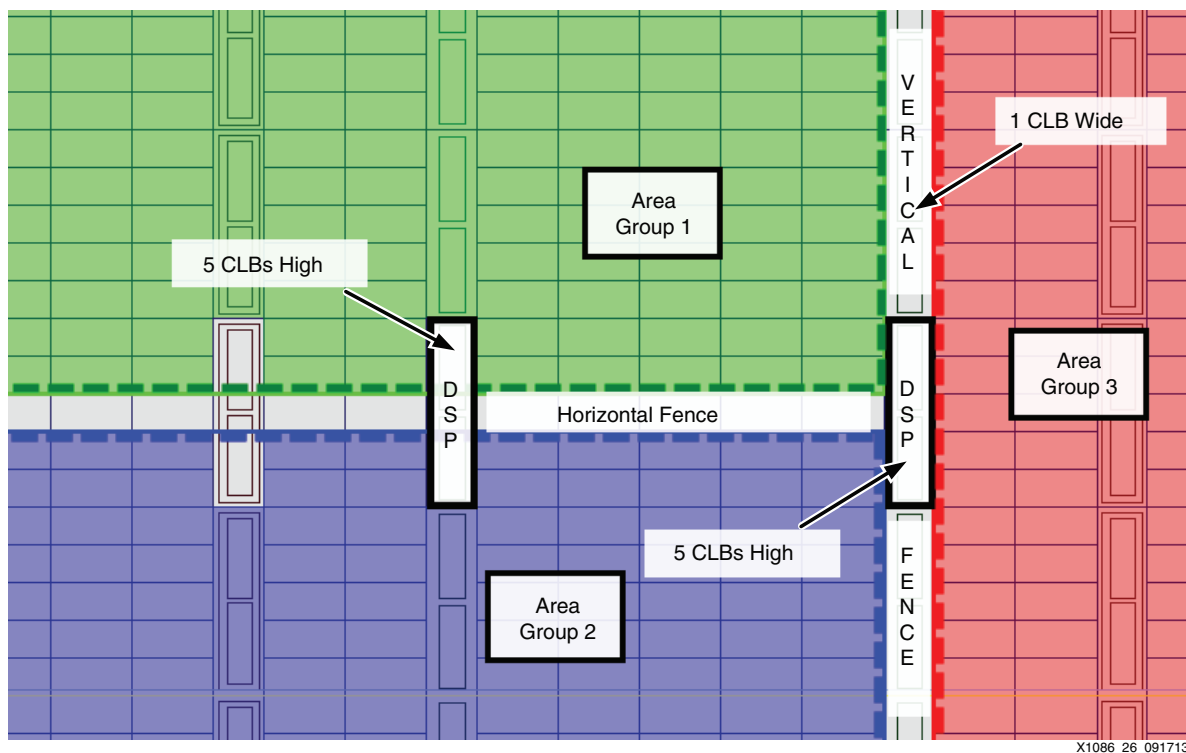
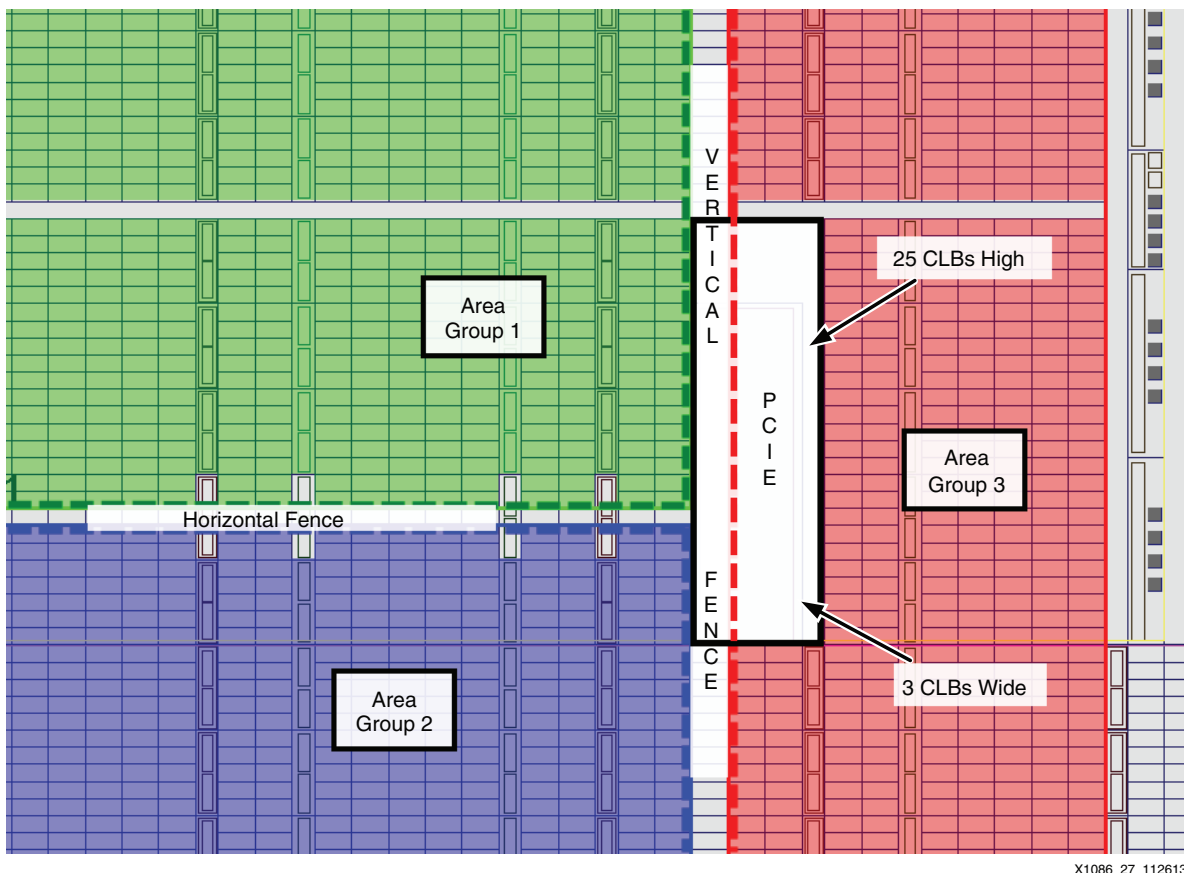


Figure 26: PlanAhead Tool View for Horizontal and Vertical Isolation Fences using the DSP Tile

## PCIe Tile Fence Tile Usage

The PCIe 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 tile in both a horizontal and vertical fence. [Figure 27](#) shows the PCIe being used as a fence tile in a vertical and a horizontal fence.

As can be seen in [Figure 27](#), the horizontal fence (white area) isolates Area Group 1 from Area Group 2. The vertical fence (white area) isolates Area Group 1 from Area Group 3 and Area Group 2 from Area Group 3. The horizontal fence is one CLB tile high and the vertical fence is one CLB tile wide, except at the PCIe tile.



**Figure 27: PlanAhead Tool View for Horizontal and Vertical Isolation Fences using the PCIe Tile**

## GTX Tile Fence Tile Usage

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 most Xilinx 7 series devices, the GTX tile is 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 GTX tile is in the middle of the device, the GTX Quad can be used as a vertical fence.

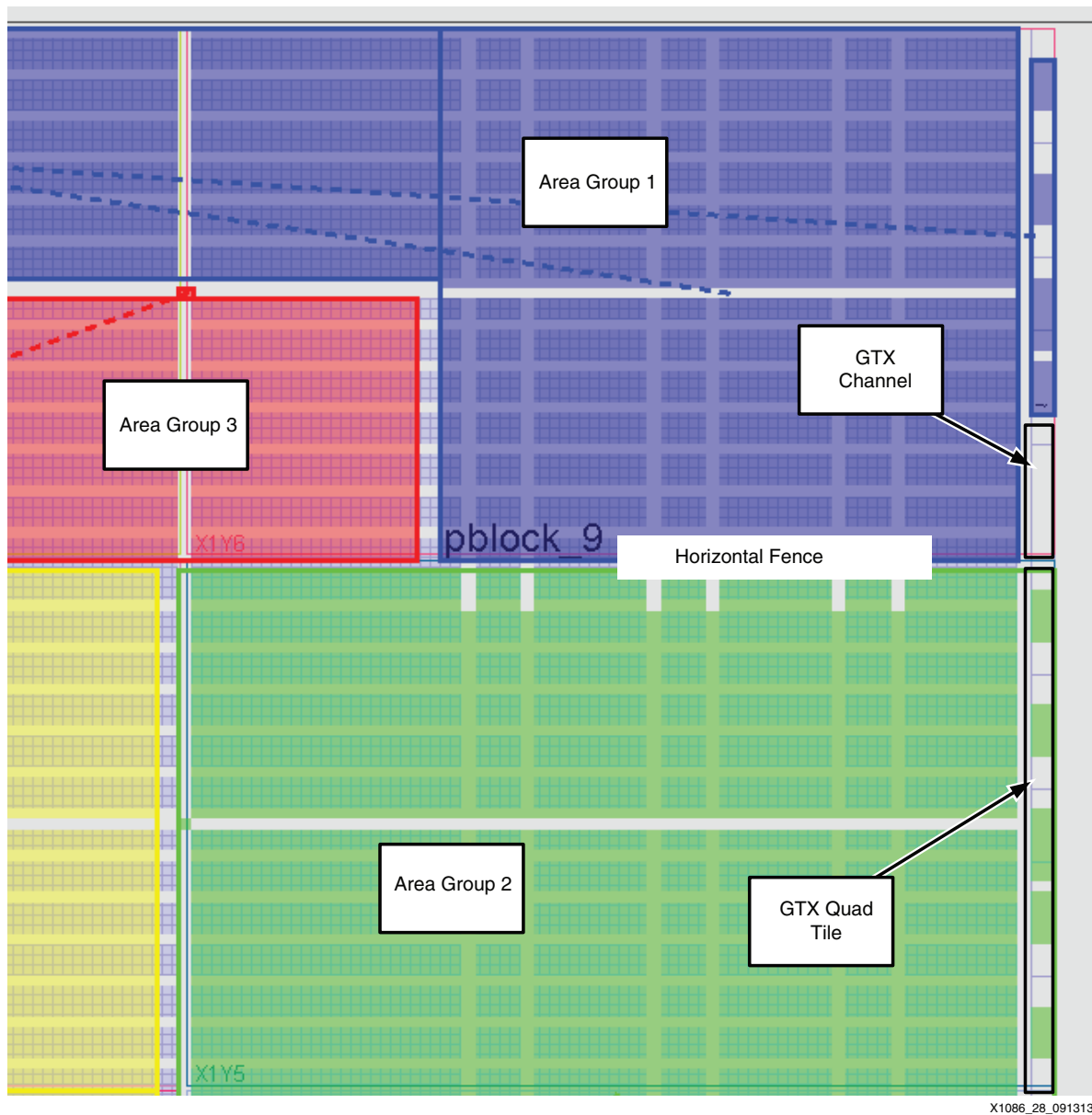


Figure 28: PlanAhead Tool View for Horizontal Fencing using the GTX Tile

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.

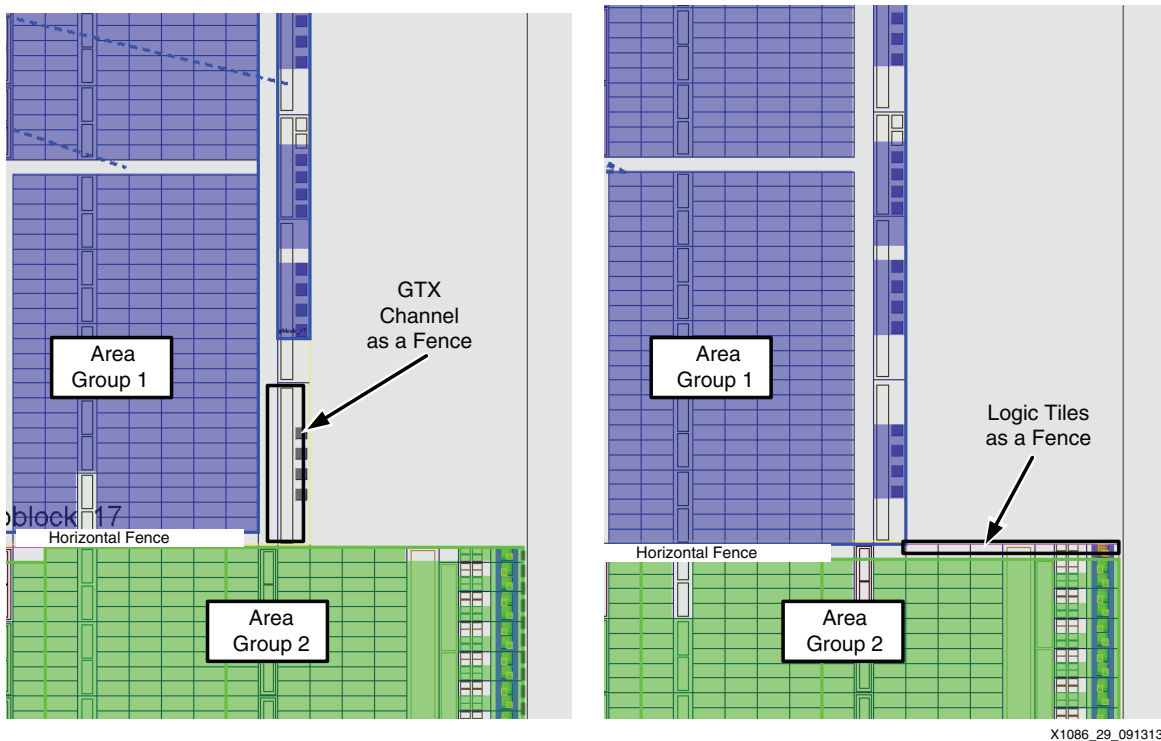


Figure 29: Example of GTX Channel Block and Logic Tiles as Fences

### ***Processor System (PS) Block Fence Tile Usage***

The following basic 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 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 are 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\_II tile below the far right corner of the PS block does not provide the required isolation on its own. Therefore, the CLB\_II 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\_II tile in order to provide the required diagonal



isolation between isolation regions (PS block and the user tiles in the PL isolated region below). Refer to [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.

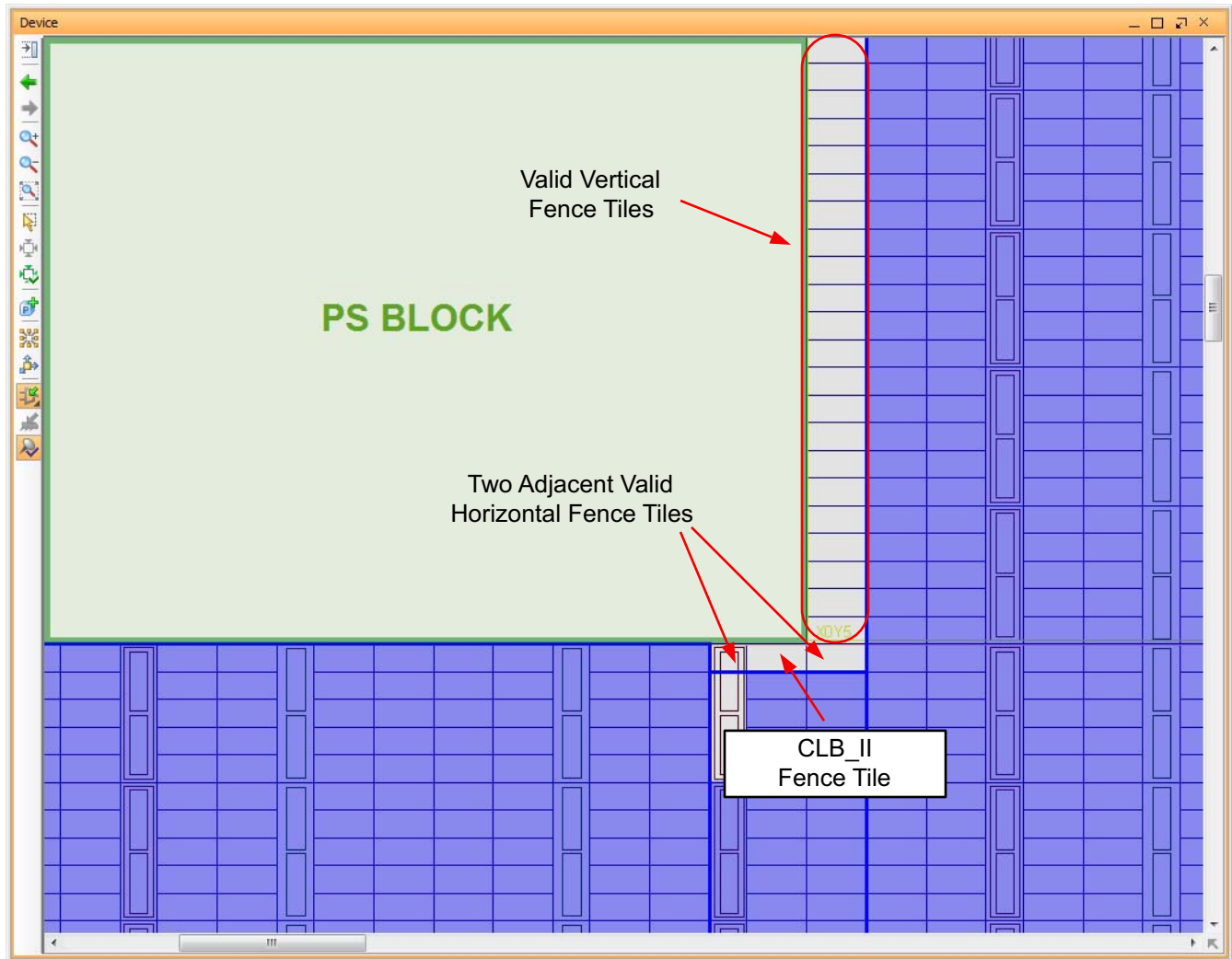


Figure 30: PlanAhead Tool View for Basic Fence Tile Usage around the PS Block

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.

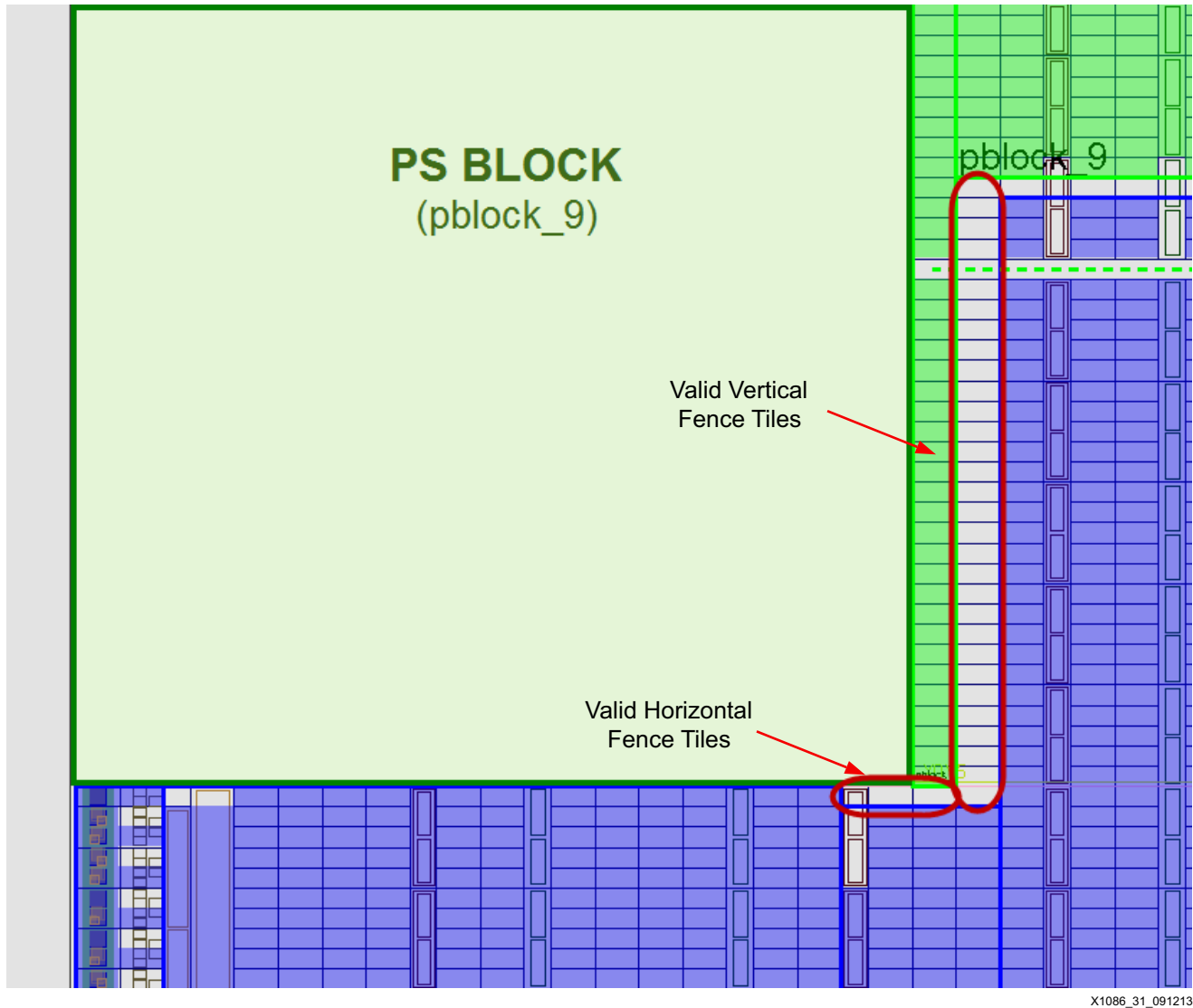


Figure 31: PlanAhead Tool View for More Realistic Fence Tile Usage around the PS Block

### I/O Tile Fence Tile Usage

With some exceptions, each I/O bank in 7 series FPGAs 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.

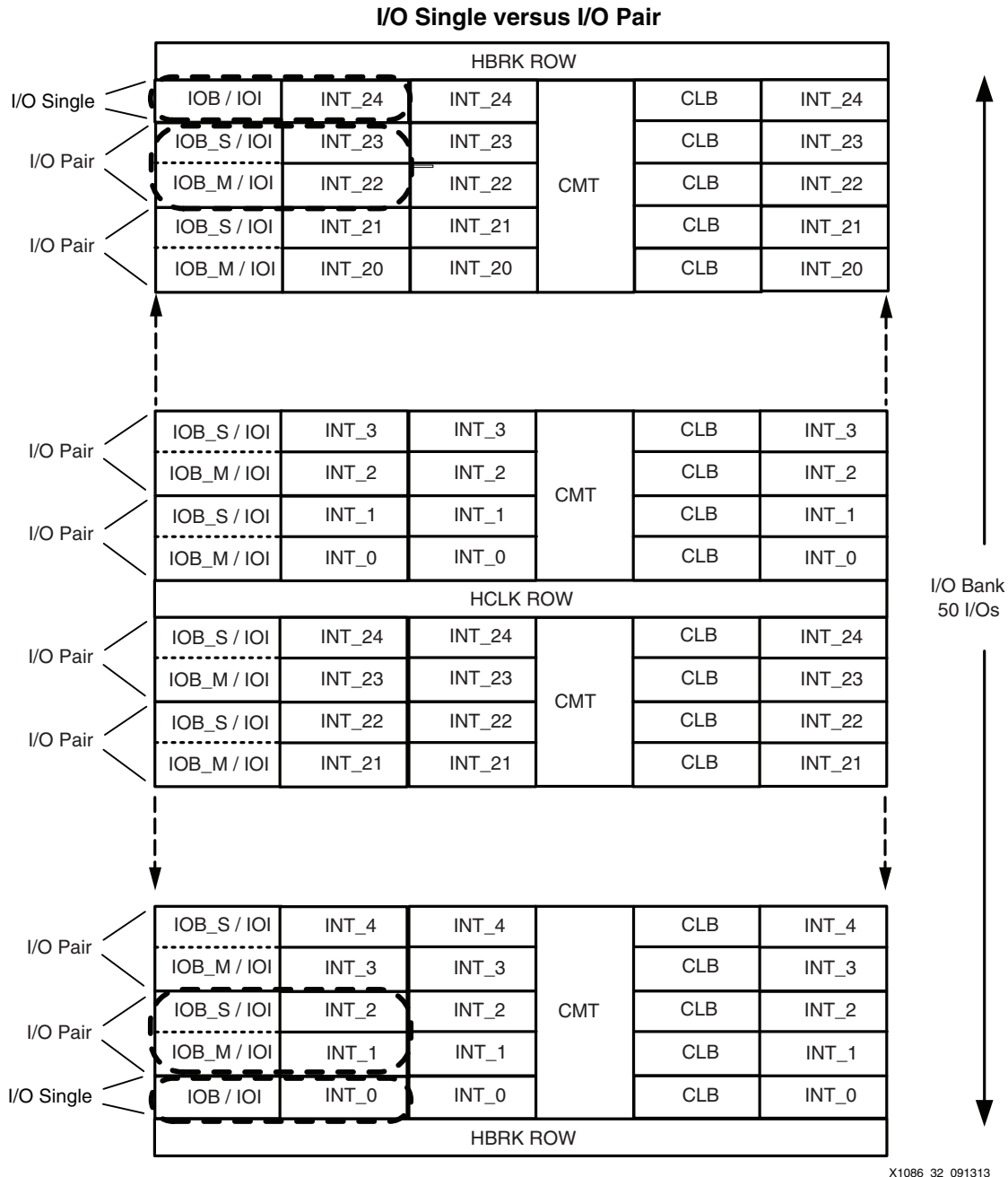
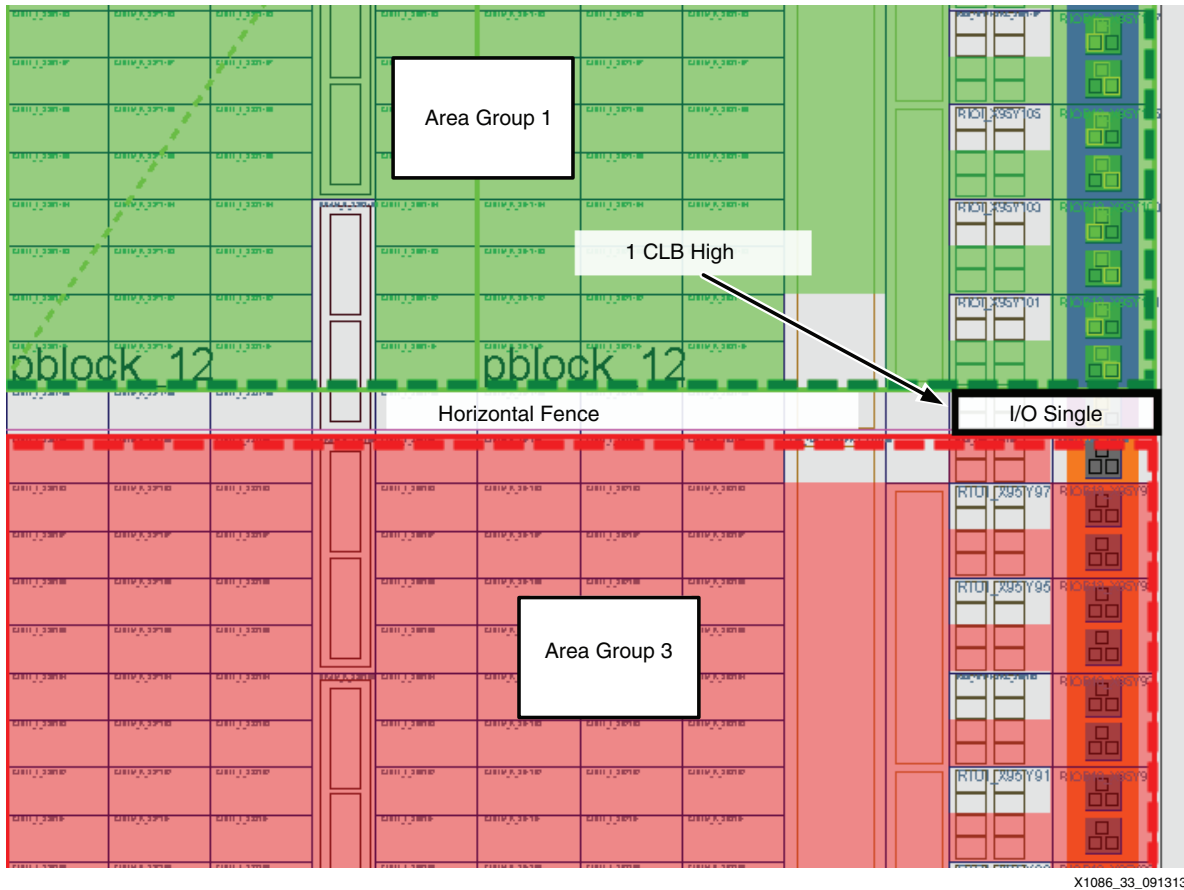


Figure 32: 7 Series I/O Single versus I/O Pair Tiles

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 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.



**Figure 33: PlanAhead 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.

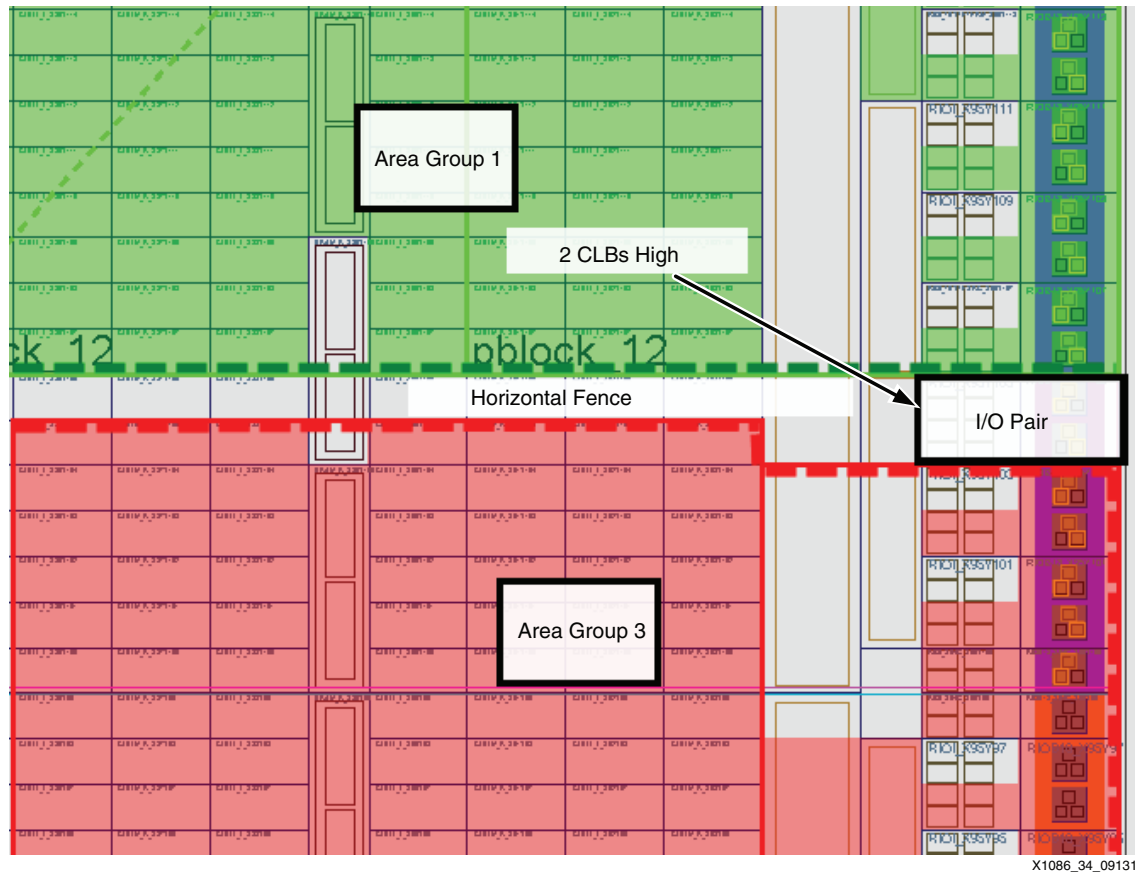


Figure 34: PlanAhead Tool View for Horizontal Isolation Fence using the I/O Pair Tile

### CMT Tile Fence Usage

Each 7 series FPGA has up to 24 clock management tiles (CMTs), each consisting of one mixed-mode clock manager (MMCM) and one phase-locked loop (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 PlanAhead 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 Area Groups 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.

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

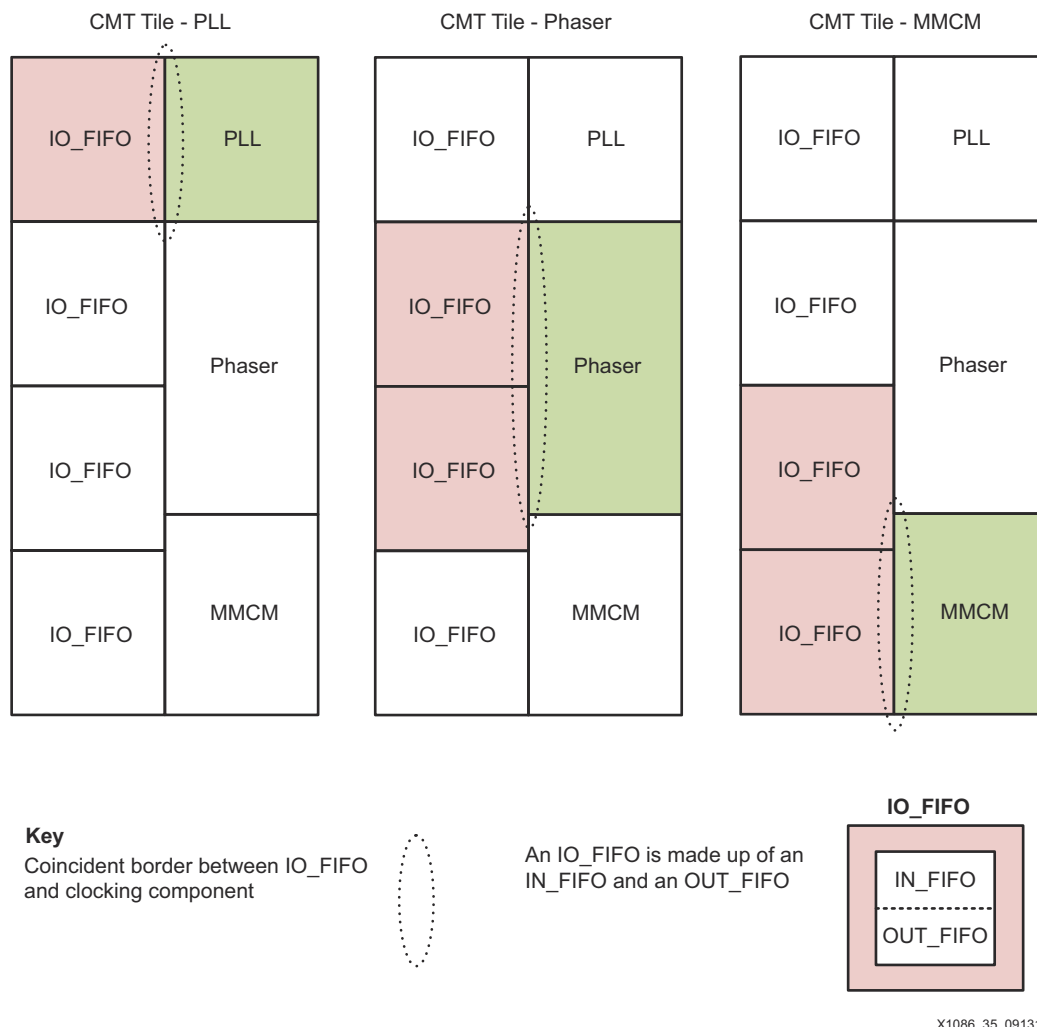


Figure 35: CMT Clocking Components and Adjacent IO\_FIFOs

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
- 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

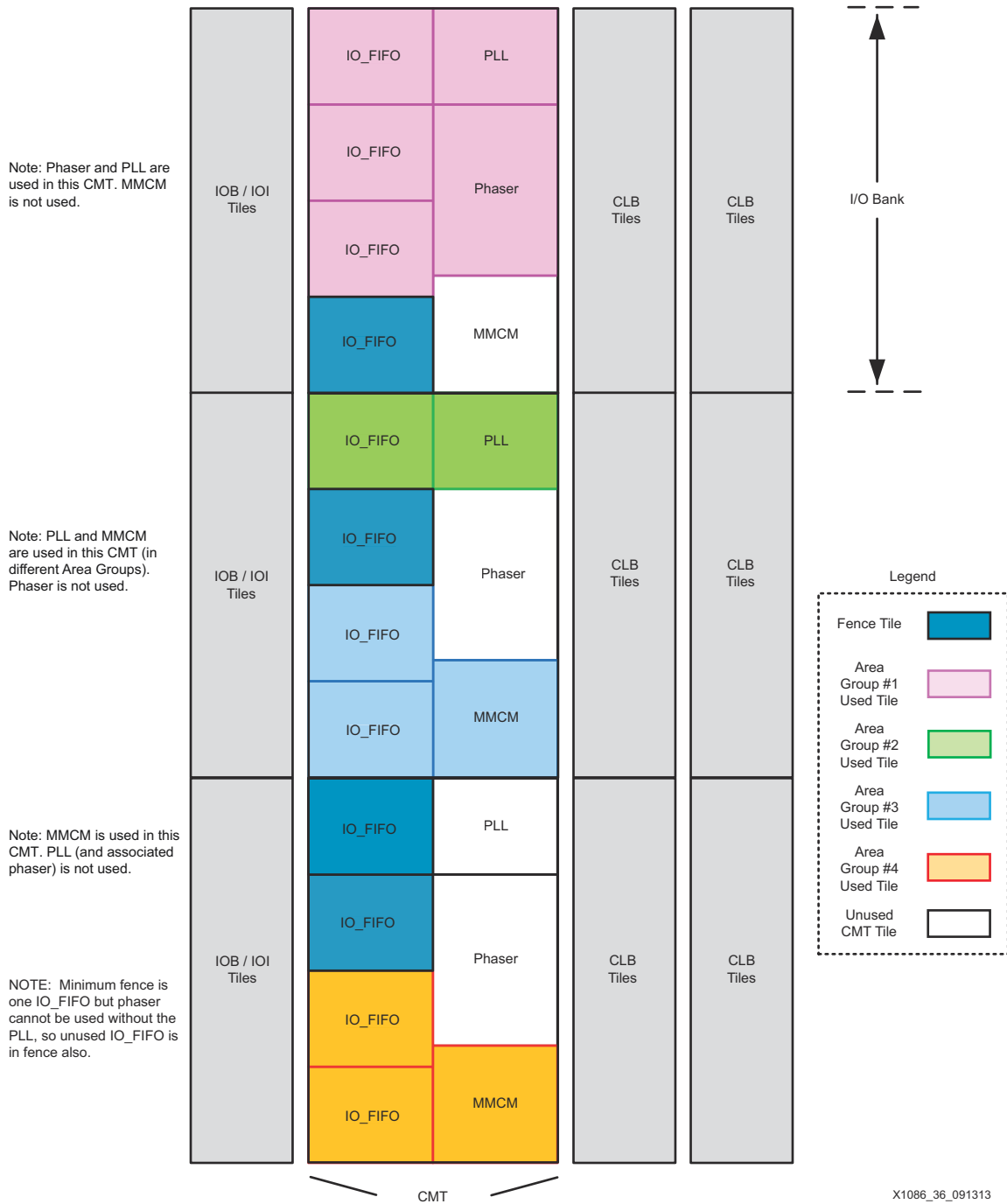


Figure 36: Minimum IO\_FIFO Fence Tiles for the 7 Series CMT Clocking Components



## 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 Area Group

It is generally advised to add all available resources within an Area Group, 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. Naturally, all used components must be included. This includes the global, regional, and I/O clock buffers (**BUFGCTRL**, **BUFHCE**, **BUFR**, and **BUFIO**) *except* the **BUFMRCE** (unless used in the design), even though such components are instantiated (logically owned) at the top level.

#### CMT Resource Exception Guideline

If *not using* CMT Components (Clocking or IO\_FIFO), then exclude those resources from the Area Group. These include: MMCME2\_ADV, PLLE2\_ADV, PHASER\_IN\_PHY, PHASER\_OUT\_PHY, PHY\_CONTROL, IN\_FIFO, and OUT\_FIFO.

**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 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 tile must include at least one IO\_FIFO, therefore, the MMCM and the Phaser cannot both be used in different Isolated Regions (Area Groups). However, the MMCM and the Phaser could be used in the same Area Group

**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.

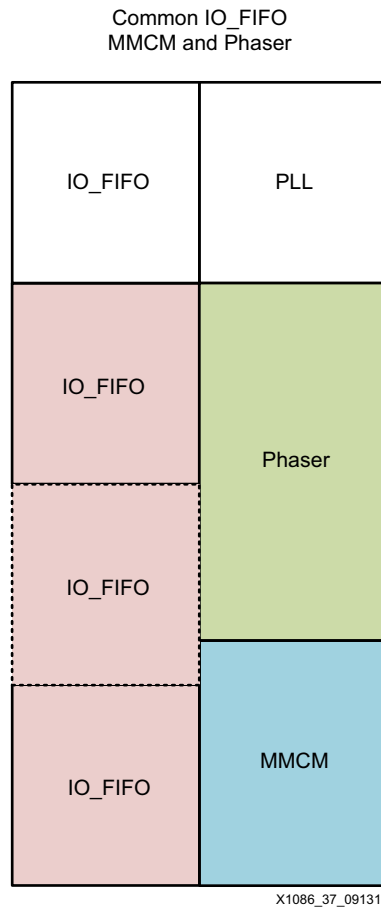


Figure 37: Common IO\_FIFO Between MMCM and Phaser Components

### ***Guidelines for Controlling Inference of Global Clocking Logic***

For an IDF design, the BUFs (global buffers) must be at the top level of the design. To achieve this in an ISE XPS design, set the C\_FCLK\_CLKx\_BUF parameter to FALSE in the System MHS file for each of the BUFs instantiated, as shown in the following example syntax. This will prevent XPS from automatically adding BUF components to the clocks, thereby allowing the user to instantiate them at the desired top level of the system design.

```
# -----
# Add these lines to prevent BUF insertion on the FCLKS to the PL
PARAMETER C_FCLK_CLK0_BUF = FALSE
PARAMETER C_FCLK_CLK1_BUF = FALSE
PARAMETER C_FCLK_CLK2_BUF = FALSE
PARAMETER C_FCLK_CLK3_BUF = FALSE
# -----
```

All clock resources used must be physically owned by one of the isolated regions. For example, in the IDF reference design referred to in this application note, the two BUFGCTRL clock buffers that are used in this design are included in the pblock\_U1\_AES1 Pblock, because this area group is physically close to these buffers. There are also BUFHCE buffers in the center column of the device that need to be included in the pblock\_U1\_AES1 Pblock. All of these buffers are enclosed in two separate rectangles, as shown in the IDF Lab reference design in [Figure 38](#).

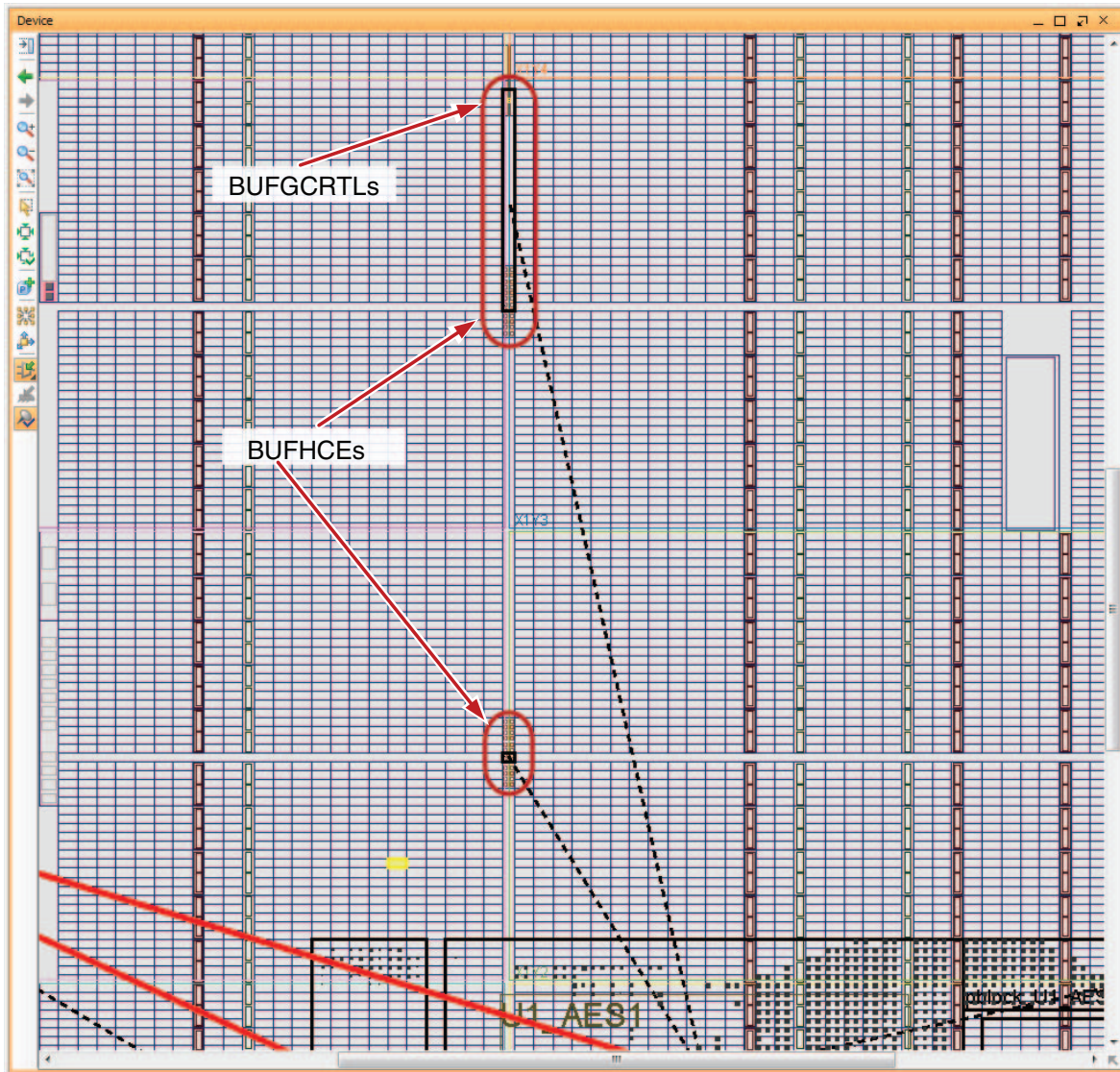


Figure 38: Additional Rectangles to Include the BUFGCTRLs and BUFHCEs

## Gaps in the Floorplan

There are apparent gaps in the floorplan as seen by the ISE tool FPGA Editor or PlanAhead tools. These gaps are an artifact of how the software model is depicted in the user GUI. 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.

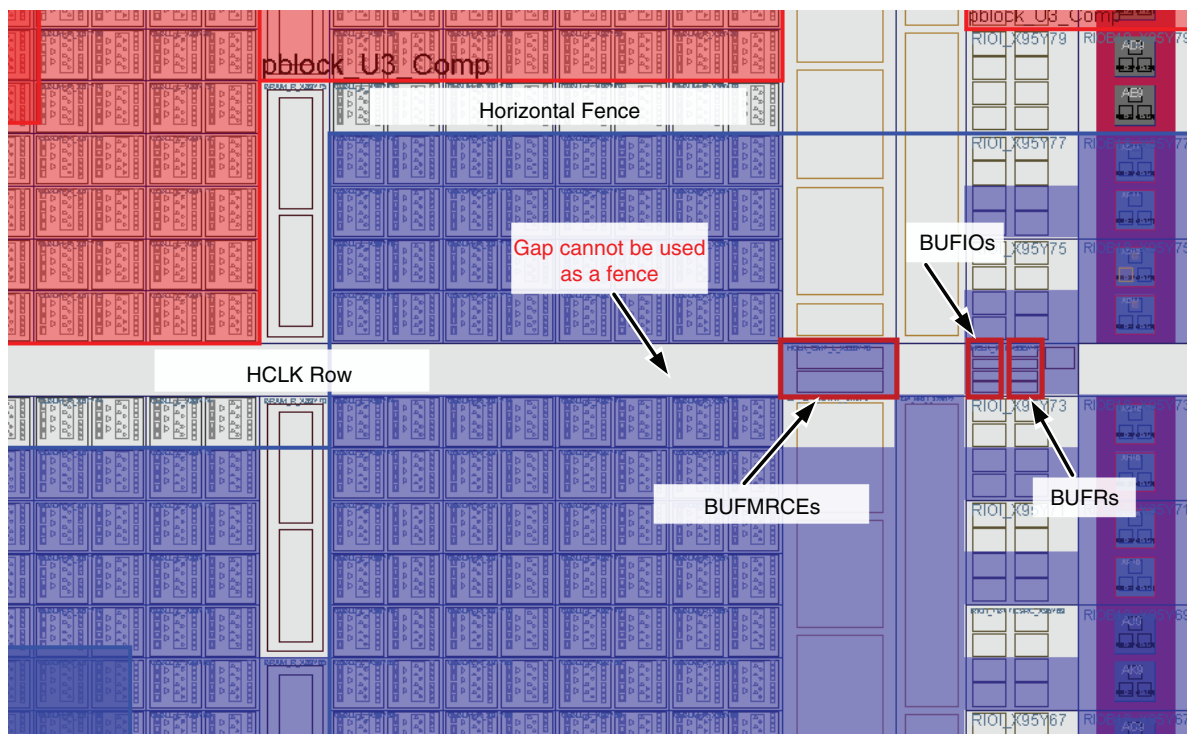
The most common gaps seen in the floorplan are the:

- 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 39](#). It contains clock routing and the following clock and I/O buffers near the I/Os:

- BUFIO (I/O Buffers)
- BUFR (Regional Clock Buffers)



X1086\_39\_092413

Figure 39: Gaps in the Floorplan—HCLK Row

## Vertical Clock Center Column Gap

The vertical clock column, in the center of device, appears as a gap between CLB tiles, as shown in [Figure 40](#). 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

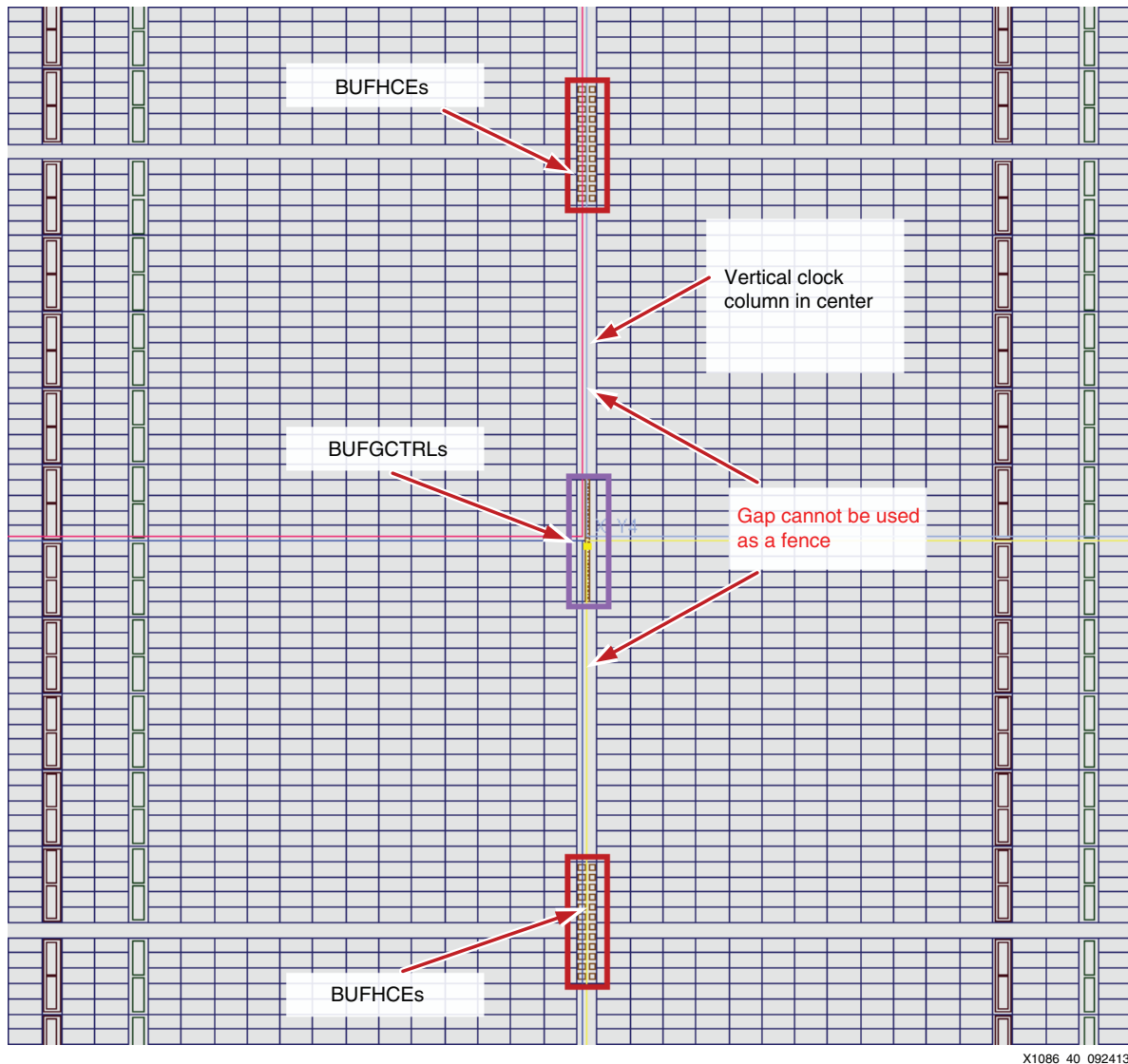


Figure 40: Gaps in the Floorplan—Vertical Clock Center Column

## Configuration Blocks Gap

The configuration block, near the center of device, appears as a gap between CLB tiles vertically and between CLB and BRAM tiles horizontally, as shown in Figure 41. This block cannot be used as isolation. It contains configuration and analog blocks such as:

- XADC
- ICAP
- BSCAN
- EFUSE
- FRAME\_ECC
- System Monitor

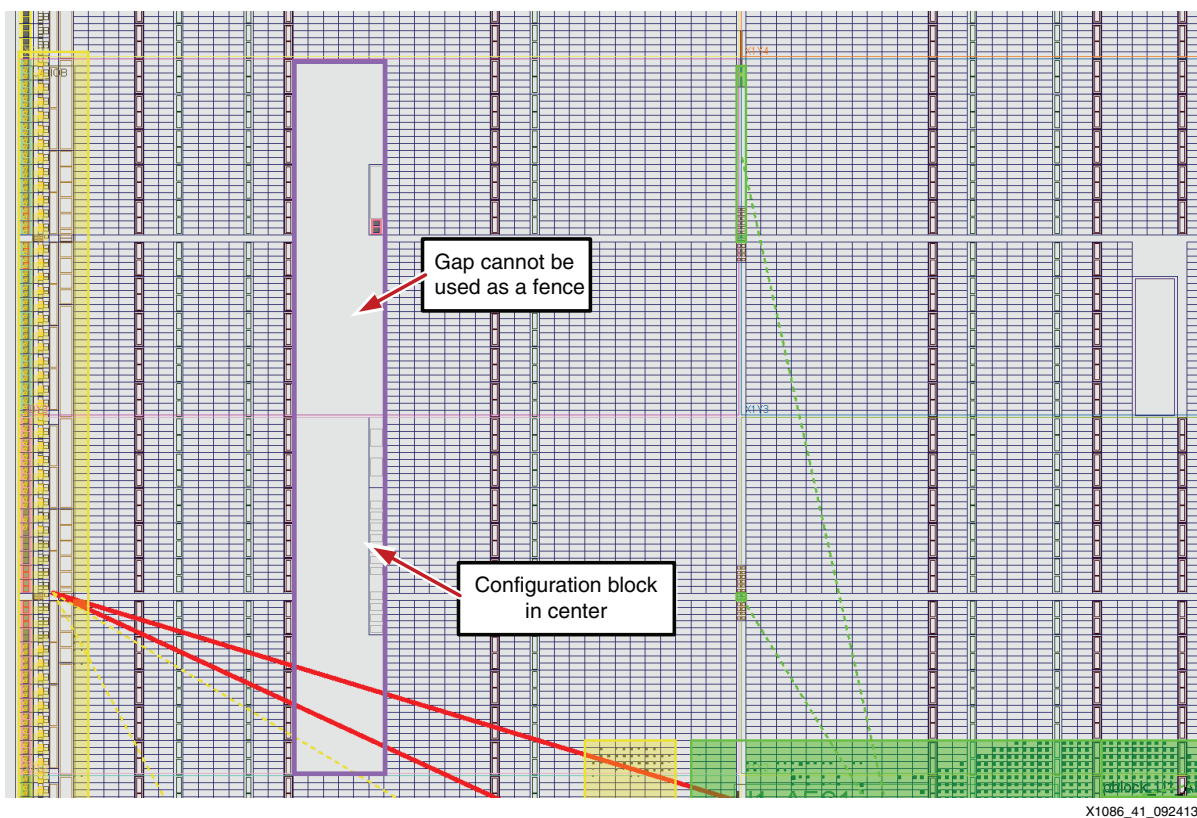


Figure 41: Gaps in the Floorplan—Configuration Blocks

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 Area Group, then all of the logic components need to be included in that Area Group. In other words, the configuration logic components cannot be split between different isolation regions (Area Groups) or be part of the isolation fence. Xilinx recommends that whether or not the user makes 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, the user must assure there is an isolation fence, using valid fence tiles, completely surrounding the block.

---

## 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 2]. The design checklist in [Table 5](#) includes simulation, implementation, and hardware details for the reference design.

*Table 5: Reference Design Checklist*

Parameter	Description
<b>General</b>	
Developer Name	Xilinx
Target devices	Kintex-7 XC7K325T FPGA
Source code provided	Yes
Source code format	VHDL
Design uses code or IP from existing Xilinx application note reference designs, CORE Generator™ tool, or third party	No
<b>Simulation</b>	
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	ISE Design Suite 14.4
SPICE/IBIS simulations?	No
<b>Implementation</b>	
Synthesis software tools and version used	XST/ ISE Design Suite 14.4
Implementation software tools and versions used	ISE Design Suite 14.4
Static timing analysis performed?	Yes
<b>Hardware Verification</b>	
Hardware verified?	No
Hardware platform used for verification	N/A

Table 6: Device Utilization

Device	Speed Grade	Package	Pre-Map	Post-Route	Slices
XC7K325T	-2	FFG900	243.30 MHz	78.46 MHz	5861 (2%)

## References

The following sites are referenced in this document:

1. [Aerospace and Defense Security Monitor IP Core](#) Product Marketing Brief
2. [Isolation Design Flow](#) website
3. *7 Series Isolation Design Flow Lab Using ISE Design Suite 14.4* ([XAPP1085](#))
4. *Constraints Guide* ([UG625](#))
5. *Hierarchical Design Methodology Guide* ([UG748](#))

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/17/2013	1.0	Initial Xilinx release.
05/23/2013	1.0.1	Replaced the Single Chip Cryptographic (SCC) lounge with the <a href="#">Isolation Design Flow</a> site in <a href="#">Reference Design, page 3</a> and <a href="#">Reference Design Files, page 55</a> .
06/21/2013	1.0.2	Removed link to <a href="#">Isolation Design Flow</a> page in <a href="#">Reference Design, page 3</a> . In <a href="#">Reference Design Files, page 55</a> , added URL to download reference design. Renamed <a href="#">Table 6</a> to "Device Utilization."
09/30/2013	1.1	Added information for Zynq-7000 AP SoC. Under <a href="#">Figure 3</a> , fourth function became <i>fifth</i> function. <a href="#">Figure 10</a> shows an example of SVG output as run on the Kintex-7 reference design. Added <a href="#">Figure 11, Zynq-7000 SVG UCF Output Run (Left) Next to the Actual PlanAhead Tool Floorplan (Right)</a> . Added the Zynq-7000 fence tile requirement to <a href="#">Table 4</a> . Added <a href="#">Figure 23, High-Level PlanAhead Tool View for Zynq-7000 Processor System (PS) Block</a> . Added section <a href="#">Processor System (PS) Block Fence Tile Usage, page 40</a> . Added <a href="#">Figure 30, PlanAhead Tool View for Basic Fence Tile Usage around the PS Block</a> . Added <a href="#">Figure 31, PlanAhead Tool View for More Realistic Fence Tile Usage around the PS Block</a> . Added information to <a href="#">Guidelines for Controlling Inference of Global Clocking Logic, page 50</a> and <a href="#">Gaps in the Floorplan, page 51</a> . Updated the Reference Design link in <a href="#">Reference Design Files, page 55</a> .
12/10/2013	1.2	The second link in <a href="#">Summary</a> was updated. PCIe tile fence equivalence is clarified in <a href="#">PCIe Tile Fence Tile Usage, page 38</a> . In the sentence before <a href="#">Figure 34</a> , I/O <i>single</i> tile changed to I/O <i>pair</i> tile.



Date	Version	Revision
04/18/2014	1.3	In the paragraph before <a href="#">Figure 33</a> , the sentence now mentions that an I/O single tile can be used as part of a vertical fence if the tile is in the middle of the device.
02/05/2015	1.3.1	Updated the title and links.

---

## Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

---

© Copyright 2013–2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.