WP386 (v1.0) February 15, 2011

Hierarchical Design Using Synopsys and Xilinx FPGAs

By: Kate Kelley

Xilinx® FPGAs offer up to two million logic cells currently, and they continue to expand. Hierarchical design is becoming more popular with designs of this complexity because it allows users to preserve completed portions of the design, deliver complex IP place-and-route results, and develop multiple blocks in parallel. These methods can lead to fewer design runs, reduced verification time, and more consistent timing closure, resulting in reduced time to market.

Complex design issues can be addressed using block-based flows where working blocks can be preserved at the netlist level, and optionally at the placement or even the routing level. Unchanged blocks are automatically preserved during synthesis and implementation.

The main benefit of this flow is to reduce the number of implementation iterations during the timing closure phase.

© Copyright 2011 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, 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.

Synopsys Synplify Compile Point Feature and Setup

Compile Points Overview

Using the Synplify compile point synthesis feature, block-based design has been available in Synplify Pro and Synplify Premier tools for several years. Blocks (partitions) are specified at the RTL level prior to synthesis; these points identify areas that can then be compiled and mapped separately. During subsequent runs, the software automatically detects which blocks have changed and recompiles and maps only the changed blocks. With a traditional bottom-up flow — a manual process — design dependencies often make it difficult to put the design together. Automated block-based design affords an easy-to-use bottom-up flow that is devoid of these traditional drawbacks.

Design Practices for Compile Points

In a traditional bottom-up flow, the design is divided into different parts that can be processed independently. This approach has been used when parts of the design need to be isolated to stabilize the results or to isolate the IP blocks. The designer can freeze these portions of the design, while at the same time can work independently (or even in parallel) on the rest of the design.

Another reason for a bottom-up flow is to speed up the design process. And on very large designs (e.g., for designs targeting devices over 500,000 LUTs), a top-down approach is often not ideal due to memory and run-time limits.

Compile point synthesis lets the user define design blocks as needed to compose the ideal mix of incremental synthesis and to change that mix at any time during the design process. The designer can choose which blocks of the FPGA can be synthesized independently of each other. The Synplify Pro and Synplify Premier tools treat each compile point as an independent block, which allows the designer or other team members to work separately on other parts of the design.

Constraints are not automatically budgeted for the compile point block; therefore, manual time budgeting — the creation of constraints for each block — is required. Since compile point constraints directly affect the quality of results, it is key to provide timing constraints for each compile point block.

For best results when using compile point and design preservation flows, a design change in one module must *not* affect any preserved modules. This is accomplished by rendering the module boundary "hard" during synthesis and implementation; this means that there is no optimization during synthesis or implementation across this hard boundary.

To get the best QoR and utilization, several design guidelines must be followed closely:

- Inputs and outputs of modules should be registered with no combinatorial logic between the registers and module boundaries. This prevents a path that crosses a partition boundary from being critical because there is no logic optimization across boundaries. If it is not possible to register both inputs and outputs, it is better to register the outputs.
- Having constants as inputs or outputs to a compile point is not recommended because of the impact this has on QoR. Constants are not propagated across boundaries. This can cause utilization and QoR issues if input constants are used to disable unused design features in a module. In a flat flow, unused logic is

optimized away, but this does not occur when compile points are used.

- Active-Low control signals on resets and clock enables must be used with caution. Because logic cannot cross a boundary, even inverters on control signals can be affected. This is especially critical for reset and clock enables because no local inverters exist on these pins. If the design uses an active-Low control signal, the signal should be inverted in the top level and then the active-Low signal can be used throughout the design.
- Unused inputs or outputs in a module with compile points must be avoided. Because the boundaries are hard, inputs unused inside the module or outputs unconnected to any logic outside the module are *not* optimized away. This can lead to packing errors or higher utilization. If it is not possible to remove unused inputs or outputs from a module, then a wrapper that contains only the used inputs and outputs can be created. The compile point should be on this wrapper, and the unused inputs and outputs of the critical module can then be optimized away.

Compile points and partitions follow the logical hierarchy of the design. Some common rules for creating good hierarchy for compile points include:

- Logic that needs to be optimized, implemented, and verified together should be kept in the same hierarchy. All designs have hierarchy, but to attain maximum performance, the hierarchy should be defined in the context of the FPGA layout. This might require adding additional levels of hierarchy so that two critical modules can be synthesized and implemented in the same partition.
- Logic that needs to be packed together should be kept in the same level of hierarchy. This includes registers that need to be packed in larger components like block RAM and DSP. This rule also affects I/O logic that needs to be packed together.
- The number of compile points and partitions is also important. Too many compile points can degrade performance because there are no optimizations across boundaries. In one design, for example, there might be just one compile point on a critical portion of the design; other designs might have five to ten compile points. The ideal number of compile points is design dependent. The best candidate for a compile point is a module or core that is expected to have limited/no changes or difficult-to-meet timing.

Synplify's Compile Point Flow

User-defined compile points can be used on any FPGA whether the design is small or very large. Since a compile point is an independent synthesis unit, it is up to the user to determine how many and how large the compile points are.

There are a few principle reasons why a designer should consider using compile points during the FPGA design process:

- Memory consumption is smaller (compared to top-down processing without compile points).
- Incremental synthesis maintains design stability.
- A run-time advantage is realized because incremental synthesis is executed only on those blocks that have changed.
- A run-time advantage is realized for multiple instantiations of a compile point.
- Synplify automatically determines which compile points should be re-synthesized based on design changes.

- Synplify Compile Points can be used in conjunction with the new Xilinx Design Preservation flow, which improves the repeatability of results.

If the designer's only goal is synthesis run-time savings, Synplify Pro and Synplify Premier software include an automatic compile points flow, which automatically divides the design into compile point blocks for faster parallel synthesis on multiple processors. This flow has minimal impact on QoR because interface time budgeting is performed for the compile points. However, this automatic compile point flow is not compatible with the Xilinx Design Preservation Flow.

Resynthesizing Compile Points

Synplify Pro and Premier determine which compile points need to be resynthesized based on the design changes that occurred.

The following cause a compile point to be resynthesized:

- Changing an RTL design
- Changing the compile point SDC file
- Changing any synthesis option
- Adding or removing a compile point

The following *do not* cause a compile point to be resynthesized:

- Adding comments to the HDL file
- Changing the timestamp of a design file
- Changing the top level SDC file

Synplify Setup for Compile Points

This setup procedure is designed to be used with the new Xilinx ISE® software Design Preservation flow.

For the FPGA design flow to take advantage of the Design Preservation feature, **Use Xilinx Partition Flow** must be enabled in the Synplify **Option** menu on the **Device** tab, as shown in Figure 1.

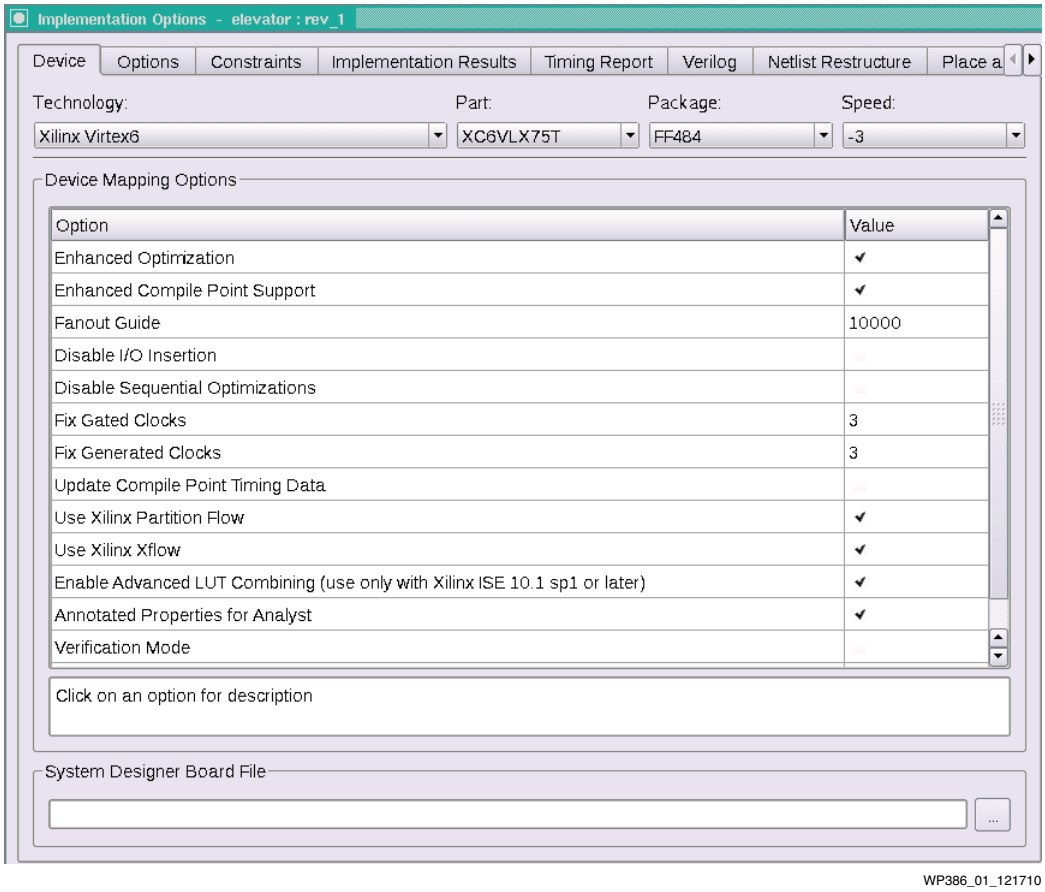

Figure 1: **Synthesis Implementation Options**

Next, the compile points for the design must be identified and set up. The design must be compiled first so that the tool has the needed database to identify potential compile points. New compile points can be defined through the Pro/Premier SCOPE GUI, where the setup is automated. The first step is to create a new constraint file using SCOPE and choose **Compile Point** as the file type, as shown in Figure 2.

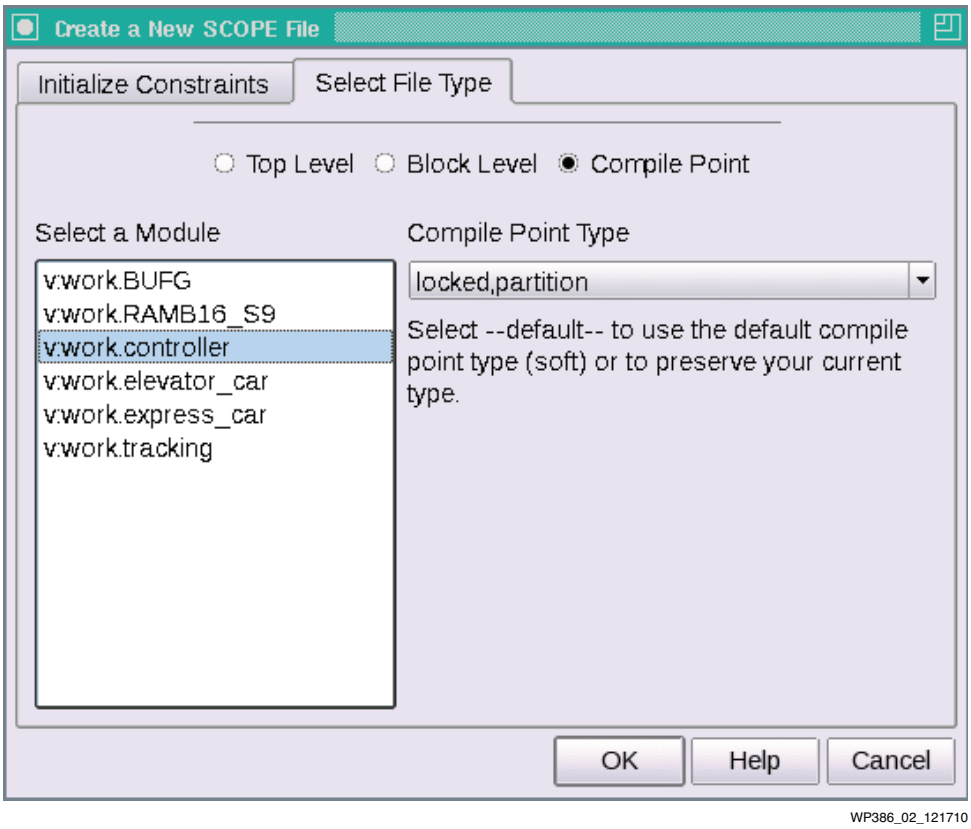

Figure 2: **Create a New Scope File**

Because the design is compiled prior to this step, all available modules in the design are listed, and any of them can be chosen by the user as new compile points.

Multiple compile point types are available to synthesis users. For the Xilinx Design Preservation flow, however, only two types are supported: 1) *locked, partition* and 2) *locked*. The *locked, partition* type is available for backward compatibility to the older Xilinx Project Navigator partition flow. Xilinx strongly recommends that the new partition flow in PlanAhead™ software v12.1 and later be used. Both compile point types are available for the PlanAhead software v12.1 and later releases.

When the compile point is defined, the Synplify tool automatically adds a new constraint file corresponding to the newly created compile point. It is the user's responsibility to add any relevant constraints (such as PERIOD constraints) to the specific compile points to maintain quality of results.

The Synplify tools give detailed reports on the individual compile points during synthesis. The reports include data indicating if individual compile points were resynthesized due to source file changes. If there was no need to resynthesize a specific compile point, the report shows it as `Unchanged`, as shown in Figure 3.

```
Summary of Compile Points :
***************************
Name              Status         Reason
-----------------------------------------------
controller        Remapped       Design changed
elevator_car      Unchanged      -
express_car       Unchanged      -
tracking          Unchanged      -
top               Unchanged      -
===============================================
```

WP386_03_021011

Figure 3: **Summary of Compile Points**

Because the **Use Xilinx Partition Flow** option was set during synthesis, Synplify can create additional data (`xpartition.pxml` file) needed for the ISE software to take advantages of its design preservation flow. This file can be created in the TCL console by using the following command:

```
sxml2pxml -design_name elevator -idir /project/syn/rev_1 -odir
/project/implementation/par_1
```

This file can also be created in a script file by using the Synplify batch command. The above TCL command must be included in a TCL script:

```
synplify_pro -batch script.tcl
```

The command must be run in batch mode with the TCL command using the **`-tclcmd`** option. For example:

```
synplify_pro -batch -tclcmd "sxml2pxml -design_name <value> [-idir <value>
-odir <value>]"
```

Xilinx Design Preservation Flow and Setup

Benefits of Design Preservation Flow

The main benefit of Design Preservation is to reduce the number of implementation iterations during the timing closure phase. When timing has been met on a portion of the design, the implementation results (placement and routing) are used in the next iteration. This prevents portions of the design that previously met timing from failing timing in the current run.

A second benefit of Design Preservation is to reduce time during the verification phase. Since the same implementation is used, it is not necessary to do a full verification on modules that have not changed.

Reducing the implementation run time is not a primary goal but is often a secondary benefit. The implementation run time changes for each design run depends on which modules are being implemented. If the changed module has very tight timing requirements, run time might be longer. If the changed module meets timing easily and the critical paths are in preserved modules, run time might be shorter.

Design Preservation Flow Impact

As long as the design guidelines specified in Synopsys Synplify Compile Point Feature and Setup are followed, there is little QoR impact to the design. The positive impact is repeatable QoR for the unchanged compile points, which can greatly reduce the timing closure phase.

Design Preservation Flow Setup

Existing design scripts can be used with compile points and partitions. During synthesis, the compile points define which modules are to be used as partitions. During implementation, an `xml` file called `xpartition.pxml` defines the partitions and specifies if they should be imported or implemented. Figure 4 is an example of a simple `pxml` file. This file is created by Synplify as described in Synplify Setup for Compile Points.

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- File written by the Synopsys Synplify-->
<!-- Users may only modify the Project Name attribute below -->

<Project  FileVersion="1.2" Name="TOP_SETUP" ProjectVersion="2.0" >
    <Partition  Name="/top" State="implement" ImportLocation=".">
      <Partition  Name="/top/Tracking_Module" State="implement" ImportLocation="."/>
      <Partition  Name="/top/Express_Car" State="implement" ImportLocation="."/>
      <Partition  Name="/top/Main_Car" State="implement" ImportLocation="."/>
      <Partition  Name="/top/Control_Module" State="implement" ImportLocation="."/>
    </Partition>
</Project>
```

WP386_04_021011

Figure 4: **`Xpartition.pmxl` File**

The user decides if the Xilinx implementation tools use a command line (or batch) flow or use a PlanAhead tool flow. In the command line flow, the `pxml` file generated by Synplify is used to control the partitions. In the PlanAhead tool flow, the user imports a `pxml` file to define the partitions, but further control (for example, what to implement versus what to import) is controlled by the user.

Implementation via Command Line

After the `pxml` file is created by Synplify, the Xilinx implementation tools can be run. The `pxml` file must be copied to the implementation directory and then the Xilinx tools or any existing batch file can be run. Synplify has set the partition properties to implement or import based on which compile points were synthesized. The same timing closure techniques are used with partitions as without. There are a few exceptions; the map options for global optimization, the high and extra high options for power, and SmartGuide are not compatible with partitions.

Implementation via PlanAhead Tool

In ISE Design Suite 12.2 and later, the initial `pxml` file can be imported into PlanAhead tool to define the partitions, but after the project is created, management of the partitions is done manually. After the `pxml` file is imported, the design is implemented in the standard way.

After the design is implemented, partitions can be promoted for use in the next run, as shown in Figure 5. The user selects which partitions should be promoted and the PlanAhead tool manages the rest. Promoted partitions are automatically set to Import on the next run.

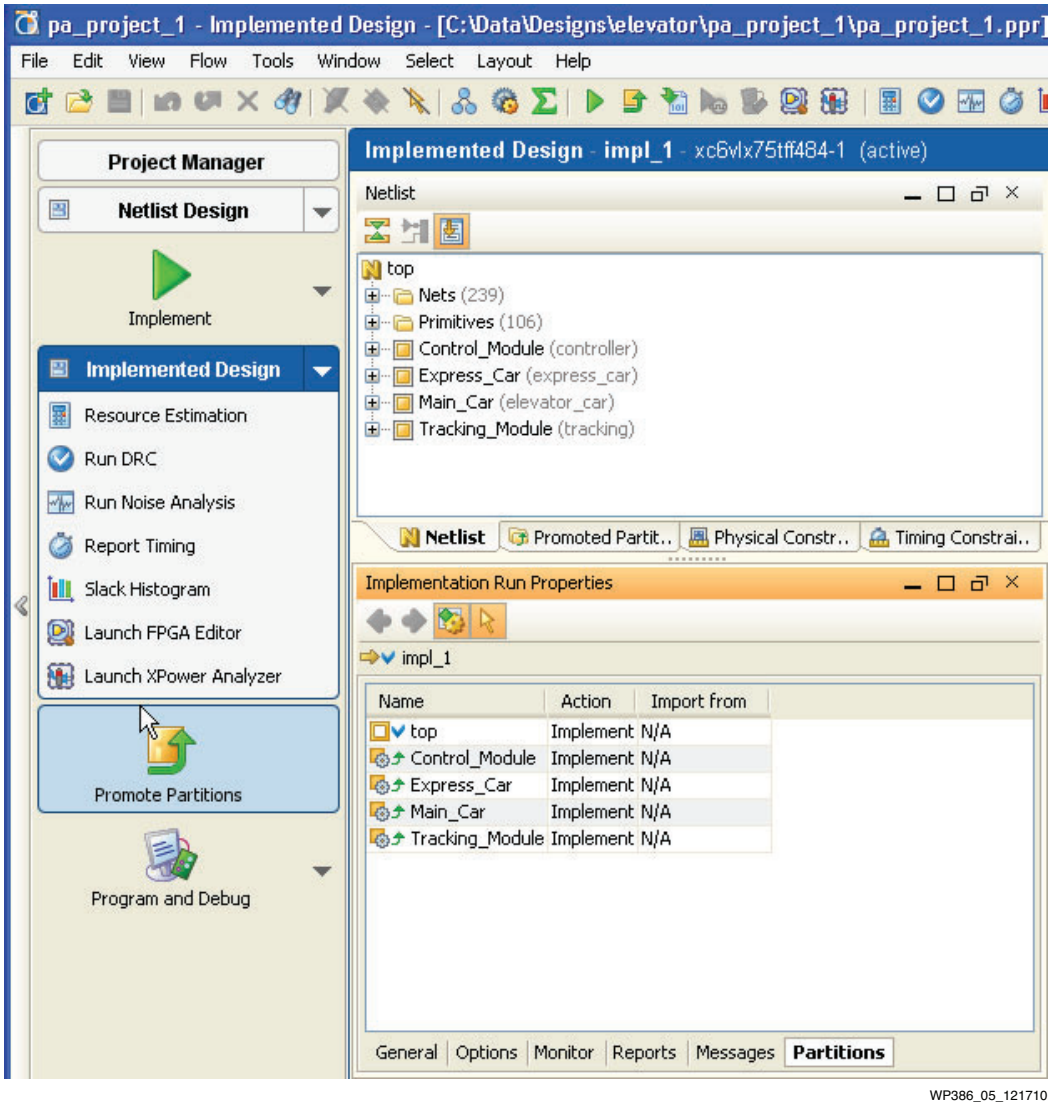

Figure 5: **Promoting Partitions**

After a design change, any partitions associated with changed compile points must be set to Implement. This is done in the **Partitions** tab in the Implementation Run Properties window, as shown in Figure 6.

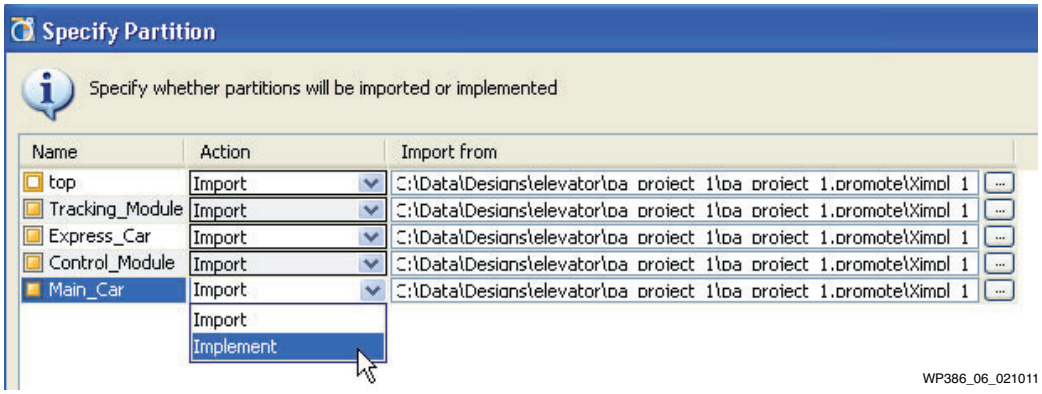

Figure 6: **Partition State in the Implementation Run Properties Window**

For partitions that are set to Import, the placement and routing information is copied and pasted into the new design. Partitions that are set to Implement are placed and routed around the imported partitions.

Whether the command line/batch flow or the PlanAhead tool flow is chosen, there are partition reporting sections in the ngdbuild, map, and par reports, as shown in this example:

```
Partition Implementation Status
-------------------------------

  Preserved Partitions:

    Partition "/top/Express_Car"

    Partition "/top/Main_Car"

    Partition "/top/Tracking_Module"


  Implemented Partitions:

    Partition "/top":
Attribute STATE set to IMPLEMENT.

    Partition "/top/Control_Module":
Attribute STATE set to IMPLEMENT.

-------------------------------
```

Conclusion

The use of Synplify's compile points in conjunction with the Xilinx Design Preservation flow helps solve complex design issues. By working on one module and then reusing the synthesis and implementation results in the next iteration, these flows allow a user to focus on solving one issue at a time. Used together, these two flows reduce the number of required iterations through the implementation tools and improve the *repeatability of results* when targeting Xilinx FPGAs. To take advantage of these benefits, however, it is important to carefully consider the design's hierarchy during the RTL design phase.

Related Documents

UG748, *Hierarchical Design Methodology Guide*

WP362, *Repeatable Results with Design Preservation*

Synopsys FPGA Synthesis User Guide

Synopsys FPGA Synthesis Reference Manual

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
02/15/11	1.0	Initial Xilinx release.

Notice of Disclaimer

The information disclosed to you hereunder (the "Information") is provided "AS-IS" with no warranty of any kind, express or implied. Xilinx does not assume any liability arising from your use of the Information. You are responsible for obtaining any rights you may require for your use of this Information. Xilinx reserves the right to make changes, at any time, to the Information without notice and at its sole discretion. Xilinx assumes no obligation to correct any errors contained in the Information or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE INFORMATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS.